

Инструменты разработки

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

Понятие рабочей среды	9
Среда разработки	9
Предпромышленная рабочая среда	12
Промышленная рабочая среда	12
Контроль версий в Creatio IDE	12
Установка пакета из хранилища SVN	13
Обновление пакета из хранилища SVN	17
Фиксация пакета в хранилище SVN	18
Принципы разработки в Creatio IDE	20
Настроить доступ к разделу Конфигурация	20
Открыть раздел Конфигурация	21
Разработка конфигурации	21
Закрыть раздел Конфигурация	35
NLog	35
Библиотека логирования NLog	35
Настройка логирования в Creatio	35
Логирование для приложений on-site	35
Логирование для приложений cloud	38
Настроить логирование NLog	38
Настройка логирования через конфигурационный файл	38
Настройка логирования через конфигурационный объект LoggingConfiguration	39
Back-end отладка	39
Выполнение back-end отладки	39
Возможные проблемы при отладке	40
Выполнить отладку серверного кода	42
1. Выполнить настройку приложения	43
2. Выгрузить исходные коды конфигурации	43
3. Создать проект Visual Studio для отладки	44
4. Добавить в проект файлы с исходным кодом	45
5. Подключить проект к рабочему процессу IIS	46
6. Выполнить отладку	47
Разработка конфигурационных элементов	49
Клиентский модуль	50
Объект	56
Исходный код	65
Front-end отладка	66

Интегрированные инструменты отладки	66
Скрипты и точки останова	67
Управление выполнением отладки	67
Использование консоли браузера	68
Режим клиентской отладки isDebug	73
Контроль версий в Subversion	76
Основные понятия	77
Модели версионирования	78
Работа с файлами в SVN	80
Рабочая копия, используемая приложением Creatio	80
Клиентское приложение для работы с SVN	81
Создать пакет в режиме разработки в файловой системе	81
1. Создать пакет	81
2. Выгрузить пакет в файловую систему	82
3. Создать каталоги для пакета в хранилище SVN	83
4. Создать рабочую копию версионной ветки пакета	84
5. Зафиксировать пакет в хранилище	85
Установить пакет из SVN в режиме разработки в файловой системе	87
Вручную установить пакет из SVN в режиме разработки в файловой системе	88
Автоматически установить пакет из SVN в режиме разработки в файловой системе	93
Настроить взаимодействие с хранилищем SVN (опционально)	94
Привязать к SVN не связанный с хранилищем пакет	94
1. Выгрузить пакет в файловую систему	95
2. Создать каталоги для пакета в хранилище SVN	95
3. Создать рабочую копию версионной ветки пакета	96
4. Зафиксировать в хранилище каталог пакета	98
Привязать к SVN не связанный с хранилищем пакет через запрос к базе данных	99
1. Подключите к приложению хранилище SVN	99
2. Привязать хранилище SVN к пакету	100
3. Выгрузить пакет в файловую систему	101
Обновить и зафиксировать пакет в SVN в режиме разработки в файловой системе	102
1. Обновить пакет из хранилища SVN	102
2. Изменить содержимое пакета	103
3. Зафиксировать пакет в хранилище	104
Создать пакет при переходе в режим разработки в файловой системе	105
1. Задать путь к каталогу для рабочих копий пакетов	106
2. Создать пакет	106
3. Выгрузить пакет в файловую систему	107
4. Зафиксировать пакет в хранилище	108

Логирование входящих http-запросов в Creatio для .NET Core	109
Стандартное логирование входящих http-запросов	110
Расширенное логирование входящих http-запросов	110
Процесс управления поставками	111
1. Разработать новую функциональность	112
2. Экспортировать пакет в *.zip-архив	112
3. Импортировать пакет в предпромышленную среду	113
4. Создать резервную копию базы данных промышленной среды	113
5. Импортировать пакет в промышленную среду	113
Пакет-проект	113
Особенности пакета-проекта	114
Структура пакета-проекта	114
Инструменты для разработки пакета-проекта	114
Импортировать пакет-проект	115
Внешние IDE	115
Разработка в файловой системе	116
Настроить Creatio для работы в файловой системе	116
Работа с пакетами	119
Разработка C# кода	120
Разработка JavaScript кода	127
Разработать C# код в конфигурационном проекте	130
Алгоритм реализации примера	131
Разработать C# код в пользовательском проекте	136
Предварительные настройки	136
Разработка C# кода для on-site приложения	137
Разработка C# кода для cloud приложения	142
Разработать клиентский код	145
Последовательность действий при разработке исходного хода клиентских схем в файловой системе	145
Автоматическое отображение изменений клиентского кода	148
Последовательность настройки автоматического отображения изменений	149
Общие принципы работы с пакетами	152
Классификация пакетов	152
Структура пакета	154
Зависимости и иерархия пакетов	155
Привязка данных к пакету	161
Создать пользовательский пакет	161
1. Создать пакет	161
2. Заполнить свойства пакета	162
3. Определить зависимости пакета	163

4. Проверить зависимости пакета Custom	164
Привязать данные к пакету	164
1. Создать раздел	164
2. Добавить в раздел демонстрационные записи	165
3. Привязать к пакету данные	166
4. Проверить привязки данных	169
Контроль версий в Git	170
Особенности работы с Git в Creatio	171
Общая последовательность работы в Git	171
Управление поставками в Creatio IDE	176
Перенос пакета	177
Перенос пакета с использованием SVN	177
Перенос схемы конфигурационного элемента	178
Перенести пакеты	178
Экспортировать пакет	179
Импортировать пакет	180
Перенести пакеты с использованием SVN	183
1. Подключить хранилище системы контроля версий SVN	183
2. Включить автоматическое применение изменений	184
3. Проверить привязки данных	185
4. Проверить возможность переноса зависимостей пакета	185
5. Установить пакет из хранилища SVN	185
Перенести схемы конфигурационных элементов	185
Экспортировать схему конфигурационного элемента	185
Импортировать схему конфигурационного элемента	186
Файловый контент пакетов	187
Структура хранения файлового контента пакета	188
Bootstrap-файлы пакета	189
Версионность файлового контента	190
Генерация вспомогательных файлов	191
Предварительная генерация статического файлового контента	191
Генерация файлового контента	192
Совместимость с режимом разработки в файловой системе	194
Перенос изменений между рабочими средами	195
Локализовать файловый контент с помощью конфигурационных ресурсов	195
1. Создать модуль с локализуемыми ресурсами	195
2. Импортировать модуль локализуемых ресурсов	196
Локализовать файловый контент с помощью плагина i18n	196
1. Добавить плагин	196

2. Создать папку с локализуемыми ресурсами	196
3. Создать папки культур	197
4. Добавить файлы с локализуемыми ресурсами	197
5. Отредактировать файл bootstrap.js	198
6. Использовать ресурсы в клиентском модуле	199
Использовать TypeScript при разработке клиентской функциональности	199
1. Установить TypeScript	200
2. Перейти в режим разработки в файловой системе	200
3. Создать структуру хранения файлового контента	201
4. Реализовать валидацию на языке TypeScript	202
5. Выполнить компиляцию исходных кодов TypeScript в исходные коды JavaScript	203
6. Выполнить генерацию вспомогательных файлов	205
7. Проверить результат выполнения примера	205
Создать Angular-компонент для использования в Creatio	208
Создание пользовательского Angular-компонента	208
Подключение Custom Element в Creatio	212
Работа с данными	213
Использование Shadow DOM	215
Управление поставками в WorkspaceConsole	217
Настройка утилиты WorkspaceConsole	217
Использование утилиты WorkspaceConsole	219
Экспортировать пакеты из базы данных	228
1. Сформировать команду для экспорта пакета из базы данных	229
2. Экспортировать пакет из базы данных	229
Экспортировать пакет из SVN	230
1. Сформировать команду для экспорта пакета из SVN-хранилища	231
2. Экспортировать пакет из SVN	231
Импортировать пакет в базу данных	232
1. Сформировать команду для импорта пакета в базу данных	233
2. Импортировать пакет в базу данных	233
3. Сформировать команду для генерации статического контента в файловую систему	234
4. Сгенерировать статический контент в файловую систему	234
Параметры утилиты WorkspaceConsole	235
NUnit	242
Добавить тест для пользовательского класса	243
1. Установить адаптер NUnit для Visual Studio	243
2. Выгрузить пакет в файловую систему	245
3. Настроить проект Unit-тестов	246
4. Создать тесты	247

5. Выполнить тестирование	249
Логирование. log4net	249
Хранение логов	250
Уровни логирования	251
Настройка логирования	251

Понятие рабочей среды



Основы

Рабочая среда представляет собой отдельное приложение Creatio со своей базой данных. Рабочая среда может быть дополнена системой контроля версий. **Назначение** рабочих сред — обеспечение поставок новой функциональности на разных этапах ее жизненного цикла: разработка, тестирование и использование.

Виды рабочих сред:

- Среда разработки.
- Предпромышленная рабочая среда.
- Промышленная рабочая среда.

Среда разработки

Среда разработки (Development Environment) — отдельное приложение или несколько приложений Creatio, в которых выполняется разработка новой функциональности.

Для фиксации изменений рекомендуется дополнить среду разработки [системой контроля версий](#).

Важно. Систему контроля версий SVN разрешено использовать для переноса изменений только между [средами разработки](#). Запрещено использовать SVN на [предпромышленной](#) и [промышленной](#) среде, поскольку это может привести к неработоспособности или ухудшению производительности приложения. Подробнее читайте в статье [Контроль версий в Subversion](#).

Используйте систему контроля версий [Git](#), если:

- планируется вести разработку в **файловой системе**;
- используется **on-site** приложение.

Используйте систему контроля версий [Subversion \(SVN\)](#), если:

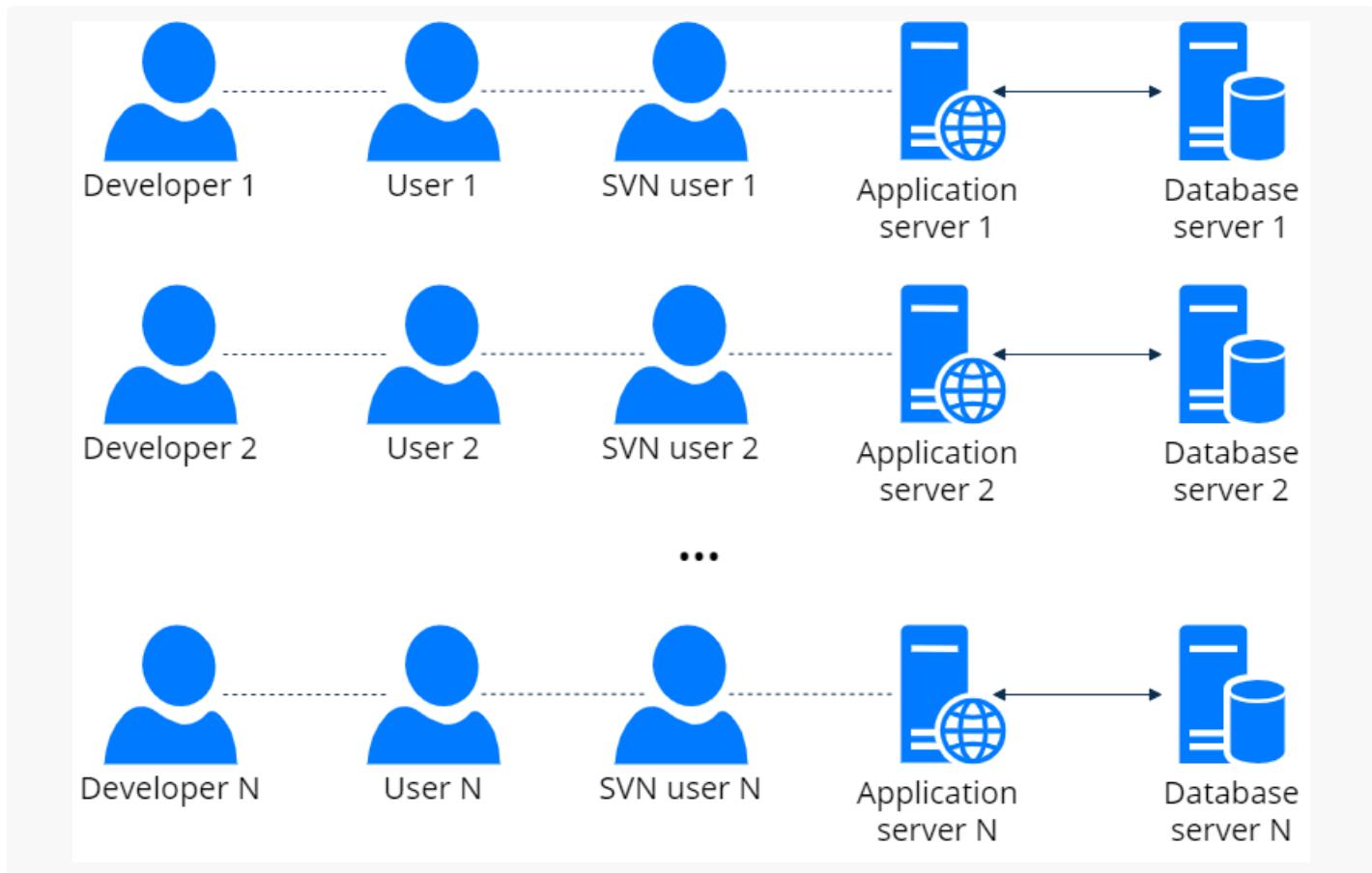
- планируется вести разработку **low-code** инструментами;
- используется **cloud** приложение.

Использование системы контроля версий SVN доступно только для приложения Creatio на платформе **.NET Framework**.

Настройка систем контроля версий описана в разделе [Системы контроля версий](#).

Для среды разработки рекомендуется выбрать [on-site развертывание](#).

При **on-site развертывании** для каждого разработчика разворачивается on-site приложение и база данных.



Варианты **cloud развертывания** приложения:

1. Воспользоваться [страницей создания пробной версии](#). На протяжении 14-дневного пробного периода можно ознакомиться с основными возможностями приложения. По завершению пробного периода используемая демоверсия приложения может быть перенесена на основную площадку Creatio.
2. Обратиться к менеджеру по продажам Creatio о необходимости развернуть новое приложение в облаке или перенести приложение клиента в облачный сервис Creatio. После согласования условий с менеджером соответствующее подразделение компании выполнит необходимые работы.

При создании облачных приложений Creatio на площадке Terrasoft необходимо учитывать некоторые ограничения. Если эти требования не будут выполнены, продукт не может быть развернут.

1. Запрещено использование SQL Agent

Нельзя создавать задания (Jobs) и другие действия, выполняемые SQL Agent. Вместо этого необходимо использовать [планировщик заданий Creatio](#).

2. Запрещено использование DB Mail

Отправку Email-уведомлений необходимо делать с использованием возможностей платформы Creatio.

3. Запрещено использование Extended Stored Procedure

Всю необходимую логику необходимо реализовывать или с использованием стандартных хранимых процедур на языке T-SQL, или с использованием возможностей сервера приложений.

4. Запрещена привязка к именам пользователей СУБД

В СУБД на площадке Terrasoft не создаются пользователи базы данных. Вместо этого используются

доменные пользователи и доменная аутентификация.

5. Запрещено изменение файла `Web.config` приложения

Все необходимые параметры необходимо хранить в системных настройках приложения Creatio.

6. Запрещена привязка к IP-адресам серверов приложений и СУБД

IP-адреса серверов могут меняться. Поэтому привязываться к ним нельзя. Необходимо всегда работать с доменными именами приложений.

7. Запрещена установка дополнительного ПО

Никакое дополнительное программное обеспечение не может быть установлено на серверах облачного сервиса Terrasoft.

8. Запрещена работа с файловой системой

Работа с файловой системой для сервера приложений и СУБД ограничена правами доступа, которые настроены в операционной системе. Вместо этого необходимо работать с файлами с использованием протоколов FTP и HTTP(S).

9. Запрещен запуск сторонних приложений на сервере

Возможность запуска сторонних приложений ограничена правами доступа, которые настроены в операционной системе. Вся необходимая логика должна быть реализована в приложении.

10. База данных должна работать на SQL Server 2016 и выше.

Для обеспечения совместимости с облачной инфраструктурой площадки Terrasoft предоставляемая база данных приложения должна быть создана в SQL Server 2016 и выше.

11. Приложение должно работать как по протоколу HTTP, так и HTTPS

Нельзя использовать логику, связанную с использованием конкретного протокола. Вместо этого необходимо определять текущий протокол приложения.

12. Приложение должно работать с правами обычного пользователя

Нельзя использовать функции, требующие административных привилегий.

13. Приложение должно работать от имени пользователя без профиля

На площадке создаются пользователи, не имеющие возможности фактического логина в ОС и не имеющие профиля.

Дополнительные рекомендации:

- В качестве значения системной настройки [*Издатель*] (код [*Maintainer*]) необходимо установить название партнера, например `FineSolution`.
- Значение системной настройки [*Префикс названия объекта*] (код [*SchemaNamePrefix*]) должно характеризовать партнера, например, `FS`.
- Решение не должно использовать замещение модулей. Замещать можно только схемы.
- Серверная логика должна быть сосредоточена в C# классах и вызываться в нужных местах.
- Public API серверных классов и клиентских схем должен быть покрыт unit-тестами.
- Все необходимые данные, скрипты, библиотеки должны быть прикреплены к пакету.
- Разработка продукта должна вестись с использованием системы контроля версий и все пакеты должны храниться в системе контроля версий.

Для разработки сложных проектных решений можно воспользоваться рекомендациями, представленными в документации [Обзор методологии Project Life Cycle](#).

Предпромышленная рабочая среда

Предпромышленная рабочая среда (Pre-Production Environment) — отдельное приложение, в котором выполняется тестирование функциональности, разработанной в среде разработки. Как правило, тестирование выполняется аналитиком группы разработки или заказчиком функциональности. Для предпромышленной среды разработки можно выбрать [on-site](#) или cloud развертывание приложения.

Промышленная рабочая среда

Промышленная рабочая среда (Production Environment) — отдельное приложение Creatio, которое используется в повседневной работе. **База данных** промышленной среды идентична базе данных предпромышленной среды. Поскольку разработка практически всегда сопряжена с возникновением ошибок, их обнаружением, отладкой приложения, компиляцией и т. д., то запрещается вести разработку в промышленной среде.

Для промышленной среды разработки можно выбрать [on-site](#) или cloud развертывание приложения:

- При выборе варианта развертывания **cloud** приложения настройка промышленной среды не отличается от настройки среды разработки.
- При выборе варианта развертывания **on-site** приложения настройка промышленной среды не отличается от настройки предпромышленной среды.

Контроль версий в Creatio IDE



Легкий

Creatio IDE предоставляет функциональность для работы с системой контроля версий SVN.

Важно. Систему контроля версий SVN разрешено использовать для переноса изменений только между [средами разработки](#). Запрещено использовать SVN на [предпромышленной](#) и [промышленной](#) среде, поскольку это может привести к неработоспособности или ухудшению производительности приложения. Подробнее читайте в статье [Контроль версий в Subversion](#).

Использование [встроенных средств](#) Creatio IDE позволяет:

- Устанавливать пакет из хранилища SVN.
- Обновлять пакет из хранилища SVN.
- Фиксировать пакет в хранилище SVN.

Работать с системой контроля версий SVN встроенными средствами Creatio IDE невозможно при включенном [режиме разработки в файловой системе](#). По умолчанию режим разработки в файловой системе отключен.

Установка пакета из хранилища SVN

Установка пакета — это процесс добавления пакета и всех его зависимостей из хранилища системы контроля версий SVN.

Устанавливать пакет необходимо в следующих **случаях**:

- Одновременная работа нескольких разработчиков над функциональностью пакета.
- Перенос изменений между [рабочими средами](#).

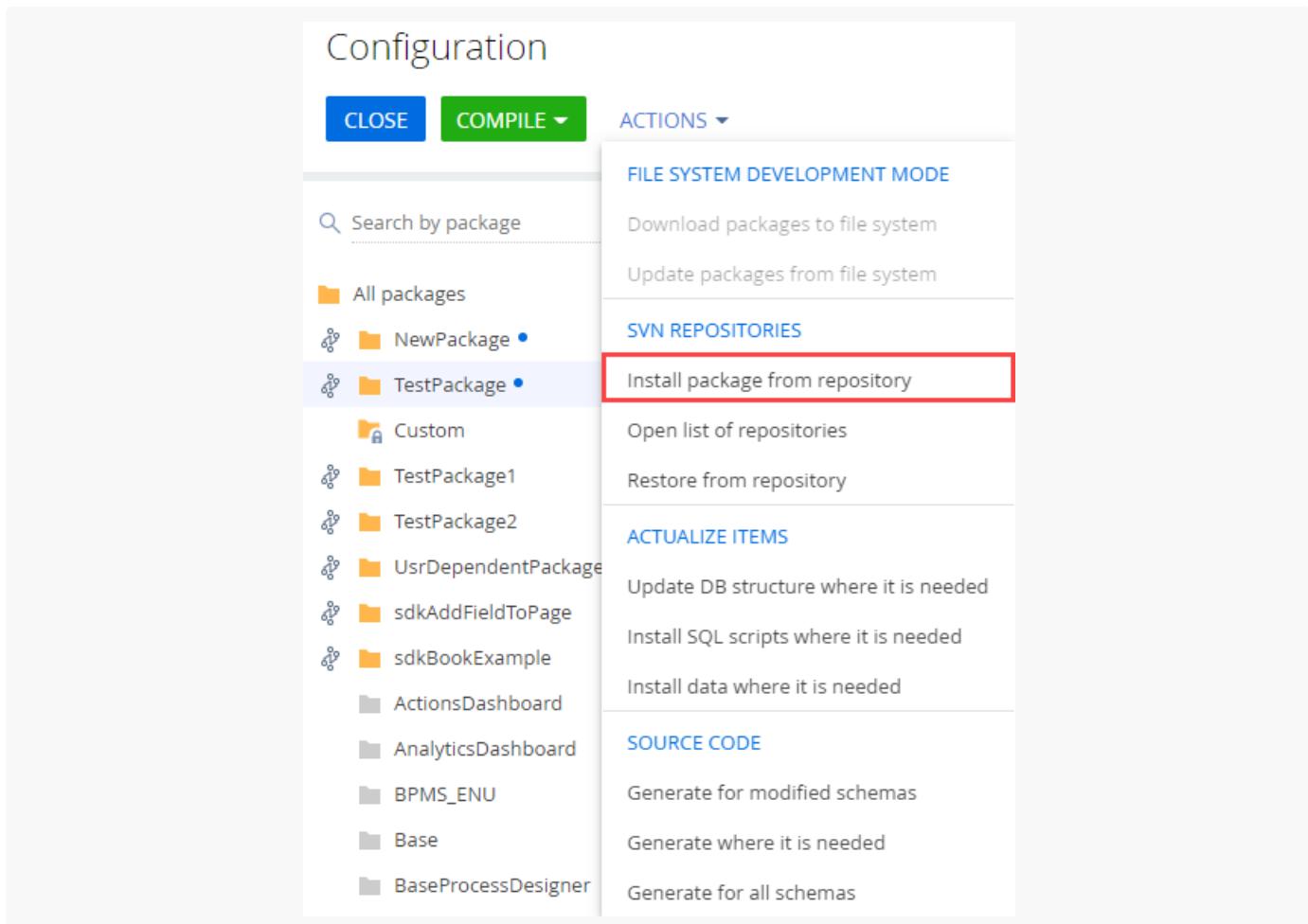
При использовании **cloud приложения** для установки пакета рекомендуется обратиться в службу поддержки Creatio.

Чтобы **установить пакет из хранилища SVN** при использовании on-site приложения:

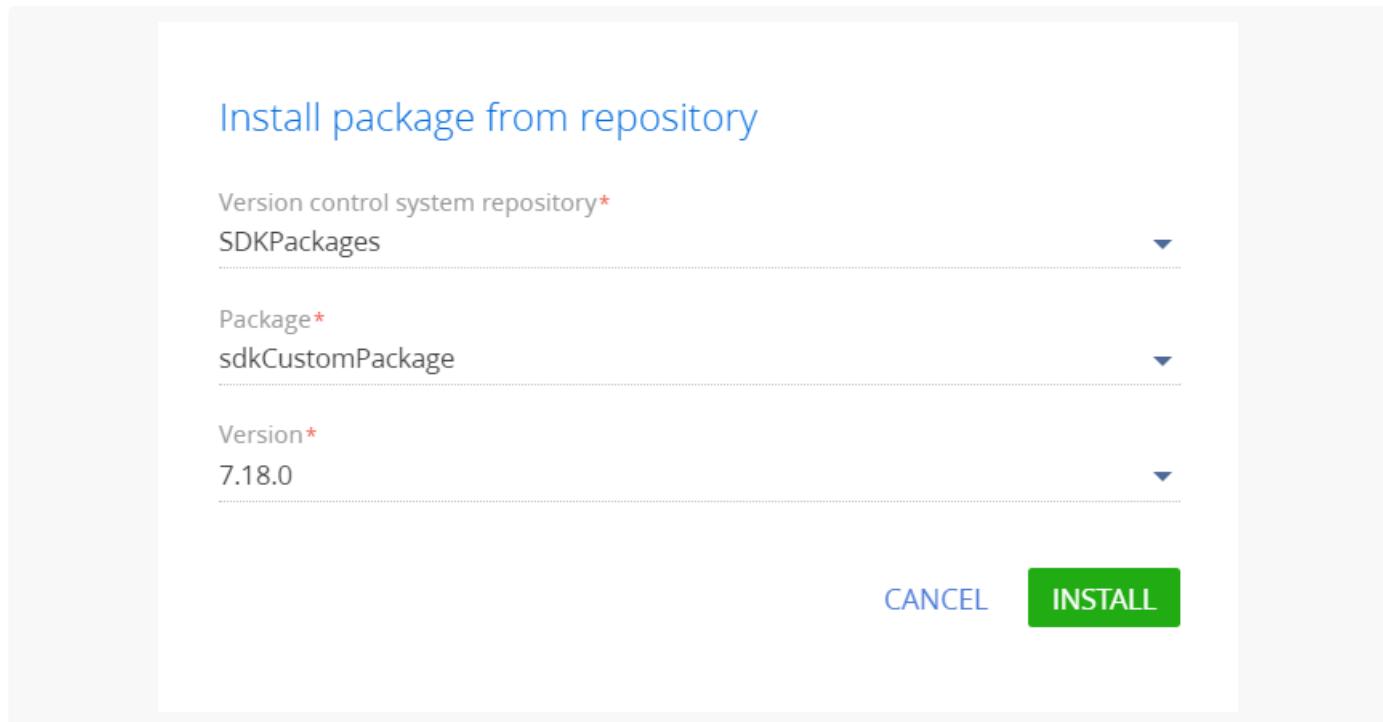
1. Выполните резервное копирование базы данных.

Резервное копирование базы данных необходимо выполнить, поскольку возможность возврата на предыдущую версию посредством системы контроля версий SVN не предусмотрена.

2. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]).
3. В меню действий в группе [Хранилища SVN] ([SVN repositories]) выберите [Установить пакет из хранилища] ([Install package from repository]).



4. В появившемся окне выберите хранилище SVN, название и версию устанавливаемого пакета, после чего нажмите кнопку [Установить] ([Install]).



Во время установки пакета будут автоматически применены привязанные данные, а также будут установлены зависимости.

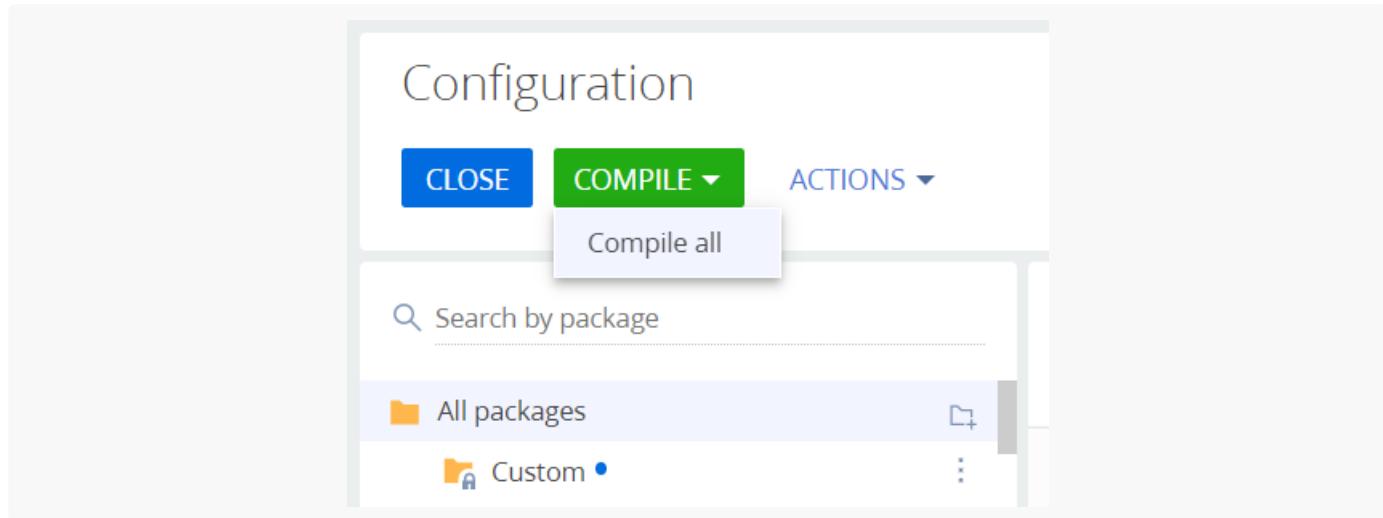
В некоторых случаях изменения могут не применяться автоматически. Если это произошло, их нужно применить вручную.

Чтобы **вручную применить изменения** для установленного пакета:

- a. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]).
- b. Сгенерируйте исходные коды для элементов, требующих генерации.
- c. Выполните компиляцию.
- d. Обновите структуру базы данных.
- e. При необходимости установите SQL-скрипты.
- f. Установите привязанные данные.

На заметку. Необходимость обновления структуры базы данных, установки SQL-скриптов и привязанных данных отображается в свойствах [Требует установки в БД] ([Needs to be installed in database]) и [Требуется обновление в БД] ([Needs to be updated in database]). В случае возникновения ошибок будет отображено всплывающее окно.

5. Выполните действие [Компилировать все] ([Compile all]). Это необходимо для [генерации статического контента](#).



При установке пользовательского пакета система проверяет его зависимости и дополнительно устанавливает либо обновляет все пакеты-зависимости текущего пакета. Например, при установке из хранилища SVN пакета [*sdkCustomPackage*] также будет установлен и пакет-зависимость [*sdkDependentPackage*], который до этого еще не был установлен в рабочее пространство. При этом будет изменена иерархия пакетов в приложении. Если пакет [*sdkCustomPackage*] был установлен ранее, то он будет изменен.

Changes

UsrEntitySchema	Schema	Added
ExternalAssembly.dll	External Assembly	Added
UsrCustomSQLScript	SQL script	Added
UsrEntitySchema	Data	Added
UsrClientUnitSchema	Schema Resource	Added
UsrEntitySchema	Schema Resource	Added
UsrEntitySchema	Schema Resource	Added
sdkDependentPackage	Package	Added

CLOSE

Процесс выполнения изменений в иерархии пакетов при установке пользовательского пакета из хранилища SVN:

1. Определяются зависимости устанавливаемого пакета, которые указаны в его метаданных в свойстве `DependsOn`.

Метаданные пакета

```
{
  "Descriptor": {
    "UIId": "8bc92579-92ee-4ff2-8d44-1ca61542aa1b",
    "PackageVersion": "7.18.0",
    "Name": "sdkCustomPackage",
    "ModifiedOnUtc": "\/Date(1522671879000)\/",
    "Maintainer": "Customer",
    "Description": "Package created by user",
    "DependsOn": [
      {
        "UIId": "51b3ed42-678c-4da3-bd16-8596b95c0546",
        "PackageVersion": "7.18.0",
        "Name": "sdkDependentPackage"
      },
      {
        "UIId": "e14dcfb1-e53c-4439-a876-af7f97083ed9",
        "PackageVersion": "7.18.0",
        "Name": "SalesEnterprise"
      }
    ]
  }
}
```

2. Выполняется проверка установки пакетов-зависимостей в конфигурацию (если пакеты установлены, то они обновляются, если нет — устанавливаются).

Важно. Если указанные в пакете зависимости не найдены в хранилищах SVN (например, хранилище SVN отсутствует в списке зарегистрированных или неактивно), то появится сообщение об ошибке установки пакета. При установке пакета обновляется вся иерархия его зависимостей, поэтому все хранилища SVN, в которых могут содержаться пакеты-зависимости, должны быть включены в конфигурацию и активированы.

3. При установке пакета устанавливаются или обновляются только те зависимости, которые установлены из системы контроля версий SVN. Не обновляются пакеты, установленные из *.zip-архивов, а также предустановленные пакеты.

Необходимо предварительно установить пакеты, от которых зависит устанавливаемый пользовательский пакет, либо его пакеты-зависимости. Установка пакета не будет выполнена, если в рабочем пространстве отсутствует какой-либо из предустановленных пакетов-зависимостей, установленных из *.zip-архивов.

Обновление пакета из хранилища SVN

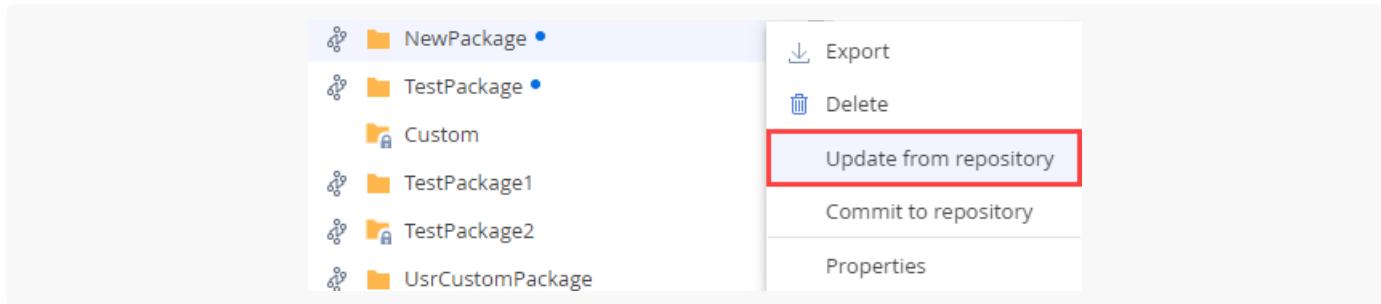
Обновление пакета — это процесс загрузки в приложение изменений выбранного пакета, а также всех изменений [зависимостей](#) пакета из хранилища системы контроля версий SVN. В процессе обновления система определяет зависимости устанавливаемого пакета, которые указаны в метаданных пакета в свойстве `DependsOn`.

Обновлять пакет необходимо в следующих **случаях**:

- Одновременная работа нескольких разработчиков над функциональностью пакета.
- Перенос изменений между [рабочими средами](#).
- Перед выполнением фиксации изменений.

Чтобы **обновить пакет из хранилища SVN**:

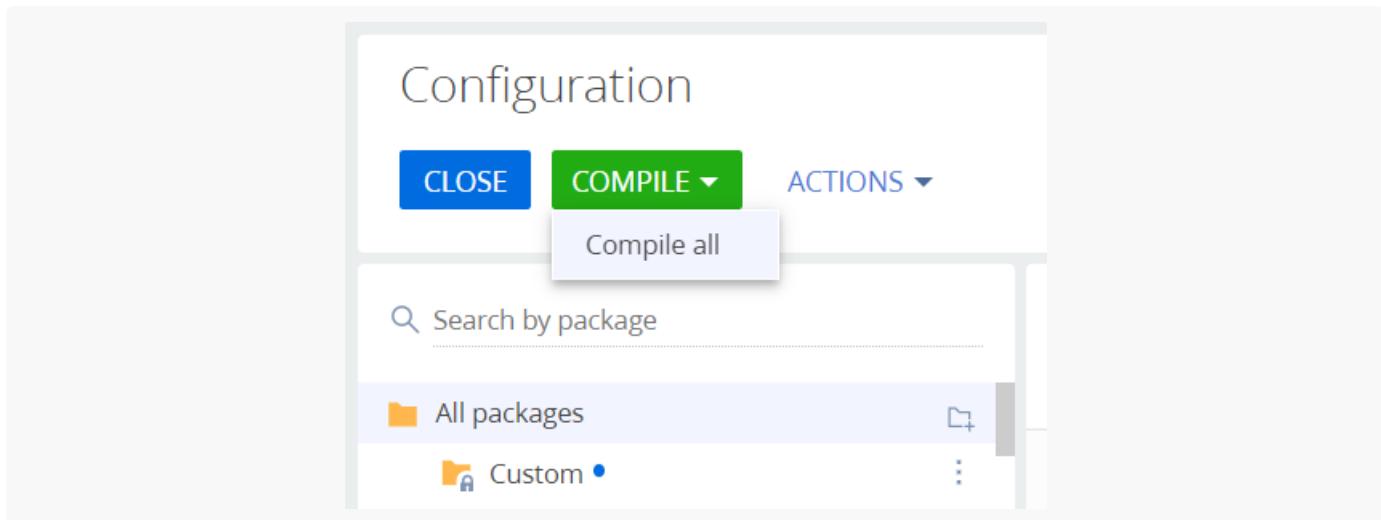
1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]).
2. В меню пакета выберите [Обновить из хранилища] ([Update from repository]).



После этого будет запущен процесс обновления выбранного пакета и всех его пакетов-зависимостей из активных хранилищ SVN.

Важно. Если указанные в пакете зависимости находятся в неактивных хранилищах SVN, то появится сообщение об ошибке обновления пакета. При обновлении пакета обновляется вся иерархия его зависимостей, а значит все хранилища SVN, в которых могут быть расположены пакеты-зависимости, должны быть активизированы.

3. Выполните действие [Компилировать все] ([Compile all]). Это необходимо для [генерации статического контента](#).



Фиксация пакета в хранилище SVN

Фиксация пакета — это процесс сохранения внесенных в пакет изменений в хранилище системы контроля версий SVN. В хранилище SVN фиксируется только тот пакет, для которого было вызвано действие фиксации. Изменения других пакетов конфигурации при этом не фиксируются.

Фиксировать пакет необходимо в следующих **случаях**:

- [Создание](#) пользовательского пакета.
- Добавление новых [конфигурационных элементов](#) пакета.
- Изменение существующих конфигурационных элементов пакета.
- Удаление конфигурационных элементов пакета.
- [Изменение свойств](#) пакета.

Наличие репозитория отображается возле имени пакета, а при наведении курсора на значок отображается **название подключенного репозитория**.

Для **незафиксированных пользовательских пакетов** отображаются:

- Название пакета.
- Название хранилища SVN, в которое этот пакет будет зафиксирован. При этом номер ревизии пакета в хранилище SVN не указывается и будет добавлен после фиксации.

Незафиксированные пользовательские пакеты по умолчанию являются заблокированными.



Для **зафиксированных пользовательских пакетов** отображаются:

- Название пакета.
- Название хранилища SVN.

- Номер последней ревизии пакета в хранилище SVN.

Стиль отображения зафиксированного пользовательского пакета в неизмененном состоянии не отличается от отображения базового пакета.

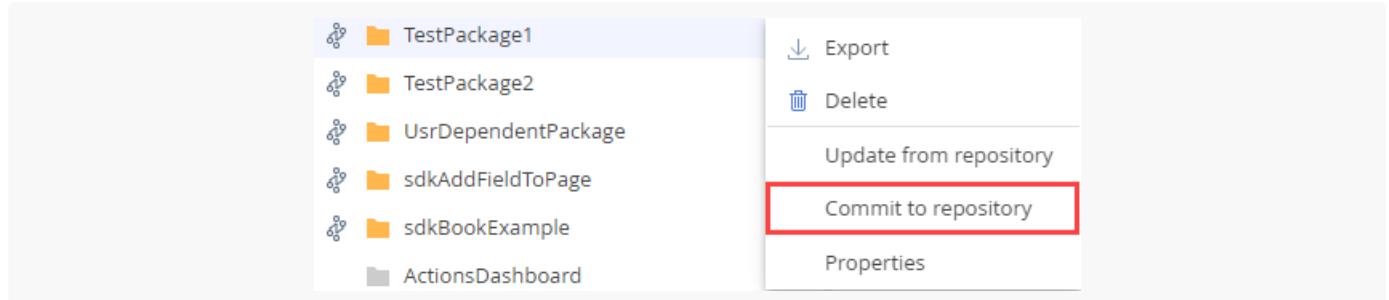


Если в пользовательский пакет были внесены изменения (например, добавлены схемы или изменены его свойства), то возле его названия появляется .

Важно. Если из пакета удалялся конфигурационный элемент, то пакет будет выглядеть как неизмененный.

Чтобы **зарегистрировать пакет в хранилище SVN**:

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]).
2. В меню пакета выберите [Зафиксировать в хранилище] ([Commit to repository]).



В результате откроется окно фиксации изменений.

Commit package to repository

Description *

Name	Type	Status
TestPackage	Package	Added

[CANCEL](#) [COMMIT CHANGES](#)

3. В обязательном поле [*Описание*] ([*Description*]) добавьте комментарий к фиксации пакета. В комментарии рекомендуется описать изменения пакета по сравнению с последней фиксацией. В нижней части окна отображаются изменения пакета, которые будут зафиксированы. После нажатия на кнопку [*Зафиксировать изменения*] ([*Commit changes*]) пакет будет зафиксирован и изменения станут доступными для других пользователей системы.

Важно. Пакет фиксируется в то хранилище SVN, которое указано в его свойствах. Зафиксировать пакет можно только в активное хранилище SVN.

При фиксации пакета в хранилище SVN снимается блокировка с пакета, а также с его конфигурационных элементов, и они становятся доступными для редактирования для других пользователей системы.

Принципы разработки в Creatio IDE



Легкий

Creatio IDE — встроенная среда разработки для управления конфигурацией. Управление конфигурацией подразумевает реализацию сложной бизнес-логики, интеграции и настройки. Управление конфигурацией выполняется с помощью Creatio IDE, которая реализована в виде раздела [*Конфигурация*] ([*Configuration*]). **Назначение** раздела [*Конфигурация*] — управление конфигурационными элементами, при помощи которых реализована функциональность системы.

Настроить доступ к разделу [Конфигурация]

Доступ к разделу [Конфигурация] настраивается на уровне системных операций. По умолчанию доступ к основным системным операциям есть только у администраторов системы. Но его можно [настроить](#) для пользователей или групп пользователей. **Настройка доступа** к разделу [Конфигурация]:

1. Перейдите в дизайнер системы по кнопке . В блоке [Пользователи и администрирование] ([Users and administration]) перейдите по ссылке [Права доступа на операции] ([Operation permissions]).
2. Выберите системную операцию [Доступ к разделу "Конфигурация"] (код [CanManageSolution]).
3. На детали [Доступ к операции] ([Operation permission]) нажмите и укажите получателя прав. Запись появится на детали в колонке [Уровень доступа] ([Access level]) со значением [Да] ([Yes]), а пользователи, входящие в указанную роль, получат доступ к системной операции [Доступ к разделу "Конфигурация"] (код [CanManageSolution]).

Если у пользователя нет доступа к разделу [Конфигурация], то выдается стандартное сообщение с указанием операции и недостающих прав.

Открыть раздел [Конфигурация]

Способы перехода в раздел [Конфигурация] для фреймворка **.NET Framework**:

- По кнопке через дизайнер системы. В блоке [Конфигурирование разработчиком] ([Admin area]) перейдите по ссылке [Управление конфигурацией] ([Advanced settings]).
- По ссылке [Адрес приложения Creatio]/0/ClientApp/#/WorkspaceExplorer . Например, `http://my.creatio.com/0/ClientApp/#/WorkspaceExplorer` .
- По алиасу /we . Например, `http://my.creatio.com/0/we` .
- По алиасу /conf . Например, `http://my.creatio.com/0/conf` .
- По алиасу /dev . Например, `http://my.creatio.com/0/dev` .

Для фреймворка **ASP.NET Core** способы перехода в раздел аналогичны. Различие — не нужно использовать приставку /0 .

Раздел [Конфигурация] будет открыт в новой вкладке.

Разработка конфигурации

Функциональные области интерфейса раздела [Конфигурация]:

- Панель инструментов (1).
- Область работы с пакетами (2).
- Рабочая область (3).

The screenshot shows the 'Configuration' interface in the Creatio IDE. At the top left are 'CLOSE' and 'COMPILE' buttons, followed by an 'ACTIONS' dropdown menu. A blue circle labeled '1' is positioned above the 'ACTIONS' button. To the right is the 'Creatio' logo.

Below the header is a search bar with placeholder 'Search by package' and a 'Type' dropdown. A blue circle labeled '2' is positioned above the search bar. To the right is a table with columns: Name, Title, Status, Type, Object, Modified on, and Package. A blue circle labeled '3' is positioned above the table header.

The left sidebar lists 'All packages' under a folder icon. The list includes: Custom, ActionsDashboard, AddressFinance, AnalyticsDashboard, BankAccountBase, BankAccountOpportunity, BankCardAndAccount, BankCardBase, BankCardOpportunity, BankOnboarding, BankOnboardingCardAndAccou..., BankOnboardingSoftkey_ENU, BankSales, BankSalesSoftkey_ENU, BankSales_BCI_Lending_Market..., BankSpecification, and Base. Each item has a three-dot ellipsis icon to its right.

The main table displays 15 rows of configuration items. The first row is expanded to show its details:

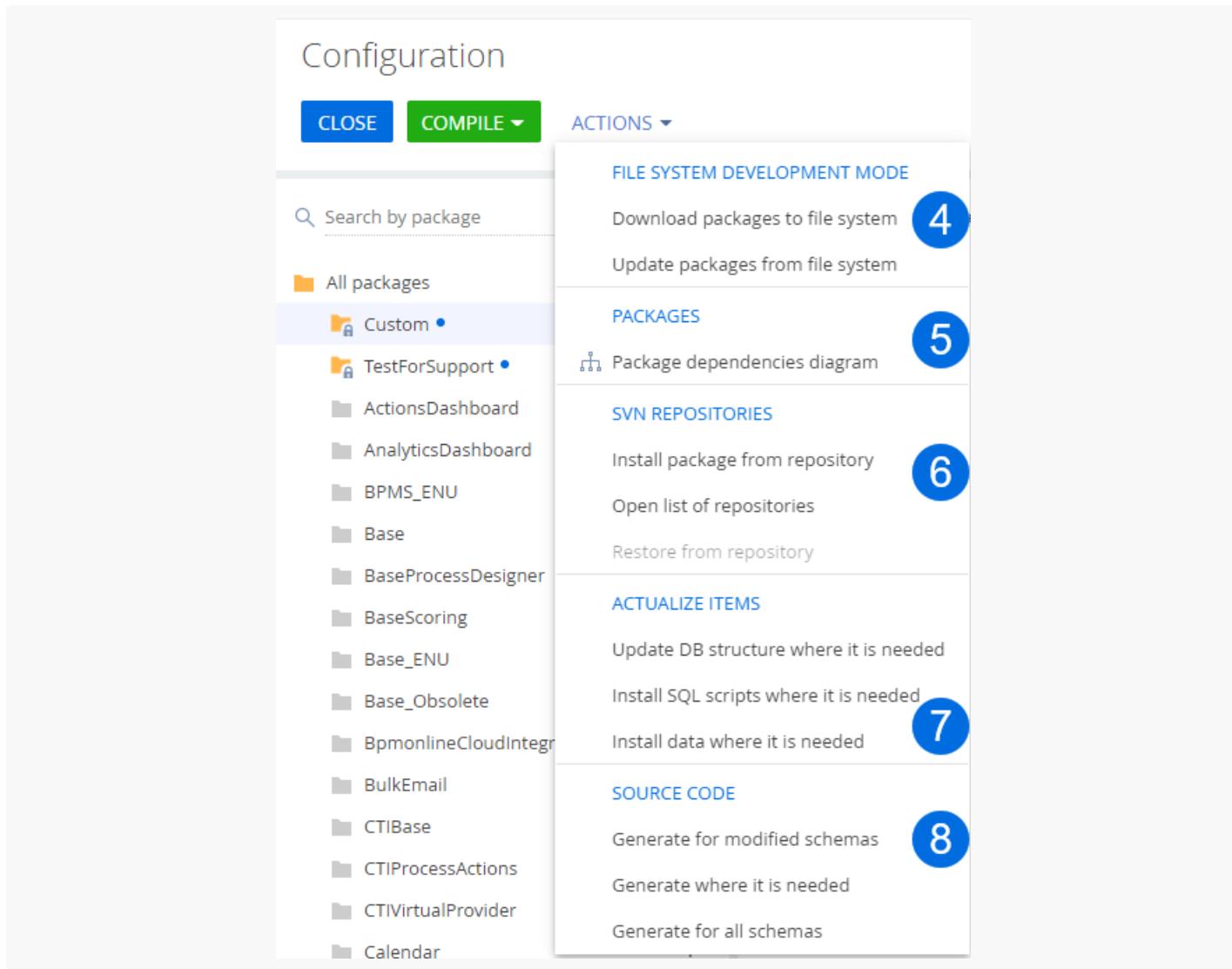
Name	Title	Status	Type	Object	Modified on	Package
AbstractChartJsConfigBuilder	AbstractChartJsConfigBuilder	PM	Client module		10/17/2019, 3:26:19 PM	NUI
AbstractChartProvider	AbstractChartProvider	PM	Client module		9/30/2019, 6:52:28 PM	NUI
AcademyURL	Academy URL	PM	Object	AcademyURL	4/20/2016, 4:51:54 PM	Base
AcademyURL_SysLookup		AM	Data		7/22/2020, 1:42:55 AM	Base
AcademyUtilities	Academy utility module	AM	Client module		3/16/2016, 11:44:35 AM	NUI
AccessTokenInfo	Access Token Info	AM	Source code		5/15/2015, 10:57:21 AM	FacebookIntegration
Account		PM	Data	Account	10/12/2015, 2:19:17 PM	Base
Account	Legal entity	PM	Object		1/14/2020, 12:36:50 PM	BankOnboarding
Account	Account	PM	Object		7/1/2020, 4:04:26 PM	Base
Account	Legal entity	AM	Object		10/5/2016, 1:50:19 AM	AddressFinance
Account	Account	AM	Object		3/23/2017, 11:40:16 AM	EmailMining
Account	Account	AM	Object		6/24/2015, 9:42:58 AM	Completeness
Account	Account	AM	Object		6/15/2016, 4:32:31 AM	Portal

Инструменты раздела [Конфигурация] позволяют управлять:

- Разработкой в файловой системе.
- Пакетами.
- Конфигурационными элементами пакетов.
- Хранилищами системы контроля версий.
- Компиляцией конфигурации.

Разработка в файловой системе

Действия, связанные с разработкой в файловой системе, реализованы в группе [*Разработка в файловой системе*] ([*File system development mode*]) (4) выпадающего списка [*Действия*] ([*Actions*]) панели инструментов (1).



Группа действий [Разработка в файловой системе] ([File system development mode]) (4) позволяет:

- Загрузить пакеты из базы данных приложения в каталог `...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg` (пункт [Выгрузить все пакеты в файловую систему] ([Download packages to file system])).
- Загрузить пакеты из каталога `...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg` в базу данных (пункт [Обновить пакеты из файловой системы] ([Update packages from file system])).

Пункты группы действий [Разработка в файловой системе] ([File system development mode]) (4) доступны только при включенном режиме разработки в [файловой системе](#). Подсказка по включению режима отображается при наведении курсора на название любого пункта текущей группы действий. Включение режима разработки в файловой системе описано в статье [Настроить Creatio для работы в файловой системе](#).

Пакеты

Инструменты управления пакетами:

- Группа [Пакеты] ([Packages]) (5) выпадающего списка [Действия] ([Actions]) панели инструментов

(1).

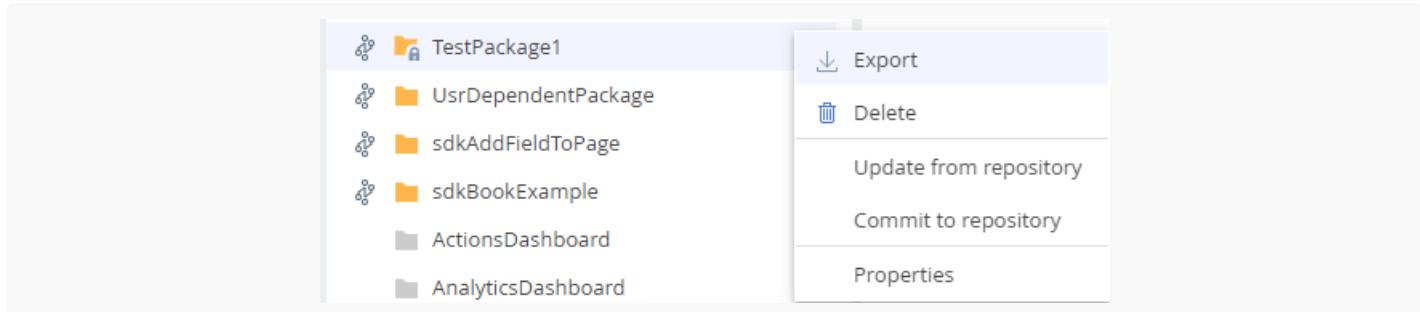
- Область работы с пакетами (2).
- Меню пакета.

Группа действий [Пакеты] ([Packages]) (5) позволяет открыть [диаграмму зависимостей](#) пакетов ([Диаграмма зависимостей пакетов] ([Package dependencies diagram])).

Область работы с пакетами (2) позволяет:

- Выполнить поиск пакета по имени (строка поиска [Поиск по пакетам] ([Search by package])).
- Создать пакет (кнопка ). При нажатии на кнопку будет отображено окно создания нового пакета, в котором можно задать название и описание пакета, добавить зависимости, а также указать хранилище системы контроля версий. Создание пакета описано в статье [Создать пользовательский пакет](#).
- Посмотреть перечень пакетов приложения (группа [Все пакеты] ([All packages])). Пакеты отображаются в алфавитном порядке. Вверху перечня отображаются измененные пакеты и пакеты, доступные для редактирования. При выборе текущей группы конфигурационные элементы всех пакетов приложения будут отображены в алфавитном порядке в рабочей области раздела [Конфигурация] (3). При выборе пакета группы в алфавитном порядке будут отображены конфигурационные элементы текущего пакета. Измененные пакеты находятся вверху перечня пакетов группы [Все пакеты] ([All packages]) и содержат символ  возле имени пакета.

Меню пакета можно вызвать по нажатию на  в строке с названием пакета.



Меню пакета позволяет:

- Удалить пакет (пункт [Удалить] ([Delete])). Пункт неактивен для предустановленных пакетов. Можно удалить пустые пакеты и пакеты с конфигурационными элементами, которые не являются родительскими. При попытке удалить пакет с родительскими конфигурационными элементами будет отображен перечень зависимых пакетов и элементов, зависимых от конфигурационных элементов удаляемого пакета, которые препятствуют удалению.
- Посмотреть свойства пакета (пункт [Свойства] ([Properties])). Вкладка [Свойства пакета] ([Package properties]) позволяет настроить зависимости текущего пакета (если пакет доступен для редактирования). Также можно просмотреть системную информацию: кто создал и отредактировал пакет, даты создания и изменения, уникальный идентификатор, первичный ключ пакета в таблице базы данных. Открыть свойства пакета можно двойным кликом по имени пакета.

Конфигурационные элементы пакетов

Инструменты управления конфигурационными элементами пакетов:

- Группа [Актуализировать элементы] ([Actualize items]) (7) выпадающего списка [Действия] ([Actions]) панели инструментов (1).
- Группа [Исходный код] ([Source code]) (8) выпадающего списка [Действия] ([Actions]) панели инструментов (1).
- Панель инструментов рабочей области (3) раздела [Конфигурация].
- Реестр рабочей области (3) раздела [Конфигурация].

Группа действий [Актуализировать элементы] ([Actualize items]) (7) позволяет:

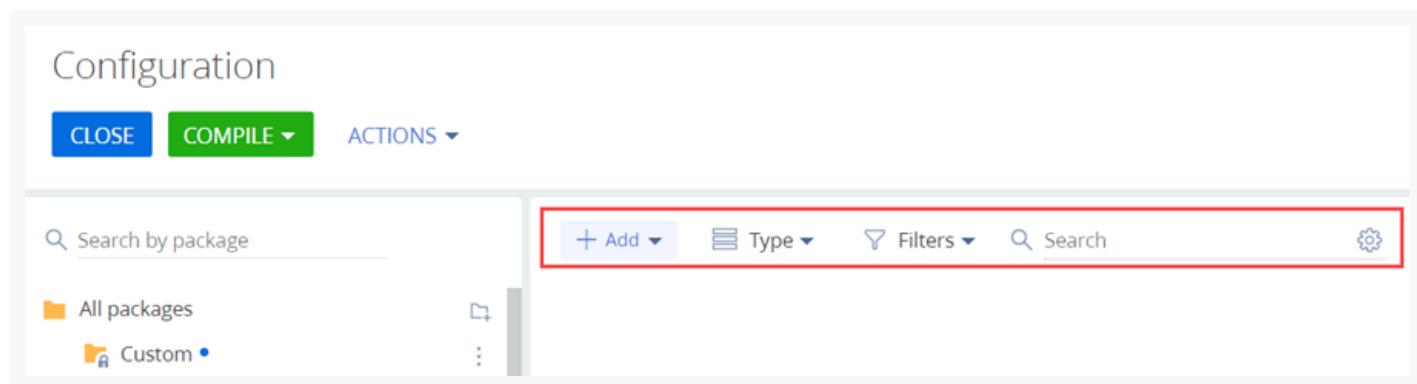
- Обновить структуру базы данных для конфигурационных элементов, которые требуют обновления (пункт [Обновить структуру БД для требующих обновления] ([Update DB structure where it is needed])).
- Установить SQL-сценарии, которые требуют установки (пункт [Установить SQL сценарии для требующих установки] ([Install SQL scripts where it is needed])).
- Установить данные, которые требуют установки (пункт [Установить данные для требующих установки] ([Install data where it is needed]))).

После завершения актуализации конфигурационных элементов вы получите уведомление.

Группа действий [Исходный код] ([Source code]) (8) позволяет:

- Генерировать исходный код для схем, которые были изменены в текущей конфигурации (пункт [Генерировать для измененных] ([Generate for modified schemas])).
- Генерировать исходный код для схем, которые требуют генерации исходного кода (пункт [Генерировать для требующих генерации] ([Generate where it is needed]))).
- Генерировать исходный код для всех без исключения схем текущей конфигурации (пункт [Генерировать для всех схем] ([Generate for all schemas]))). Эта операция может занять длительное время (больше 10 минут).

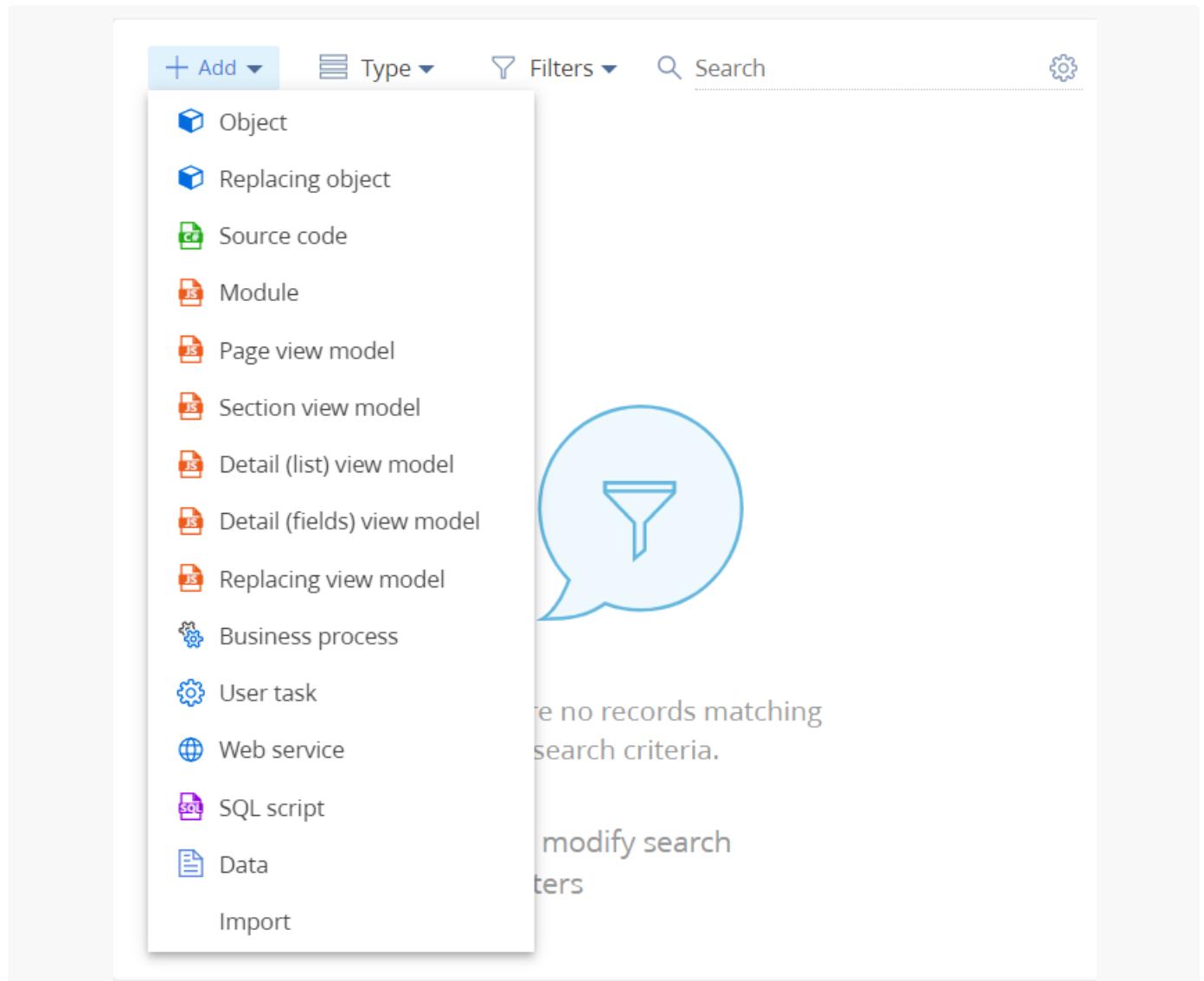
Панель инструментов рабочей области (3) раздела [Конфигурация] представлена на рисунке ниже.



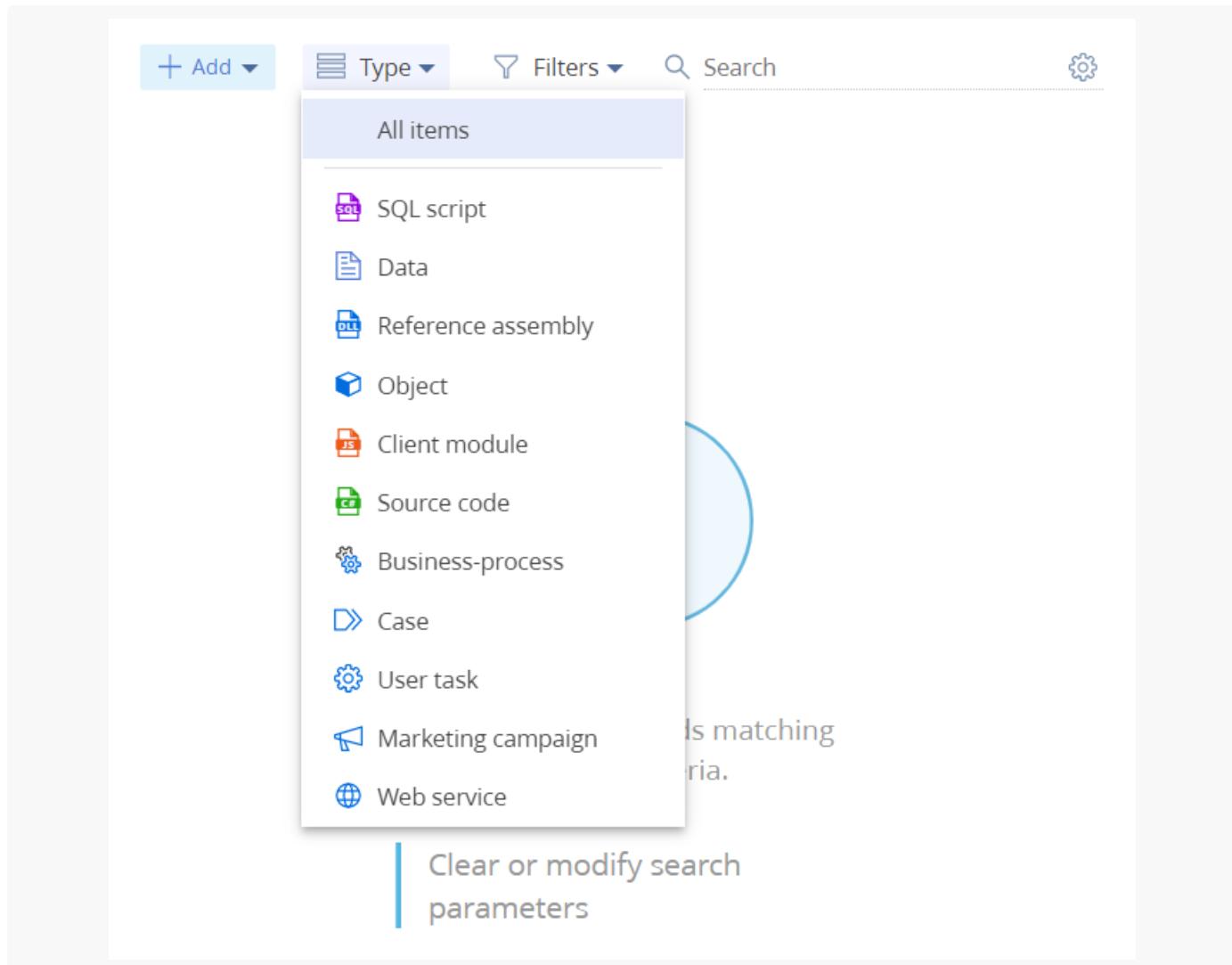
Панель инструментов позволяет:

- Выбрать вид конфигурационного элемента для добавления (выпадающий список [Добавить] ([Add])). Предварительно необходимо выбрать пакет. Без выбора пакета пункты выпадающего списка [Добавить] ([Add]) остаются неактивными. В предустановленные пакеты конфигурационные

элементы добавить невозможно. Возможные **виды** конфигурационных элементов для добавления представлены на рисунке ниже.



- Загрузить в пользовательский пакет схему (*.md) или внешнюю сборку (*.dll) (пункт [Импортировать] ([Import]) выпадающего списка [Добавить] ([Add])).
- Выбрать тип конфигурационных элементов для отображения в реестре раздела (выпадающий список [Тип] ([Type])). Возможные **типы** конфигурационных элементов для отображения представлены на рисунке ниже.



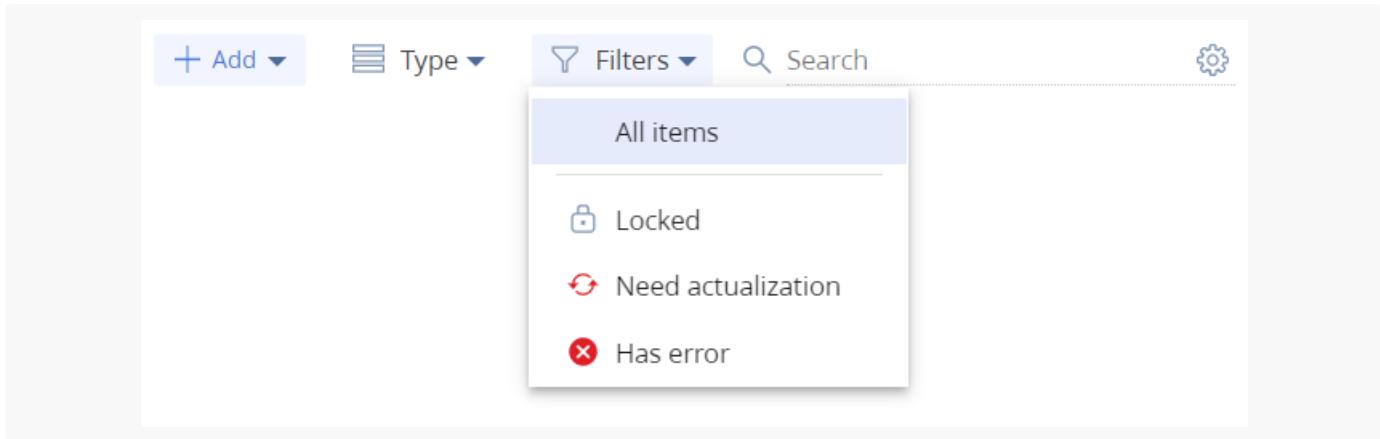
Конфигурационные элементы с типом [Клиентский модуль] ([*Client module*]):

- [Модуль] ([*Module*]).
- [Модель представления страницы] ([*Page view model*]).
- [Модель представления раздела] ([*Section view model*]).
- [Модель представления детали с реестром] ([*Detail (list) view model*]).
- [Модель представления детали с полями] ([*Detail (fields) view model*]).
- [Замещающая модель представления] ([*Replacing view model*]).

Конфигурационные элементы с типом [Объект] ([*Object*]):

- [Объект] ([*Object*]).
- [Замещающий объект] ([*Replacing object*]).

- Выбрать состояния конфигурационных элементов для отображения в реестре раздела (выпадающий список [Фильтры] ([*Filters*])). Настройки выпадающего списка [Фильтры] ([*Filters*]) сохраняются в профиле пользователя и отображаются при входе в раздел [Конфигурация]. Возможные **состояния** конфигурационных элементов для отображения представлены на рисунке ниже.



Конфигурационные элементы со значением [Требующие актуализации] ([*Need actualization*]) выпадающего списка [Фильтры] ([*Filters*]):

- [SQL сценарий] ([*SQL script*]) и [Данные] ([*Data*]) — при установленном свойстве [Требует установки в БД] ([*Needs to be installed in database*]).
- [Объект] ([*Object*]) — при установленном свойстве [Требуется обновление в БД] ([*Needs to be updated in database*]).
- [Клиентский модуль] ([*Client module*]), [Исходный код] ([*Source code*]), [Кейс] ([*Case*]), [Маркетинговая кампания] ([*Marketing campaign*]), [Бизнес-процесс] ([*Business-process*]), [Действие процесса] ([*User task*]) — при установленном свойстве [Требует генерации исходного кода] ([*Needs generate source code*]).
- Выполнить поиск конфигурационного элемента по имени в реестре раздела (строка поиска [Поиск] ([*Search*])). Поиск выполняется в текущем пакете. Чтобы выполнять поиск по всем конфигурационным элементам пакетов, необходимо перейти в группу [Все пакеты] ([*All packages*]) области работы с пакетами (2). По нажатию настраиваются **параметры поиска**:
 - Поиск по заголовку (опция [Поиск по полю "Заголовок"] ([*Search by column "Title"*]) — по умолчанию включена).
 - Поиск по уникальному идентификатору (опция [Поиск по полю "UID"] ([*Search by column "UID"*]) — по умолчанию выключена).
 - Режим поиска. Возможные значения:
 - Пункт [Начинается] ([*Starts with*]) — по умолчанию включен. Название конфигурационного элемента начинается с текста, который введен в строке поиска [Поиск] ([*Search*]).
 - Пункт [Содержит] ([*Contains*]) — по умолчанию выключен. Название конфигурационного элемента содержит текст, который введен в строке поиска [Поиск] ([*Search*]).
 - Пункт [Равно] ([*Equals*]) — по умолчанию выключен. Название конфигурационного элемента соответствует тексту, который введен в строке поиска [Поиск] ([*Search*]).

Для применения внесенных изменений нажмите [Применить] ([*Apply*]).

Настройки поиска сохраняются в профиле пользователя и отображаются при входе в раздел [Конфигурация].

Реестр раздела рабочей области (3) раздела [Конфигурация] содержит перечень конфигурационных элементов. **Свойства** конфигурационных элементов реестра раздела представлены в таблице.

Свойства конфигурационных элементов

Колонка	Описание	Дополнительные сведения
[Название] ([Name])	Имя конфигурационного элемента, которое было задано при создании	Все конфигурационные элементы отсортированы в алфавитном порядке. Колонка позволяет посмотреть перечень измененных конфигурационных элементов (символ * возле имени элемента). Измененные конфигурационные элементы находятся вверху перечня конфигурационных элементов реестра раздела.
[Заголовок] ([Title])	Заголовок конфигурационного элемента	
[Статус] ([Status])	Состояние конфигурационного элемента	<p>Содержит значения выпадающего списка [Фильтры] ([Filters]). Колонка [Статус] ([Status]) будет содержать  , если необходима актуализация конфигурационного элемента. При наведении на  можно увидеть всплывающие подсказки:</p> <ul style="list-style-type: none"> [Требует установки в БД] ([Needs to be installed in database]) — для конфигурационных элементов [SQL сценарий] ([SQL script]) и [Данные] ([Data]).] [Требуется обновление в БД] ([Needs to be updated in database]) — для конфигурационного элемента [Объект] ([Object]).] [Требует генерации исходного кода] ([Needs generate source code]) — для конфигурационных элементов [Клиентский модуль] ([Client module]), [Исходный код] ([Source code]), [Кейс] ([Case]), [Маркетинговая кампания] ([Marketing campaign]), [Бизнес-процесс] ([Business-process]), [Действие процесса] ([User task]).] <p>Колонка [Статус] ([Status]) будет содержать  , если конфигурационный элемент содержит ошибку. При наведении на  можно увидеть текстовое описание ошибки.</p> <p>Если необходима актуализация конфигурационного элемента и при этом он содержит ошибку, то колонка [Статус] ([Status]) будет содержать  . При наведении курсора можно увидеть информацию о необходимости актуализации конфигурационного элемента и текстовое описание ошибки.</p>

Колонка [Тип]	Описание [Type]	Дополнительные сведения
[Тип]	конфигурационного элемента	
[Объект] ([Object])	Объект, с которым связаны привязываемые к пакету данные	Колонка заполняется только для конфигурационного элемента [Данные] ([Data]).
[Дата изменения] ([Modified on])	Дата изменения конфигурационного элемента	
[Пакет] ([Package])	Название пакета, который содержит конфигурационный элемент	

На заметку. Отсортировать данные в колонке по возрастанию или по убыванию можно нажатием по имени колонки.

При нажатии на в строке конфигурационного элемента будет отображено меню, которое зависит от значения колонки [Тип] ([Type]). Описание действий с конфигурационными элементами, которые можно выполнить с помощью меню конфигурационного элемента, приведено в таблице.

Действия над конфигурационными элементами

Пункт меню	Действие	Дополнительные сведения	Конфигурационный элемент	
[Удалить] ([Delete])	Удалить конфигурационный элемент	Неактивно для конфигурационных элементов предустановленных пакетов.	[SQL сценарий] ([SQL script]) [Данные] ([Data]) [Внешняя сборка] ([Reference assembly]) [Объект] ([Object]) [Клиентский модуль] ([Client module]) [Исходный код] ([Source code]) [Кейс] ([Case]) [Маркетинговая кампания] ([Marketing	

Пункт меню	Действие	Дополнительные сведения	<small>campaign])</small> Конфигурационный элемент [Web service]
[Установить] ([Install])	Установить конфигурационный элемент	Для элемента [SQL сценарий] ([SQL script]) выполнится установка в базу данных, для элемента [Данные] ([Data]) выполнится установка данных для объекта колонки [Объект] ([Object]). При некорректной установке данных для объекта колонки [Объект] ([Object]) можно посмотреть описание ошибки в колонке [Статус] ([Status]). Описание ошибки также можно посмотреть в свойствах конфигурационных элементов.	[SQL сценарий] ([SQL script]) [Данные] ([Data])
[Отменить изменения] ([Discard changes])	Отменить внесенные изменения	Пункт доступен, если к пакету с конфигурационным элементом подключено хранилище системы контроля версий.	[SQL сценарий] ([SQL script]) [Данные] ([Data]) [Внешняя сборка] ([Reference assembly]) [Объект] ([Object]) [Клиентский модуль] ([Client module]) [Исходный код] ([Source code]) [Кейс] ([Case]) [Маркетинговая кампания] ([Marketing campaign])

Пункт меню	Действие	Дополнительные сведения	Справка
			[Конфигурационный элемент] [Бизнес-процесс] [Business-process]
[Обновить структуру БД] ([Update database structure])	Обновить структуру базы данных для объекта		[Действие процесса] ([User task])
[Сгенерировать исходный код] ([Generate source code])	Сгенерировать исходный код конфигурационного элемента	Будет выполнено, если процесс содержит компилируемые элементы.	[Объект] ([Object]) [Бизнес-процесс] [Business-process] [Действие процесса] ([User task]) [Веб-сервис] ([Web service])
[Открыть метаданные] ([Open metadata])	Открыть окно метаданных конфигурационного элемента.		[Объект] ([Object]) [Клиентский модуль] ([Client module]) [Исходный код] ([Source code]) [Кейс] ([Case]) [Маркетинговая кампания] ([Marketing campaign]) [Бизнес-процесс] ([Business-process]) [Действие процесса] ([User task]) [Веб-сервис] ([Web service])

Реестр позволяет удалить конфигурационный элемент (кнопка ). Данная кнопка появляется при наведении курсора на запись реестра раздела [Конфигурация] только для конфигурационных элементов пользовательских пакетов.

Хранилища системы контроля версий

Инструменты работы с хранилищами системы контроля версий:

- Группа [Хранилища SVN] ([SVN repositories]) (6) выпадающего списка [Действия] ([Actions]) панели инструментов (1).
- Меню пакета.

Группа действий [Хранилища SVN] ([SVN repositories]) (6) позволяет:

- Установить пакет из хранилища системы контроля версий (пункт [Установить пакет из хранилища] ([Install package from repository])).
- Открыть вкладку [Список хранилищ] ([List of repositories]) (пункт [Открыть список хранилищ] ([Open list of repositories])), которая позволяет создавать, настраивать и удалять ссылки на доступные хранилища системы контроля версий.
- Синхронизировать конфигурацию с хранилищем до последней ревизии (пункт [Восстановить из хранилища] ([Restore from repository])). Это приведет к потере изменений, не зафиксированных в хранилище системы контроля версий.

Использование системы контроля версий SVN доступно только для приложения Creatio на платформе .NET Framework. Описание работы с хранилищами системы контроля версий содержится в статье [Контроль версий в Creatio IDE](#).

Меню пакета

- Выгрузить пакет в zip-архив (пункт [Экспортировать] ([Export])). Экспорт пакетов описан в статье [Экспорт и импорт пакетов](#).
- Обновить пакет из подключенного хранилища системы контроля версий (пункт [Обновить из хранилища] ([Update from repository])). Обновление пакета описано в статье [Обновить пакет из системы контроля версий](#).
- Зафиксировать пакет в подключенном хранилище системы контроля версий (пункт [Зафиксировать в хранилище] ([Commit to repository])). Фиксация пакета в системе контроля версий описана в статье [Фиксировать пакет в системе контроля версий](#).

Компиляция конфигурации

Запуск компиляции изменений в конфигурации выполняется нажатием [Компилировать] ([Compile]) на панели инструментов (1).

Выбор пункта [Перекомпилировать все] ([Compile all]) в выпадающем списке кнопки [Компилировать] ([Compile]) запускает компиляцию конфигурации всех без исключения конфигурационных элементов. В результате будут обновлены исполняемые файлы и [статический контент](#) будет выгружен в каталог `...\\Terrasoft.WebApp\\conf`. После завершения компиляции пользователь получит уведомление и изменения вступят в силу для пользователей, работающих в текущей конфигурации.

При компиляции могут возникнуть **ошибки**, которые отображаются в диалоговом окне. **Свойства ошибок компиляции:**

- Иконка типа ошибки (ошибка  или предупреждение ).
- Имя файла с ошибкой.
- Описание ошибки.

- Код ошибки.
- Номер строки с ошибкой.

Закрыть раздел [Конфигурация]

Чтобы закрыть вкладку с разделом [Конфигурация], нажмите [Закрыть] ([Close]) на панели инструментов (1).

NLog



Библиотека логирования NLog

Для проверки корректности работы новой функциональности, мы рекомендуем включить логирование. Чтобы избежать ухудшения производительности, рекомендуется включать логирование только при тестировании и отладке приложения. В Creatio выполняется логирование всех основных операций. Для этого используется [NLog](#) — бесплатная библиотека логирования для .NET с широкими возможностями маршрутизации и управления высококачественными логами для приложения, независимо от его размера или сложности. Библиотека может обрабатывать диагностические сообщения, отправляемые с любого языка .NET, дополнять контекстной информацией, форматировать в соответствии с предпочтениями пользователя и отправлять одному или нескольким получателям сообщений, таким как файл или база данных.

Документация по работе с библиотекой NLog доступна на [сайте GitHub](#).

Настройка логирования в Creatio

Логирование ведется отдельно для загрузчика приложения и для конфигурации `Default`. Настройки логирования выполняются в каталоге `..\Terrasoft.WebApp` в следующих конфигурационных файлах:

- `nlog.config` .
- `nlog.targets.config` .
- `nlog.cloud.config` (для приложений, развернутых в облаке).
- `nlog.cloud.settings.config` (для приложений, развернутых в облаке).

Пример настройки логирования содержится в статье [Реализовать модальное окно](#).

Логирование для приложений on-site

Хранение логов

Местоположение файлов логов зависит от значения системных переменных Windows.

Путь к файлам с логами загрузчика (по умолчанию)

```
[TEMP]\Creatio\Site_[{SiteId}]\[{ApplicationName}]\Log\[{DateTime.Today}]
```

Пример пути

```
C:\Windows\Temp\Creatio\Site_1\creatio7121\Log\2018_05_22
```

Путь к файлам с логами конфигурации Default

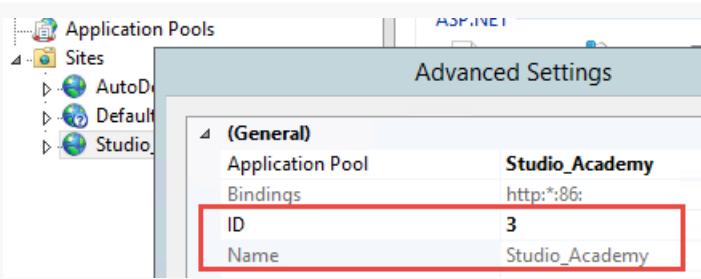
```
[TEMP]\Creatio\Site_[{SiteId}]\[{ApplicationName}]\[ConfigurationNumber]\Log\[{DateTime.Today}]
```

Пример пути

```
C:\Windows\Temp\Creatio\Site_1\creatio7121\0\Log\2018_05_22
```

В приведенных выше примерах в квадратных скобках указаны следующие переменные:

- [TEMP] — базовый каталог. По умолчанию для IIS используется каталог C:\Windows\Temp , а для Visual Studio (IIS Express) — C:\Users\{Имя пользователя}\AppData\Local\Temp .
- [{SiteId}] — номер сайта. Для IIS указан в расширенных настройках сайта. Для Visual Studio номер содержит значение 2.
- [{ApplicationName}] — название приложения.
- [ConfigurationNumber] — номер конфигурации. Конфигурация Default , как правило, имеет номер 0.
- [{DateTime.Today}] — дата логирования.



Получатели логирования

Получатели логирования используются для отображения, хранения или передачи сообщений лога другому получателю. Есть получатели, которые получают и обрабатывают сообщения и те, которые буферизируют или направляют сообщения другому получателю.

Получатели логирования задаются в файле `nlog.targets.config` с помощью следующих атрибутов:

- `name` — имя получателя.
- `xsi:type` — тип получателя. Может принимать значения "File", "Database", "Mail".
- `fileName` — файл и путь к файлу, куда будут записываться логи.
- `layout` — шаблон, по которому будет заполняться файл.

Получатели логирования подробно описаны в [документации NLog](#).

Правила логирования

Правила логирования устанавливаются в файле `nlog.config` с помощью следующих атрибутов:

- `name` — имя лога.
- `minlevel` — минимальный уровень логирования.
- `maxlevel` — максимальный уровень логирования.
- `level` — логирование событий одного уровня логирования.
- `levels` — логирование событий нескольких уровней логирования (записываются через запятую).
- `writeTo` — имя получателя логирования.
- `final` — после совпадения окончательного правила правила не обрабатываются.
- `enabled` — отключить правило (установить в значение `false`), не удаляя его.
- `ruleName` — идентификатор правила.

Атрибуты правил обрабатываются в следующем порядке:

1. `level`.
2. `levels`.
3. `minlevel` и `maxlevel` (имеют одинаковый приоритет).

Если `minLevel = "Warn" level = "Info"`, то будет использоваться только уровень логирования `Info`.

Атрибут `level` имеет высший приоритет обработки, чем `minLevel`.

Уровни логирования

Уровни логирования устанавливаются в файле `nlog.config`. По умолчанию уровень логирования для всех компонентов Creatio установлен таким образом, чтобы обеспечить максимальную производительность приложения. Возможные уровни логирования в порядке возрастания приоритета:

- `Trace` — логирование событий при отладке трассировки (указываются начальный и конечный методы).
- `Debug` — логирование всех событий при отладке.
- `Info` — нормальная работа приложения.
- `Warn` — логирование предупреждений (приложение будет продолжать работу).

- `Error` — логирование ошибок (возможно прекращение работы приложения).
- `Fatal` — логирование ошибок, приводящих к прекращению работы приложения.
- `Off` — логирование отключено, не используется для записей.

Способы настройки логирования

Логирование можно настраивать следующими способами:

- через конфигурационный файл;
- через конфигурационный объект `LoggingConfiguration`.

Логирование для приложений cloud

Для настройки логирования в приложении, развернутом в облаке, необходимо обратиться в службу технической поддержки Terrasoft.

Важно. При настройке логирования приложения cloud нет возможности добавить дополнительный получатель логирования.

Настроить логирование NLog



Средний

Настройка логирования через конфигурационный файл

1. Задать путь к файлу с настройками логирования

Путь к файлу `nlog.config` задается в конфигурационном файле `..\Terrasoft.WebApp\Web.config`.

`..\Terrasoft.WebApp\Web.config`

```
<common>
  <logging>
    <factoryAdapter type="Common.Logging.NLog.NLogLoggerFactoryAdapter, Common.Logging.NLog45">
      <arg key="configType" value="FILE" />
      <arg key="configFile" value="~/nlog.config" />
    </factoryAdapter>
  </logging>
</common>
```

2. Задать получателей логирования

Для этого необходимо в XML-элементе `<target></target>` файла `..\Terrasoft.WebApp\nlog.targets.config` определить свойства получателя логирования.

```
..\Terrasoft.WebApp\nlog.targets.config

<target name="universalAppender" xsi:type="File"
    layout="${DefaultLayout}"
    fileName="${LogDir}${LogDay}${logger:shortName=True}.log" />
```

3. Задать правило логирования

Пример правила, которое реализует запись лога в базу данных

```
<logger name="IncidentRegistration" writeTo="AdoNetBufferedAppender" minlevel="Trace" final="true">
```

Настройка логирования через конфигурационный объект LoggingConfiguration

Для программной настройки логирования необходимо выполнить следующие шаги:

1. Создать объект `LoggingConfiguration`, который будет описывать конфигурацию.
2. Создать одного или нескольких получателей.
3. Установить свойства получателей.
4. Определить правила с помощью объектов `LoggingRule` и добавить их в конфигурационные `LoggingRules`.
5. Активировать конфигурацию, указав в `LogManager.Configuration` созданный объект конфигурации.

Back-end отладка



Сложный

Back-end отладка — отладка серверных схем исходного кода. Это могут быть, например, существующие базовые схемы, пользовательские конфигурационные классы, веб-сервисы или скрипты бизнес-процессов, написанные на языке C#.

Выполнение back-end отладки

Выполнять отладку серверных схем исходного кода Creatio можно исключительно с помощью интегрированных функций отладки **внешних IDE**, например, Visual Studio. Отладчики внешних IDE позволяют:

- приостанавливать выполнение методов;
- проверять значения переменных;
- изменять значения переменных;
- получать полное представление о том, что делает код.

В этом руководстве описано выполнение back-end отладки на примере Visual Studio.

Необходимые условия выполнения отладки с использованием Visual Studio:

- Приложение Creatio развернуто on-site.
- Режим разработки в файловой системе отключен.
- В Visual Studio включен признак [*Suppress JIT Optimization*] (меню [*Options*], вкладки [*Debugging*] —> [*General*]). Это позволяет во время отладки узнать значения переменных. Подробнее об оптимизированном и неоптимизированном коде во время отладки можно узнать из [документации Visual Studio](#).

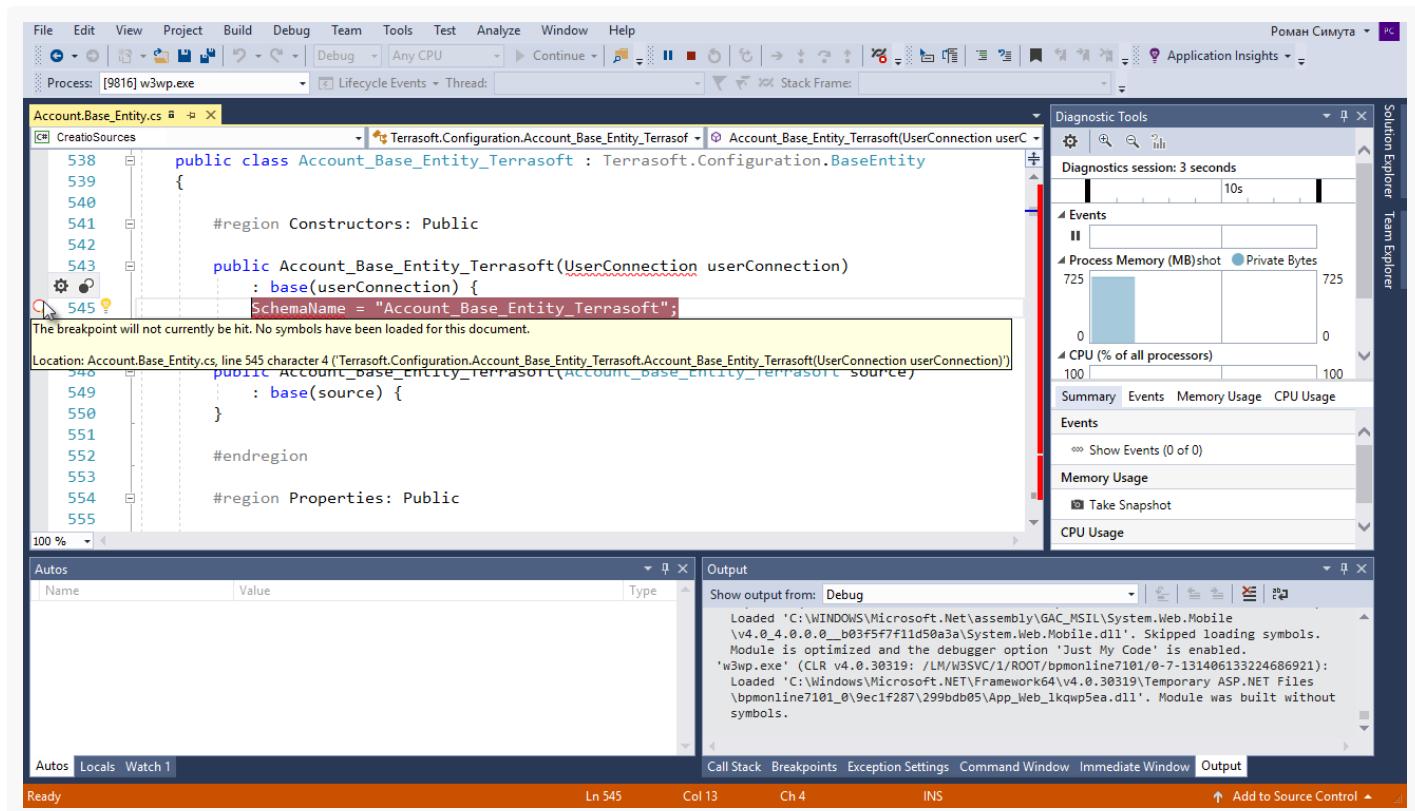
Последовательность back-end отладки:

1. Выгрузить исходные коды конфигурации Creatio в файлы локального каталога.
2. Создать новый проект в Visual Studio, в котором будет выполняться отладка.
3. Добавить в проект Visual Studio выгруженные файлы с исходным кодом.
4. Подключить проект к рабочему процессу сервера IIS и начать процесс отладки.

Возможные проблемы при отладке

Символ точки останова отображается в виде белого круга, ограниченного красной окружностью.

Такая точка останова является неактивной и на ней не будет прерываться выполнение скрипта. При наведении курсора на символ неактивной точки останова появится подсказка, уведомляющая о проблеме.

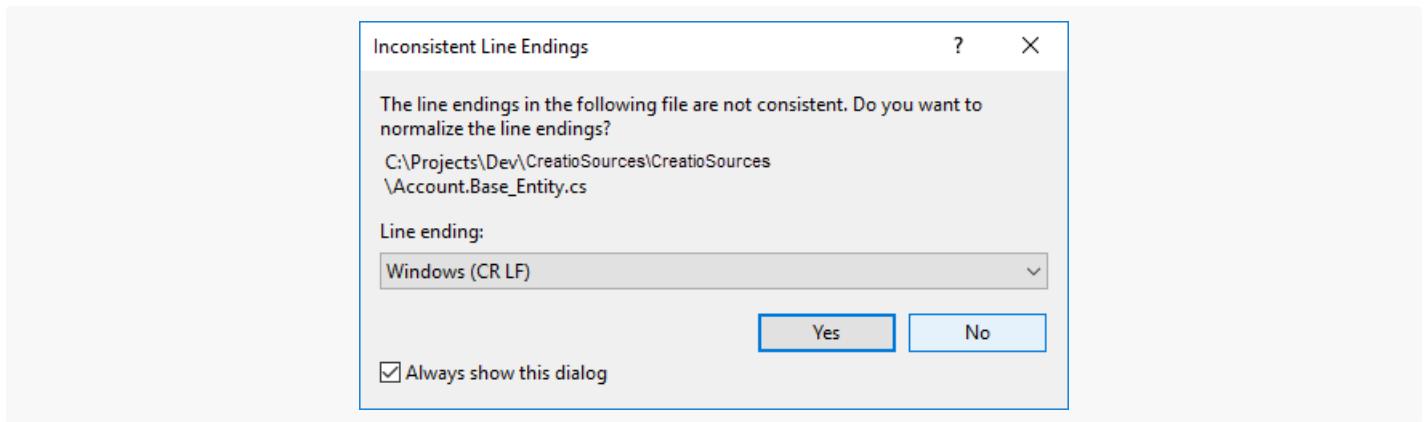


Решение:

1. Завершить выполнение отладки (`Debug` —> `Stop Debugging`).
2. Закрыть файл с исходным кодом, для которого выполняется отладка.
3. В разделе [Конфигурация] ([*Configuration*]) приложения выполнить действие [Компилировать все] ([*Compile all*]).
4. Во время выполнения компиляции и перевыгрузки файлов с исходными кодами подключить проект к процессу IIS заново.
5. После выполнения компиляции переоткрыть файл с исходным кодом, для которого выполняется отладка.

На заметку. В некоторых случаях может помочь повторная компиляция без открепления и прикрепления к IIS.

После повторного открытия файла с исходным кодом появилось сообщение о неединобразных символах в конце строк.

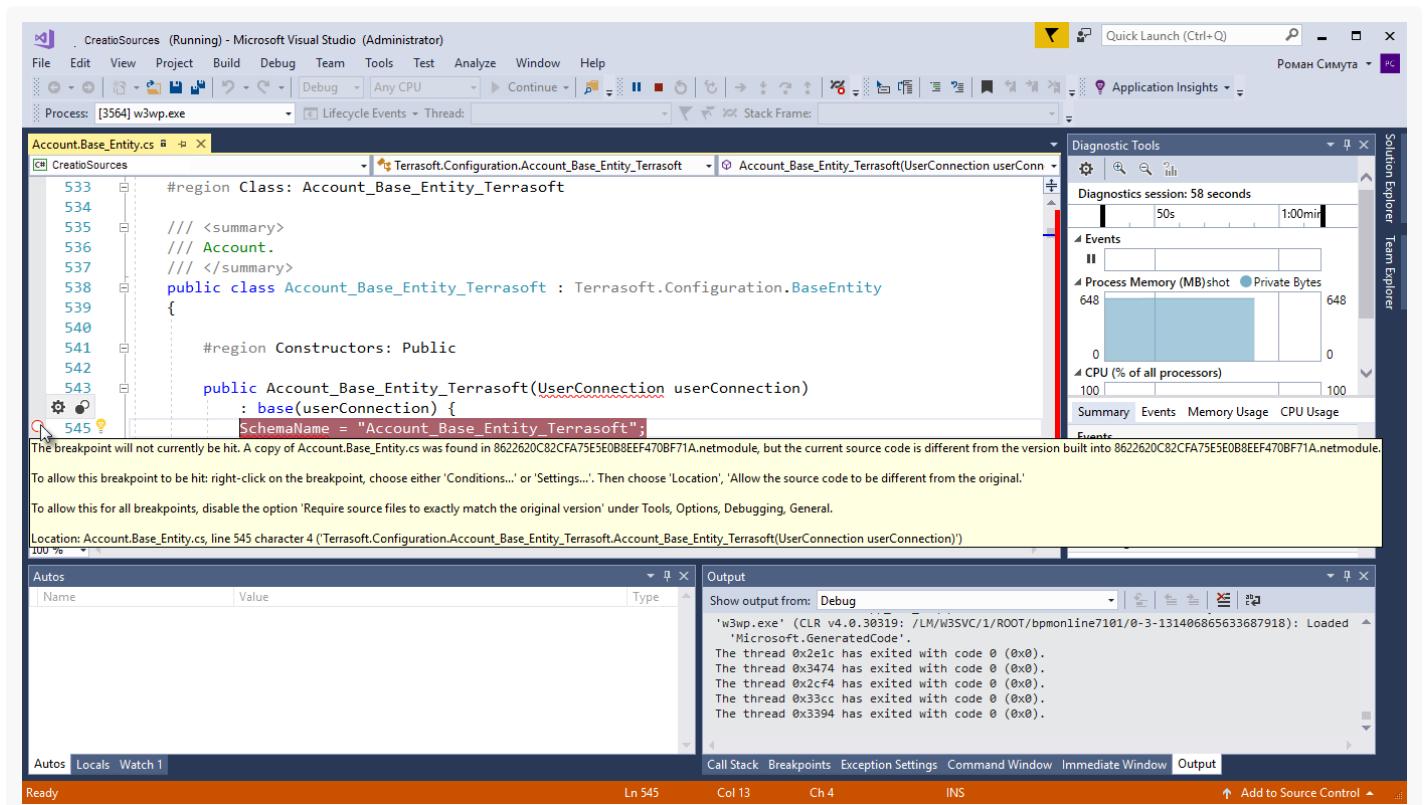


Решение:

- Нажать кнопку [Нет] ([No]). Если согласиться с нормализацией символов (кнопка [Да] ([Yes])), то точка останова снова может стать неактивной.

Причина проблемы отобразится в подсказке — несоответствие версий файла. Также в подсказке отображены варианты решения проблемы.

- Выполнить один из предложенных вариантов решения проблемы.



Выполнить отладку серверного кода

Сложный

1. Выполнить настройку приложения

Для выполнения отладки необходимо внести изменения в конфигурационные файлы приложения.

Чтобы **выполнить настройку приложения**:

- Настройте "внешний" `Web.config`.

В файле `Web.config`, расположенном в корневом каталоге приложения, для атрибута `debug` элемента `<compilation>` установите значение `true`.

`Web.config`

```
<compilation debug="true" targetFramework="4.5" />
```

После внесения изменений сохраните файл.

- Настройте "внутренний" `Web.config`.

В файле `Web.config`, расположенном в каталоге `Terrasoft.WebApp` приложения, укажите значения для следующих элементов:

- `IncludeDebugInformation` — `true`.
- `ExtractAllCompilerSources` — `true`, если необходимо выгружать все схемы при выполнении действия [Компилировать] ([Compile]) раздела [Конфигурация] ([Configuration]).
- `ExtractAllCompilerSources` — `false`, если необходимо выгружать только измененные схемы (значение по умолчанию).

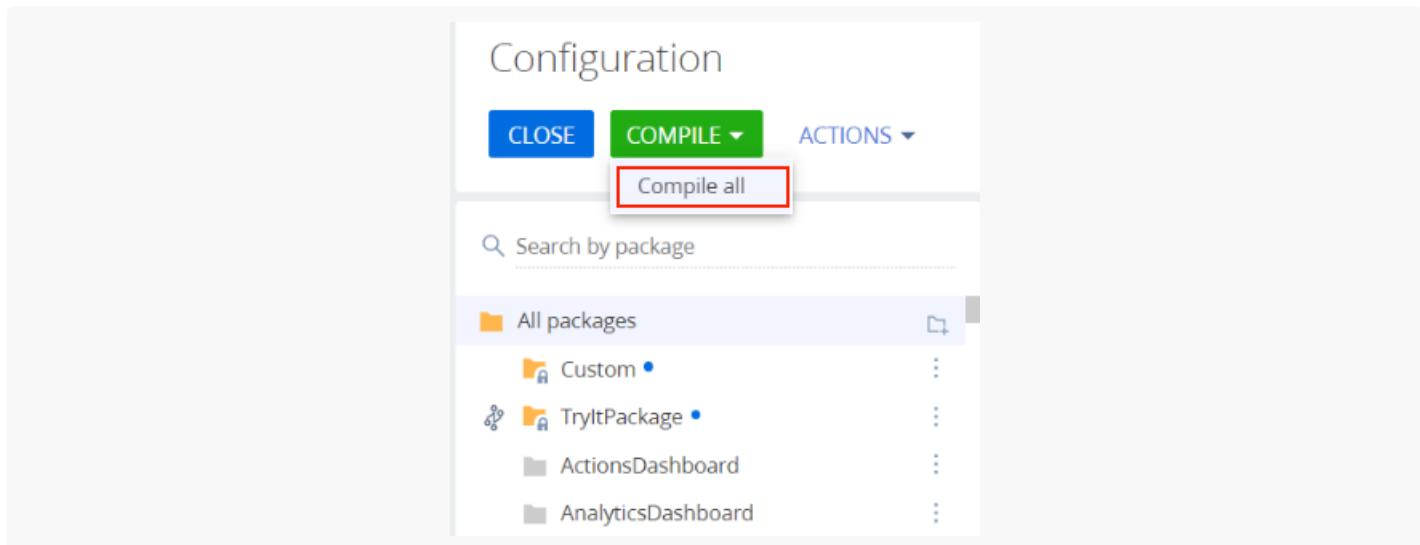
`Web.config`

```
<add key="IncludeDebugInformation" value="true" />
<add key="ExtractAllCompilerSources" value="false" />
```

После внесения изменений сохраните файл.

2. Выгрузить исходные коды конфигурации

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



Во время компиляции в папку `../Terrasoft.WebApp/Terrasoft.Configuration/Autogenerated/Src` будут выгружены файлы с исходными кодами конфигурационных схем приложения, а также конфигурационные библиотеки, их модули и файлы с отладочной информацией (*.pdb). Исходные коды схем будут выгружены заново при каждой последующей компиляции приложения.

Файлы выгруженных исходных кодов конфигурационных схем именуются в определенном формате:

[Название схемы в конфигурации].[Название пакета]_[Тип схемы].cs .

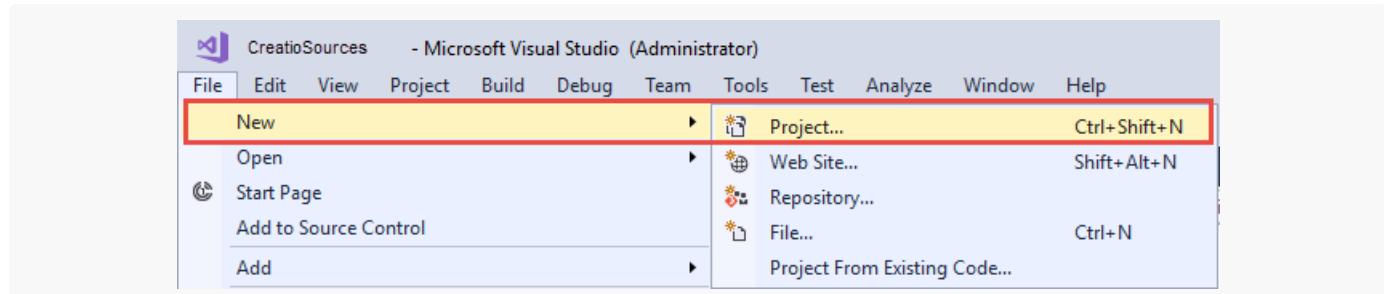
Например: `Contact.Base_Entity.cs` , `ContractReport.Base_Report.cs` .

3. Создать проект Visual Studio для отладки

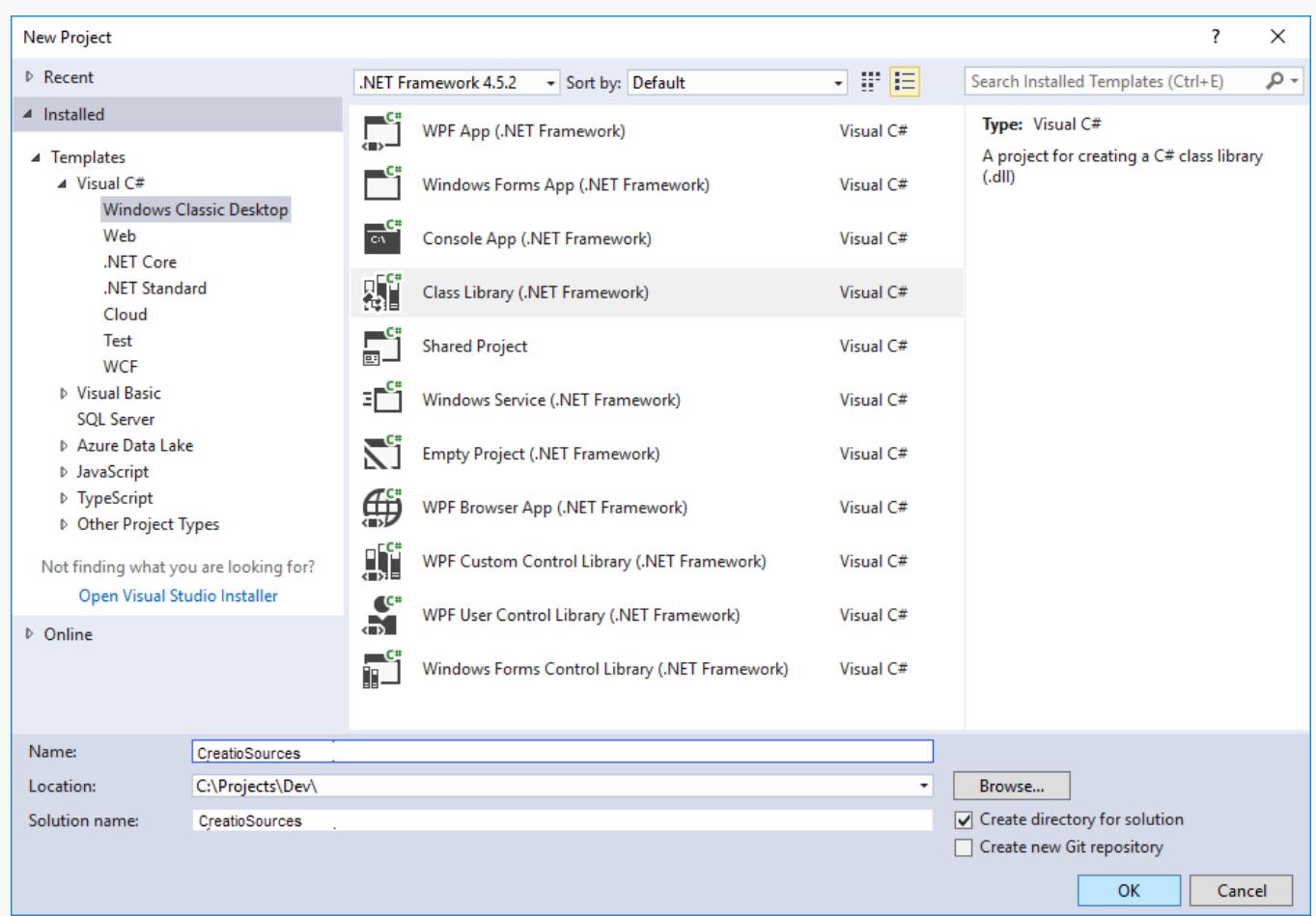
Важно. Для отладки исходного кода создавать проект Visual Studio не обязательно. Достаточно открыть в Visual Studio нужные файлы. Однако если отладка выполняется часто или необходимо работать с большим количеством файлов одновременно, то создание проекта сделает работу более удобной.

Чтобы **создать проект Visual Studio** для отладки:

1. В Visual Studio выполните команду меню [*File*] —> [*New*] —> [*Project*].



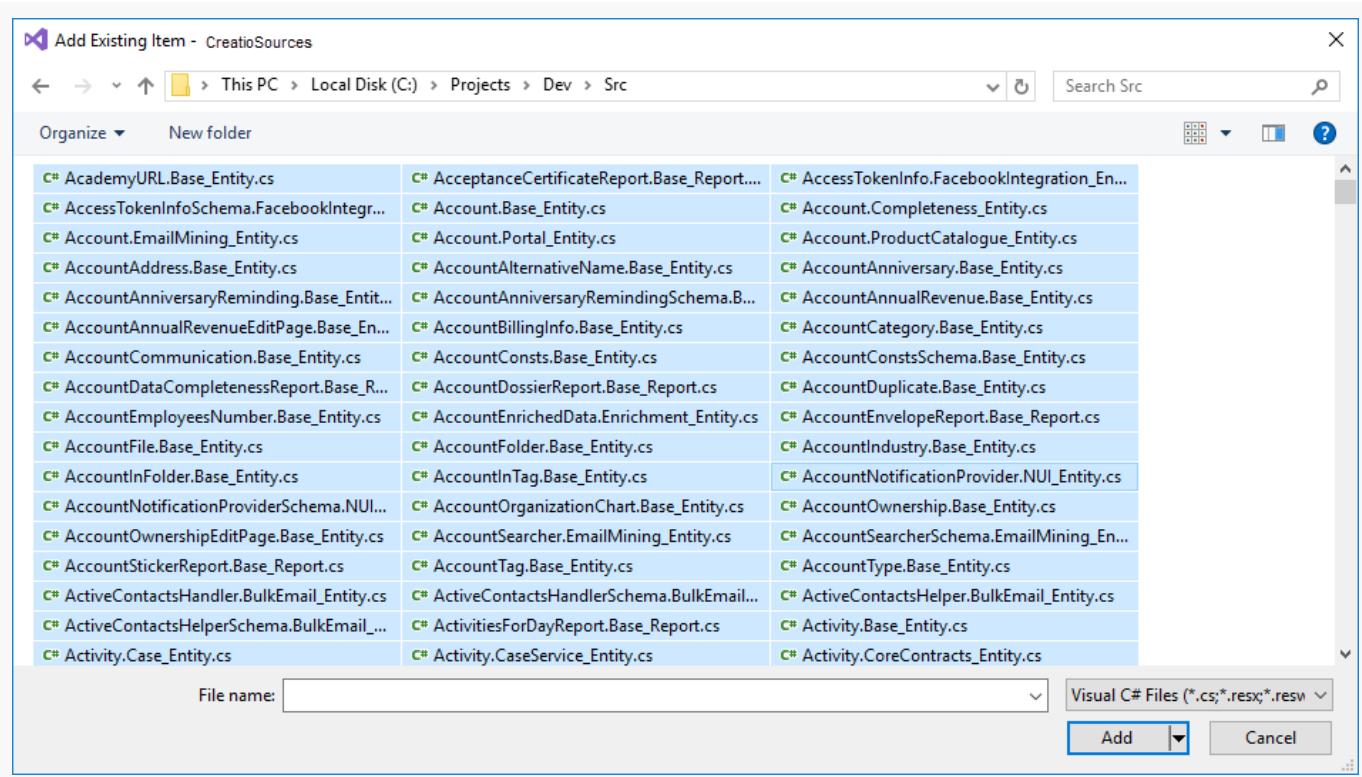
2. В окне свойств создаваемого проекта выберите тип проекта [*Class Library (.NET Framework)*], укажите название и расположение проекта.



- После создания проекта удалите из него файл `Class1.cs`, который был создан по умолчанию, и сохраните проект.

4. Добавить в проект файлы с исходным кодом

- В контекстном меню проекта в проводнике решения выберите команду [*Add*] —> [*Existing Item*].
- Перейдите в каталог с выгруженными файлами с исходным кодом и выберите все файлы.

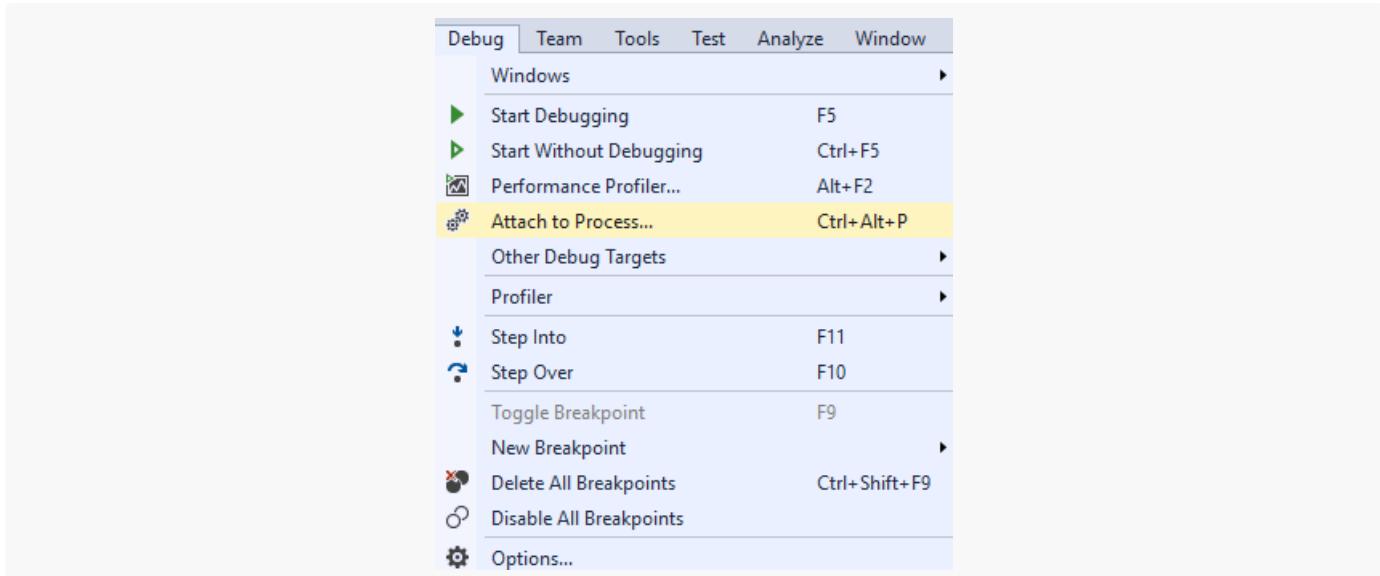


На заметку. В проект Visual Studio можно добавлять только необходимые для отладки файлы. Но тогда переход между методами при отладке будет ограничен только методами классов, реализованных в добавленных в проект файлах.

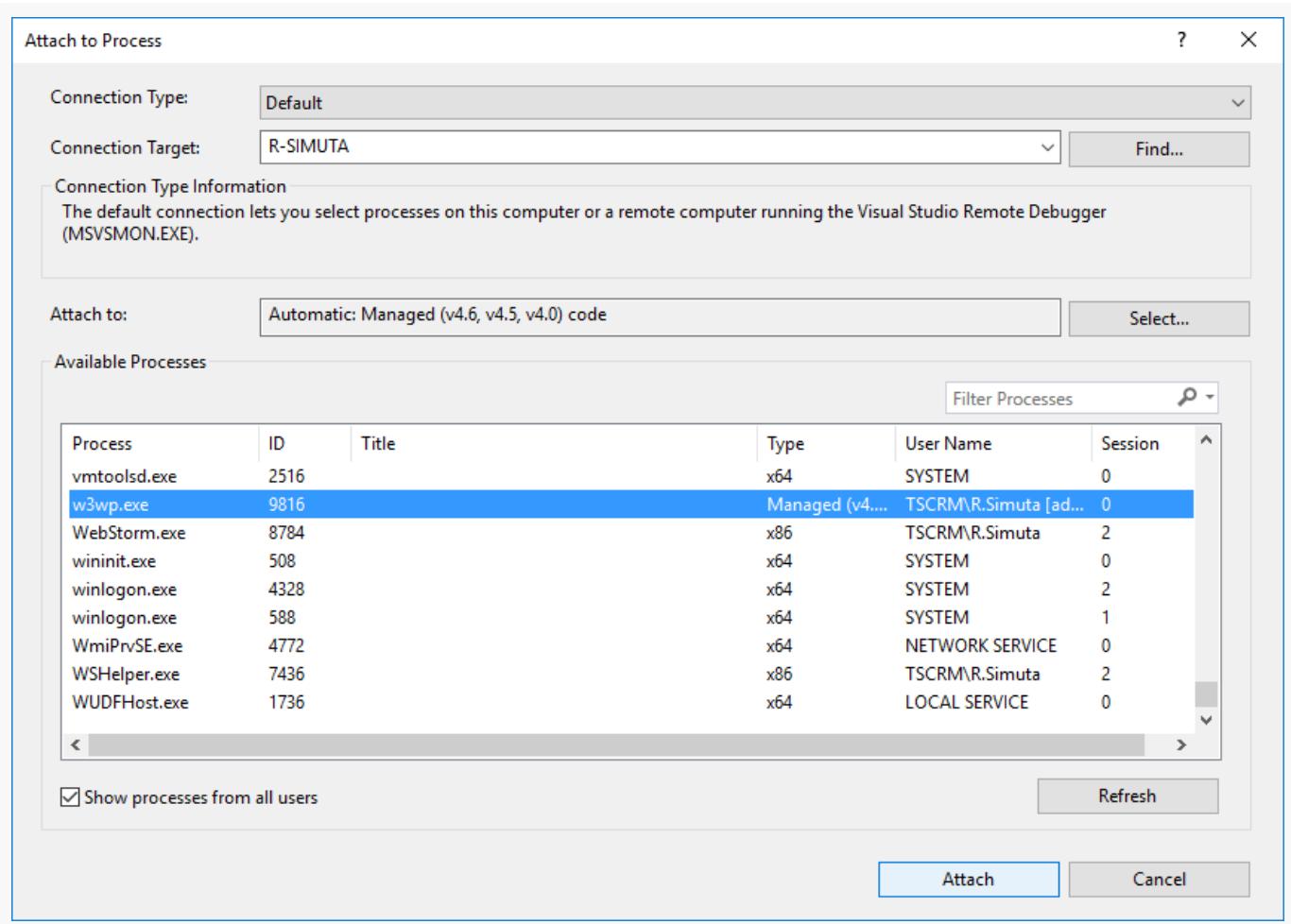
3. После добавления файлов сохраните проект.

5. Подключить проект к рабочему процессу IIS

1. В меню Visual Studio выберите команду [Debug] —> [Attach to process].



2. В списке процессов выберите рабочий процесс IIS, в котором запущен пул приложения Creatio.

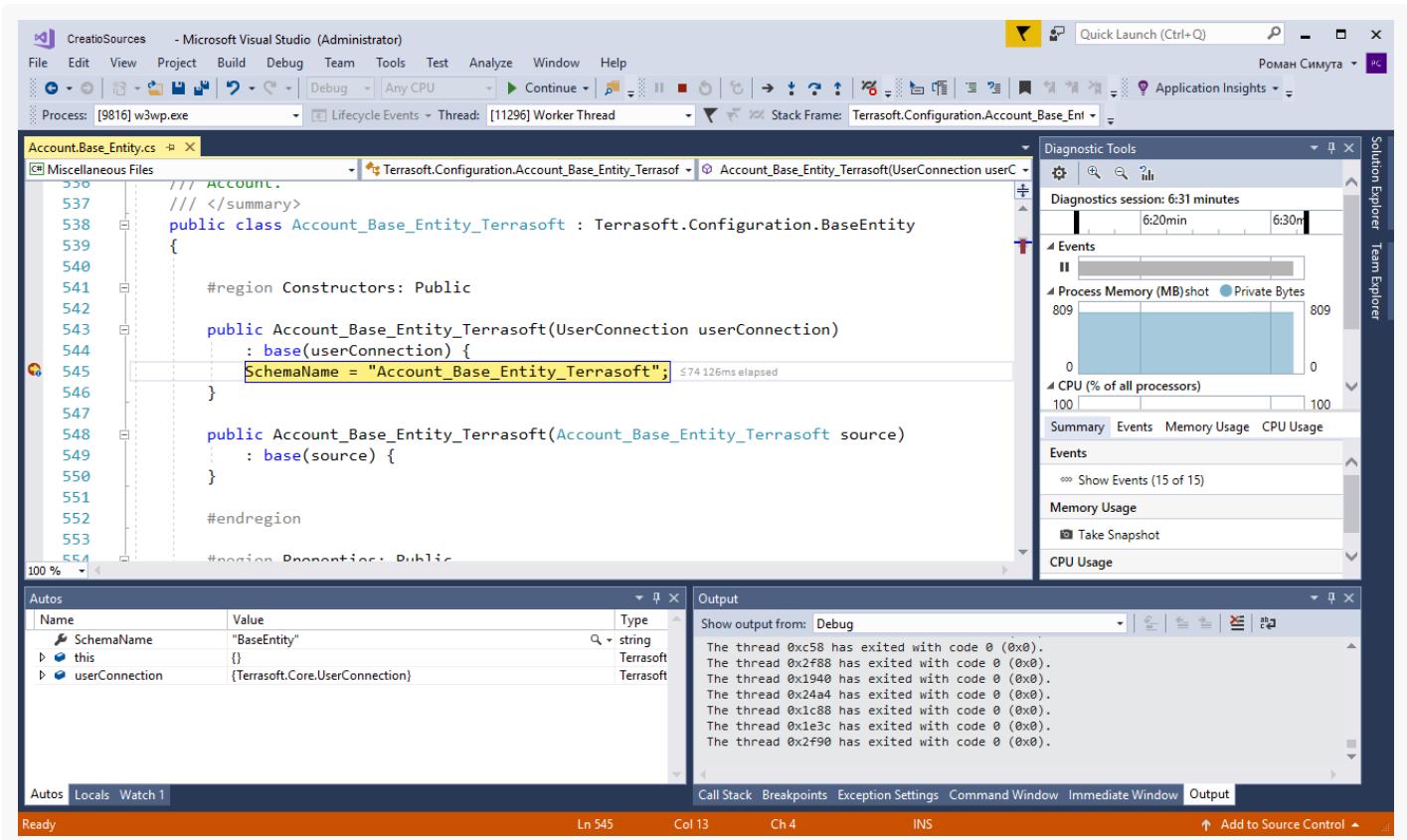


Важно. Название рабочего процесса может различаться в зависимости от конфигурации используемого сервера IIS. Для полнофункционального IIS Web Server процесс называется `w3wp.exe`, для IIS Express — `iisexpress.exe`.

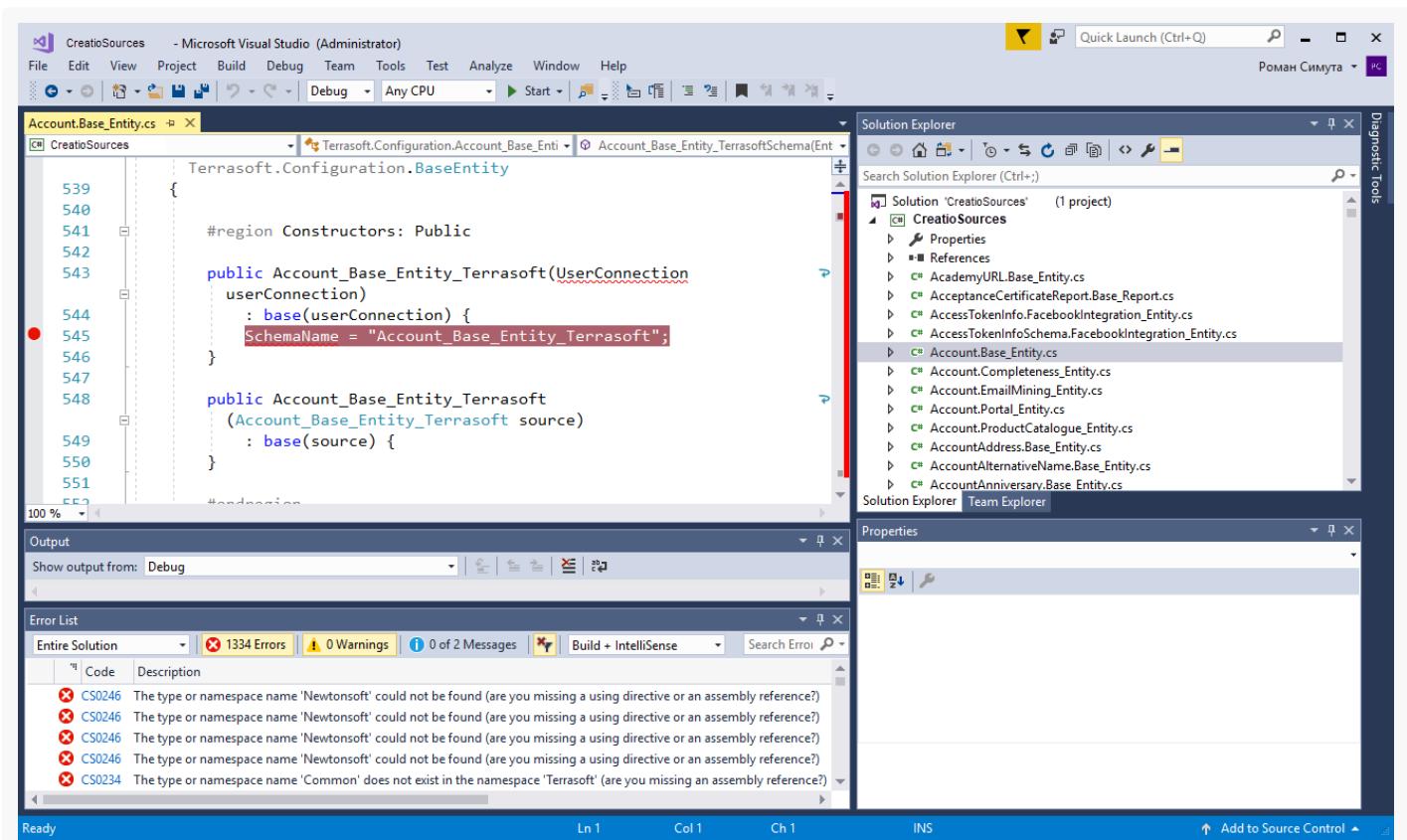
По умолчанию рабочий процесс IIS запущен под учетной записью, имя которой совпадает с именем пула приложения. Чтобы отобразить процессы всех пользователей, а не только текущего, необходимо установить признак [*Show processes from all users*].

6. Выполнить отладку

Откройте файл с необходимым исходным кодом и установите точку останова.



Как только будет задействован метод, на котором была установлена точка останова, программа будет остановлена и можно будет проверить текущее состояние переменных и выполнить трассировку кода.



Разработка конфигурационных элементов



Сложный

Схема — основа конфигурации Creatio. С точки зрения программной реализации схема — это класс ядра, который наследуется от базового класса `Schema`. Конфигурационные элементы (значения выпадающего списка [Добавить] ([Add])) панели инструментов реестра раздела [Конфигурация] ([Configuration])), представлен схемой соответствующего типа (выпадающий список [Тип] ([Type])).

Виды конфигурационных элементов

The screenshot shows the 'Configuration Registry' interface with the 'Types' section open. The left sidebar lists various schema types with their icons:

- Object
- Replacing object
- Source code
- Module
- Page view model
- Section view model
- Detail (list) view model
- Detail (fields) view model
- Replacing view model
- Business process
- User task
- Web service
- SQL script
- Data
- Import

The main pane displays a search results area with a funnel icon and the message: "No records matching search criteria. Modify search criteria".

Типы схем

The screenshot shows a search interface within the Creatio application. At the top, there are buttons for '+ Add' and 'Type' (with a dropdown arrow), a 'Filters' button, a search bar with the placeholder 'Search', and a gear icon for settings. A dropdown menu titled 'All items' is open, listing various item types with corresponding icons:

- SQL script
- Data
- Reference assembly
- Object
- Client module
- Source code
- Business-process
- Case
- User task
- Marketing campaign
- Web service

Below the dropdown menu, there is a link 'Clear or modify search parameters'.

Клиентский модуль

Клиентский модуль — это отдельный блок функциональности, который загружается и запускается по требованию. В соответствии с подходом [AMD](#) и несмотря на некоторые функциональные различия, клиентские модули Creatio имеют одинаковую структуру описания.

Клиентские модули используются для front-end разработки (на языке JavaScript) в приложении Creatio.

Виды схем клиентских модулей:

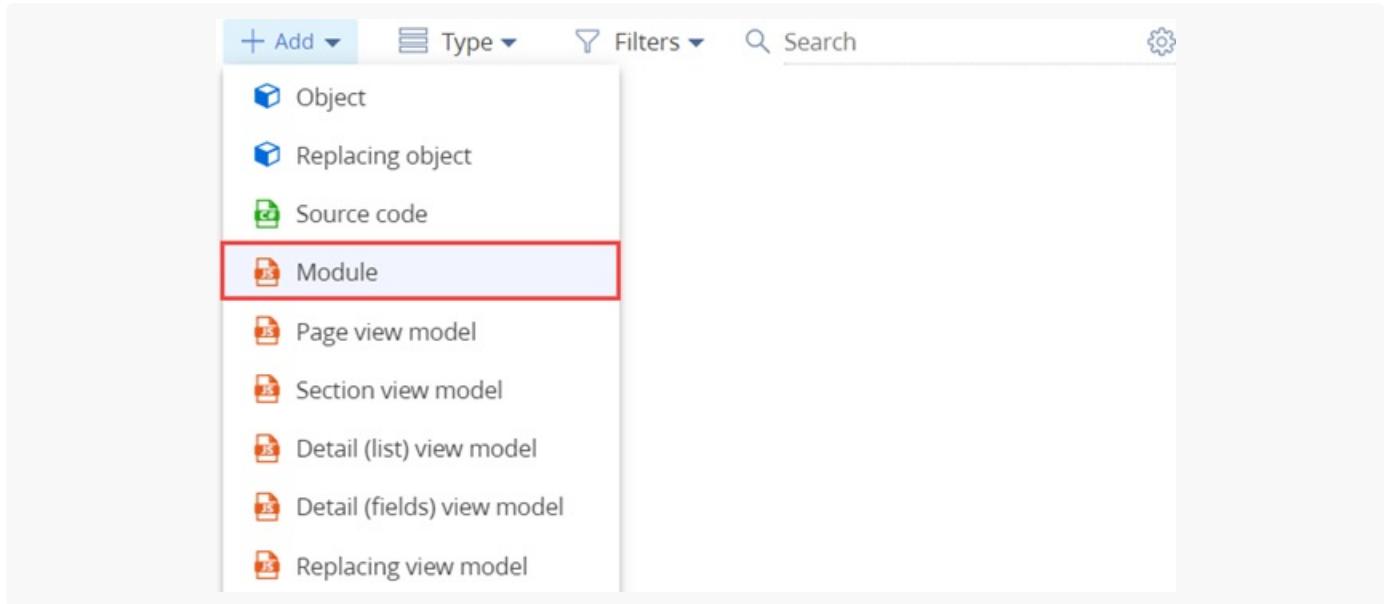
- Невизуальный модуль (схема модуля).
- Визуальный модуль (схема модели представления).
- Модуль расширения и замещающий клиентский модуль (схема замещающей модели представления).

Модули описаны в статье [Виды модулей](#).

Схема модуля

Алгоритм разработки схемы:

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Модуль] ([Add] —> [Module]).



3. В дизайнере модуля заполните свойства схемы.

Основные **свойства** схемы:

- [Код] ([Code]) — название схемы (обязательное свойство). Должно содержать префикс (по умолчанию `Usr`), указанный в системной настройке [Префикс названия объекта] (код `SchemaNamePrefix`), символы латинского алфавита и цифры.
- [Заголовок] ([Title]) — локализуемый заголовок схемы (обязательное свойство).
- [Пакет] ([Package]) — пользовательский пакет, в котором создается схема. Заполняется автоматически и недоступно для редактирования.
- [Описание] ([Description]) — локализуемое описание схемы.

The screenshot shows a configuration dialog titled "Module". It contains the following fields:

- Code ***: UsrClientUnit
- Title ***: Client Schema (with a red asterisk)
- Package**: Custom
- Description**

At the bottom right are two buttons: "CANCEL" and a blue "APPLY" button.

Для применения заданных свойств нажмите [Применить] ([*Apply*]).

Панель свойств позволяет изменить основные свойства схемы (кнопка) и задать дополнительные (кнопка). Дополнительными свойствами являются [Локализуемые строки] ([*Localizable strings*]), [Сообщения] ([*Messages*]), [Изображения] ([*Images*]).

4. В дизайнере модуля добавьте исходный код. Название модуля в функции `define()` должно совпадать с названием схемы (свойство [Код] ([*Code*])).

Если при написании кода допущена ошибка, то слева возле номера строки отображается тип ошибки (ошибка или предупреждение). При наведении курсора на тип ошибки отображается всплывающая подсказка с текстовым описанием.

5. На панели инструментов дизайнера модуля нажмите [Сохранить] ([*Save*]).

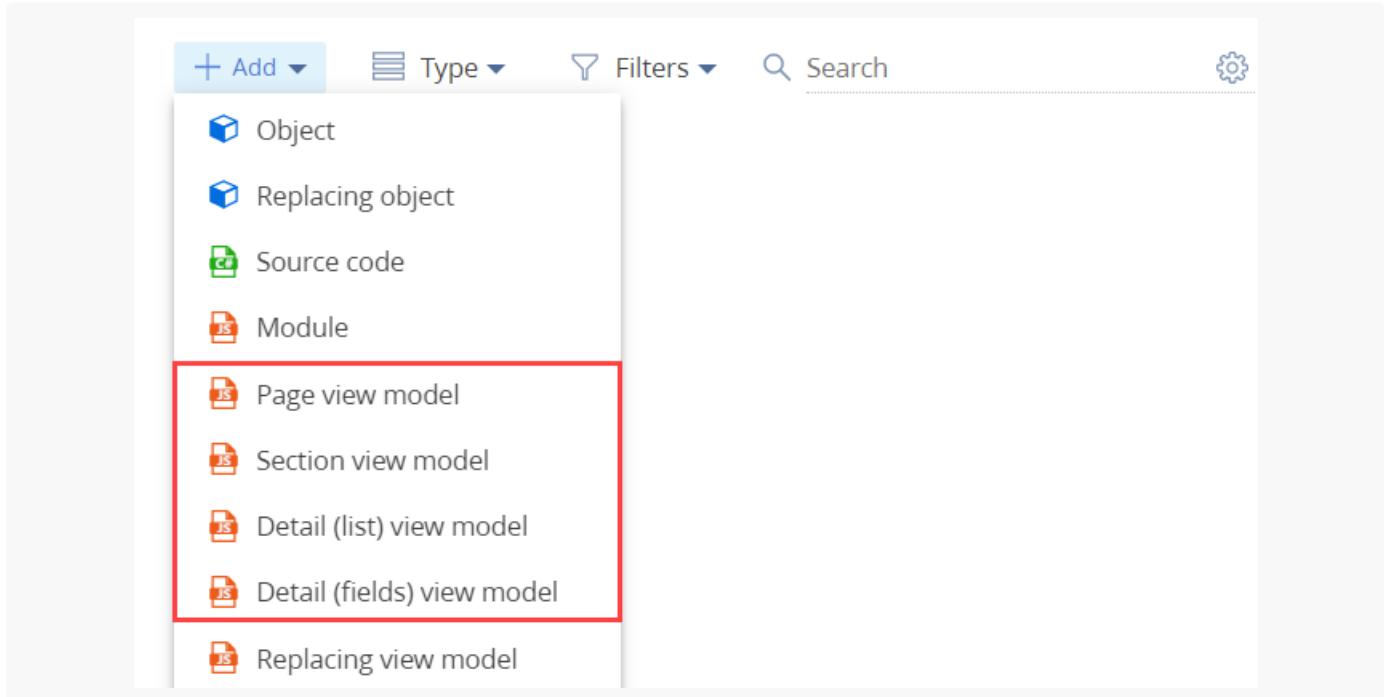
Схема модели представления

Виды схемы модели представления:

- Схема страницы записи раздела (пункт [*Модель представления страницы*] ([*Page view model*])).
- Схема страницы раздела с реестром и итогами (пункт [*Модель представления раздела*] ([*Section view model*])).
- Схема страницы детали с реестром (пункт [*Модель представления детали с реестром*] ([*Detail (list) view model*])).
- Схема страницы детали с полями ([*Модель представления детали с полями*] ([*Detail (fields) view model*])).

Алгоритм разработки схемы:

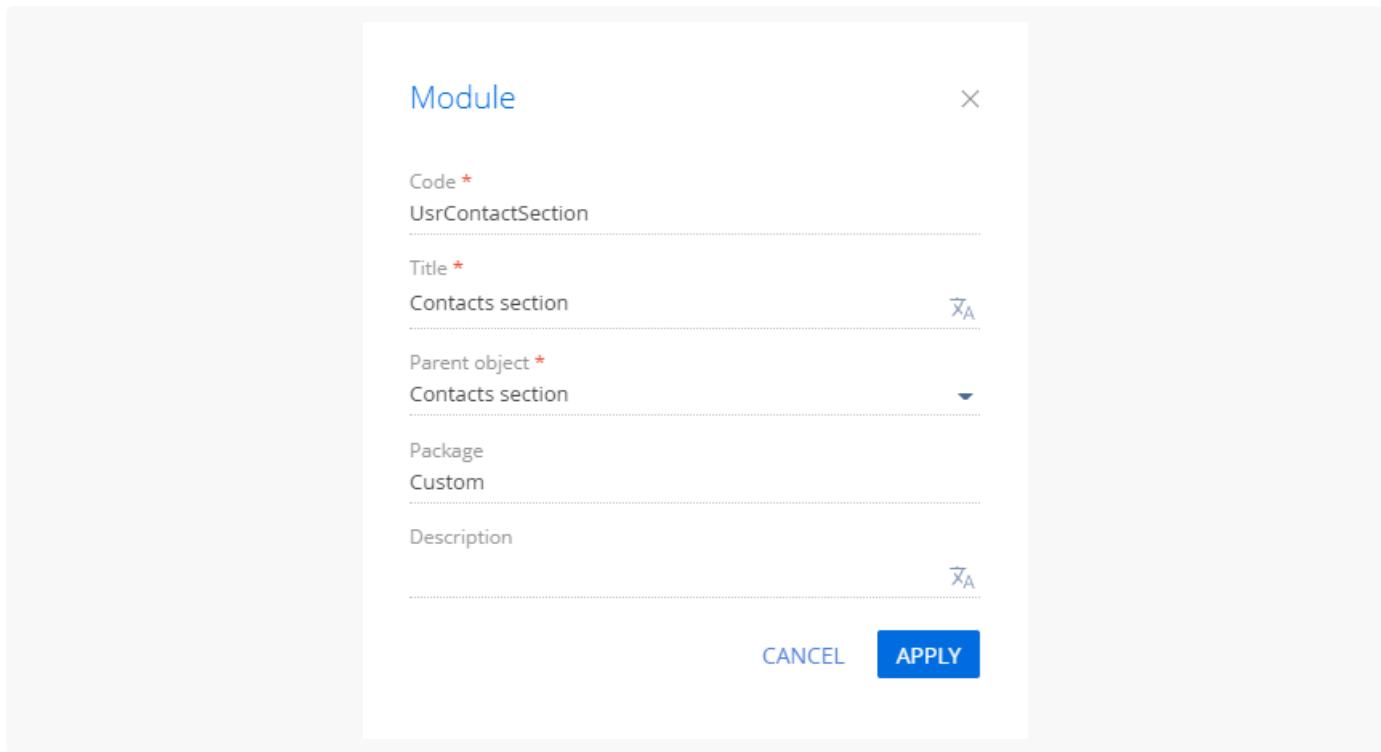
1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] ([Add]) и выберите вид схемы модели представления.



3. В дизайнере модуля заполните свойства схемы.

Основные **свойства** схемы:

- [Код] ([Code]) — название схемы (обязательное свойство). Должно содержать префикс (по умолчанию `Usr`), указанный в системной настройке [Префикс названия объекта] (код `SchemaNamePrefix`), символы латинского алфавита и цифры.
- [Заголовок] ([Title]) — локализуемый заголовок схемы (обязательное свойство).
- [Пакет] ([Package]) — пользовательский пакет, в котором создается схема. Заполняется автоматически и недоступно для редактирования.
- [Родительский объект] ([Parent object]) — родительский объект для текущего объекта. В выпадающем списке выберите родительский объект, свойства которого необходимо наследовать.
- [Описание] ([Description]) — локализуемое описание схемы.



Для применения заданных свойств нажмите [Применить] ([*Apply*]).

Панель свойств позволяет изменить основные свойства схемы (кнопка) и задать дополнительные (кнопка). Дополнительными свойствами являются [Локализуемые строки] ([*Localizable strings*]) и [Изображения] ([*Images*]).

4. В дизайнере модуля добавьте исходный код. Название модуля в функции `define()` должно совпадать с названием схемы (свойство [Код] ([*Code*])). Схема модели представления обязательно должна быть наследником базовой схемы `BaseModulePageV2` .

Если при написании кода допущена ошибка, то слева возле номера строки отображается тип ошибки (ошибка или предупреждение). При наведении курсора на тип ошибки отображается всплывающая подсказка с текстовым описанием.

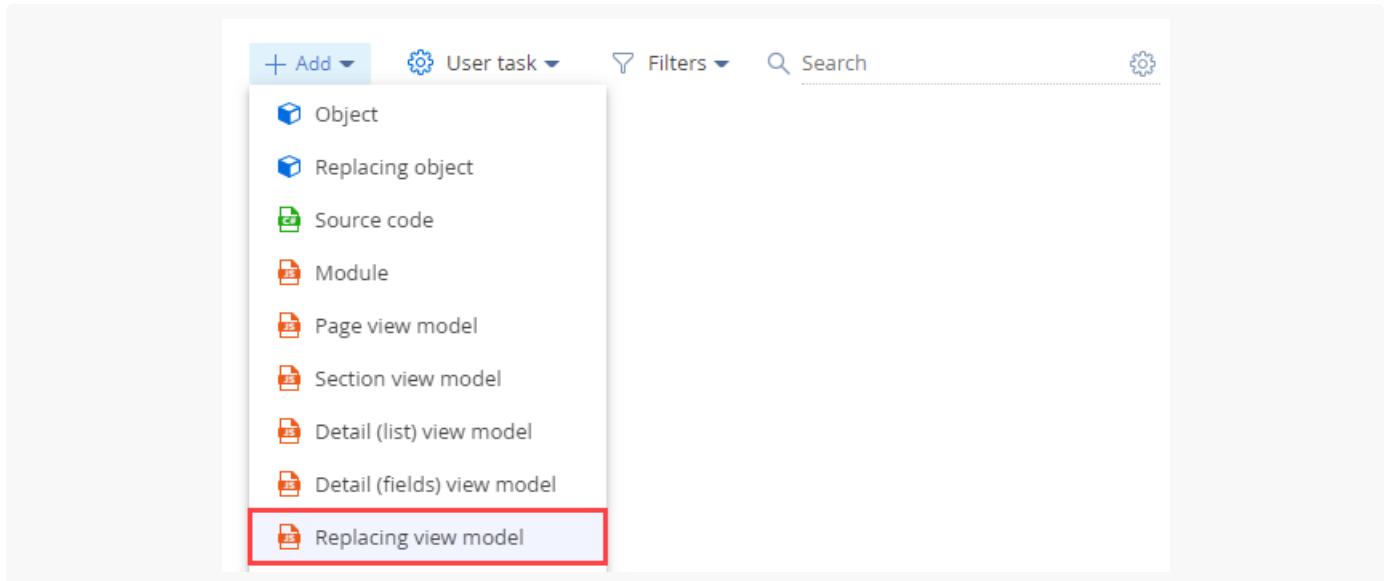
5. На панели инструментов дизайнера модуля нажмите [Сохранить] ([*Save*]).

Схема замещающей модели представления

Схемы **замещающих моделей представления** предназначены для расширения функциональности существующих схем. При этом существующие схемы также могут быть замещающими и принадлежать разным пакетам.

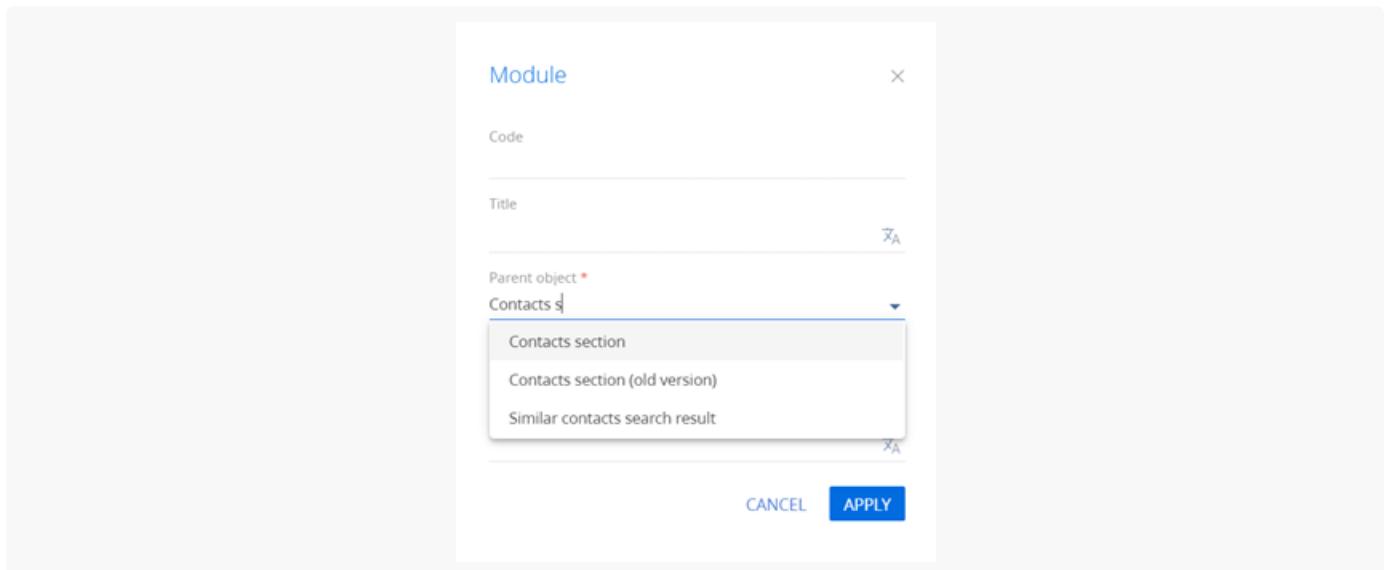
Алгоритм разработки схемы:

1. [Перейдите в раздел \[Конфигурация \]](#) ([*Configuration*]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Замещающая модель представления] ([*Add*] —> [*Replacing view model*]).

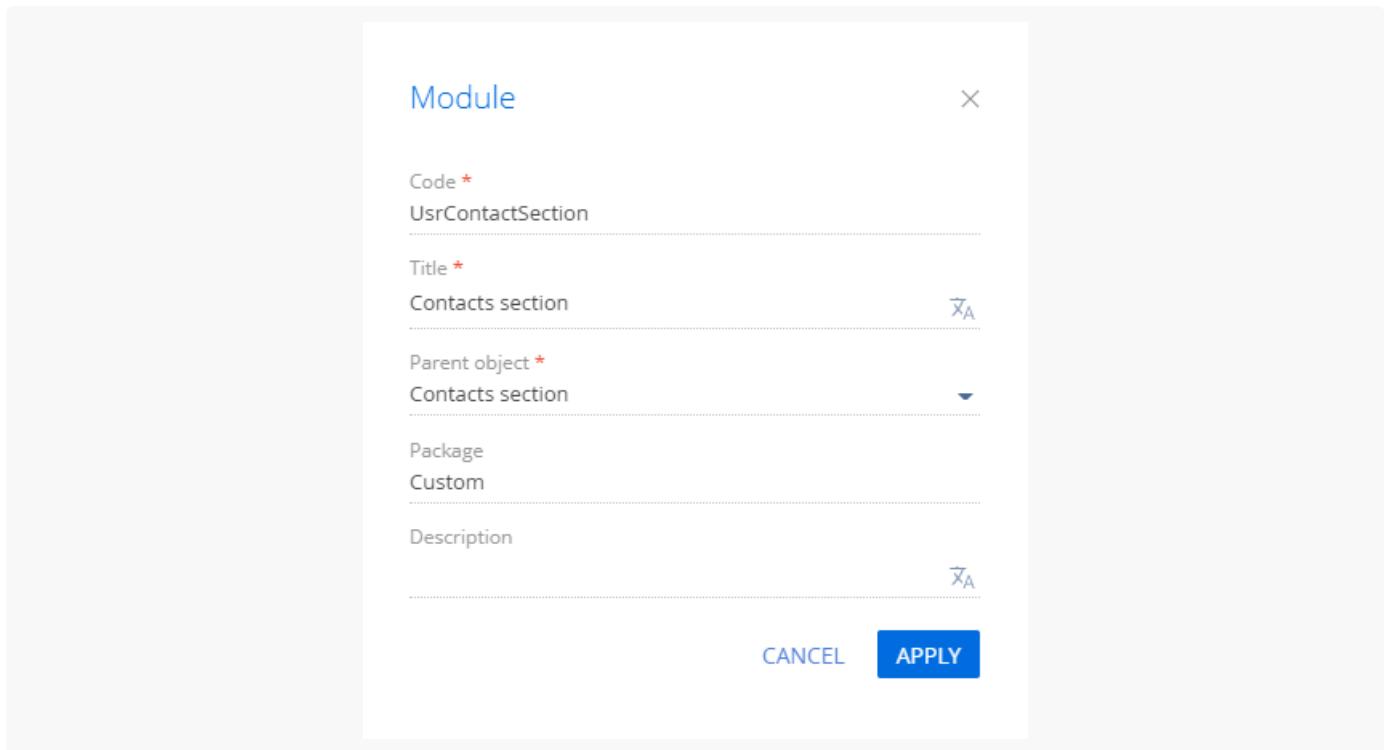


3. В дизайнере модуля выберите родительский объект.

Чтобы модуль замещал раздел или страницу, в обязательном свойстве [Родительский объект] ([Parent object]) схемы укажите заголовок той базовой схемы модели представления, которую необходимо заместить. Например, для создания замещающей схемы раздела [Контакты] ([Contacts]) в качестве родительского объекта необходимо указать схему ContactSectionV2. Для этого в поле [Родительский объект] ([Parent object]) свойств замещающей схемы необходимо начать вводить заголовок "Раздел контакты" ("Contacts section") и выбрать нужное значение из выпадающего списка.



После подтверждения выбранного родительского объекта остальные свойства будут заполнены автоматически.



Для применения заданных свойств нажмите [Применить] ([*Apply*]).

Панель свойств позволяет изменить основные свойства схемы (кнопка) и задать дополнительные (кнопка). Дополнительными свойствами являются [Локализуемые строки] ([*Localizable strings*]) и [Изображения] ([*Images*]).

4. В дизайнере модуля добавьте исходный код. Название модуля в функции `define()` должно совпадать с названием схемы (свойство [Код] ([*Code*])).

Если при написании кода допущена ошибка, то слева возле номера строки отображается тип ошибки (ошибка или предупреждение). При наведении курсора на тип ошибки отображается всплывающая подсказка с текстовым описанием.

5. На панели инструментов дизайнера модуля нажмите [Сохранить] ([*Save*]).

Объект

Объектный слой ORM ([Object-relational mapping](#)) в Creatio основан на объектах (`Entity`). **Объект** — это бизнес-сущность, которая на уровне серверного ядра позволяет объявить новый класс ORM-модели. На уровне базы данных создание объекта означает создание записи таблицы с таким же именем, как у созданного объекта, и с таким же набором колонок. То есть в большинстве случаев каждый объект в системе является системным представлением одной физической таблицы в базе данных.

Объект, как элемент конфигурации, представлен схемой, которая реализована соответствующим классом `EntitySchema` . Именно в схеме объекта описывается набор колонок, индексов и методов объекта.

Виды схем объектов:

- Базовые. Недоступны для редактирования, находятся в предустановленных [пакетах](#). Базовые схемы могут замещаться пользовательскими.
- Пользовательские. Создаются при кастомизации, находятся в пользовательских пакетах.

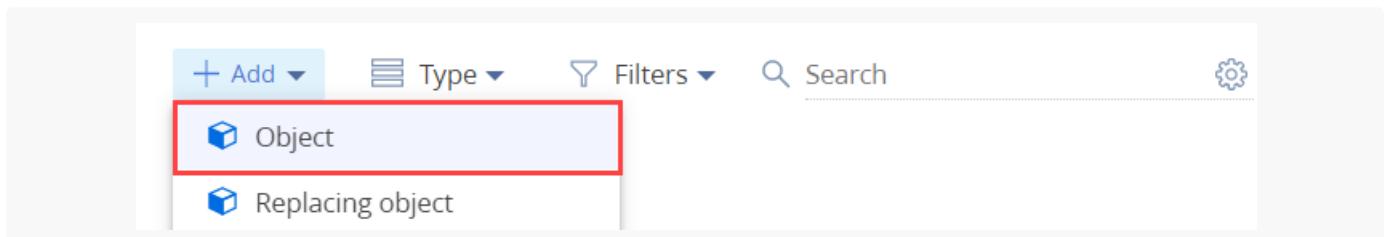
Платформа Creatio не ограничивает количество колонок объекта. Количество колонок в объекте ограничивается максимально допустимым количеством столбцов в таблицах базы данных, которую использует клиент.

Объекты используются для back-end разработки (на языке C#) в приложении Creatio.

Схема объекта

Алгоритм разработки схемы:

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Объект] ([Add] —> [Object]).



3. В дизайнере объекта заполните свойства схемы.

Основные **свойства** схемы:

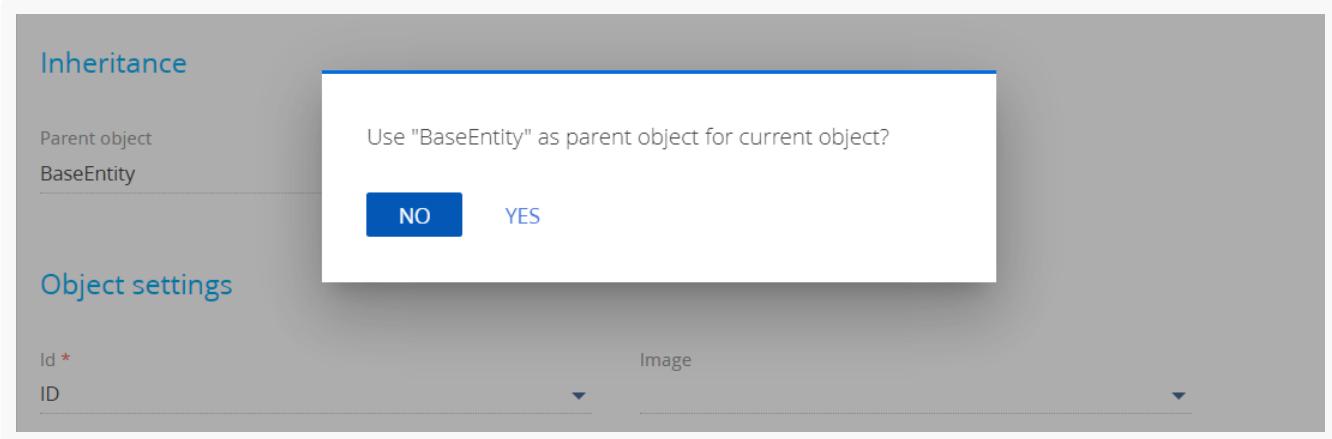
- [Код] ([Code]) — название схемы (обязательное свойство). Должно содержать префикс (по умолчанию `Usr`), указанный в системной настройке [Префикс названия объекта] (код [`SchemaNamePrefix`]), символы латинского алфавита и цифры. Допустимая длина имени объекта — 128 символов. На базах Oracle ниже версии 12.2 не допускаются к использованию объекты с длиной имени более 30 символов.
- [Заголовок] ([Title]) — локализуемый заголовок схемы (обязательное свойство).

General	
Code*	Title*
UsrEntity	Object
Package	Description
Custom	

- [Родительский объект] ([Parent object]) — родительский объект для текущего объекта.

Чтобы объект наследовал функциональность базового объекта, в свойстве [Родительский объект] ([Parent object]) схемы укажите код той базовой схемы объекта, функциональность которой необходимо наследовать. Например, для наследования функциональности базовой схемы `BaseEntity` в поле [Родительский объект] ([Parent object]) свойств схемы необходимо начать вводить код `BaseEntity` и выбрать нужное значение из выпадающего списка. После подтверждения выбранного родительского объекта к структуре объекта будут добавлены колонки, унаследованные от базового объекта.

Подтверждение использования родительского объекта



Унаследованные колонки в структуре объекта

- [**Идентификатор**] ([*Id*]) — системная колонка, используемая в качестве первичного ключа в таблице базы данных (обязательное свойство). Заполняется автоматически после установки свойства [**Родительский объект**] ([*Parent object*]).

Поскольку объект в системе является представлением таблицы в базе данных, то он обязательно должен содержать колонку-идентификатор. Для установки значения свойства [**Идентификатор**] ([*Id*]) в качестве родительского объекта укажите один из базовых объектов системы или в выпадающем списке выберите пользовательскую колонку типа [**Уникальный идентификатор**] ([*Unique identifier*]). Добавление пользовательской колонки рассмотрено ниже. Если попытаться сохранить схему объекта без идентификатора, то система выдаст предупреждение.

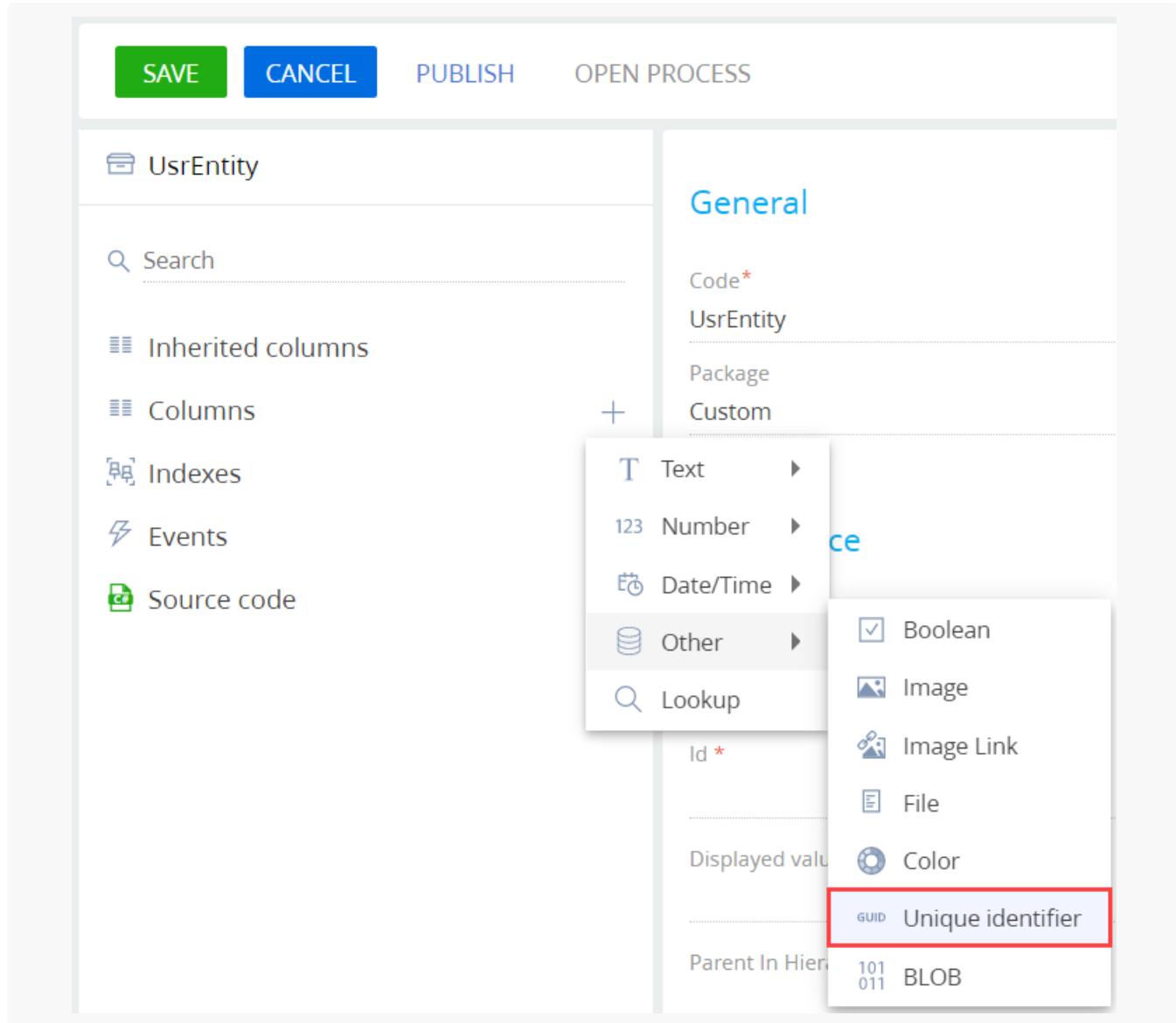
4. Добавьте пользовательскую колонку в объект.

Алгоритм добавления в объект пользовательской колонки:

- В контекстном меню узла [**Колонки**] ([*Columns*]) структуры объекта нажмите **+**.
- В выпадающем меню выберите тип колонки и задайте ее свойства.

Для добавления колонки-идентификатора нажмите [**Другие**] —> [**Уникальный идентификатор**]

([Other] —> [Unique identifier]).



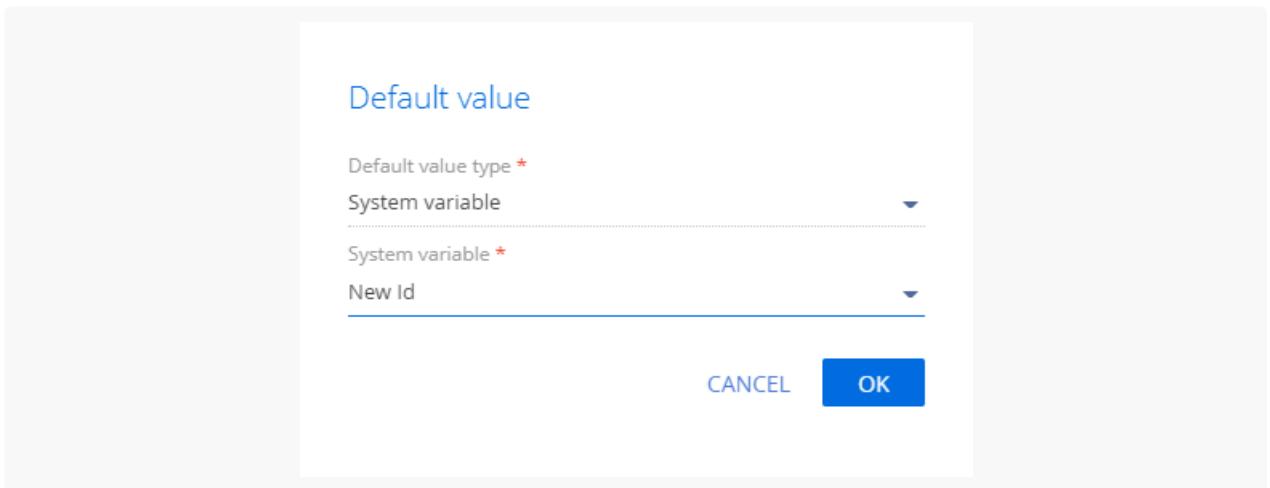
c. В дизайнере объекта заполните свойства добавляемой колонки.

Основные **свойства** добавляемой колонки:

- [Код] ([Code]) — название колонки (обязательное свойство). Значение по умолчанию устанавливается дизайнером объекта и может быть изменено.
- [Заголовок] ([Title]) — локализуемый заголовок колонки (обязательное свойство).
- [Тип данных] ([Data type]) — тип данных, содержащихся в колонке. Значение по умолчанию устанавливается дизайнером объекта в зависимости от выбранной команды добавления колонки.
- [Обязательное] ([Required]) — обязательность колонки. Выберите "На уровне приложения" ("Application Level"), поскольку колонка должна обязательно содержать значение.
- [Значение по умолчанию] ([Default value]) — значение по умолчанию.

Для установки значения по умолчанию нажмите и заполните **поля**:

- [Тип значения] ([*Default value type*]) — выберите "Системная переменная" ("System variable").
- [Системная переменная] ([*System variable*]) — выберите "Новый идентификатор" ("New Id"), поскольку идентификаторы должны быть уникальными.



- [Режим использования] ([*Usage mode*]) — выберите "Расширенный" ("Advanced").

The screenshot shows the 'General' tab of a configuration element. It includes the following fields:

- Code***: UsrId
- Title***: Id
- Data type**: Unique identifier
- Description**
- Required**: Application Level
- Default value**: (None)

A checkbox labeled "Copy this value when copying records" is checked. Below the General tab are tabs for "Data source" and "Behavior".

Режимы использования колонок, реализованные в Creatio IDE:

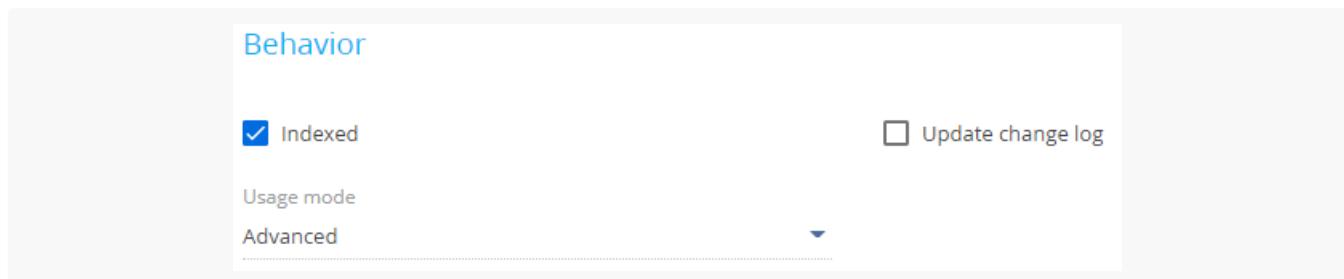
- [Общие] ([*General*]) — стандартный режим колонок в приложении.
- [Расширенный] ([*Advanced*]) — колонка отображается в конфигурации и доступна для

использования в приложении.

- [Никогда] ([*None*]) — колонка отображается в конфигурации как системная и недоступна для использования в приложении.
- I. На панели инструментов дизайнера объекта нажмите [Сохранить] ([*Save*]) для временного сохранения изменений в метаданных.
- m. Добавьте индексы в объект.

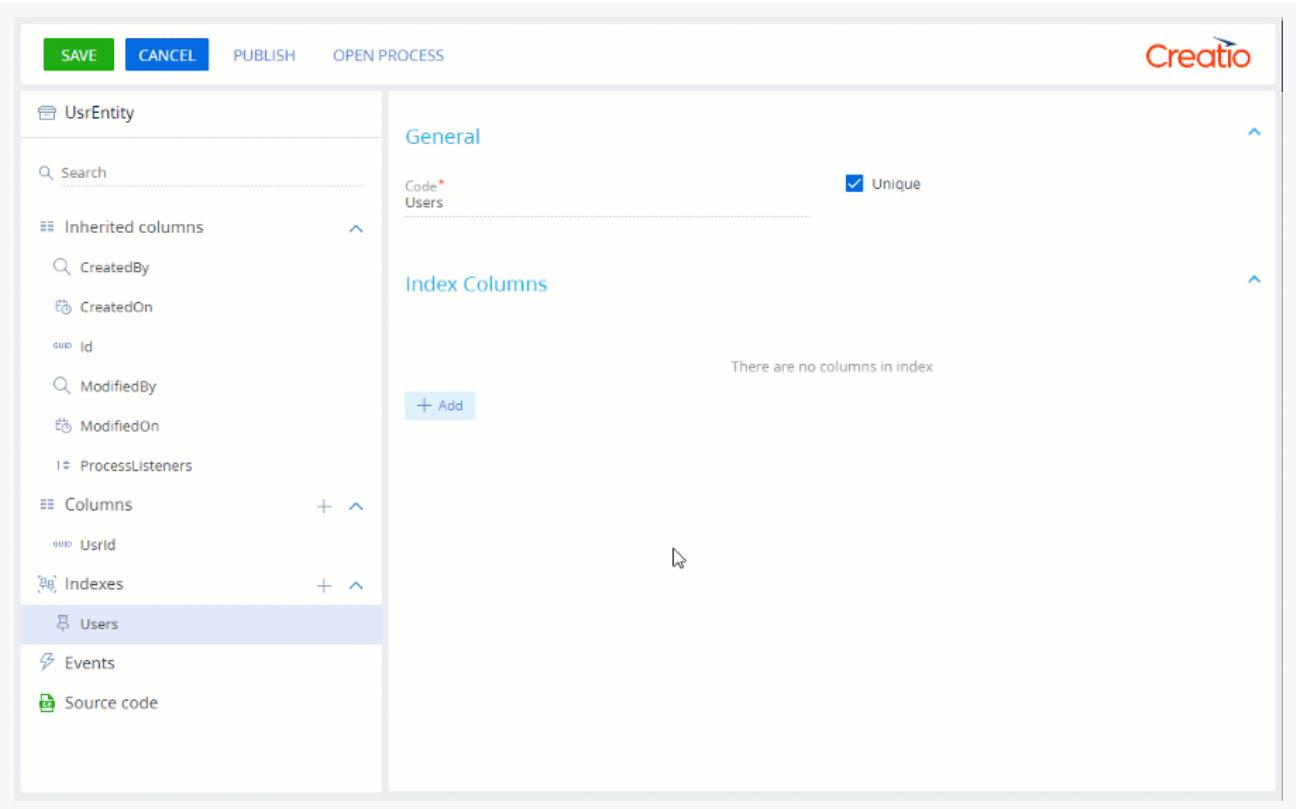
Кроме колонок, в объект могут быть добавлены индексы, которые при публикации объекта будут автоматически созданы в таблице базы данных.

В блоке свойств [Поведение] ([*Behavior*]) установите признак [Индексируемая] ([*Indexed*]), если необходимо создать индекс по одной колонке. В системе по умолчанию справочные колонки являются индексируемыми.



Алгоритм добавления составного индекса:

- a. Задайте название индекса. Для этого в контекстном меню элемента [Индексы] ([*Indexes*]) нажмите + и в поле [Код] ([*Code*]) укажите пользовательское название.
- b. Установите признак [Уникальный] ([*Unique*]) если для колонок индекса необходимо реализовать ограничение целостности (исключить возможность вставки повторяющихся комбинаций значений).
- c. Добавьте необходимые колонки в индекс. Для этого в блоке [Колонки индекса] ([*Index Columns*]) нажмите [Добавить] ([*Add*]), выберите колонку объекта и укажите направление сортировки.

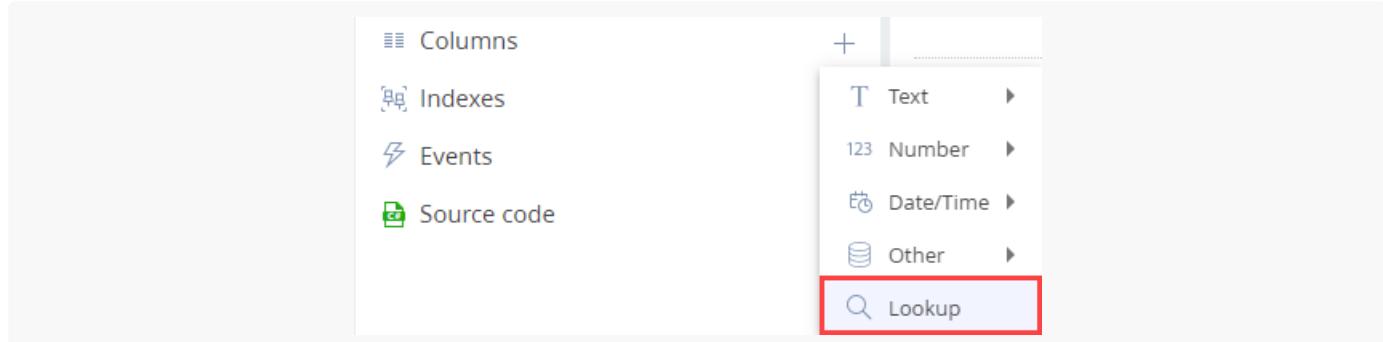


- п. На панели инструментов дизайнера объекта нажмите [Сохранить] ([Save]) для временного сохранения изменений в метаданных схемы.
- о. На панели инструментов дизайнера объекта нажмите [Опубликовать] ([Publish]) для окончательного сохранения схемы и создания соответствующей таблицы в базе данных.

Для объекта можно установить **каскадную связь**. Она настраивается для колонки типа [Справочник] ([Lookup]).

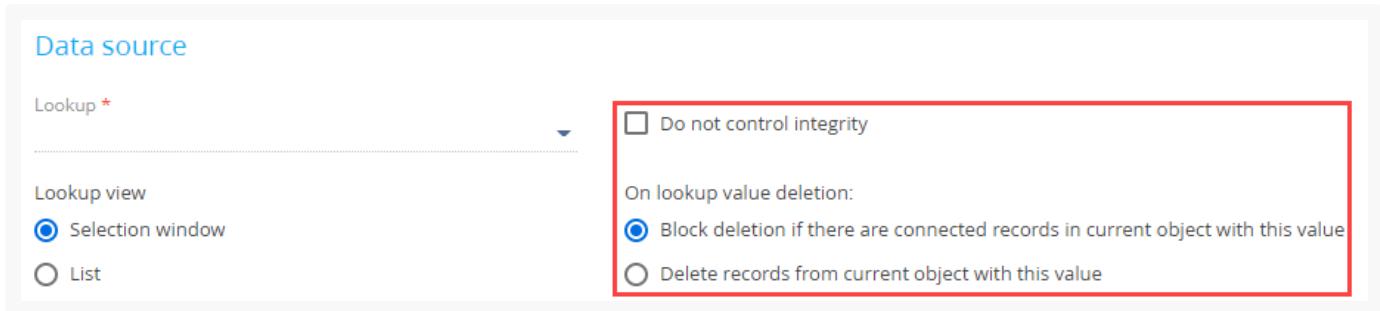
Алгоритм добавления в объект колонки типа [Справочник] ([Lookup]):

1. В контекстном меню узла [Колонки] ([Columns]) структуры объекта нажмите +.
2. Для добавления колонки типа [Справочник] ([Lookup]) нажмите [Справочник] ([Lookup]).



Каскадная связь настраивается в блоке свойств [Источник данных] ([Data source]) с помощью:

- признака [Не контролировать целостность] ([Do not control integrity]).
- опций пункта [При удалении значения справочника] ([On lookup value deletion]).



Рассмотрим каскадную связь на примере объекта [Контакт] ([Contact]), который связан по справочной колонке [AccountId] с объектом [Контрагент] ([Account]). Для этого в поле [Выбор объекта] ([Lookup]) выберите [Account].

Варианты настройки каскадной связи:

- Если установлен признак [Не контролировать целостность] ([Do not control integrity]), то удаление контрагента будет выполнено. При этом не будут удалены контакты, связанные с текущим контрагентом.
- Если не установлен признак [Не контролировать целостность] ([Do not control integrity]) и выбрана опция [Блокировать удаление, если есть связанные записи в текущем объекте с этим значением] ([Block deletion if there are connected records in current object with this value]), то удаление контрагента будет заблокировано, если присутствуют контакты, связанные с текущим контрагентом. В этом случае приложение выдаст предупреждающее сообщение. После подтверждения удаление контрагента будет выполнено. При этом не будут удалены контакты, связанные с текущим контрагентом.
- Если не установлен признак [Не контролировать целостность] ([Do not control integrity]) и выбрана опция [Удалять записи из текущего объекта с этим значением] ([Delete records from current object with this value]), то удаление контрагента будет выполнено вместе с удалением контактов, связанных с текущим контрагентом.

Схема замещающего объекта

Схемы **замещающих объектов** предназначены для расширения функциональности существующих схем. При этом существующие схемы также могут быть замещающими и принадлежать разным пакетам.

Алгоритм разработки схемы:

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Замещающий объект] ([Add] —> [Replacing object]).

The screenshot shows a list of objects in a configuration interface. The objects listed are:

	Status	Type
Object		Object
Replacing object		Object
Source code	/ page	Client module
Module		

3. В дизайнере объекта выберите родительский объект.

Чтобы объект замещал функциональность базового объекта, в обязательном свойстве [Родительский объект] ([Parent object]) схемы укажите название той базовой схемы объекта, функциональность которой необходимо заменить. Например, для замещения функциональности базовой схемы `BaseEntity` в поле [Родительский объект] ([Parent object]) свойств схемы необходимо начать вводить код `BaseEntity` и выбрать нужное значение из выпадающего списка. После подтверждения выбранного родительского объекта остальные свойства будут заполнены автоматически.

General

Code*	Title*
BaseEntity	Base object
Package	Description
Custom	

Inheritance

Parent object *	<input checked="" type="checkbox"/> Replace parent
BaseEntity	

Object settings

Id *	Image
Id	
Displayed value	Sorting In Lists
Parent In Hierarchy	Owner
Change Log Object Name	Permission Object Name
Localization Object Name	

4. На панели инструментов дизайнера объекта нажмите [Сохранить] ([Save]) для временного

сохранения изменений в метаданных схемы.

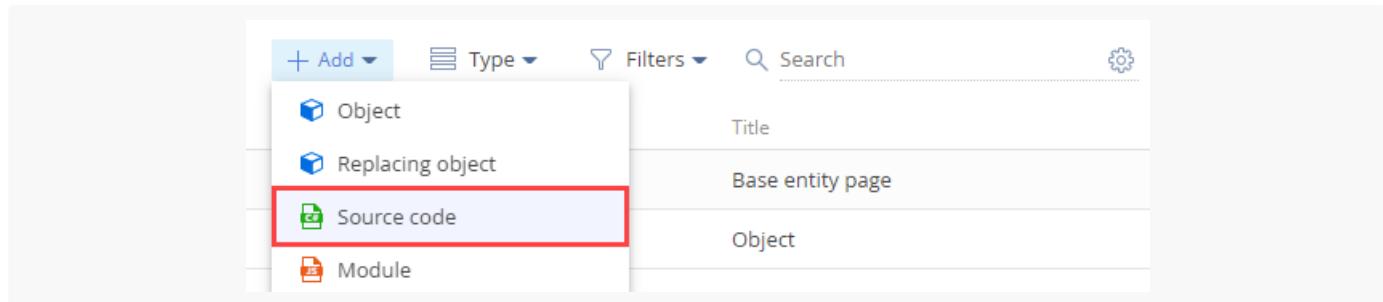
- На панели инструментов дизайнера объекта нажмите [*Опубликовать*] ([*Publish*]) для выполнения изменений на уровне базы данных. Результатом успешной публикации объекта являются созданные (или измененные) физические объекты в базе данных — таблицы, столбцы, индексы.

Исходный код

Схема [*Исходный код*] ([*Source code*]) используется для back-end разработки (на языке C#) в приложении Creatio.

Алгоритм разработки схемы:

- [Перейдите в раздел \[Конфигурация \]](#) ([*Configuration*]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
- На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).

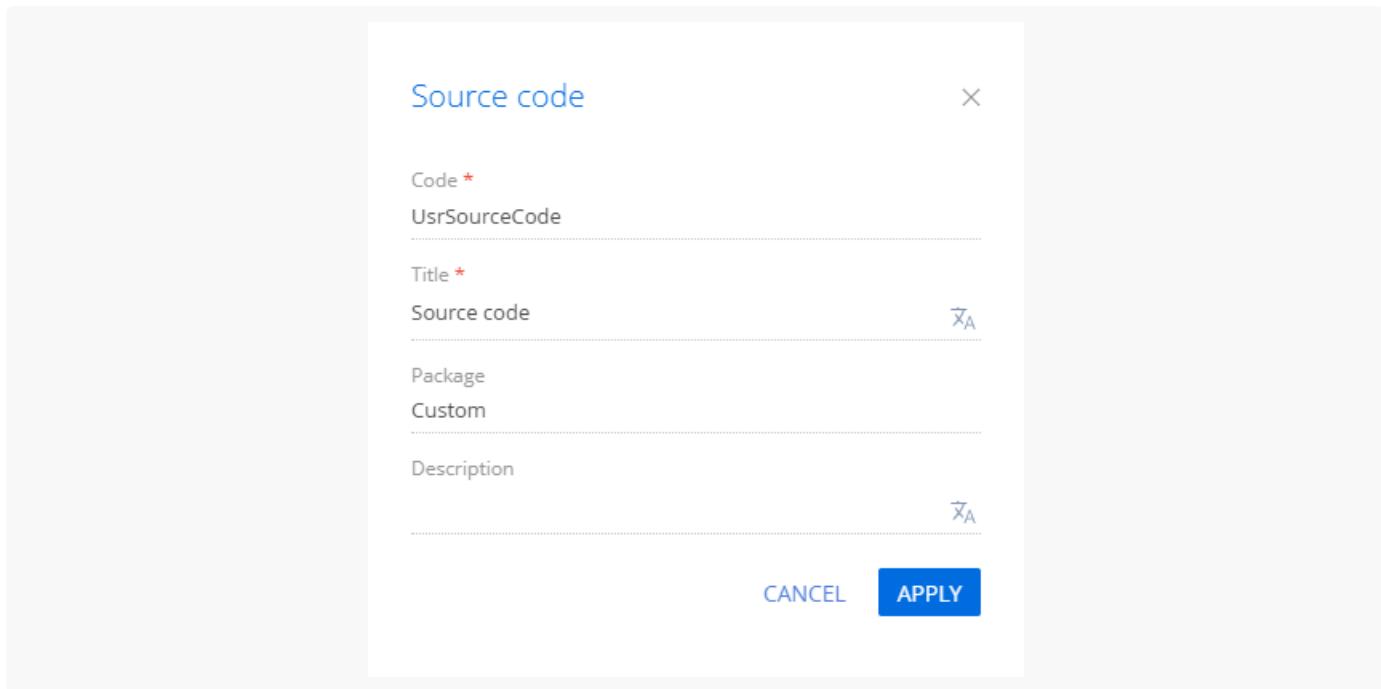


- В дизайнере исходного кода заполните свойства схемы.

Основные **свойства** схемы:

- [*Код*] ([*Code*]) — название схемы (обязательное свойство). Должно содержать префикс (по умолчанию `usr`), указанный в системной настройке [*Префикс названия объекта*] (код [*SchemaNamePrefix*]), символы латинского алфавита и цифры.
- [*Заголовок*] ([*Title*]) — локализуемый заголовок схемы (обязательное свойство).
- [*Пакет*] ([*Package*]) — пользовательский пакет, в котором создается схема. Заполняется автоматически и недоступно для редактирования.
- [*Описание*] ([*Description*]) — локализуемое описание схемы.

Панель свойств позволяет изменить основные свойства схемы (кнопка) и задать дополнительные (кнопка). Дополнительным свойством является [*Локализуемые строки*] ([*Localizable strings*]).



Для применения заданных свойств нажмите [Применить] ([*Apply*]).

4. В дизайнере исходного кода добавьте исходный код. Название класса, объявленного в исходном коде, должно совпадать с названием схемы (свойство [Код] ([*Code*])).
5. На панели инструментов дизайнера исходного кода нажмите [Сохранить] ([*Save*]) для временного сохранения изменений в метаданных схемы.
6. На панели инструментов дизайнера исходного кода нажмите [Опубликовать] ([*Publish*]) для выполнения изменений на уровне базы данных.

Front-end отладка



Front-end отладка — отладка клиентской части приложения Creatio, которая представлена [конфигурационными схемами \(модулями\)](#), написанными на языке JavaScript.

Интегрированные инструменты отладки

Отладка исходного кода конфигурационных схем выполняется непосредственно из браузера. Для этого используются интегрированные инструменты разработчика, которые предоставляют все браузеры, поддерживаемые Creatio.

Чтобы **запустить инструменты отладки**, необходимо в браузере выполнить команду:

- Chrome: F12 или Ctrl + Shift + I .
- Firefox: F12 .
- Internet Explorer: F12 .

Все поддерживаемые браузеры предоставляют одинаковый набор инструментов отладки клиентского

кода. Наиболее распространенные и часто используемые инструменты отладки описаны ниже. Более детально возможности отладки с помощью браузерных инструментов описаны в документации:

- [Инструменты разработчика Chrome](#)
- [Инструменты разработчика Firefox](#)
- [Средства разработчика Internet Explorer](#)

Скрипты и точки останова

С помощью инструментов разработчика можно посмотреть полный список скриптов, подключенных к странице и загруженных на клиент. Открыв любой скрипт, можно установить **точку останова** (**breakpoint**) в том месте, где необходимо остановить выполнение программного кода. В остановленном коде можно просмотреть текущие значения переменных, выполнить команды и т. д.

Чтобы **установить точку останова**:

1. Откройте необходимый файл скрипта (например, выполнив его поиск по имени комбинацией клавиш `Ctrl+O` или `Ctrl+P`).
2. Перейдите к строке кода, на которой необходимо установить точку останова (например, выполнив поиск по скрипту по имени метода).
3. Установите точку останова.

Способы **добавления точки останова**:

- Щелкните по номеру строки.
- Нажмите клавишу `F9`.
- Выберите в контекстном меню [Добавить точку останова].

Кроме этого, можно использовать **условную точку останова** (**conditional breakpoint**), для которой задается условие, при котором точка останова сработает.

Остановку выполнения также можно инициировать непосредственно из кода командой `debugger`.

Пример инициирования остановки выполнения в исходном коде

```
function customFunc (args) {
  ...
  debugger; // <-- Отладчик остановится здесь.
  ...
}
```

Управление выполнением отладки

После того как выполнение кода прерывается в точке останова, выполняется проверка значений переменных стека вызовов. Затем выполняется трассировка кода с целью поиска фрагментов, в которых поведение программы отклоняется от предполагаемого.

Команды для **пошаговой навигации** по коду в отладчиках браузеров:

- приостановить/продолжить выполнение скрипта (1);
- выполнить шаг, не заходя в функцию (2);
- выполнить шаг, заходя в функцию (3);
- выполнять до выхода из текущей функции (4).

Навигационная панель в отладчике браузера Firefox



Навигационная панель в отладчике браузера Internet Explorer



Дополнительно браузер Chrome предоставляет еще две команды для управления выполнением:

- отключить все точки останова (5);
- включить/отключить автоматическую остановку при ошибке (6).

Навигационная панель в отладчике браузера Chrome



Детальную информацию о возможностях и командах навигационной панели для конкретного браузера смотрите в соответствующей документации.

Использование консоли браузера

Консоль браузера позволяет:

- выполнять команды JavaScript;
- выводить отладочную информацию;
- выводить трассировочную информацию;
- выполнять замеры и профилирование кода.

Для этого используется объект `console`.

Вызов команд JavaScript

1. Откройте консоль браузера, перейдя на вкладку [*Console*], либо откройте ее в дополнение к отладчику клавишей *Esc*.
2. Вводите в консоли команды на языке JavaScript и запускайте их на выполнение нажатием *Enter*.

Вывод отладочной информации

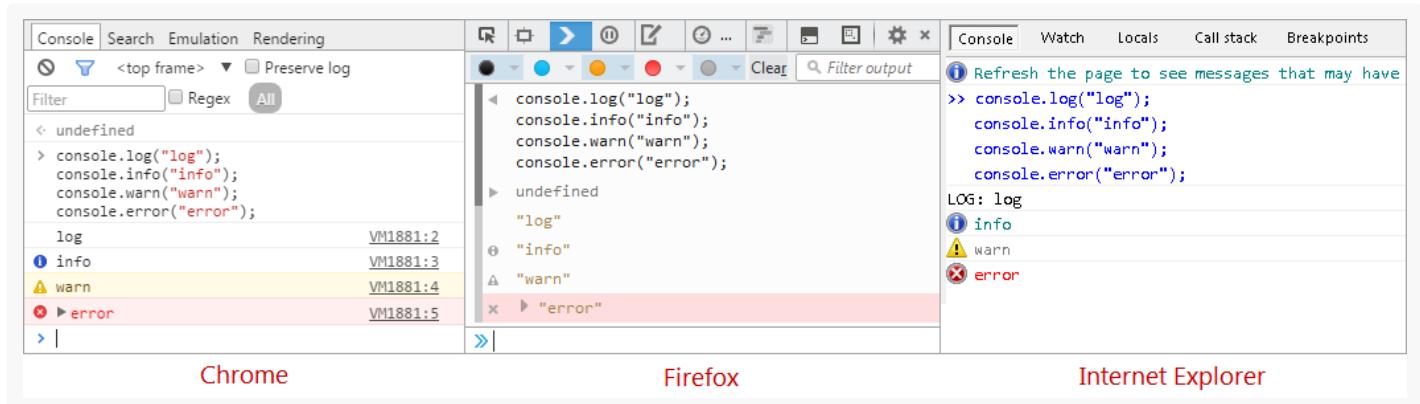
В консоли можно выводить отладочную информацию различного характера:

- информационные сообщения;
- предупреждения;
- сообщения об ошибках.

Для этого используются соответствующие методы объекта `console`.

Метод	Описание	Chrome	Firefox	Internet Explorer
<code>console.log(object [, object, ...])</code>	Выводит в консоль аргументы, разделяя их запятыми. Используется для вывода различных сообщений общего назначения.	+	+	+
<code>console.info(object [, object, ...])</code>	Аналогичен методу <code>log()</code> , но выводит сообщения в другом стиле, за счет чего позволяет акцентировать внимание на их важности.	+	+	+
<code>console.warn(object [, object, ...])</code>	Выводит в консоль сообщение предупреждающего характера.	+	+	+
<code>console.error(object [, object, ...])</code>	Выводит в консоль сообщение об ошибке.	+	+	+ (8+)

Для каждого типа выводимого сообщения в консоли применяется свой стиль.



Приведенные методы `console` поддерживают форматирование выводимых сообщений. То есть, можно использовать в тексте выводимых сообщений специальные **управляющие последовательности** (шаблоны), которые при выводе будут заменяться на соответствующие им значения — аргументы,

дополнительно передаваемые в функцию, в порядке их очередности.

Методы `console` поддерживают следующие **шаблоны форматирования**.

Шаблон	Тип данных	Пример использования
<code>%s</code>	Строка	<code>console.log("%s – один из флагманских продуктов компании %s", "Creatio sales", "Terrasoft");</code>
<code>%d</code> , <code>%i</code>	Число	<code>console.log("Платформа %s впервые была выпущена в %d году", "Creatio", 2011);</code>
<code>%f</code>	Число с плавающей точкой	<code>console.log("Число Пи равно %f", Math.PI);</code>
<code>%o</code>	DOM-элемент (не поддерживается IE)	<code>console.log("DOM-представление элемента <body/>: %o", document.getElementsByTagName("body")[0]);</code>
<code>%o</code>	Объект Java Script (не поддерживается IE, Firefox)	<code>console.log("Объект: %o", {a:1, b:2});</code>
<code>%c</code>	Стиль CSS (не поддерживается IE)	<code>console.log("%cЗеленый текст, %cКрасный текст на синем фоне, %cБольшие буквы, %cОбычный текст", "color:green;", "color:red; background:blue;", "font-size:20px;", "font:normal; color:normal; background:normal");</code>

Трассировка и проверки

С помощью методов консоли браузера можно выполнять трассировку и проверку выражений.

Метод	Описание	Chrome	Firefox	Internet Explorer
<code>console.trace()</code>	Выводит стек вызовов из точки кода, откуда был вызван метод. Стек вызовов включает в себя имена файлов, номера строк, а также счетчик вызовов метода <code>trace()</code> из одной и той же точки.	+	+	+ (11+)
<code>console.assert(expression[, object, ...])</code>	Выполняет проверку выражения, переданного в качестве параметра <code>expression</code> . Если выражение ложно, то выводит в консоль ошибку вместе со стеком вызовов (<code>console.error()</code>), иначе — ничего не выводит. Метод позволяет обеспечить соблюдение правил в коде и быть уверенным, что результаты выполнения кода соответствуют ожиданиям. С его помощью можно выполнять тестирование кода — если результат выполнения будет неудовлетворительным, будет отображено исключение.	+	+ (28+)	+

Пример использования метода `console.assert()` для тестирования результатов

```
var a = 1, b = "1";
console.assert(a === b, "A не равно B");
```

Профилирование и замеры

С помощью методов консоли браузера можно **замерять время выполнения кода**.

Метод	Описание	Chrome	Firefox	Internet Explorer
<code>console.time(label)</code>	Включает счетчик миллисекунд с меткой <code>label</code> .	+	+	+ (11+)
<code>console.timeEnd(label)</code>	Останавливает счетчик миллисекунд с меткой <code>label</code> и публикует результат в консоли.	+	+	+ (11+)

Пример использования методов `console.time()` и `console.timeEnd()`

```
var myArray = new Array();
// Включение счетчика с меткой Initialize myArray.
console.time("Initialize myArray");
myArray[0] = myArray[1] = 1;
for (i = 2; i<10; i++)
{
    myArray[i] = myArray[i-1] + myArray[i-2];
}
// Выключение счетчика с меткой Initialize myArray.
console.timeEnd("Initialize myArray");
```

С помощью методов консоли можно выполнить **профилирование кода** и вывести стек профилирования, содержащий подробную информацию о том, сколько времени и на какие операции было потрачено браузером.

Метод	Описание	Chrome	Firefox	Internet Explorer
<code>console.profile(label)</code>	Запускает профайлер Java Script, затем показывает результаты под меткой <code>label</code> .	+	+ (при открытой панели DevTools)	+ (10+)
<code>console.profileEnd(label)</code>	Останавливает профайлер Java Script.	+	+ (при открытой панели DevTools)	+ (10+)

Результаты профилирования отображаются в браузерах:

- Chrome — на вкладке [*Profiles*];

- Firefox — на вкладке [*Performance*];
- Internet Explorer — на вкладке [*Profiler*].

Режим клиентской отладки *isDebug*

Режим клиентской отладки `isDebug` необходим для получения подробной информации об ошибках приложения Creatio и их отслеживании в коде.

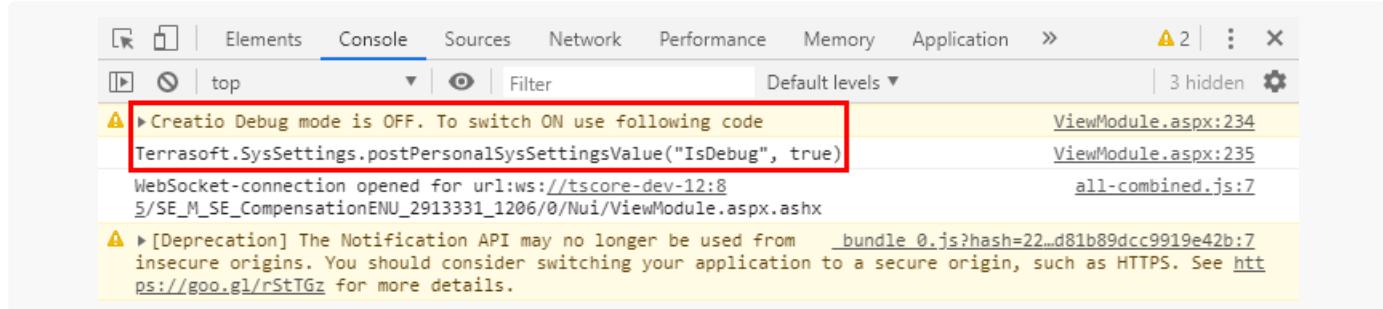
В обычном режиме работы в браузере выполняется [минификация кода](#). Это означает что сборка клиентских скриптов осуществляется в файл `all-combined.js`. Файл собирается в момент создания сборки и содержит всю функциональность. Включение режима `isDebug` отключает сборку и минификацию *.js-файлов и позволяет получить перечень клиентских скриптов в виде отдельных файлов.

Настройка режима отладки *isDebug*

1. Определите **текущий статус** режима клиентской отладки.

Откройте консоль браузера по клавише `F12` или с помощью комбинации клавиш `Ctrl+Shift+I`.

Кроме статуса режима клиентской отладки, в консоли будет отображен **код для его активации или деактивации**.



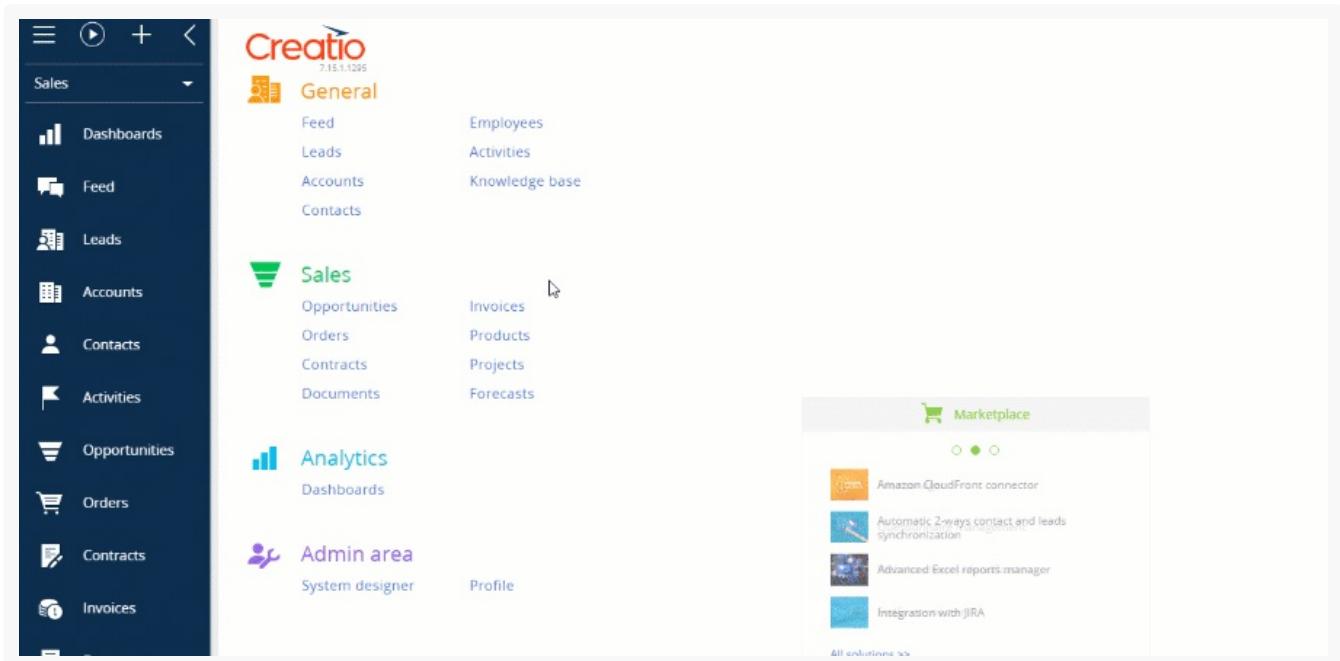
2. Включите режим клиентской отладки

Это можно сделать следующими **способами**:

- Выполнить в консоли браузера код:

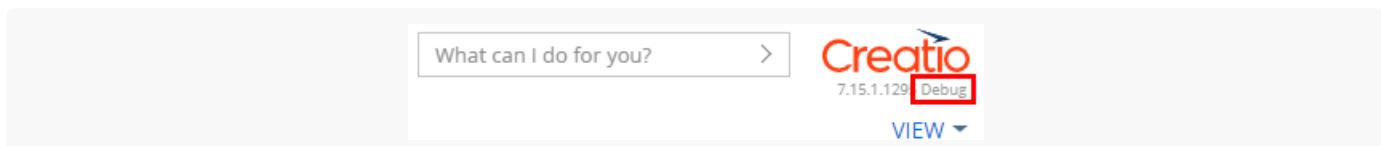
```
Terrasoft.SysSettings.postPersonalSysSettingsValue("IsDebug", true)
```

- Изменить системную настройку [*Режим отладки*] (код [*isDebug*]).



3. Чтобы применить изменения, обновите страницу или нажмите **F5**.

После включения режима клиентской отладки возле номера версии сайта отобразится **индикатор Debug**.



На заметку. Включение режима клиентской отладки влияет на производительность сайта, например, увеличивается время открытия страниц.

На рисунках ниже показаны примеры отображения в консоли информации об ошибке при выключенном и включенном режиме `isDebugEnabled`.

Отображение информации об ошибке (`isDebugEnabled` выключен)

✖ ▼Uncaught TypeError: Cannot read property 'value' of undefined [InvoicePageV2.js?has...c0f99002aff011:1741](#)
at i.setCardLockoutStatus ([InvoicePageV2.js?has...c0f99002aff011:1741](#))
at i.onEntityInitialized ([InvoicePageV2.js?has...c0f99002aff011:1752](#))
at Object.callback ([all-combined.js:6](#))
at i.<anonymous> ([BasePageV2.js?hash=6...c0f99002aff011:1108](#))
at i.e ([all-combined.js:7](#))
at Object.callback ([all-combined.js:6](#))
at i.<anonymous> ([all-combined.js:7](#))
at Object.callback ([all-combined.js:6](#))
at i.<anonymous> ([all-combined.js:7](#))
at Object.callback ([all-combined.js:6](#))
setCardLockoutStatus @ [InvoicePageV2.js?has...c0f99002aff011:1741](#)
onEntityInitialized @ [InvoicePageV2.js?has...c0f99002aff011:1752](#)
callback @ [all-combined.js:6](#)
(anonymous) @ [BasePageV2.js?hash=6...c0f99002aff011:1108](#)
e @ [all-combined.js:7](#)
callback @ [all-combined.js:6](#)
(anonymous) @ [all-combined.js:7](#)
callback @ [all-combined.js:6](#)
(anonymous) @ [all-combined.js:7](#)
callback @ [all-combined.js:6](#)
_parseGetEntityResponse @ [all-combined.js:7](#)
(anonymous) @ [all-combined.js:7](#)
callback @ [all-combined.js:7](#)
e.callback @ [all-combined.js:7](#)
callback @ [all-combined.js:6](#)
onComplete @ [all-combined.js:6](#)
onStateChange @ [all-combined.js:6](#)
(anonymous) @ [all-combined.js:6](#)
XMLHttpRequest.send (async)
request @ [all-combined.js:6](#)
request @ [all-combined.js:7](#)
executeRequest @ [all-combined.js:7](#)
callParent @ [all-combined.js:6](#)
executeRequest @ [all-combined.js:7](#)
executeQuery @ [all-combined.js:7](#)
getEntity @ [all-combined.js:7](#)
load @ [all-combined.js:7](#)
loadEntity @ [all-combined.js:7](#)
(anonymous) @ [BasePageV2.js?hash=6...c0f99002aff011:1104](#)
e @ [all-combined.js:7](#)
callback @ [all-combined.js:6](#)
XMLHttpRequest.send (async)
request @ [all-combined.js:6](#)
request @ [all-combined.js:7](#)
executeRequest @ [all-combined.js:7](#)

Отображение информации об ошибке (`isDebug` включен)

```

✖ ▼Uncaught TypeError: Cannot read property 'value' of undefined InvoicePageV2.js?has...c0f99002aff011:1741
    at constructor.setCardLockoutStatus (InvoicePageV2.js?has...c0f99002aff011:1741)
    at constructor.onEntityInitialized (InvoicePageV2.js?has...c0f99002aff011:1752)
    at Object.callback (extjs-base-debug.js:11584)
    at constructor.<anonymous> (BasePageV2.js?hash=6...c0f99002aff011:1108)
    at constructor.nextFn (commonutils.js?hash=...7c0f99002aff011:130)
    at Object.callback (extjs-base-debug.js:11584)
    at constructor.<anonymous> (entity-base-view-mod...7c0f99002aff011:977)
    at Object.callback (extjs-base-debug.js:11584)
    at constructor.<anonymous> (entity-data-model.js...7c0f99002aff011:177)
    at Object.callback (extjs-base-debug.js:11584)
setCardLockoutStatus      @ InvoicePageV2.js?has...c0f99002aff011:1741
onEntityInitialized        @ InvoicePageV2.js?has...c0f99002aff011:1752
callback                  @ extjs-base-debug.js:11584
(anonymous)                @ BasePageV2.js?hash=6...c0f99002aff011:1108
nextFn                    @ commonutils.js?hash=...7c0f99002aff011:130
callback                  @ extjs-base-debug.js:11584
(anonymous)                @ entity-base-view-mod...7c0f99002aff011:977
callback                  @ extjs-base-debug.js:11584
(anonymous)                @ entity-data-model.js...7c0f99002aff011:177
callback                  @ extjs-base-debug.js:11584
_parseGetEntityResponse   @ entity-schema-query...7c0f99002aff011:487
(anonymous)                @ entity-schema-query...7c0f99002aff011:558
callback                  @ base-service-provide...7c0f99002aff011:126
config.callback            @ ajax-provider.js?has...7c0f99002aff011:157
callback                  @ extjs-base-debug.js:11584
onComplete                @ extjs-base-debug.js:46413
onStateChange              @ extjs-base-debug.js:46349
(anonymous)                @ extjs-base-debug.js:3278
XMLHttpRequest.send (async)
request                   @ extjs-base-debug.js:45742
request                   @ ajax-provider.js?has...7c0f99002aff011:177
executeRequest             @ base-service-provide...7c0f99002aff011:289
callParent                @ extjs-base-debug.js:6836
executeRequest             @ service-provider.js?...97c0f99002aff011:73
executeQuery               @ data-provider.js?has...7c0f99002aff011:138
getEntity                 @ entity-schema-query...7c0f99002aff011:556
load                      @ entity-data-model.js...7c0f99002aff011:174
loadEntity                @ entity-base-view-mod...7c0f99002aff011:971
(anonymous)                @ BasePageV2.js?hash=6...c0f99002aff011:1104
nextFn                    @ commonutils.js?hash=...7c0f99002aff011:130
callback                  @ extjs-base-debug.js:11584
XMLHttpRequest.send (async)
request                   @ extjs-base-debug.js:45742
request                   @ ajax-provider.js?has...7c0f99002aff011:177
executeRequest             @ base-service-provide...7c0f99002aff011:289

```

Контроль версий в Subversion



Легкий

Creatio позволяет использовать любые системы контроля версий. В этой статье мы рассмотрим применение наиболее популярной из них — Subversion (SVN).

Subversion (SVN) — это бесплатная система управления версиями с открытым исходным кодом.

Основа SVN — хранилище, которое содержит данные в форме иерархии файлов и каталогов — т. н. дерева файлов.

Возможные **действия** пользователей с хранилищем SVN:

- **Чтение данных** других пользователей системы, к которым они предоставили доступ:
 - Чтение файлов других пользователей системы и дерева каталогов.
 - Чтение дерева каталогов.
 - Просмотр предыдущих версий файлов и дерева каталогов.
- **Изменение данных:**
 - Создание новых каталогов и файлов.
 - Переименование каталогов и файлов.
 - Изменение содержимого файлов.
 - Удаление каталогов и файлов.
- **Запись данных** для предоставления доступа другим пользователям системы.

В одном из нижеприведенных случаев система контроля версий SVN **рекомендуется к использованию** для:

- Приложений Creatio на платформе **.NET Framework**.
 - Приложений, в которых разработка ведется, в основном, low-code инструментами.
 - Cloud-приложений.
- Для on-site приложений рекомендуется использовать Git. Работа с системой контроля версий Git описана в статье [Контроль версий в Git](#).

Важно. Систему контроля версий SVN разрешено использовать для переноса изменений только между [средами разработки](#). Запрещено использовать SVN на [предпромышленной](#) и [промышленной](#) среде, поскольку это может привести к неработоспособности или ухудшению производительности приложения.

Неработоспособность приложения при использовании SVN на предпромышленной и промышленной среде может быть вызвана ошибками, которые получены при переносе изменений или при сохранении конфигурационных элементов (например, если в качестве текущего указан пакет, который привязан к SVN). Восстановить работоспособность приложения можно из резервной копии баз данных. Это может привести к потере изменений, которые были выполнены в приложении с момента создания последней резервной копии.

Также при использовании SVN на предпромышленной и промышленной среде изменения могут поставляться без предварительного тестирования на среде разработки, что категорически запрещено.

Инструкция по настройке и использованию SVN содержится в [документации SVN](#).

Основные понятия

Хранилище — центральная база данных, обычно расположенная на файловом сервере и содержащая файлы со своей историей версий. Хранилище может быть доступно посредством различных сетевых протоколов или с локального диска.

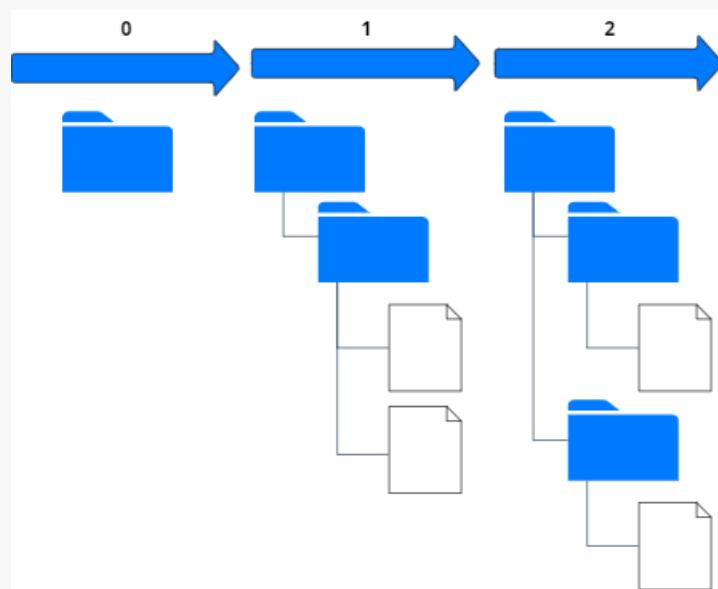
Рабочая копия — каталог на локальном компьютере, с которым работает пользователь. Рабочая копия содержит копию файлов, которые были в хранилище до того, как их начал изменять пользователь. Таким образом можно узнать, какие конкретно изменения были выполнены.

Важно. Изменения можно просмотреть только для текстовых файлов. Для бинарных файлов можно узнать только сам факт изменения.

Ревизия — состояние дерева файловой системы. Ревизия подразумевает весь набор изменений файлов и каталогов как единое изменение.

Фиксация изменений дерева файловой системы — это атомарная операция, которая позволяет зафиксировать ревизию.

Ревизии в хранилище можно представить в виде **серии деревьев файловой системы** — массива номеров ревизий, начинающегося с 0 и растущего слева направо. Под каждым номером расположено дерево файловой системы — "снимок" состояния хранилища после фиксации.



На заметку. В отличие от других систем управления версиями, номера ревизий в SVN относятся к деревьям целиком, а не к отдельным файлам.

Общая **последовательность работы** с файлами в рабочей копии:

1. Получение из хранилища последней версии файлов.
2. Локальная работа с файлами.
3. Фиксация файлов в хранилище.

Модели версионирования

При работе с SVN может возникнуть ситуация, когда разработчики работают над одной и той же

функциональностью, реализованной в одном и том же файле. Если первый разработчик сохранит свои изменения первым, а второй — несколькими секундами позже, то изменения, внесенные первым разработчиком, могут быть затерты. И хотя эти изменения содержатся в хранилище, правки, внесенные первым разработчиком, будут отсутствовать в последней ревизии файла. Чтобы избежать подобной проблемы, используются **модели версионирования**.

Типы моделей версионирования:

- Модель "Блокирование-Изменение-Разблокирование".
- Модель "Копирование-Изменение-Слияние".

Модель "Блокирование-Изменение-Разблокирование"

Хранилище разрешает вносить изменения в файл только одному пользователю за раз. До того как первый пользователь сможет внести изменения в файл, он должен сначала заблокировать этот файл. Второй пользователь не сможет зафиксировать изменения до тех пор, пока первый не внесет изменения в хранилище и не снимет блокировку.

Особенности модели:

- **Блокирование может вызвать проблемы администрирования.**

Первый разработчик может забыть снять блокировку, что приведет к потере времени вторым разработчиком.

- **Блокирование может вызвать излишнюю пошаговость.**

Если разработчики работают с непересекающимися частями файла, то можно было бы работать с файлом одновременно, предполагая корректное слияние изменений.

- **Блокирование может вызвать ложное чувство безопасности.**

Разработчики могут одновременно работать с разными файлами, содержащими зависящую друг от друга функциональность. Каждый разработчик заблокировал свой файл и считает, что начинает безопасную изолированную задачу. Это препятствует заблаговременному обсуждению изменений, которые могут быть несовместимы друг с другом, что приведет к неработоспособности разрабатываемого решения.

Эту модель необходимо использовать, если выполняется работа над файлами, не поддающимися слиянию. Например, если хранилище содержит изображения, и пользователи изменяют их в одно и то же время, то нет возможности выполнить слияние этих изменения.

Модель "Копирование-Изменение-Слияние"

Клиентское приложение каждого пользователя считывает из хранилища проект и создает персональную **рабочую копию** — локальную копию файлов и каталогов хранилища. После этого пользователи работают, одновременно изменения свои личные копии. В результате работ, личные копии сливаются в новую, финальную версию. Обычно SVN выполняет слияние автоматически, но выполнение слияния необходимо подтвердить.

Если при одновременной работе двух пользователей изменения пересекаются, то возникает **конфликт**.

Чтобы **разрешить конфликт** необходимо:

1. Обсудить изменения, которые вызвали конфликт, с пользователем, выполняющим предыдущую

фиксацию файлов.

2. Вручную выбрать, какие изменения из набора конфликтующих изменений необходимо зафиксировать.
3. Зафиксировать в хранилище объединенный файл.

Решающим фактором при использовании этой модели является взаимодействие между пользователями.

Работа с файлами в SVN

В служебный каталог `.svn` рабочей копии для каждого файла SVN записывает свойства.

Свойства файла:

- Рабочая ревизия файла — номер ревизии, на которой основан файл в рабочей копии.
- Дата и время последнего обновления локальной копии файла из хранилища.

Назначение свойств — определить состояние файла рабочей копии.

Состояния файла рабочей копии:

- **Не изменился и не устарел.**

В хранилище не фиксировались изменения файла со времени его рабочей ревизии. При попытке его обновить или зафиксировать не будет выполнено никаких действий.

- **Изменен локально и не устарел.**

В хранилище не фиксировались изменения этого файла со времени его базовой ревизии. Обновление выполняться не будет. Фиксация в хранилище выполнится успешно.

- **Не изменился и устарел.**

Файл в рабочей папке не изменился, но был изменен в хранилище. Файл необходимо обновить для соответствия текущей публичной ревизии. Фиксация выполняться не будет. Обновление выполнится успешно.

- **Изменен локально и устарел.**

Файл был изменен как в рабочей папке, так и в хранилище. Попытка фиксации потерпит неудачу.

Файл необходимо сначала обновить, попытавшись объединить опубликованные другим разработчиком изменения с локальными. Если SVN не сможет выполнить объединение самостоятельно, решение конфликта будет выполнять пользователь.

Рабочая копия, используемая приложением Creatio

В Creatio по умолчанию включен режим работы с SVN. При выключенном режиме [разработки в файловой системе](#) приложение Creatio использует собственную рабочую копию каждого пользовательского пакета, для которого подключена версионность. Эти рабочие копии размещаются в каталоге, указанном в элементе `defPackagesWorkingCopyPath` конфигурационного файла `ConnectionString.config`.

Если включен режим разработки в файловой системе, то рабочая копия может быть [создана вручную](#) в каталоге `[Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\НазваниеПакета`.

Для настройки работы с SVN необходимо изменить настройку `sourceControlAuthPath` конфигурационного файла `connectionStrings.config`. Эта настройка содержит путь к каталогу на диске, который хранит

аутентификационные данные (логин и зашифрованный пароль) подключения пользователя к SVN-репозиториоу. Аутентификационные данные сохраняются SVN-клиентом, встроенным в Creatio. В качестве значения настройки `sourceControlAuthPath` рекомендуется установить путь на фиксированный каталог, поскольку временный каталог, установленный по умолчанию, может быть очищен операционной системой.

Клиентское приложение для работы с SVN

Для работы с SVN в файловой системе рекомендуется использовать клиентское приложение [TortoiseSVN](#) версии не ниже 1.9. Оно реализовано как расширение оболочки Windows и встраивается в контекстное меню проводника Windows. Использование TortoiseSVN описано в [документации по TortoiseSVN](#).

Создать пакет в режиме разработки в файловой системе



Сложный

При включенном [режиме разработки в файловой системе](#) механизм интеграции с SVN отключен. Встроенными средствами можно только [установить](#) либо [обновить пакеты](#) из хранилища SVN. Поэтому рекомендуется создавать пакет с помощью встроенных средств, а привязывать его к хранилищу — с помощью сторонних утилит, например, [TortoiseSVN](#).

Прежде чем привязывать пакет к хранилищу SVN при включенном режиме разработки в файловой системе, необходимо удостовериться, что приложение настроено для доступа к нужному хранилищу SVN.

Пример. Создать пакет в режиме разработки в файловой системе.

1. Создать пакет

Чтобы **создать пользовательский пакет**:

- Перейдите в дизайнер системы по кнопке
- В блоке [Конфигурирование разработчиком] ([Admin area]) перейдите по ссылке [Управление конфигурацией] ([Advanced settings]).
- В области работы с пакетами нажмите кнопку
- Заполните **свойства пакета**:

The screenshot shows the 'Package' creation dialog. It has the following fields:

- Name***: sdkTestPackage
- Description**: (empty)
- Version control system repository**: SDKPackages
- Version ***: 7.18.0

At the bottom are three buttons: **CANCEL**, **CREATE AND ADD DEPENDENCIES**, and a green **SAVE** button.

- [Название] ([*Name*]) — "sdkTestPackage".
- [Хранилище системы контроля версий] ([*Version control system repository*]) — "SDKPackages".

Если заполнить поле [Хранилище системы контроля версий] ([*Version control system repository*]), то пакет будет привязан к хранилищу. При включенном режиме разработки в файловой системе каталог с названием, соответствующим пакету, необходимо вручную добавить в хранилище.

Если не заполнять поле [Хранилище системы контроля версий] ([*Version control system repository*]), то пакет не будет привязан к хранилищу. Вести версионную разработку этого пакета можно будет только подключив его вручную из файловой системы.

- [Версия] ([*Version*]) — "7.18.0".

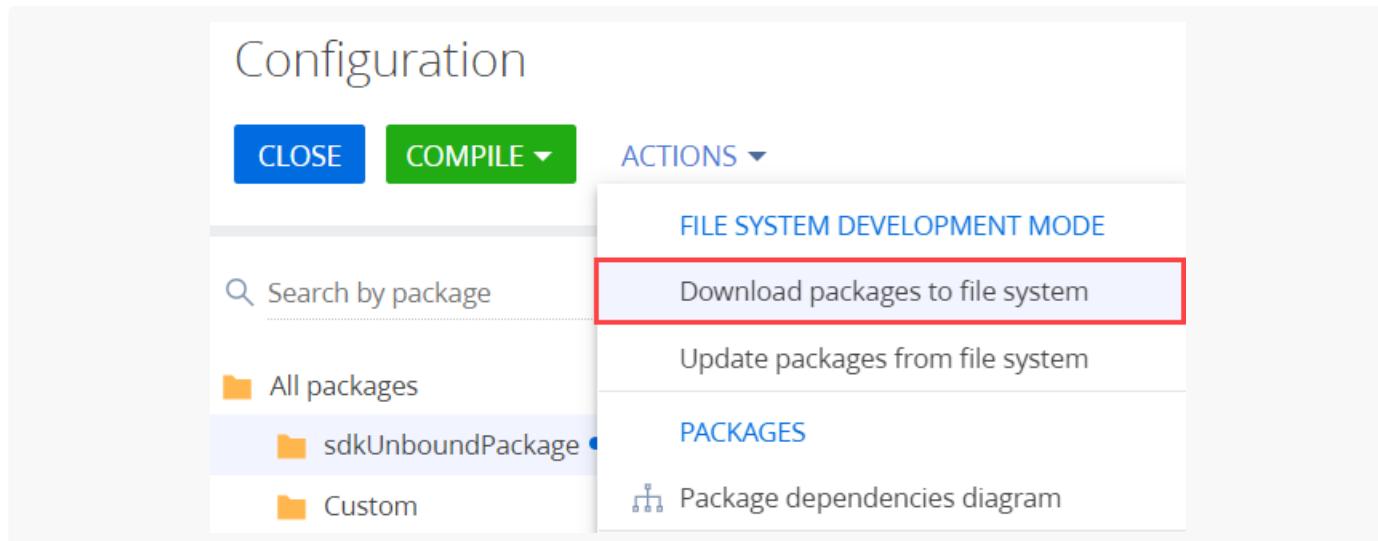
Важно. Название репозитория в свойствах пакета указывает только на то, что каталог для пакета будет создан сторонними средствами в этом репозитории. Это позволит в дальнейшем выполнять обновление пакета из раздела [Конфигурация] ([*Configuration*]).

5. Нажмите кнопку [Создать и добавить зависимости] ([*Create and add dependencies*]) и установите зависимости пакета.

2. Выгрузить пакет в файловую систему

Чтобы **выгрузить пакет** в файловую систему:

1. Настройте Creatio для работы в файловой системе. Настройка описана в статье [Внешние IDE](#).
2. На панели инструментов в группе действий [Разработка в файловой системе] ([*File system development mode*]) выберите [Выгрузить все пакеты в файловую систему] ([*Download packages to file system*]).



В результате все пакеты будут выгружены по пути `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в каталог с соответствующим названием пакета.

3. Создать каталоги для пакета в хранилище SVN

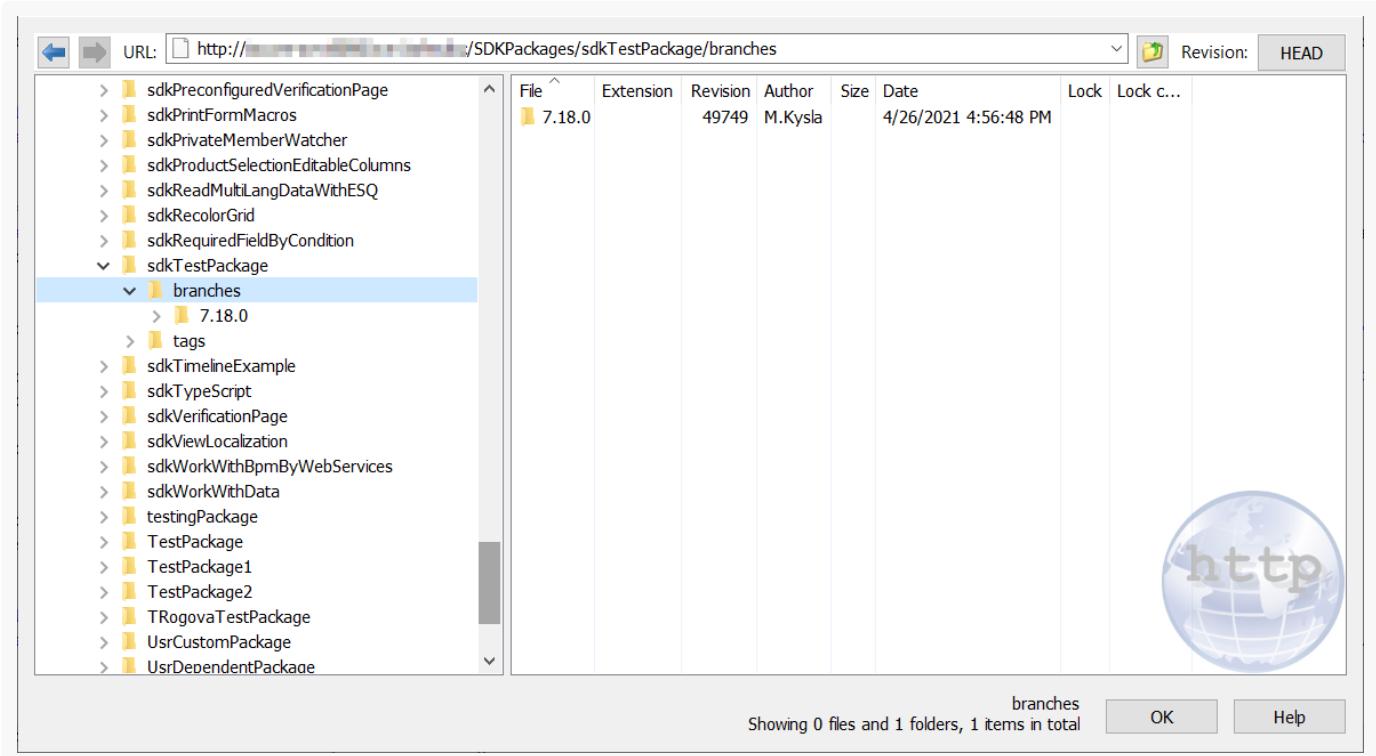
Чтобы создать каталоги для пакета в хранилище SVN необходимо использовать клиентское приложение для работы с SVN (например, [TortoiseSvn](#)).

Чтобы **создать каталоги** для пакета в хранилище SVN:

- Перейдите в репозиторий, указанный в свойствах пакета.
- В репозитории создайте каталог, название которого совпадает с названием созданного в приложении пакета. В нашем примере это `sdkTestPackage`.

File	Extension	Revision	Author	Size	Date
AddDetailMobile		39429	R.Simuta		4/16/2018 12:02:38 P
mobileSinglePage		43939	T.Rogova		6/25/2019 3:08:28 AM
NewPackage		39220	R.Simuta		4/2/2018 1:15:33 PM
NewPackage2		39216	R.Simuta		4/2/2018 1:01:30 PM
NewPackage3		39219	R.Simuta		4/2/2018 1:08:11 PM
sdkActionsDashboardAdding		39731	T.Rogova		5/9/2018 2:31:28 AM
sdkActionsDashboardNewChannel		46032	T.Rogova		2/2/2020 4:41:49 AM
sdkAddActionToEditPage		39137	T.Rogova		3/29/2018 4:10:52 AM
sdkAddAlignableContainer		40716	T.Rogova		7/8/2018 3:28:26 AM
sdkAddButtonToEditPage		39305	T.Rogova		4/6/2018 2:54:30 AM
sdkAddButtonToEditPage		39371	T.Rogova		4/12/2018 1:25:08 AM
sdkAddButtonToNewPage		42975	T.Rogova		2/26/2019 2:14:58 PM
sdkAddButtonToSection		39450	T.Rogova		4/17/2018 12:18:08 P
sdkAddColorButton		36586	R.Simuta		9/11/2017 10:37:45 A
sdkAddCustomNotification		36661	R.Simuta		9/19/2017 12:43:21 P
sdkAddCustomPredictionModel		35246	R.Simuta		6/14/2017 12:25:13 P
sdkAddCustomSectionToMobileApp		43547	T.Rogova		4/25/2019 2:36:26 AM
sdkAddCustomWidget		36486	R.Simuta		8/30/2017 8:42:56 AM
sdkAddCustomWidgetTypeMobile		39120	T.Rogova		3/28/2018 3:38:24 AM
sdkAddFieldToPage		36028	R.Simuta		7/14/2017 10:43:06 A
sdkAddFileLinkDetailToSectionMobile		38790	T.Rogova		3/13/2018 3:19:28 AM
sdkAddFiltrationRuleToPage		38733	T.Rogova		3/7/2018 3:19:54 AM
sdkAddFixedFilters					

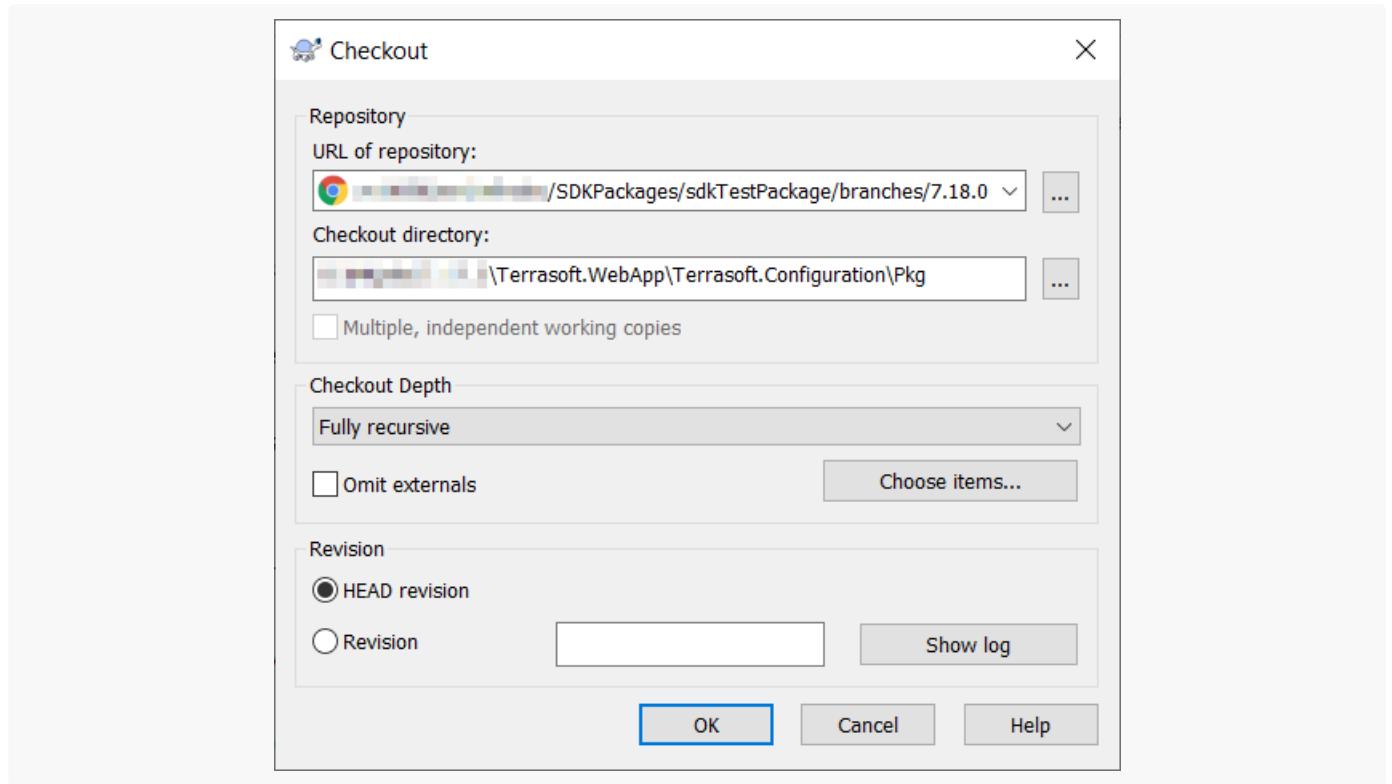
3. В созданном каталоге создайте подкаталоги `branches` и `tags`.
4. В каталоге `branches` создайте каталог, название которого совпадает с номером версии пакета — `7.18.0`.



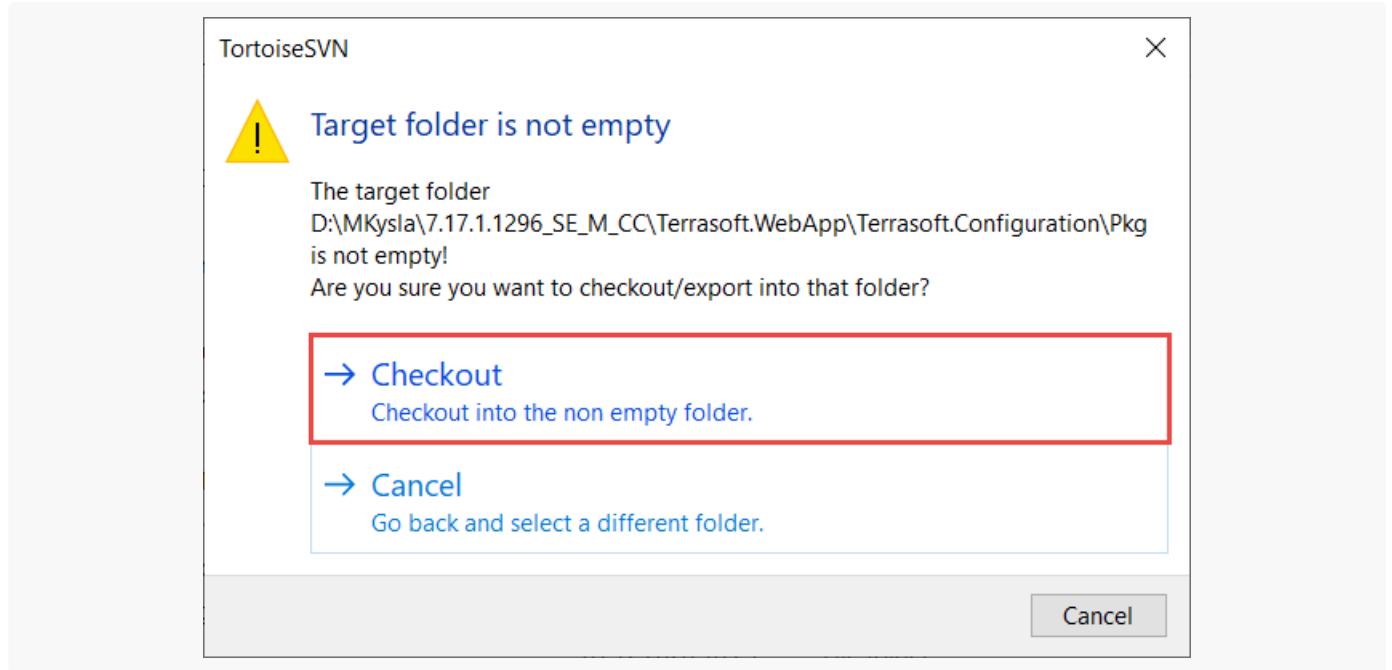
4. Создать рабочую копию версионной ветки пакета

Чтобы **создать рабочую копию версионной ветки пакета**:

1. Выгрузите (команда `SVN Checkout...`) созданный на предыдущем шаге каталог из хранилища в каталог пакета в файловой системе. В нашем примере это каталог `7.18.0`.



2. Подтвердите выгрузку в существующий каталог.



В результате каталог пакета в файловой системе

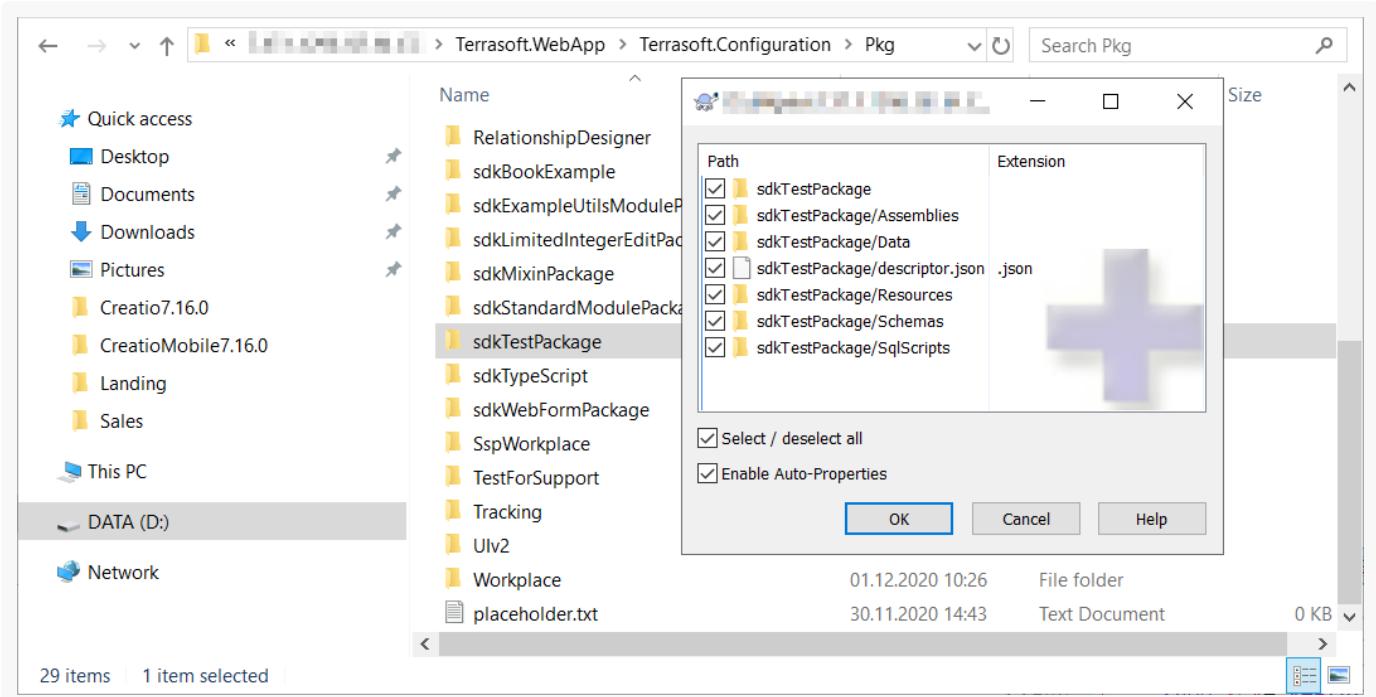
`...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\sdkTestPackage` будет связан с веткой версии `7.18.0` пакета в хранилище.

5. Зафиксировать пакет в хранилище

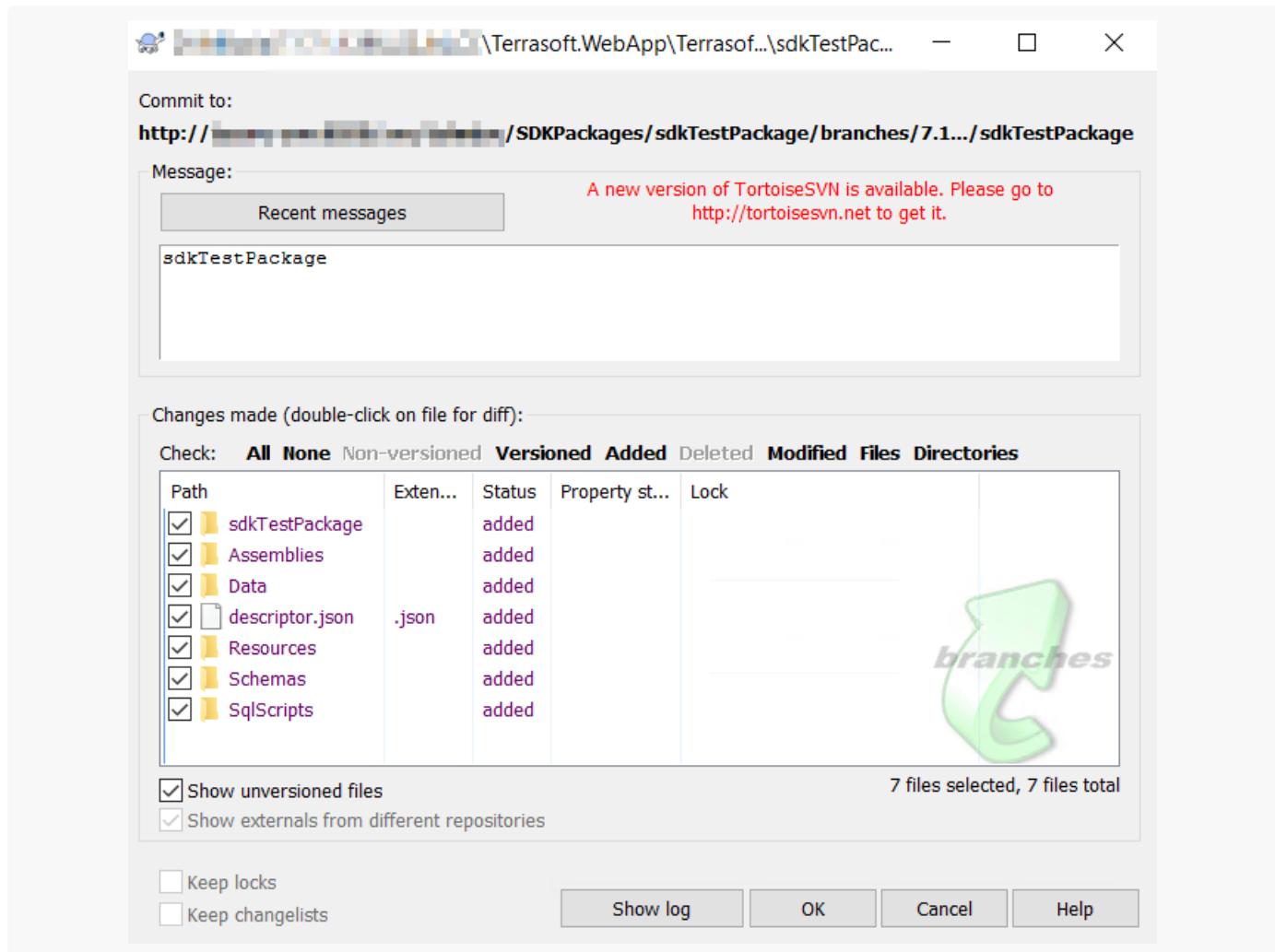
Чтобы **зафиксировать пакет в хранилище**:

1. Добавьте (команда `SVN Commit`) в хранилище содержимое каталога

`...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\sdkTestPackage .`



2. Выполните фиксацию каталога в хранилище.



Установить пакет из SVN в режиме разработки в файловой системе

 Средний

Пример. В приложение Creatio в режиме разработки в файловой системе установить пакет из хранилища SVN.

Адрес пакета в хранилище SVN — .../SDKPackages/sdkCreateDetailWithEditableGrid/branches/7.18.0 .

В пакете содержится функциональность [детали с редактируемым реестром](#).

Creatio предоставляет возможность установить существующий пакет из SVN в режиме разработки в файловой системе автоматически и вручную.

Последовательность действий при **ручной установке пакета:**

1. Установить пакет в файловую систему.
2. Установить пакет в приложение.

3. Выполнить генерацию исходных кодов.
4. Скомпилировать изменения.
5. Обновить структуру базы данных.
6. Установить SQL-сценарии и привязанные данные (опционально).

Последовательность действий при **автоматической установке пакета**:

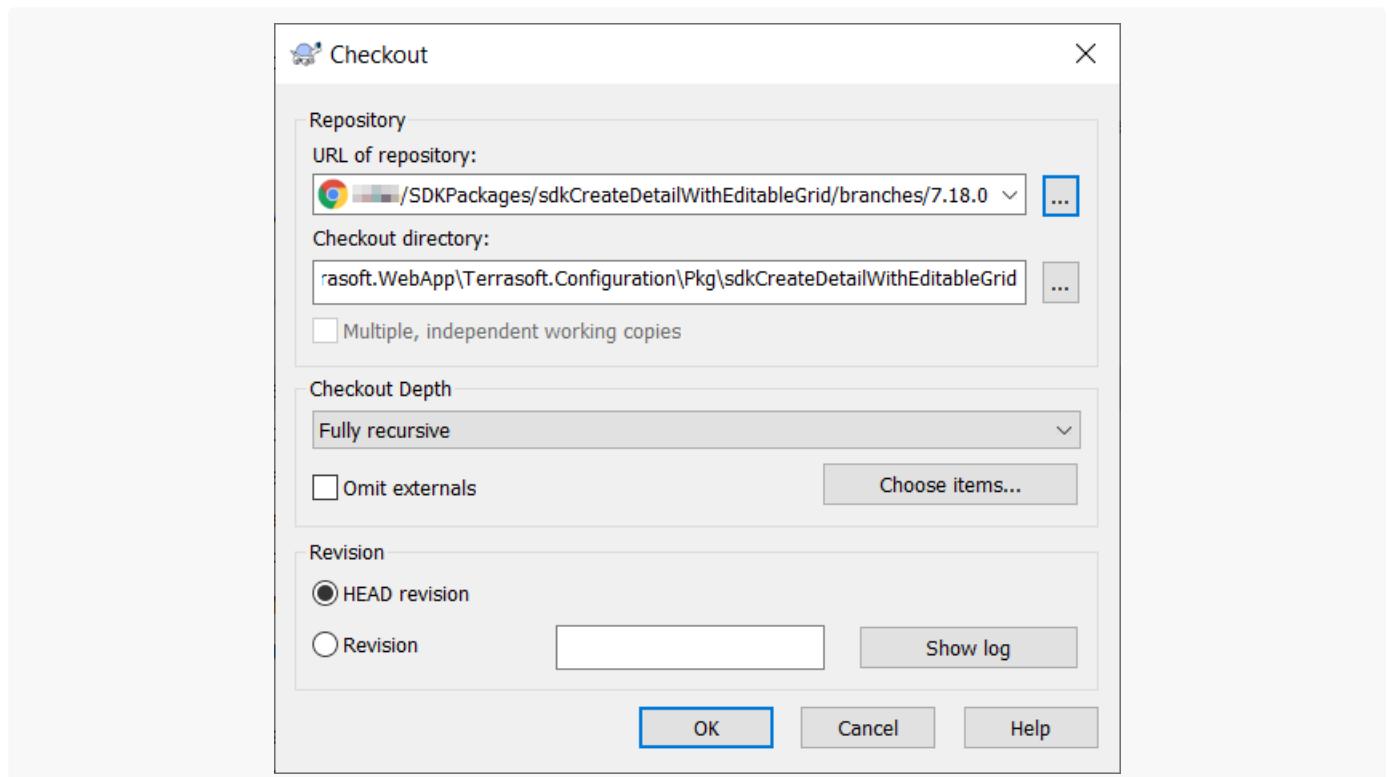
1. Включить автоматическое применение изменений.
2. Установить пакет в файловую систему.
3. Установить пакет в приложение.

Вручную установить пакет из SVN в режиме разработки в файловой системе

1. Установить пакет в файловую систему

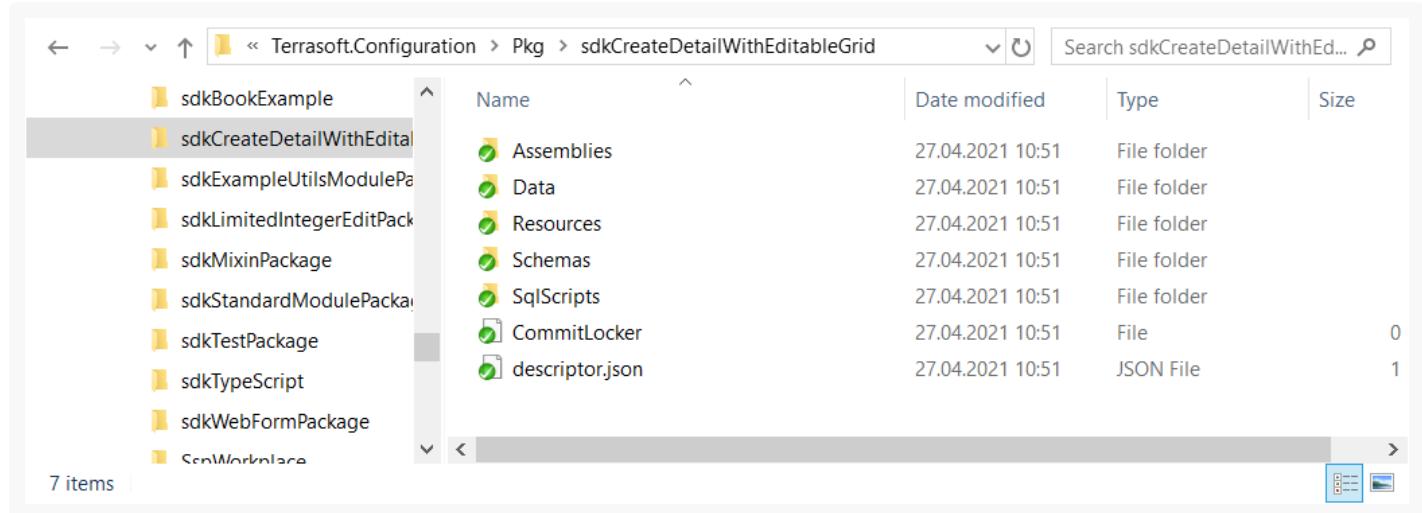
Чтобы **установить пакет в файловую систему**:

1. В каталоге приложения `...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg` создайте каталог, название которого совпадает с названием пакета.
2. Выгрузите (команда `SVN Checkout...`) созданный на предыдущем шаге каталог из хранилища в каталог пакета в файловой системе.
3. Укажите адрес хранилища, по которому размещено содержимое пакета, и каталог для выгрузки содержимого пакета.



Название каталога для выгрузки содержимого пакета должно совпадать с названием пакета.

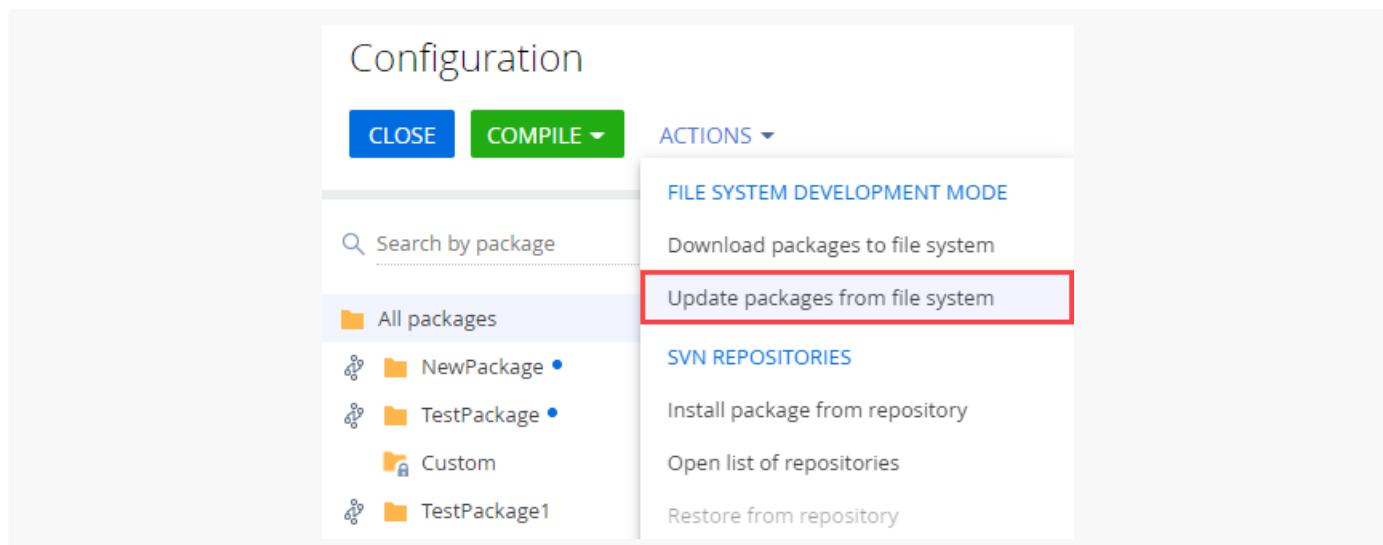
После выгрузки в каталоге `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` будет создана рабочая копия пакета.



2. Установить пакет в приложение

Чтобы **установить пакет в приложение**:

- Перейдите в дизайнер системы по кнопке
- В блоке [Конфигурирование разработчиком] ([Admin area]) перейдите по ссылке [Управление конфигурацией] ([Advanced settings]).
- На панели инструментов в группе действий [Разработка в файловой системе] ([File system development mode]) выберите [Обновить пакеты из файловой системы] ([Update packages from file system]).



В результате пакет будет добавлен в приложение.

Информационное окно статуса загрузки пакета в приложение

Changes

Name	Type	Status
✓ sdkCreateDetailWithEditableGrid	Package	Added
OrderPageV2	Schema	Added
UsrCourierService	Schema	Added
UsrCourierServiceDetail	Schema	Added
OrderPageV2	Schema Resource	Added

CLOSE

Пакет в области работы с пакетами

Отсутствие названия репозитория в названии пакета свидетельствует о том, что все изменения могут быть зафиксированы в репозитории только из файловой системы.

3. Выполнить генерацию исходных кодов

Чтобы **выполнить генерацию исходных кодов**:

- На панели инструментов в группе действий [*Исходный код*] ([*Source code*]) выберите [*Сгенерировать для требующих генерации*] ([*Generate where it is needed*]).

Configuration

CLOSE COMPILE ▾ ACTIONS ▾

Search by package

- All packages
 - Custom
 - TestForSupport
 - sdkBookExample
 - sdkCreateDetailWithEc
 - sdkExampleUtilsModule**
 - sdkLimitedIntegerEditF
 - sdkMixinPackage
 - sdkStandardModulePa
- sdkTestPackage
- sdkWebFormPackage
- ActionsDashboard
- AnalyticsDashboard
- BPMS_ENU
- Base
- BaseProcessDesigner
- BaseScoring

FILE SYSTEM DEVELOPMENT MODE

- Download packages to file system
- Update packages from file system

PACKAGES

- Package dependencies diagram

SVN REPOSITORIES

- Install package from repository
- Open list of repositories
- Restore from repository

ACTUALIZE ITEMS

- Update DB structure where it is needed
- Install SQL scripts where it is needed
- Install data where it is needed

SOURCE CODE

- Generate for modified schemas
- Generate where it is needed
- Generate for all schemas

4. Скомпилировать изменения

Чтобы **скомпилировать изменения**, на панели инструментов нажмите [Компилировать] ([Compile]).

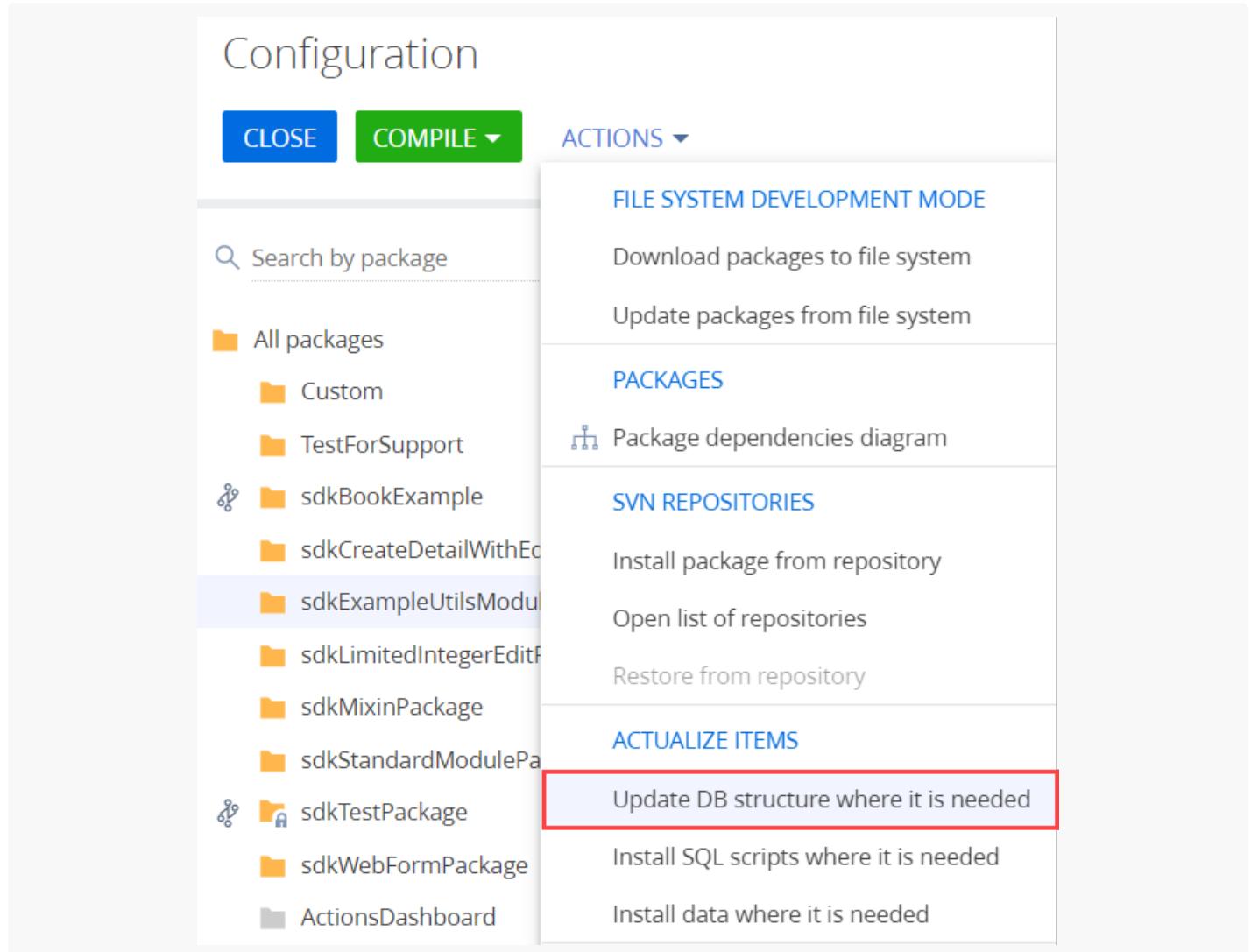
Configuration

CLOSE COMPILE ▾ ACTIONS ▾

Необходимость обновления структуры базы данных, установки SQL-скриптов и привязанных данных отображается в колонке [Статус] ([Status]) рабочей области раздела [Конфигурация] ([Configuration]).

5. Обновить структуру базы данных

Чтобы **обновить структуру базы данных**, на панели инструментов в группе действий [Актуализировать элементы] ([Actualize items]) выберите [Обновить структуру БД для требующих обновления] ([Update DB structure where it is needed]).



6. Установить SQL-сценарии и привязанные данные (опционально)

Если пакет содержит привязанные SQL-сценарии или данные, то необходимо выполнить соответствующие действия для их выполнения или установки.

После установки в приложении станет доступна реализованная в пакете функциональность. В нашем примере это функциональность детали с редактируемым реестром.

The screenshot shows the 'ORD-1 (sample)' order details screen in the Creatio application. The 'DELIVERY' tab is active. On the right side, there is a vertical toolbar with icons for user profile, settings, help, phone, email, messaging, notifications, and file sharing. In the main area, under 'Delivery type', it says 'Courier'. Under 'Payment type', it says 'Non-cash payment'. Below these, there are sections for 'Delivery address' and 'Recipient information'. The 'Recipient information' section is expanded, showing a list item 'Courier Service'. A red box highlights the 'Account' field, which contains the value 'Accom (sample)'. There is also a search icon and a button labeled 'Enter a value'.

Для отображения примененных изменений может понадобиться обновление страницы с очисткой кэша.

Автоматически установить пакет из SVN в режиме разработки в файловой системе

1. Включить автоматическое применение изменений

Чтобы **включить автоматическое применение изменений**, в файле `..\Terrasoft.WebApp\Web.config` установите значение `true` для ключей элемента `<appSettings>`:

- `AutoUpdateOnCommit` — ключ отвечает за автоматическое обновление пакетов из хранилища SVN перед их заливкой. Если для этого ключа установлено значение `false`, то перед заливкой в хранилище SVN приложение предупредит пользователя о необходимости обновления, если схемы пакета были изменены.
- `AutoUpdateDBStructure` — ключ отвечает за автоматическое обновление структуры базы данных.
- `AutoInstallSqlScript` — ключ отвечает за автоматическую установку SQL-сценариев.
- `AutoInstallPackageData` — ключ отвечает за установку привязанных данных.

`..\Terrasoft.WebApp\Web.config`

`<appSettings>`

```

...
<add key="AutoUpdateOnCommit" value="true" />
<add key="AutoUpdateDBStructure" value="true" />
<add key="AutoInstallSqlScript" value="true" />
<add key="AutoInstallPackageData" value="true" />
</appSettings>

```

Затем выполните шаги 1—2 последовательность действий пункта [Вручную установить пакет из SVN в режиме разработки в файловой системе](#).

Настроить взаимодействие с хранилищем SVN (опционально)

Creatio позволяет настроить взаимодействие с хранилищем SVN как из раздела [Конфигурация] ([Configuration]), так и из файловой системы.

Чтобы **настроить взаимодействие с хранилищем SVN**:

- Перейдите в дизайнер системы по кнопке .
- В блоке [Конфигурирование разработчиком] ([Admin area]) перейдите по ссылке [Управление конфигурацией] ([Advanced settings]).
- Установите пакет из хранилища SVN. Подробная инструкция содержится в статье [Контроль версий в Creatio IDE](#).
- Выгрузите пакет в файловую систему. Подробная инструкция содержится в статье [Настроить Creatio для работы в файловой системе](#).

Затем выполните шаги 3—6 последовательность действий пункта [Вручную установить пакет из SVN в режиме разработки в файловой системе](#).

Привязать к SVN не связанный с хранилищем пакет



Может возникнуть задача, когда пользователю необходимо привязать к хранилищу пакет с файловой системы. Выполнить эту привязку можно только в on-site приложении Creatio. После привязки пакета к SVN в файловой системе его можно [установить](#), используя встроенные средства Creatio.

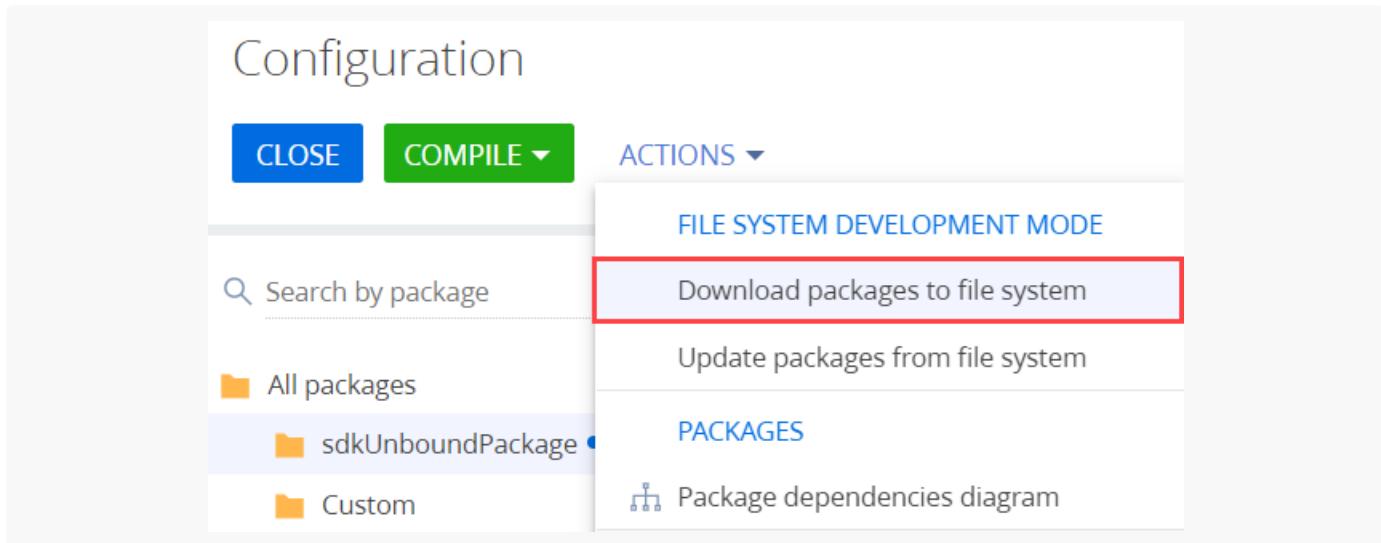
Пример. Привязать к хранилищу не связанный с хранилищем SVN существующий пользовательский пакет `sdkUnboundPackage`.

Адрес пакета в хранилище SVN — `..../SDKPackages/sdkUnboundPackage`.

1. Выгрузить пакет в файловую систему

Чтобы **выгрузить пакет в файловую систему**:

1. Настройте Creatio для работы в файловой системе. Настройка описана в статье [Внешние IDE](#).
2. На панели инструментов в группе действий [*Разработка в файловой системе*] ([*File system development mode*]) выберите [*Выгрузить все пакеты в файловую систему*] ([*Download packages to file system*]).



В результате все пакеты будут выгружены по пути `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в каталог с соответствующим названием пакета.

2. Создать каталоги для пакета в хранилище SVN

Чтобы создать каталоги для пакета в хранилище SVN необходимо использовать клиентское приложение для работы с SVN (например, [TortoiseSvn](#)).

Чтобы **создать каталоги** для пакета в хранилище SVN:

1. Перейдите в репозиторий, указанный в свойствах пакета.
2. В репозитории создайте каталог, название которого совпадает с названием созданного в приложении пакета. В нашем примере это `sdkUnboundPackage` .

Screenshot of a web-based SVN client showing the contents of the `/SDKPackages` directory. The left pane displays a tree view of folders and files, while the right pane shows a detailed list of items with columns for File, Extension, Revision, Author, Size, and Date.

File	Extension	Revision	Author	Size	Date
AddDetailMobile		39429	R.Simuta		4/16/2018 12:02:45
mobileSinglePage		43939	T.Rogova		6/25/2019 3:08:20
NewPackage		39220	R.Simuta		4/2/2018 1:15:33
NewPackage2		39216	R.Simuta		4/2/2018 1:01:30
NewPackage3		39219	R.Simuta		4/2/2018 1:08:11
sdkActionsDashboardAdding		39731	T.Rogova		5/9/2018 2:31:28
sdkActionsDashboardNewChannel		46032	T.Rogova		2/2/2020 4:41:49
sdkAddActionToEditPage		39137	T.Rogova		3/29/2018 4:10:51
sdkAddAlignableContainer		40716	T.Rogova		7/8/2018 3:28:26
sdkAddButtonToEditPage		39305	T.Rogova		4/6/2018 2:54:30
sdkAddButtonToNewPage		39371	T.Rogova		4/12/2018 1:25:08
sdkAddButtonToSection		42975	T.Rogova		2/26/2019 2:14:58
sdkAddColorButton		39450	T.Rogova		4/17/2018 12:18:01
sdkAddCustomNotification		36586	R.Simuta		9/11/2017 10:37:41
sdkAddCustomPredictionModel		36661	R.Simuta		9/19/2017 12:43:11
sdkAddCustomSectionToMobileApp		35246	R.Simuta		6/14/2017 12:25:11
sdkAddCustomWidget		43547	T.Rogova		4/25/2019 2:36:20
sdkAddCustomWidgetTypeMobile		36486	R.Simuta		8/30/2017 8:42:51
sdkAddFieldToPage		39120	T.Rogova		3/28/2018 3:38:26
sdkAddFileLinkDetailToSectionMobile		36028	R.Simuta		7/14/2017 10:43:01
sdkAddFiltrationRuleToPage		38790	T.Rogova		3/13/2018 3:19:28
sdkAddFixedFilters		38733	T.Rogova		3/7/2018 3:19:54
sdkAddFloatingIcon					

SDKPackages
Showing 0 files and 113 folders, 113 items in total

OK Help

3. В созданном каталоге создайте подкаталоги `branches` и `tags`.

4. В каталоге `branches` создайте каталог, название которого совпадает с номером версии пакета —

`7.18.0`.

Screenshot of a web-based SVN client showing the contents of the `/SDKPackages/sdkUnboundPackage/branches` directory. The left pane displays a tree view of folders and files, while the right pane shows a detailed list of items with columns for File, Extension, Revision, Author, Size, Date, Lock, and Lock c... (partially visible).

File	Extension	Revision	Author	Size	Date	Lock	Lock c...
7.18.0		49756	M.Kysla		4/27/2021 3:39:03 PM		

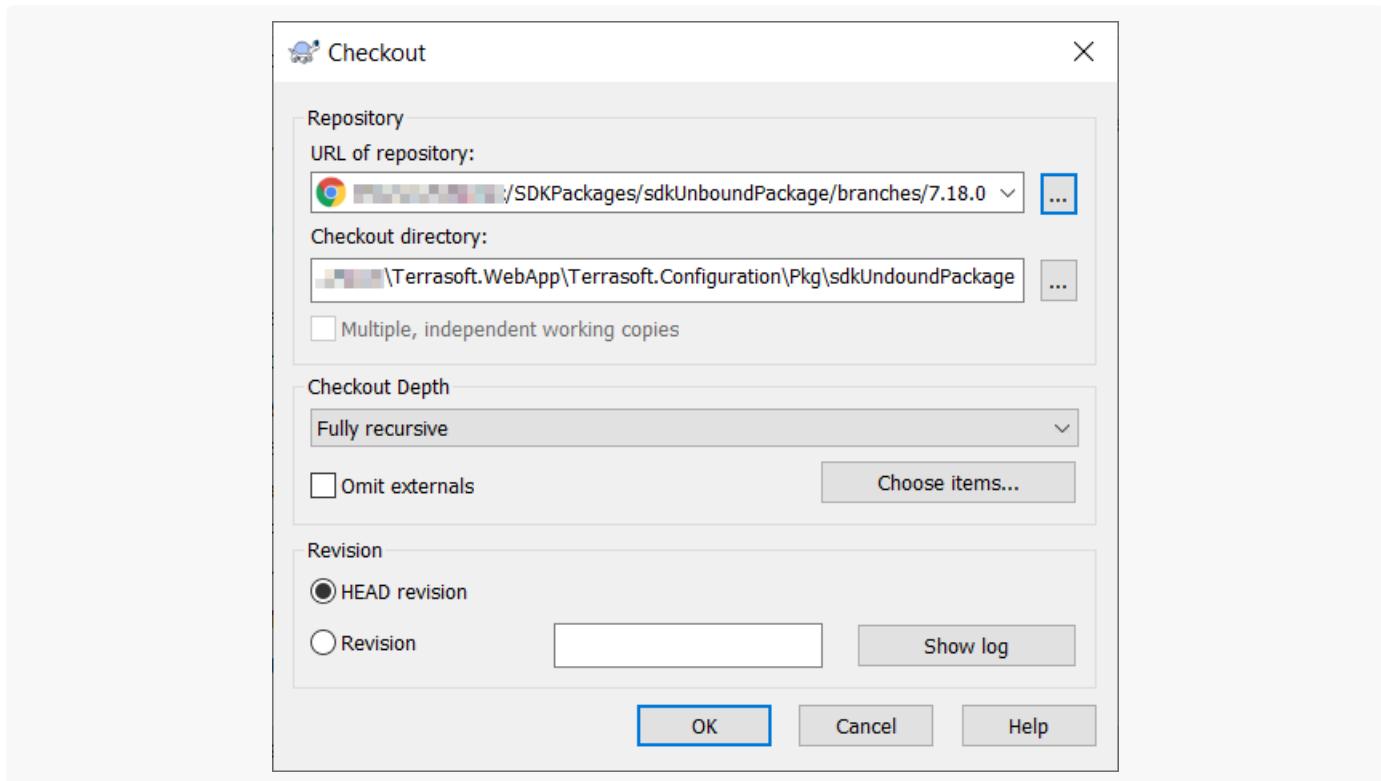
branches
Showing 0 files and 1 folders, 1 items in total

OK Help

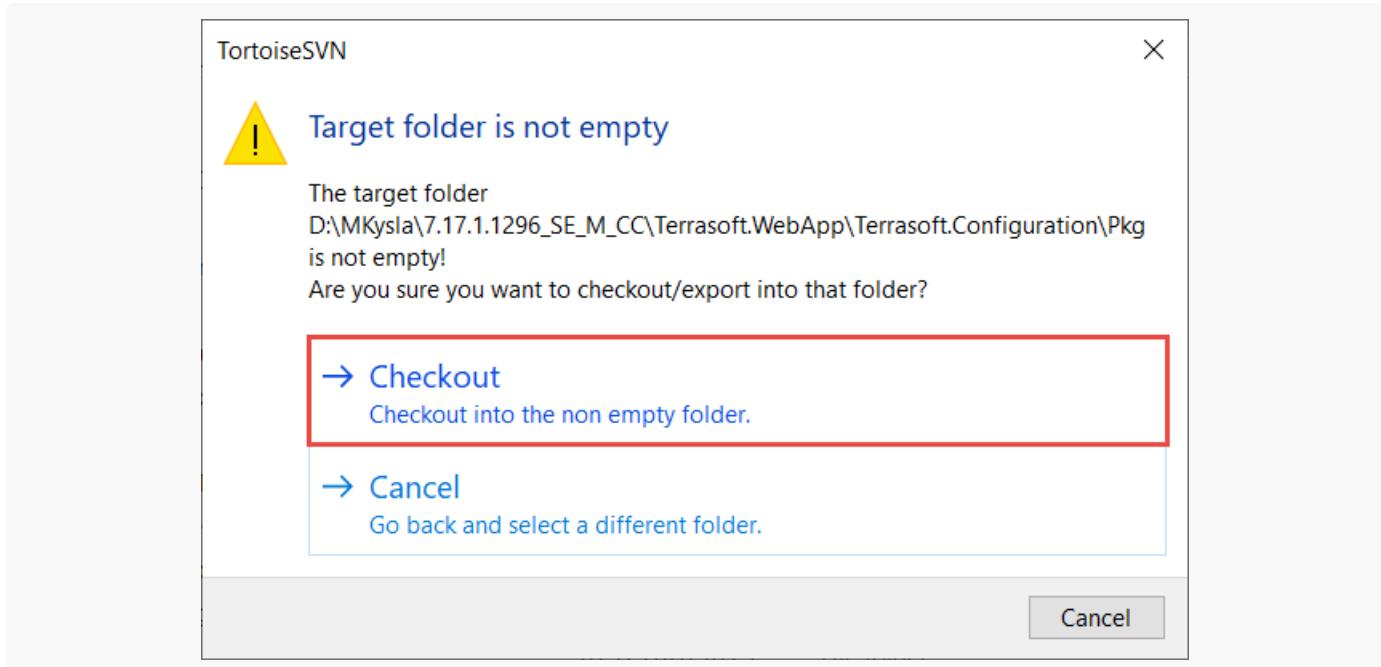
3. Создать рабочую копию версионной ветки пакета

Чтобы **создать рабочую копию версионной ветки пакета**:

- Выгрузите (команда `SVN Checkout...`) созданный на предыдущем шаге каталог из хранилища в каталог пакета в файловой системе. В нашем примере это каталог `7.18.0`.



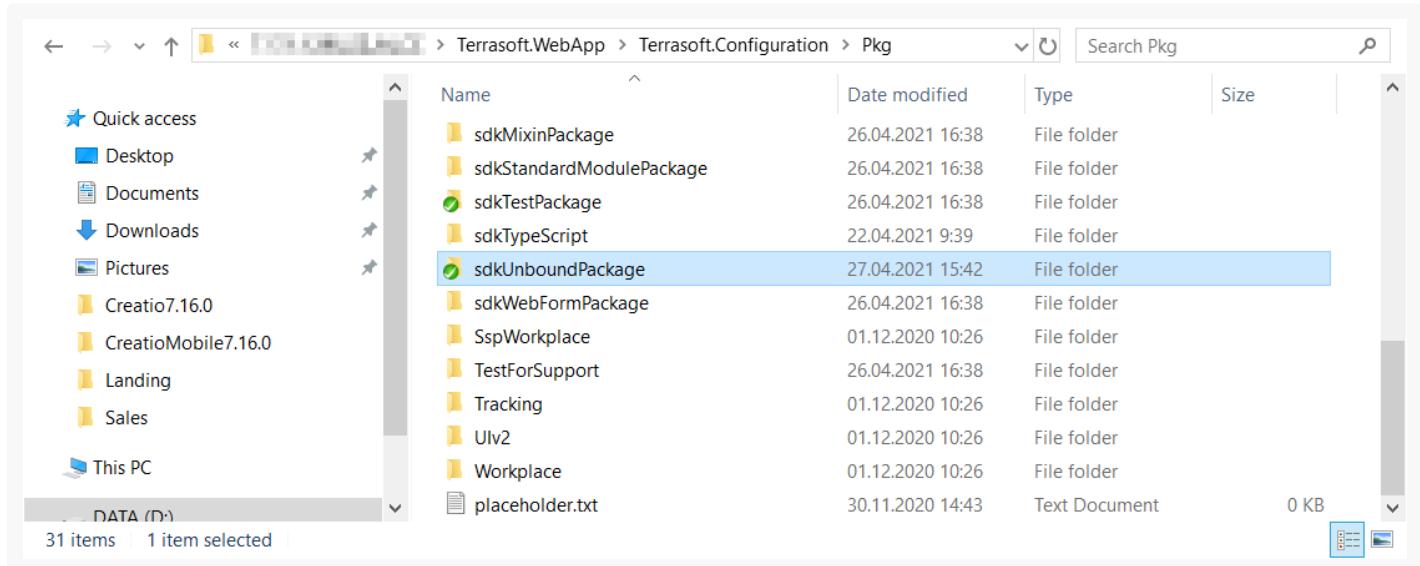
- Подтвердите выгрузку в существующий каталог.



В результате каталог пакета в файловой системе

`...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\sdkUnboundPackage` будет связан с веткой версии `7.18.0`

пакета в хранилище.

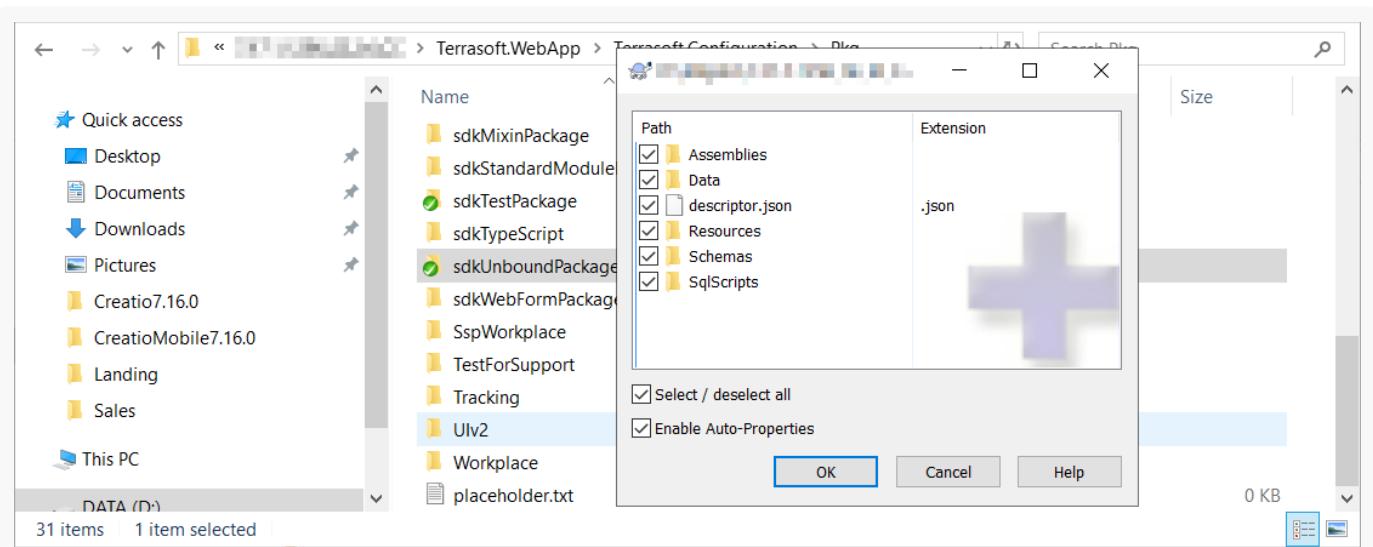


4. Зафиксировать в хранилище каталог пакета

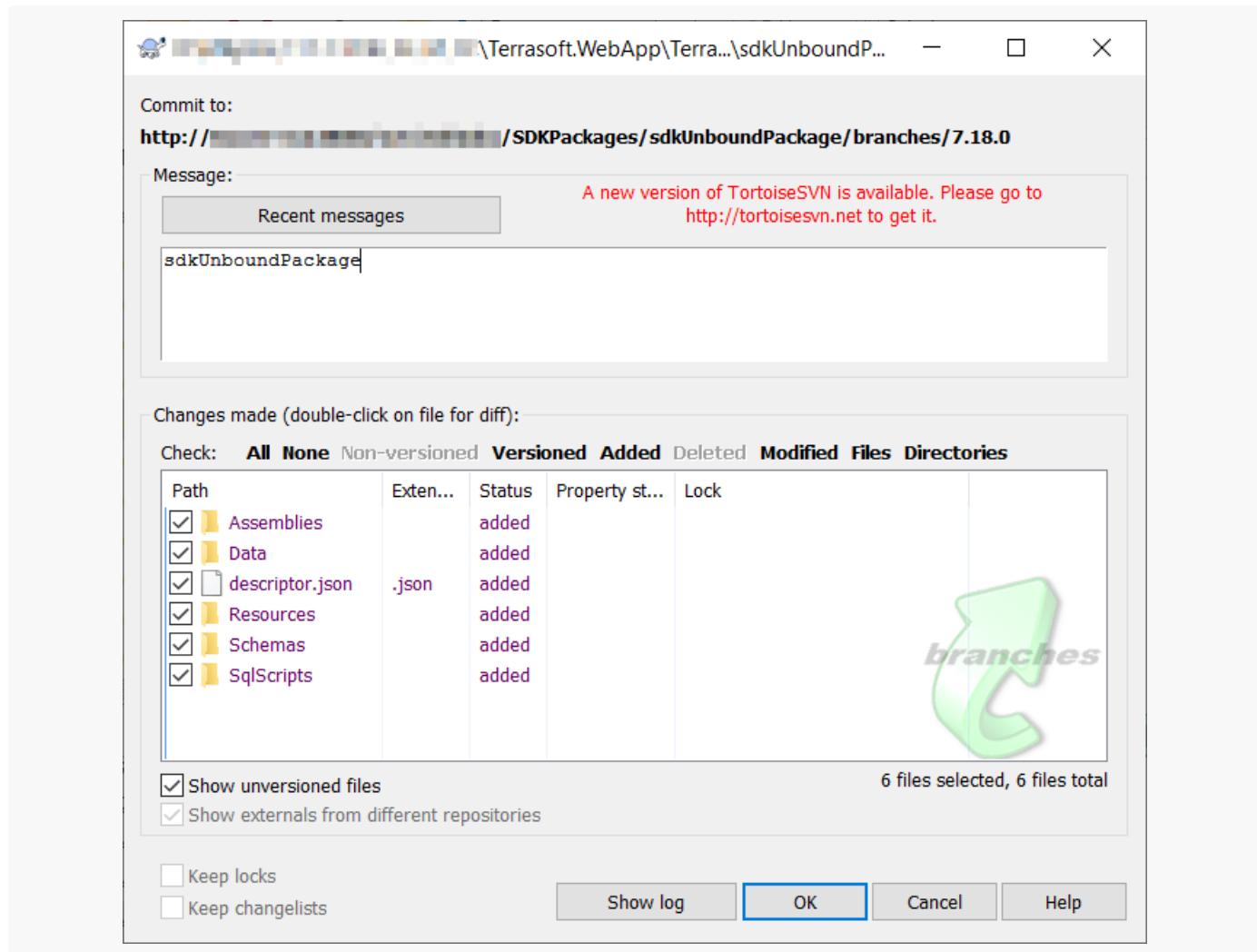
Чтобы **зафиксировать в хранилище каталог пакета**:

- Добавьте в хранилище содержимое каталога

```
...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\sdkUnboundPackage .
```



- Выполните фиксацию каталога в хранилище.



Привязать к SVN не связанный с хранилищем пакет через запрос к базе данных

 Средний

Пример. Привязать к хранилищу не связанный с хранилищем SVN существующий пользовательский пакет `sdkUnboundPackage` через прямой запрос к базе данных.

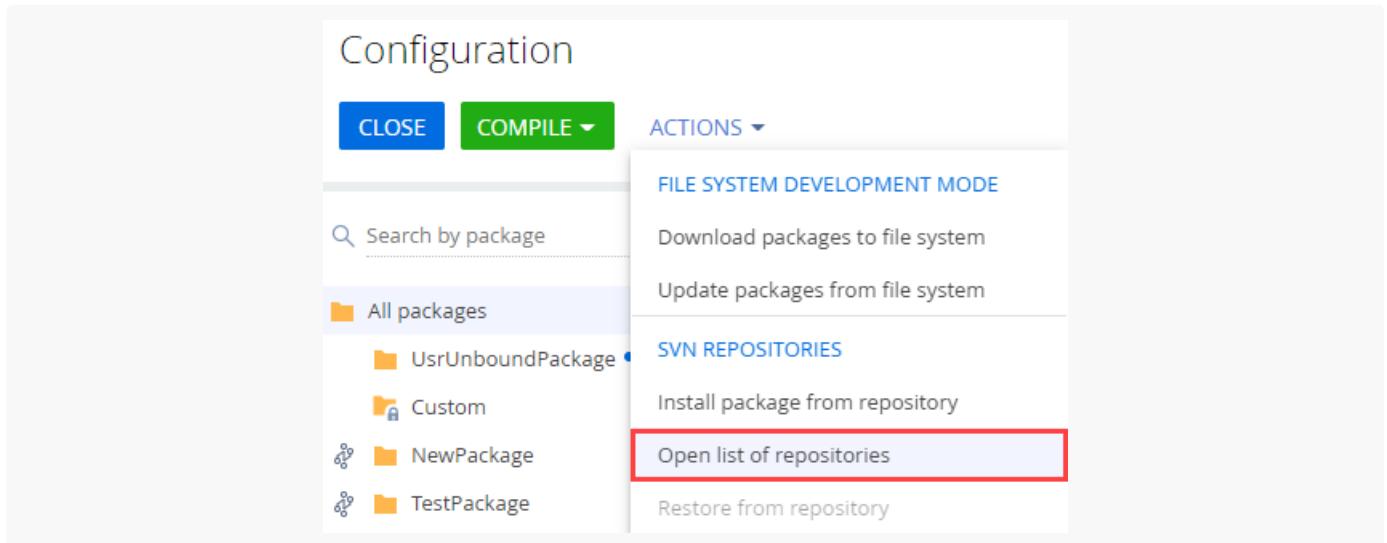
Адрес пакета в хранилище SVN — `.../SDKPackages/sdkUnboundPackage`.

1. Подключите к приложению хранилище SVN

Если хранилище SVN подключено к приложению, то перейдите к следующему шагу.

Чтобы **подключить к приложению хранилище SVN**:

- Перейдите в дизайнер системы по кнопке .
- В блоке [Конфигурирование разработчиком] ([Admin area]) перейдите по ссылке [Управление конфигурацией] ([Advanced settings]).
- На панели инструментов в группе действий [Хранилища SVN] ([SVN repositories]) выберите [Открыть список хранилищ] ([Open list of repositories]).



- Нажмите [Добавить] ([Add]) и добавьте репозиторий. В нашем примере это `.../SDKPackages`.

The screenshot shows the 'SVN repository list' dialog. It has a 'CLOSE' button, an 'ADD' button, and a search bar. On the left, there's a sidebar with 'Name' and 'SVNRepositories'. The main form has fields for 'Name *' (filled with 'SDKPackages'), 'SVN storage address *' (filled with 'http://.../SDKPackages'), 'Login', 'Password', and a checked 'Active' checkbox. At the bottom are 'CANCEL', 'AUTHENTICATE', and another 'ADD' button.

- Нажмите [Аутентификация] ([Authenticate]) и выполните аутентификацию.

2. Привязать хранилище SVN к пакету

Чтобы **привязать хранилище SVN к пакету**:

1. Перейдите в базу данных приложения.
2. В базе данных выполните SQL-запрос.

Пример SQL-запроса

```
UPDATE SysPackage
SET
    [SysRepositoryId] =
(
    select top 1 Id from SysRepository
    where Name = 'SDKPackages'-- Название хранилища.
)
where [Name]='sdkUnboundPackage'-- Название пользовательского пакета.
```

Этот запрос позволяет:

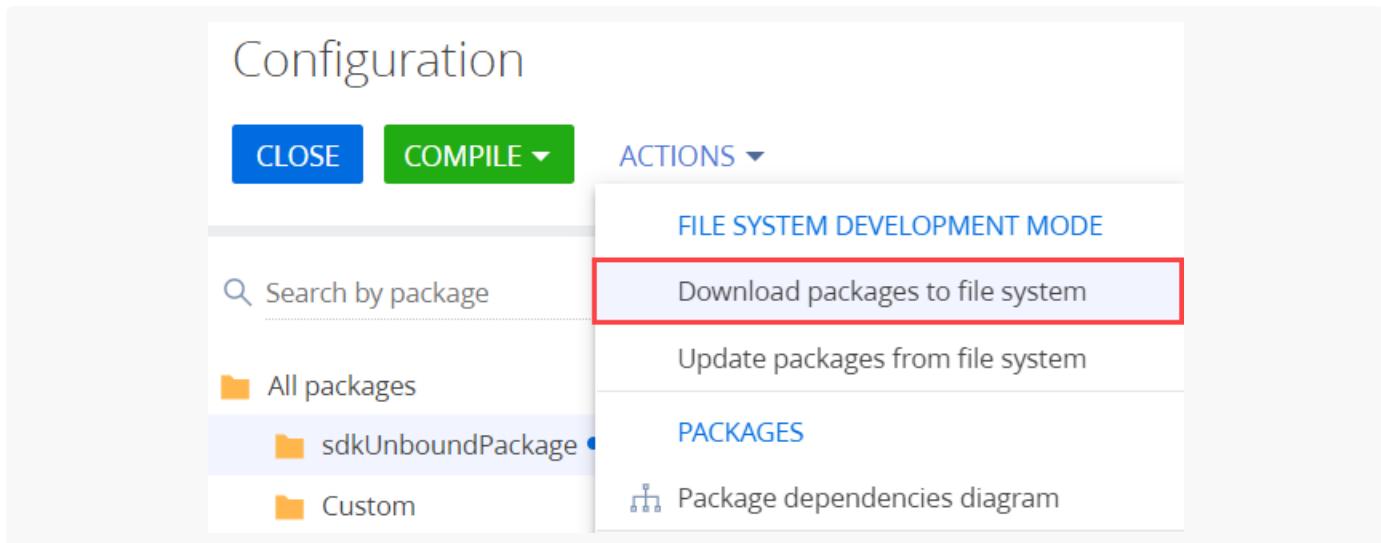
- Из таблицы `[SysRepository]` считать идентификатор записи, содержащей адрес хранилища SVN.
- Добавить полученный идентификатор в колонку `[SysRepositoryId]` записи, которая содержит имя не привязанного к хранилищу SVN пакета, таблицы `[SysPackage]`.

Чтобы изменения применились в приложении, необходимо выйти из приложения и зайти повторно.

3. Выгрузить пакет в файловую систему

Чтобы **выгрузить пакет** в файловую систему:

1. Настройте Creatio для работы в файловой системе. Настройка описана в статье [Внешние IDE](#).
2. На панели инструментов в группе действий [*Разработка в файловой системе*] ([*File system development mode*]) выберите [*Выгрузить все пакеты в файловую систему*] ([*Download packages to file system*]).



В результате все пакеты будут выгружены по пути `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в каталог с соответствующим названием пакета.

Обновить и зафиксировать пакет в SVN в режиме разработки в файловой системе

Средний

Пример. В конфигурацию Creatio установлен пользовательский пакет `sdkPackageInFileSystem`. В режиме разработки в файловой системе необходимо выполнить его обновление, а после изменения содержимого — фиксацию в хранилище.

1. Обновить пакет из хранилища SVN

Для получения последней ревизии пакета необходимо использовать клиентское приложение для работы с SVN (например, [TortoiseSvn](#)).

Чтобы **обновить пакет из хранилища SVN**:

1. В каталоге `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` выберите необходимый пакет. В нашем примере это `sdkPackageInFileSystem`.
2. Обновите пакет (команда [*SVN Update*]).
После выполнения команды будут обновлены дата редактирования пакета в файле дескриптора пакета `descriptor.json` и исходный код схемы типа [*Исходный код*] ([*Source code*]) `UsrGreetingService`.

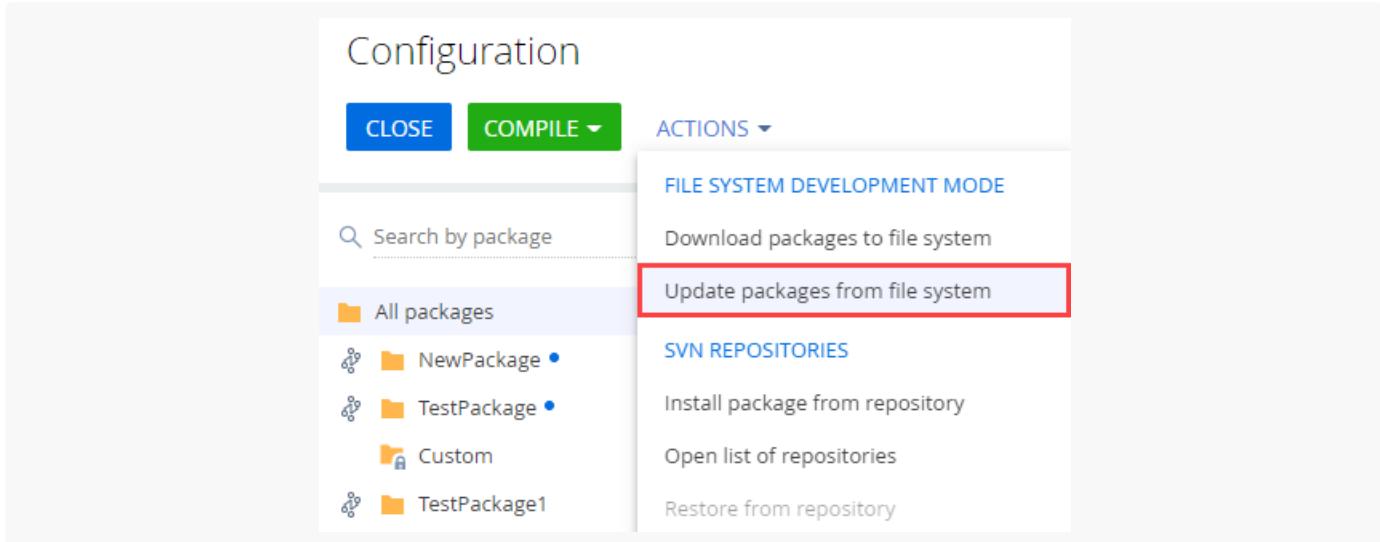
Схема `UsrGreetingService`

```

namespace Terrasoft.Configuration
{
    using System.ServiceModel;
    using System.ServiceModel.Activation;
    using System.ServiceModel.Web;
    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.R
public class UsrGreetingService : System.Web.SessionState.IReadOnlySessionState
{
    [OperationContract]
    [WebInvoke(Method = "GET", UriTemplate = "Hello")]
    public string TestHello()
    {
        return "Hello!";
    }
}
}

```

3. Перейдите в дизайнер системы по кнопке .
4. В блоке [*Конфигурирование разработчиком*] ([*Admin area*]) перейдите по ссылке [*Управление конфигурацией*] ([*Advanced settings*]).
5. На панели инструментов в группе действий [*Разработка в файловой системе*] ([*File system development mode*]) выберите [*Обновить пакеты из файловой системы*] ([*Update packages from file system*]).



6. Если были изменены схемы объектов или схемы исходного кода, то для применения изменений также необходимо выполнить шаги 3—6 статьи [Установить пакет из SVN в режиме разработки в файловой системе](#).

2. Изменить содержимое пакета

Чтобы **изменить содержимое пакета**, добавьте в схему типа [*Исходный код*] ([*Source code*])

```
UserGreetingService МЕТОД TestHelloWorld()
```

Схема UserGreetingService

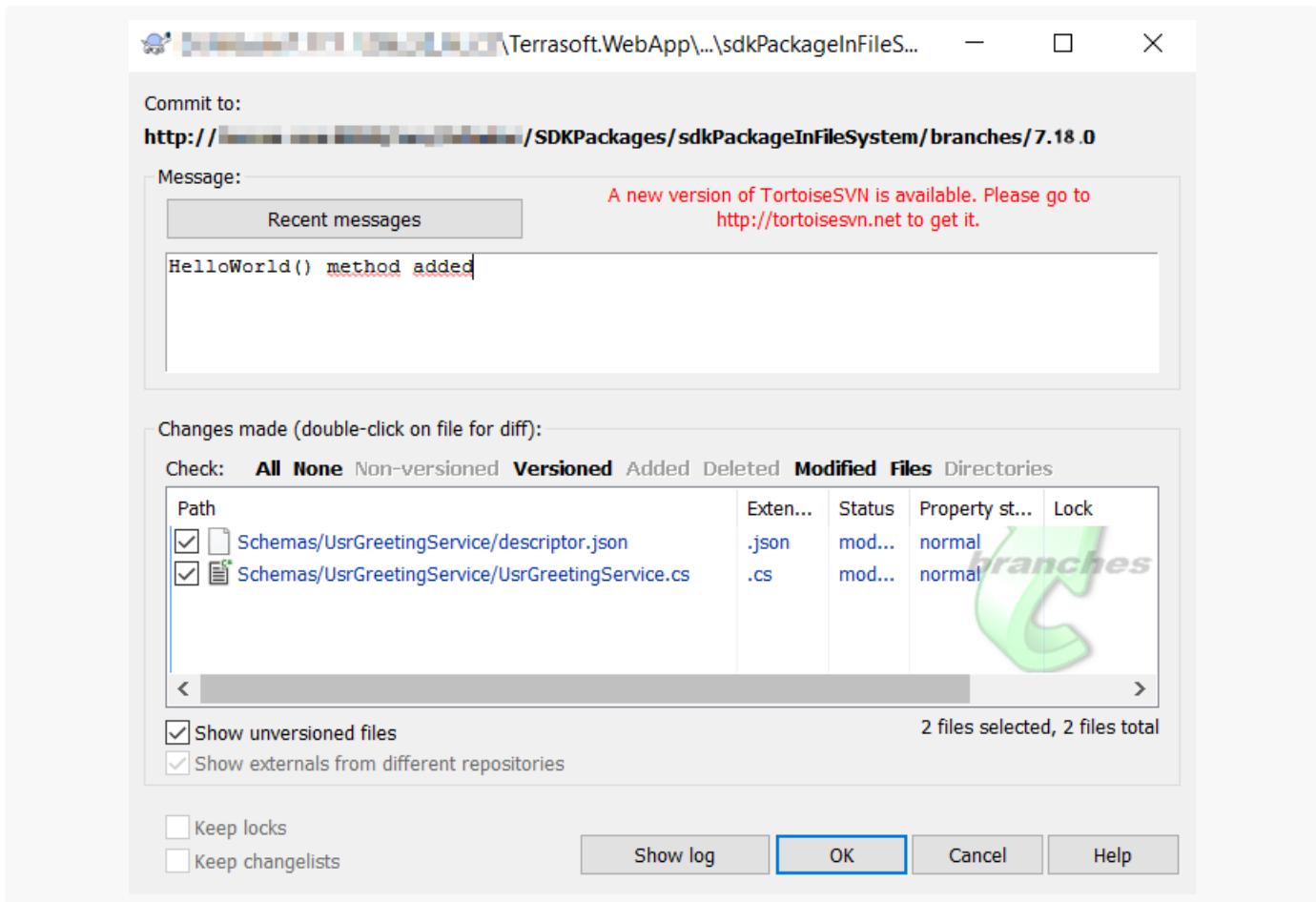
```
namespace Terrasoft.Configuration
{
    using System.ServiceModel;
    using System.ServiceModel.Activation;
    using System.ServiceModel.Web;
    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.RequireSSL)]
    public class UserGreetingService : System.Web.SessionState.IReadOnlySessionState
    {
        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "Hello")]
        public string TestHello()
        {
            return "Hello!";
        }

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "HelloWorld")]
        public string TestHelloWorld()
        {
            return "Hello world!";
        }
    }
}
```

3. Зафиксировать пакет в хранилище

Чтобы **записать пакет в хранилище**:

1. В каталоге ..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg выберите необходимый пакет. В нашем примере это `sdkPackageInFileSystem`.
2. Выполните фиксацию (команда `SVN Commit`) каталога в хранилище.



Создать пакет при переходе в режим разработки в файловой системе

 Средний

Пример. Создать пользовательский пакет `sdkPackageForFileSystem`, привязанный к хранилищу SVN. Выполнить настройку Creatio таким образом, чтобы в режиме разработки в файловой системе после выгрузки пакета его содержимое в файловой системе также было привязано к хранилищу SVN.

Приведенный в этой статье пример требует четкого понимания **разницы между режимами разработки**.

Общие рекомендации:

- В режиме разработки в файловой системе работать с хранилищем SVN следует только из файловой системы.
- В режиме разработки с помощью встроенных средств работать с SVN нужно только встроенными средствами раздела [Конфигурация] ([Configuration]).

1. Задать путь к каталогу для рабочих копий пакетов

В режиме разработки в файловой системе после выполнения действия [Выгрузить пакеты в файловую систему] ([*Download packages to file system*]) все пользовательские пакеты будут выгружены в каталог `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg`. При этом содержимое пользовательского пакета, выгруженное в файловую систему, не будет привязано к хранилищу SVN, даже если пакет был привязан к хранилищу в разделе [Конфигурация] ([*Configuration*]).

Если при создании пакета заполнить поле [Хранилище системы контроля версий] ([*Version control system repository*]) с помощью встроенных средств, то пакет будет привязан к хранилищу SVN. При этом в файловой системе будет создана рабочая копия пакета. Путь к каталогу, в котором создаются рабочие копии пакетов, задается в файле `ConnectionStrings.config` с помощью настройки

`defPackagesWorkingCopyPath`. Если в настройке `defPackagesWorkingCopyPath` указать путь к каталогу `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg`, то после выгрузки пакета в файловую систему он будет автоматически привязан к нужному хранилищу SVN.

Чтобы **задать путь к каталогу для рабочих копий пакетов**, в настройке `defPackagesWorkingCopyPath` файла `ConnectionStrings.config` укажите полный путь к каталогу `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg`.

Пример файла `ConnectionStrings.config`

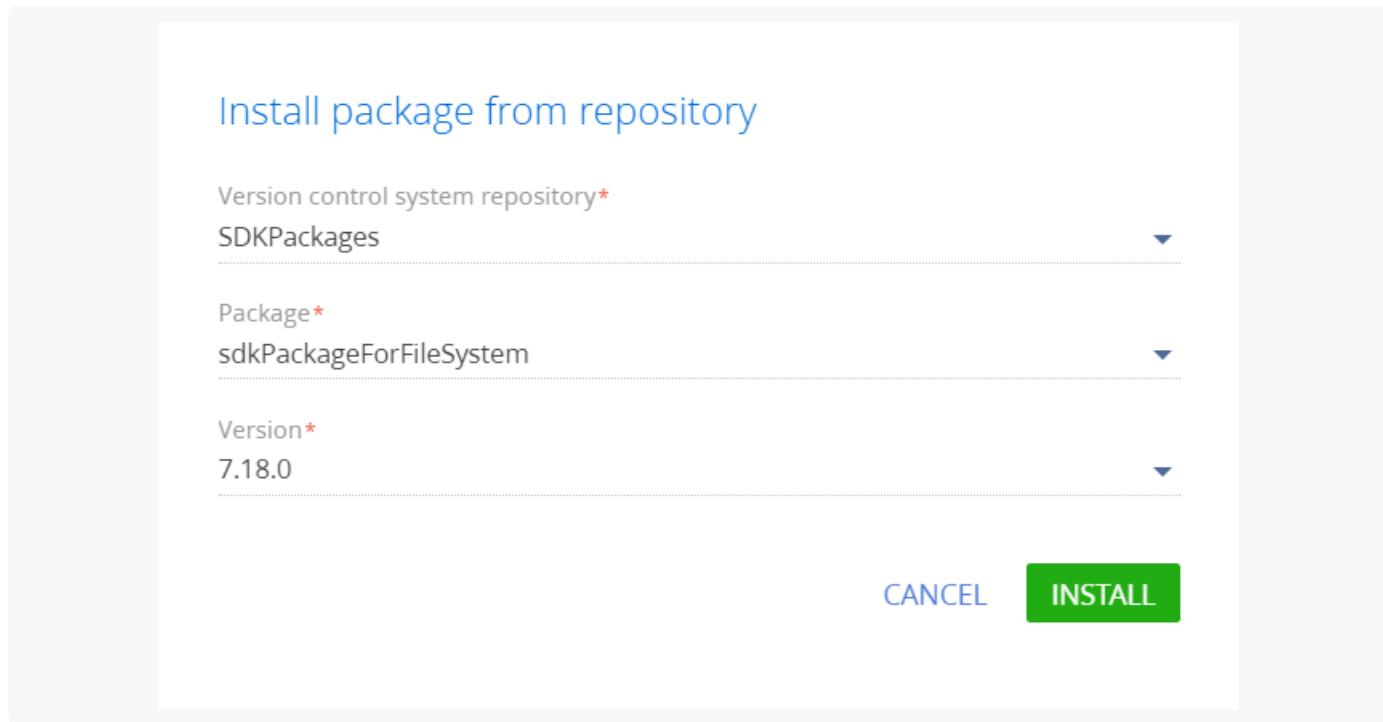
```
<?xml version="1.0" encoding="utf-8"?>
<connectionStrings>
    ...
    <add name="defPackagesWorkingCopyPath" connectionString="C:\creatio\Terrasoft.WebApp\Terrasoft
    ...
</connectionStrings>
```

Это изменение позволяет совместить каталог, в котором содержатся рабочие копии пользовательских пакетов, с каталогом, в который выгружаются пакеты в режиме разработки в файловой системе.

2. Создать пакет

Чтобы **создать пользовательский пакет**:

- Перейдите в дизайнер системы по кнопке .
- В блоке [Конфигурирование разработчиком] ([*Admin area*]) перейдите по ссылке [Управление конфигурацией] ([*Advanced settings*]).
- В области работы с пакетами нажмите кнопку .
- Заполните **свойства пакета**:



- [Название] ([Name]) — "sdkPackageForFileSystem".
- [Хранилище системы контроля версий] ([Version control system repository]) — "SDKPackages".
- [Версия] ([Version]) — "7.18.0".

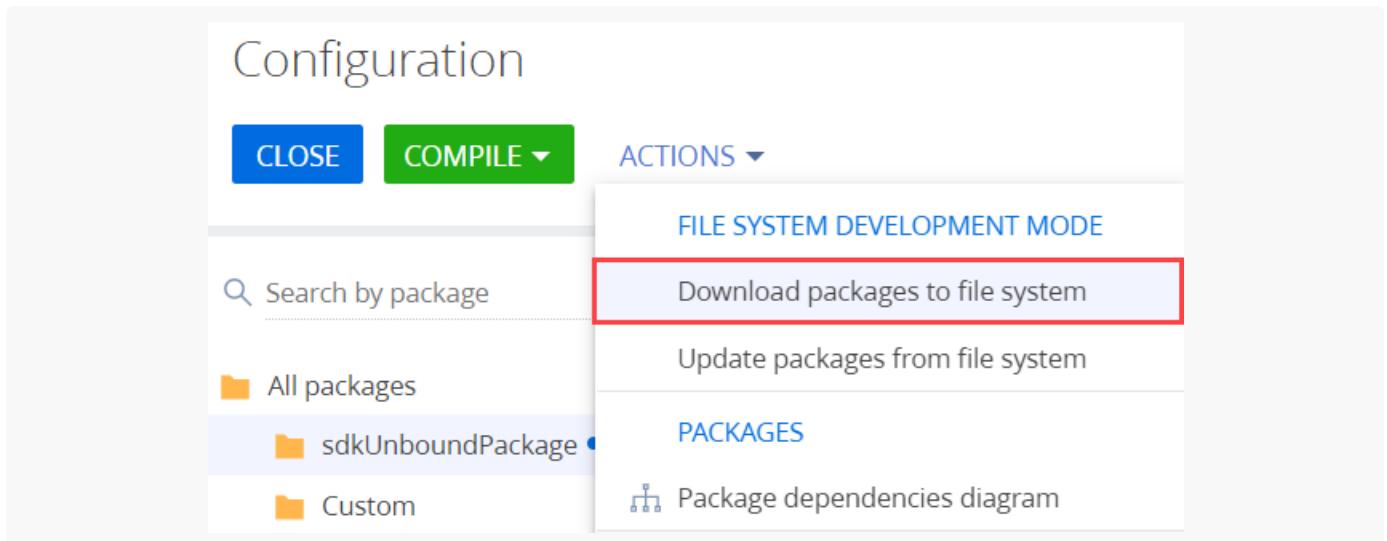
5. Нажмите кнопку [Создать и добавить зависимости] ([Create and add dependencies]) и установите зависимости пакета.

В результате пакет будет автоматически зафиксирован в SVN, а в каталоге
`..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` будет создана рабочая копия пакета.

3. Выгрузить пакет в файловую систему

Чтобы **выгрузить пакет** в файловую систему:

1. Настройте Creatio для работы в файловой системе. Настройка описана в статье [Внешние IDE](#).
2. На панели инструментов в группе действий [Разработка в файловой системе] ([File system development mode]) выберите [Выгрузить все пакеты в файловую систему] ([Download packages to file system]).



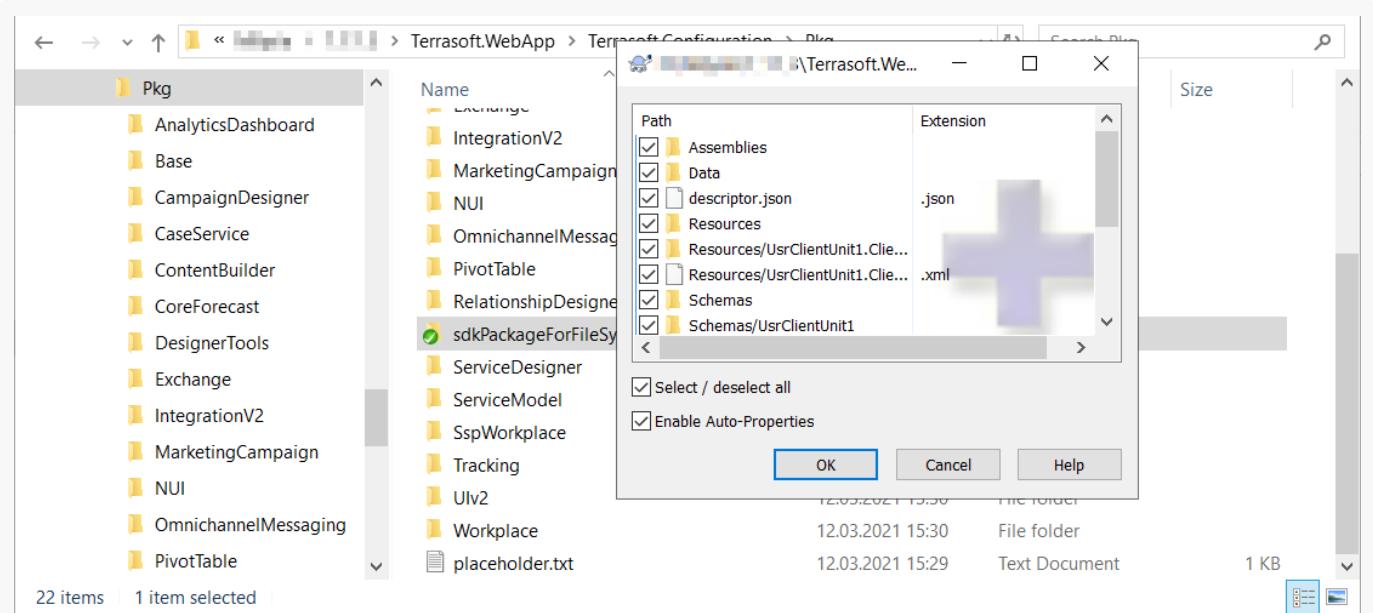
В результате все пакеты будут выгружены по пути `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в каталог с соответствующим названием пакета.

4. Зафиксировать пакет в хранилище

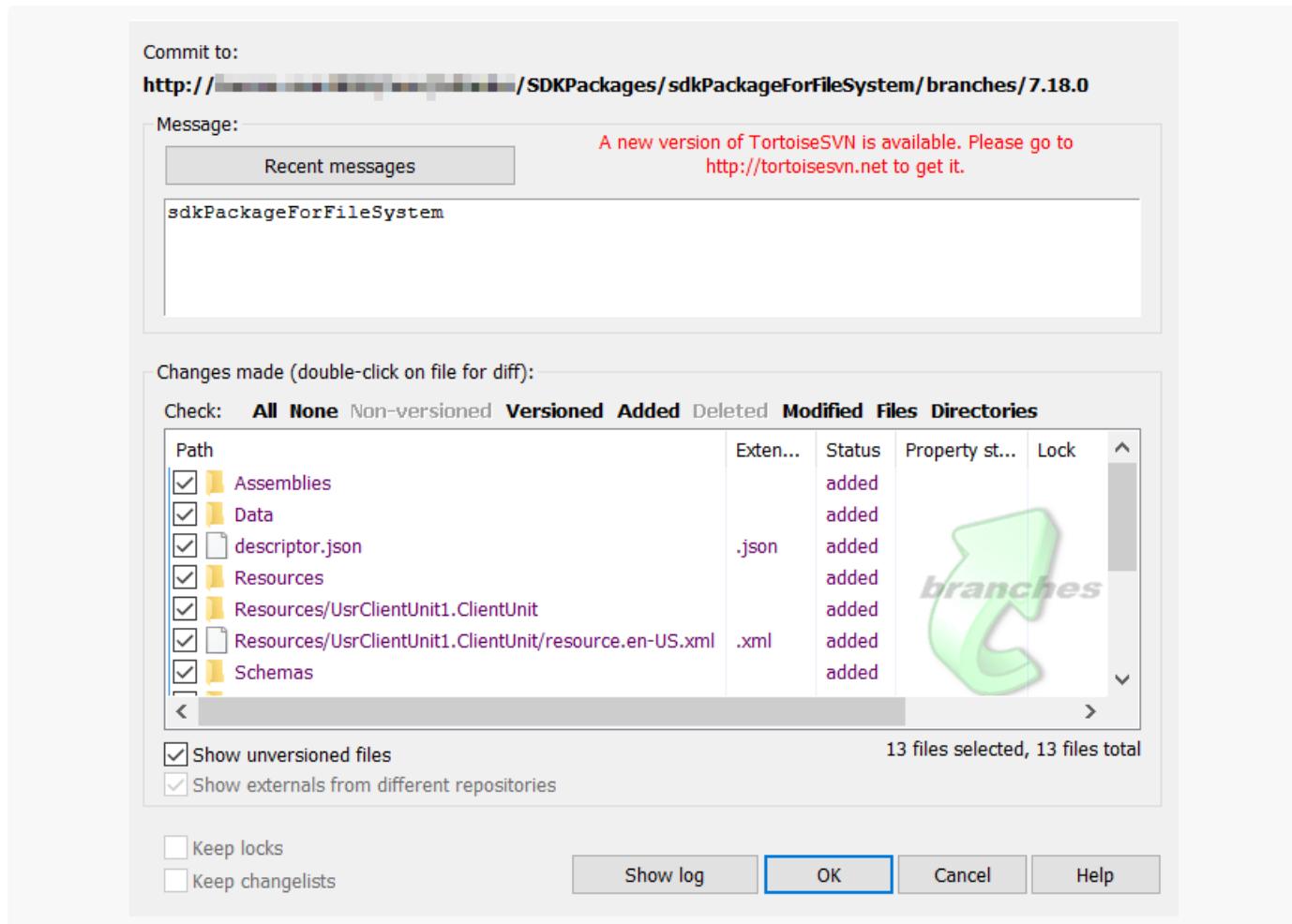
Чтобы **зафиксировать пакет в хранилище**:

- Добавьте (команда `SVN Commit`) в хранилище содержимое каталога

`...\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\sdkPackageForFileSystem`.



- Выполните фиксацию каталога в хранилище.



Логирование входящих http-запросов в Creatio для .NET Core

Легкий

Возможность логирования входящих http-запросов доступна для приложений Creatio на платформе **.NET Core** с версии 7.17.3.

Для приложений Creatio на платформе **.NET Core** стандартное логирование включено по умолчанию. Для использования расширенного логирования необходимо выполнить его настройку. Настройка логирования входящих http-запросов выполняется в секции `RequestLogging` конфигурационного файла `appsettings.json`, который находится в корневом каталоге приложения.

Секция настройки логирования файла `appsettings.json`

```
"RequestLogging": {
    "Standard": {
        "Enabled": true,
        "StatusCodes": [ 200, 300, 302 ]
    },
}
```

```

    "Extended": {
        "Enabled": true,
        "LogRequestBody": false,
        "MaxBodySizeBytes": 500,
        "StatusCodes": [ 400, 401, 403 ]
    }
}

```

Стандартное логирование входящих http-запросов

Стандартное логирование позволяет получить информацию о логах входящих http-запросов.

Стандартные логи записываются в файл `Request.log`, который находится в каталоге `Logs` корневого каталога приложения.

Настройте стандартное логирование:

1. Откройте конфигурационный файл `appsettings.json`, который находится в корневом каталоге приложения.
2. Перейдите в секцию `standard`. Стандартное логирование по умолчанию включено (значение `true` флага `Enabled`).
3. В флаге `StatusCodes` перечислите http-коды ответа, для которых необходимо выполнять запись логов. Если включить флаг `Enabled` и оставить незаполненным флаг `StatusCodes`, то запись логов будет выполняться для всех входящих http-запросов.

Для анализа логов можно использовать утилиту Log Parser Studio, поскольку формат логов аналогичен формату логов для IIS.

Расширенное логирование входящих http-запросов

Расширенное логирование позволяет получить детальную информацию о логах входящих http-запросов.

Расширенные логи записываются в файл `ExtendedRequest.log`, который находится в каталоге `Logs` корневого каталога приложения. При включенном расширенном логировании информация о теле ответа входящего http-запроса будет записана автоматически.

Настройте расширенное логирование:

1. Откройте конфигурационный файл `appsettings.json`, который находится в корневом каталоге приложения.
2. Перейдите в секцию `Extended`. Расширенное логирование по умолчанию отключено (значение `false` флага `Enabled`).
3. Включите логирование, установив значение `true` для флага `Enabled`.
4. При необходимости получить в логе информацию о теле http-запроса, установите значение `true` для флага `LogRequestBody` (по умолчанию — `false`).
5. Задайте размер выводимого тела запроса/ответа (первые N байт) для флага `MaxBodySizeBytes`.
6. В флаге `StatusCodes` перечислите http-коды ответа, для которых необходимо выполнять запись

логов. Если включить флаг `Enabled` и оставить незаполненным флаг `StatusCodes`, то запись логов будет выполняться для всех входящих http-запросов.

Допускается одновременное использование стандартного и расширенного логирования, которые могут отличаться значением флага `StatusCodes`. Если для стандартного и расширенного логирования совпадают значения флага `StatusCodes`, то логи входящих http-запросов будут продублированы в файлах `Request.log` и `ExtendedRequest.log` с разной степенью детализации.

Важно. Расширенное логирование влияет на производительность при большом количестве запросов или большом значении флага `MaxBodySizeBytes`.

Процесс управления поставками

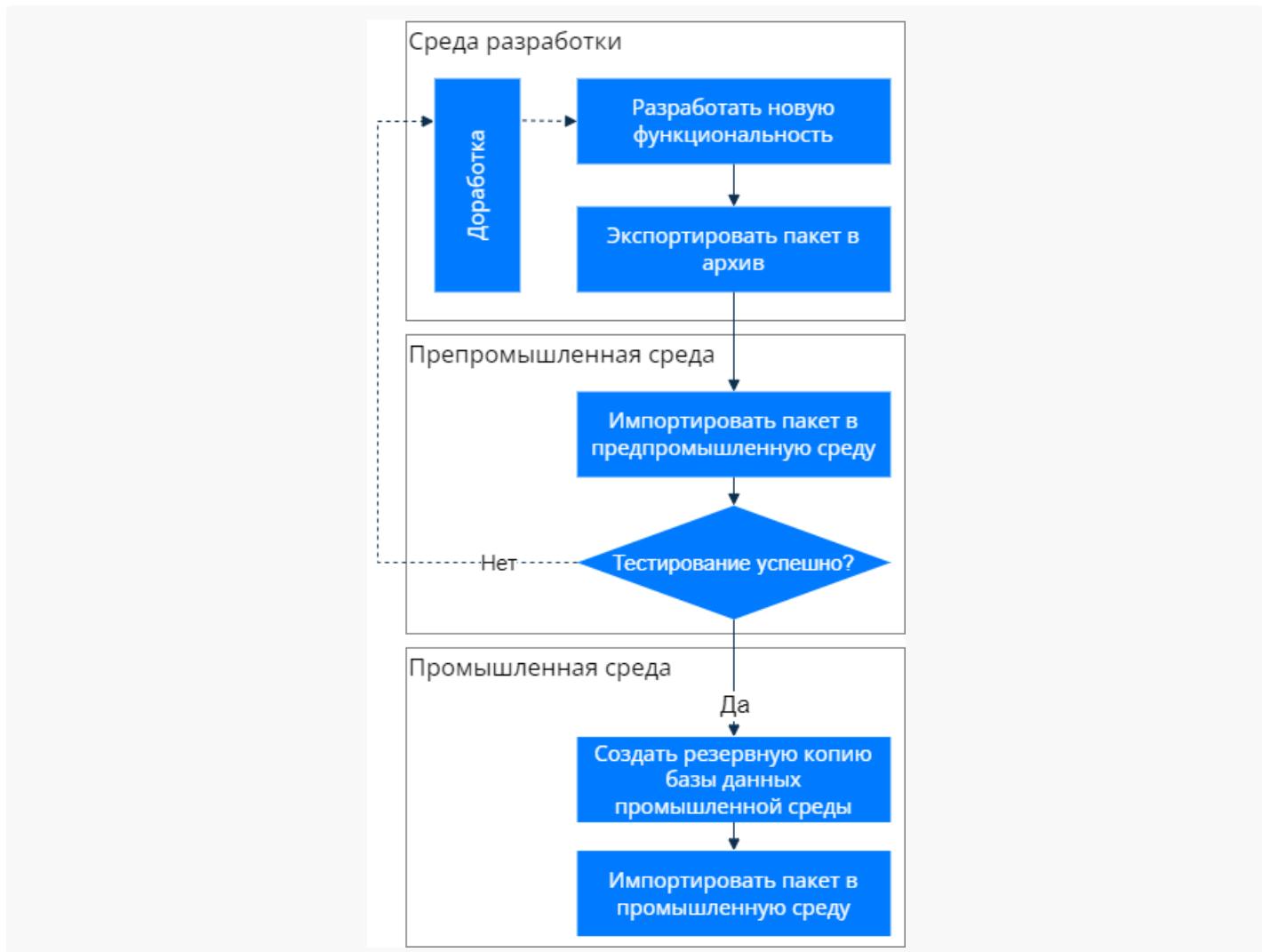


Рабочие среды, которые задействованы в процессе разработки и внедрения новой функциональности:

- Среда разработки.
- Предпромышленная среда.
- Промышленная среда.

Подробнее о рабочих средах читайте в статье [Понятие рабочей среды](#).

Чтобы избежать нарушений в работе системы и критических ошибок на промышленной среде, при переносе функциональности между рабочими средами необходимо придерживаться определенной последовательности действий, которая приведена на рисунке ниже.



1. Разработать новую функциональность

Разработку новой функциональности рекомендуется выполнять в [среде разработки](#) с персональной базой данных для каждого разработчика. Для переноса изменений между средами разработки рекомендуется использовать [систему контроля версий](#) ([Subversion](#), [Git](#) и т. д.).

Важно. Для переноса изменений в промышленную среду нельзя использовать SVN. Перенос изменений с помощью SVN следует использовать только для сред разработки.

2. Экспортировать пакет в *.zip-архив

Способы экспорта пакета в *.zip-архив:

- Из раздела [Конфигурация] ([Configuration]). Для этого воспользуйтесь инструкцией, которая приведена в статье [Перенести пакеты](#).
- С помощью утилиты WorkspaceConsole. Для этого воспользуйтесь инструкцией, которая приведена в статье [Управление поставками в WorkspaceConsole](#).

3. Импортировать пакет в предпромышленную среду

Способы импорта пакета в приложение:

- Из пользовательского интерфейса приложения. Удобен, если предпромышленная среда размещена в облаке. Для этого воспользуйтесь инструкцией, которая приведена в статье [Управление поставками в Creatio IDE](#).
- С помощью утилиты WorkspaceConsole. Удобен, если используются процессы непрерывной интеграции и предпромышленная среда размещена on-site. Для этого воспользуйтесь инструкцией, которая приведена в статье [Управление поставками в WorkspaceConsole](#).

Важно. Для переноса изменений в приложение, которое размещено в облаке, рекомендуется использовать возможности пользовательского интерфейса Creatio. Использование WorkspaceConsole невозможно, поскольку у пользователя нет непосредственного доступа к базе данных облачного приложения.

Импорт пакета отличается для рабочей среды с балансировщиком нагрузки. Чтобы **импортировать пакет на среду с балансировщиком**, воспользуйтесь инструкцией, которая приведена в статье [Установить приложение Marketplace](#).

В случае возникновения ошибок при тестировании разработанной функциональности, выполните ее доработку, устранив все ошибки. Затем повторить шаги 1–3.

4. Создать резервную копию базы данных промышленной среды

Перед поставкой пакетов с разработанной функциональностью в приложение промышленной среды, выполните резервное копирование базы данных. Для этого воспользуйтесь инструкцией, которая приведена в [инструкции по обновлению on-site](#). Этот шаг является обязательным, поскольку существует вероятность, что функциональность, разработанная разными сторонними разработчиками, может влиять на общую работоспособность приложения.

Важно. Чтобы создать резервную копию базы данных приложения, которое размещено в облаке, обратитесь в службу поддержки. При размещении приложения on-site резервная копия базы данных создается клиентом самостоятельно.

5. Импортировать пакет в промышленную среду

Способы импорта пакетов в промышленную среду аналогичны предпромышленной среде (шаг 3).

Пакет-проект



Пакет-проект — пакет, который позволяет разрабатывать функциональность приложения в обычном

C#-проекте.

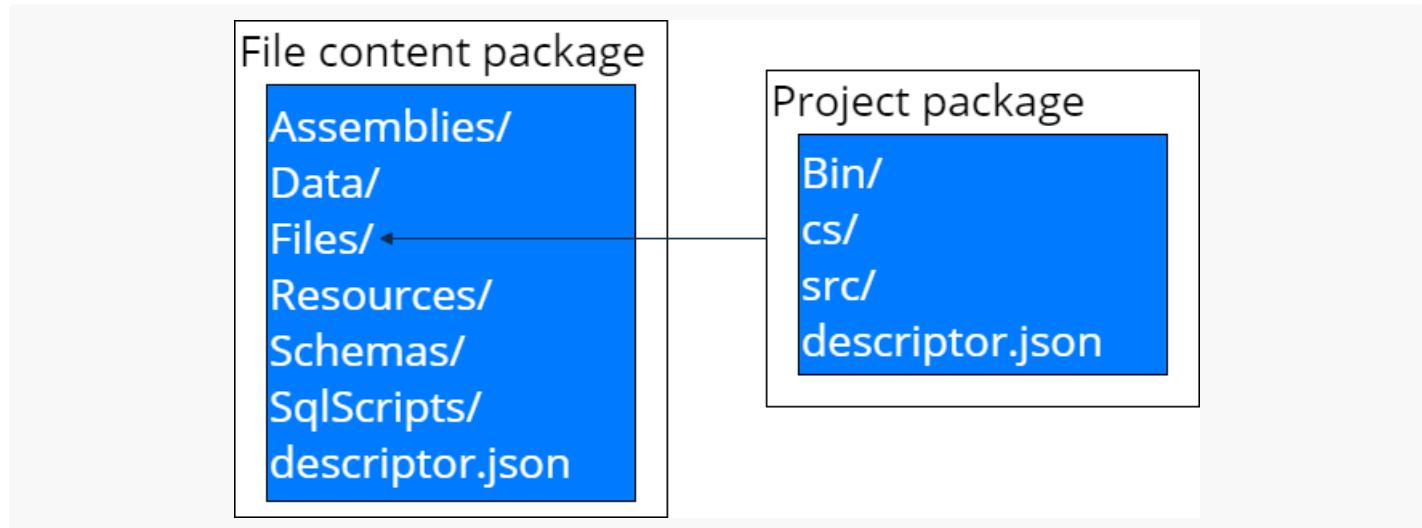
Особенности пакета-проекта

- 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.



Внешние IDE



Для увеличения производительности разработки решений Creatio можно использовать внешние интегрированные среды разработки (IDE, Integrated Development Environment), например, Visual Studio, WebStorm и т. д.

Внешние IDE позволяют не только создавать, редактировать и компилировать программный код решения Creatio, но и выполнять его отладку, вести командную разработку, использовать системы

управления версиями и многое другое.

На заметку. Для разработки в файловой системе можно использовать Microsoft Visual Studio редакций Community, Professional и Enterprise версии 2017 (с последними обновлениями) и выше.

Разработка в файловой системе

Для использования внешних IDE Creatio предоставляет механизм разработки пакетов конфигурации в файловой системе.

Режим разработки в файловой системе (РФС) позволяет:

- выгружать содержимое пакетов из базы данных в файлы;
- изменять исходный код схем с помощью IDE;
- загружать измененные пакеты обратно в базу данных.

Особенности разработки в файловой системе:

1. При включенном режиме РФС полноценная разработка поддерживается только для схем типа [Исходный код] ([*Source code*]) и [Клиентский модуль] ([*ClientUnitSchema*]).

Важно. При включенном режиме РФС после сохранения клиентских схем типа [Исходный код] ([*Source code*]) и [Клиентский модуль] ([*ClientUnitSchema*]) в [соответствующем дизайнере](#) Creatio IDE в файловую систему также сохраняются и ресурсы этих схем.

Для других элементов пакетов, например, ресурсов или SQL-скриптов, действуют следующие **правила**:

- При выгрузке из базы данных в файловую систему элементы пакетов, хранящиеся в базе данных, заменят элементы в файловой системе. Схемы типа [Исходный код] ([*Source code*]) и [Клиентский модуль] ([*ClientUnitSchema*]) заменены не будут.
- При загрузке в базу данных элементов пакетов из файловой системы они заменят элементы в базе данных. Но приложение все равно будет работать со схемами типа [Исходный код] ([*Source code*]) и [Клиентский модуль] ([*ClientUnitSchema*]) из файловой системы.

2. Интеграция с **системой контроля версий** (SVN) при включенном режиме РФС выполняется не встроенным в Creatio механизмом работы с SVN, а сторонними средствами.

Чтобы облегчить работу со связанными пакетами, в разделе [Конфигурация] ([*Configuration*]) оставлена возможность установки пакетов из хранилища SVN. Для установки единичных пакетов рекомендуется использовать **сторонние утилиты**, например, [TortoiseSVN](#).

Чтобы использовать встроенные возможности работы с хранилищем SVN, необходимо **отключить** режим разработки в файловой системе.

Настроить Creatio для работы в файловой системе

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

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

2. Отключить получение статического клиентского контента из файловой системы.

На текущий момент режим РФС не совместим с получением клиентского контента из предварительно сгенерированных файлов. Для корректной работы с РРС необходимо отключить получение статического клиентского контента из файловой системы. В файле `Web.config`, который находится в корневом каталоге установленного приложения, установите значение `false` для флага `UseStaticFileContent`.

Web.config

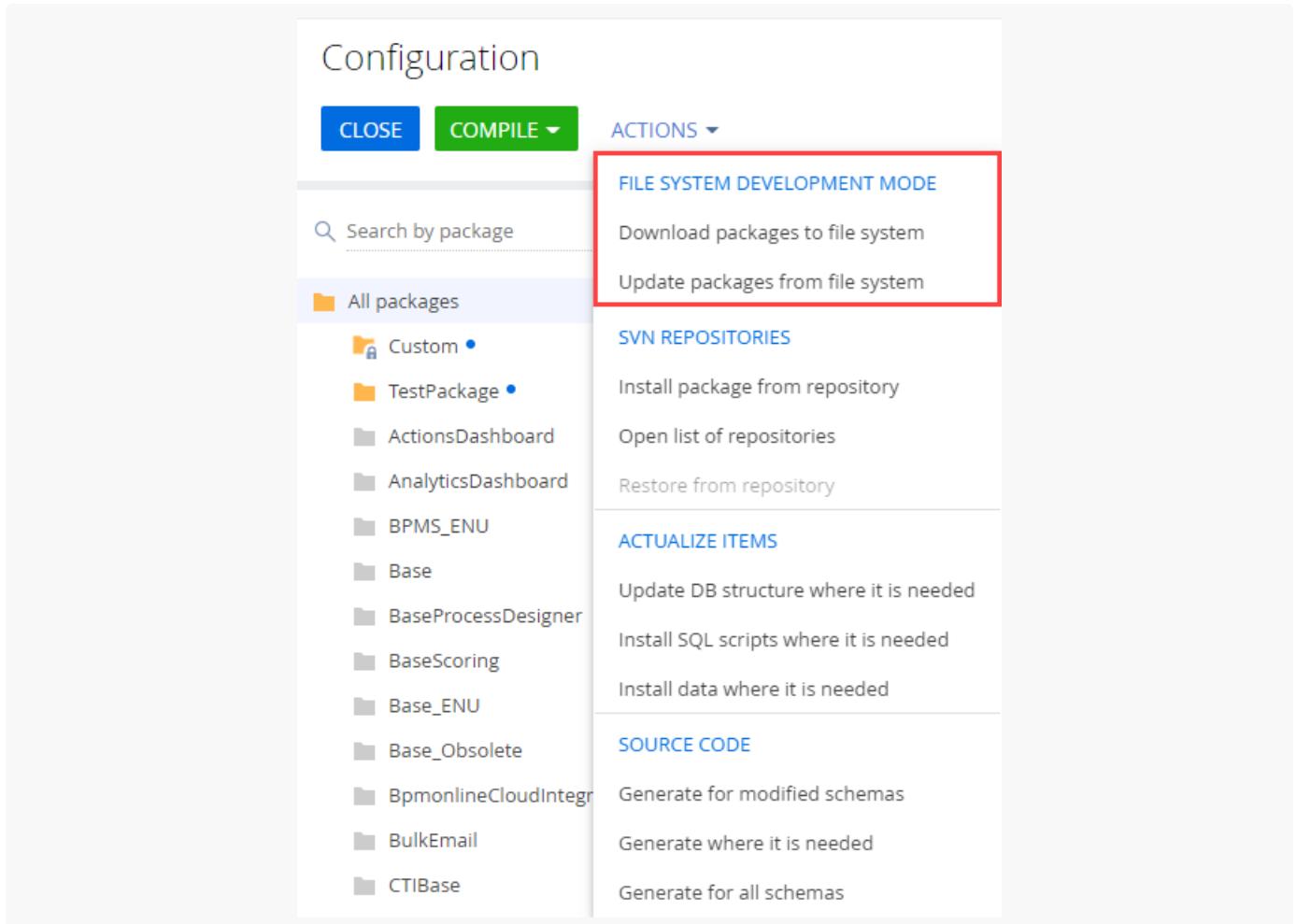
```
<fileDesignMode enabled="true"/>
...
<add key="UseStaticFileContent" value="false"/>
```

3. Скомпилировать приложение.

Выполните действие [Перекомпилировать все] ([*Compile all*]) панели инструментов раздела [Конфигурация] ([*Configuration*]).

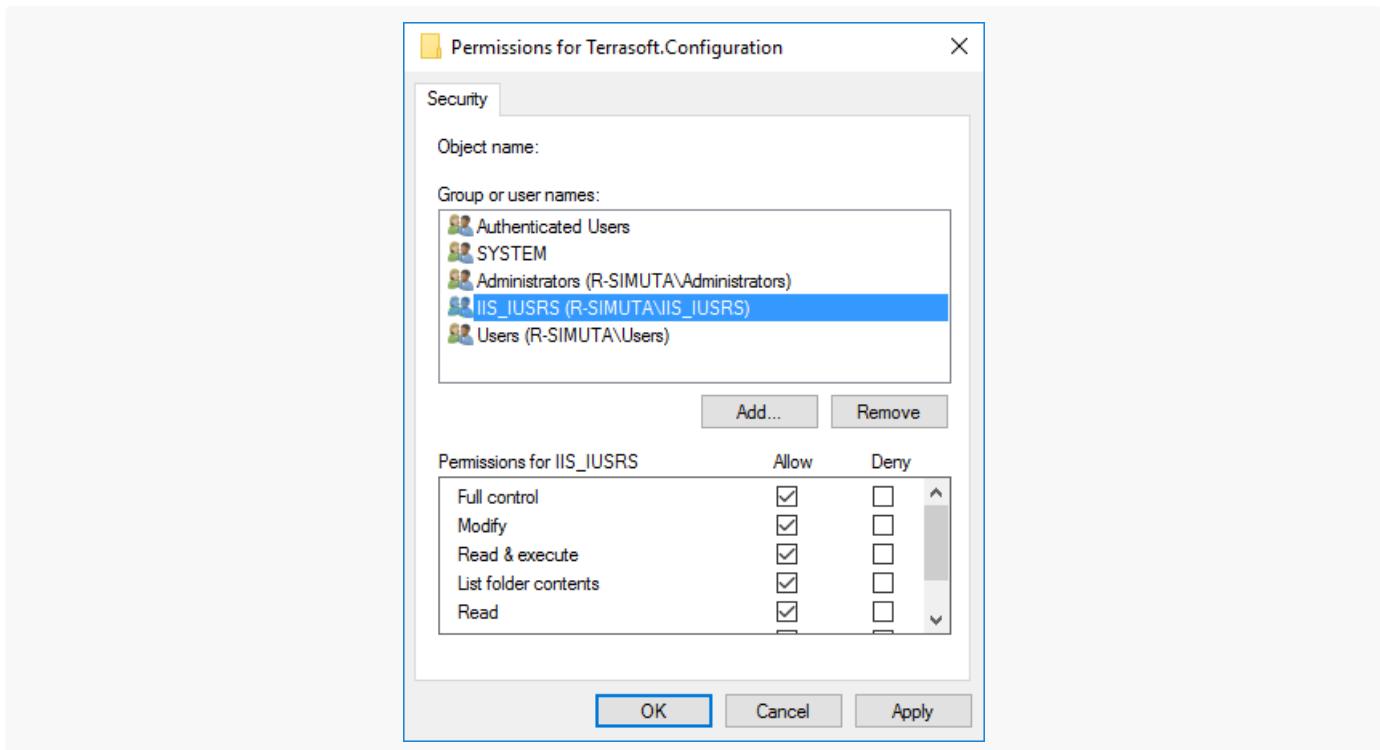
После включения режима разработки в файловой системе, в разделе [Конфигурация] ([*Configuration*]) станут активными действия группы [Разработка в файловой системе] ([*File system development mode*]):

- [Выгрузить пакеты в файловую систему] ([*Download packages to file system*]) — выгружает пакеты из базы данных приложения в каталог [Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Pkg .
- [Обновить пакеты из файловой системы] ([*Update packages from file system*]) — загружает пакеты из каталога [Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Pkg в базу данных.



4. Предоставить 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 .



Работа с пакетами

Создание пакета

Если не предполагается разработка с использованием SVN, то при включенном режиме разработки в файловой системе последовательность [создания пакета](#) ничем не отличается от обычного режима.

Важно. Если при создании пакета не заполнять поле [Хранилище системы контроля версий] ([Version control system repository]), то пакет не будет привязан к хранилищу. Вести версионную разработку этого пакета можно будет только [подключив](#) его вручную из файловой системы.

Добавить элемент пакета

Рекомендуется добавлять новые элементы (например, схему или ресурс) в пакет только из раздела [Конфигурация] ([Configuration]). Чтобы создать и отредактировать новый элемент в пользовательском пакете, необходимо выполнить следующие действия:

1. В разделе [Конфигурация] ([Configuration]) выбрать пользовательский пакет и добавить в него новый элемент ([клиентскую схему](#), [исходный код](#) и т. д.).
2. При необходимости добавить в созданную схему ресурсы, например, локализуемую строку.
3. В разделе [Конфигурация] ([Configuration]) в выпадающем списке [Действия] ([Actions]) панели инструментов выполнить действие [Выгрузить пакеты в файловую систему] ([Download packages to file system]).
4. С помощью IDE (например, Visual Studio) изменить исходный код схемы или локализуемого ресурса в

файлах, расположенных в каталоге

[Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\[Имя пакета].

- Для внесения изменений в базу данных приложения необходимо выполнить действие [*Обновить пакеты из файловой системы*] ([*Update packages from file system*]).

Важно. Изменения, сделанные в клиентских схемах, доступны в приложении сразу же, без загрузки в базу данных. Достаточно лишь обновить страницу в браузере.

- Если были изменены схемы типа [*Исходный код*], то необходимо выполнить компиляцию приложения.

На заметку. При разработке схем типа [*Исходный код*] на языке C# удобно [выполнять компиляцию](#) непосредственно в Visual Studio, а не в приложении (раздел [*Конфигурация*] ([*Configuration*])).

Разработка C# кода

Способы компиляции в Visual Studio

- С помощью **встроенного компилятора Visual Studio**. Однако он может не учесть зависимости и позиции пакетов.
- С использованием **утилиты WorkspaceConsole**, интегрируемой в Visual Studio. **Преимущества** данного способа:
 - Ускорение компиляции за счет того, что конфигурационная сборка разбивается на независимые компилируемые модули. Перекомпилируются только те модули, пакеты которых были изменены.
 - Для перекомпиляции конфигурационной сборки не нужно выходить из режима отладки или открепляться от процесса IIS.

Настроить WorkspaceConsole для компиляции

- Включить режим РФС для WorkspaceConsole.**

Утилита WorkspaceConsole поставляется вместе с приложением. Перед использованием утилиту необходимо правильно настроить. Кроме обычных настроек, в конфигурационном файле `Terrasoft.Tools.WorkspaceConsole.exe.config` также необходимо включить режим разработки в файловой системе.

Включение режима разработки в файловой системе

```
<fileDesignMode enabled="true" />
```

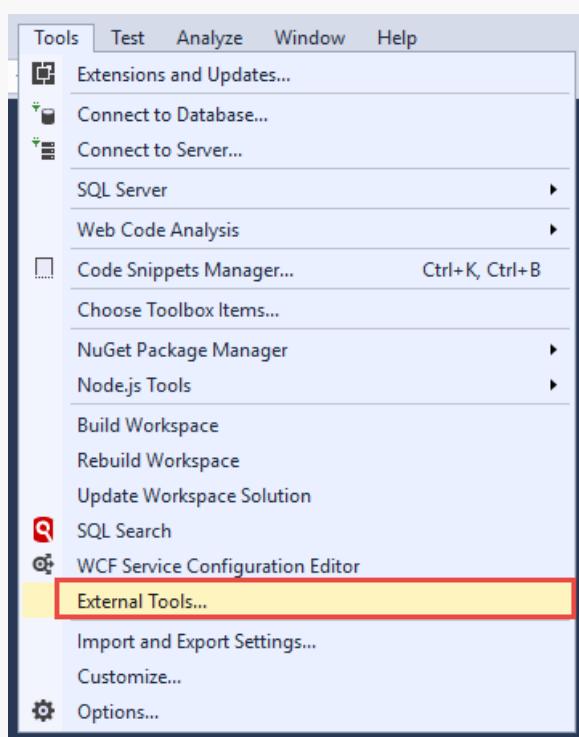
Для ускорения компиляции конфигурационную сборку необходимо разбивать на независимые компилируемые модули. В таком случае необходимо установить значение `true` для настройки `CompileByManagerDependencies` во "внутреннем" Web.config, находящемся в каталоге `Terrasoft.WebApp`, и в файле настроек `Terrasoft.Tools.WorkspaceConsole.exe.config` утилиты WorkspaceConsole.

Настройка "внутреннего" Web.Config

```
<appSettings>
  ...
  <add key="CompileByManagerDependencies" value="true" />
  ...
</appSettings>
```

2. Настройте интеграцию WorkspaceConsole в Microsoft Visual Studio.

- В среде Visual Studio выполните команду меню [*Tools*] —> [*External Tools...*].



- В появившемся диалоговом окне добавьте и настройте три **команды вызова утилиты** WorkspaceConsole:

- **Build Workspace** — компиляция изменений конфигурационного проекта.

Свойства команды Build Workspace

Command	<p>Структура</p> <p>[Путь к каталогу с установленным приложением]\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft\</p>
Arguments	<p>Структура</p> <p>--operation=BuildWorkspace --workspaceName=Default --webApplicationName=Default</p> <p>Пример</p> <p>--operation=BuildWorkspace --workspaceName=Default --webApplicationName=Default</p>

- **Rebuild Workspace** — полная компиляция конфигурационного проекта.

Свойства команды Rebuild Workspace

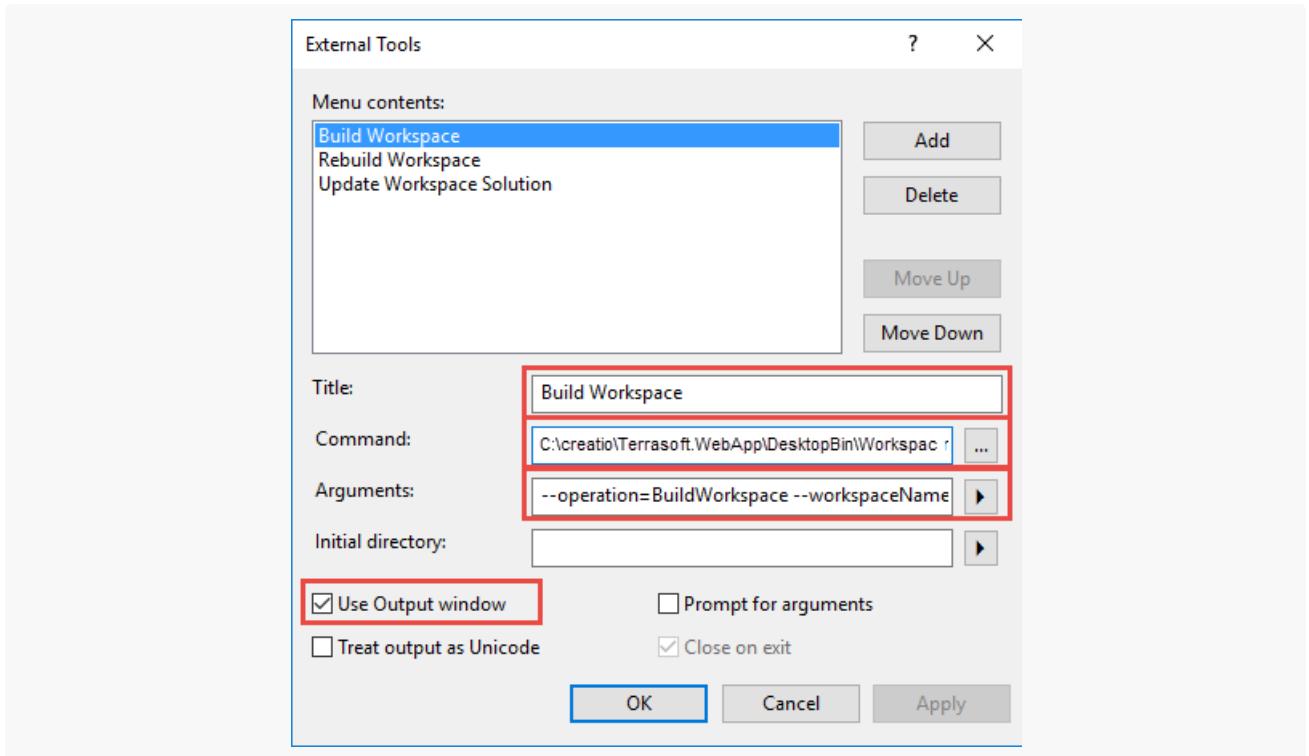
Command	<p>Структура</p> <p>[Путь к каталогу с установленным приложением]\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft\RebuildWorkspace.exe</p> <p>Пример</p> <p>C:\creatio713\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft\RebuildWorkspace.exe --operation=RebuildWorkspace --workspaceName=Default --webApplicationName=Default</p>
Arguments	<p>Структура</p> <p>--operation=RebuildWorkspace --workspaceName=Default --webApplicationName=Default</p> <p>Пример</p> <p>--operation=RebuildWorkspace --workspaceName=Default --webApplicationName=Default</p>

- **Update Workspace Solution** — обновление решения Visual Studio конфигурационного проекта из базы данных приложения. Команда применяет все изменения, внесенные пользователем через приложение.

Свойства команды Update Workspace Solution

Command	<p>Структура</p> <p>[Путь к каталогу с установленным приложением]\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft\UpdateWorkspaceSolution.exe</p>
Arguments	<p>Структура</p> <p>--operation=UpdateWorkspaceSolution --workspaceName=Default --webAppPath=C:\creatio713\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft\UpdateWorkspaceSolution.exe</p>
	<p>Пример</p> <p>--operation=UpdateWorkspaceSolution --workspaceName=Default --webAppPath=C:\creatio713\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft\UpdateWorkspaceSolution.exe</p>

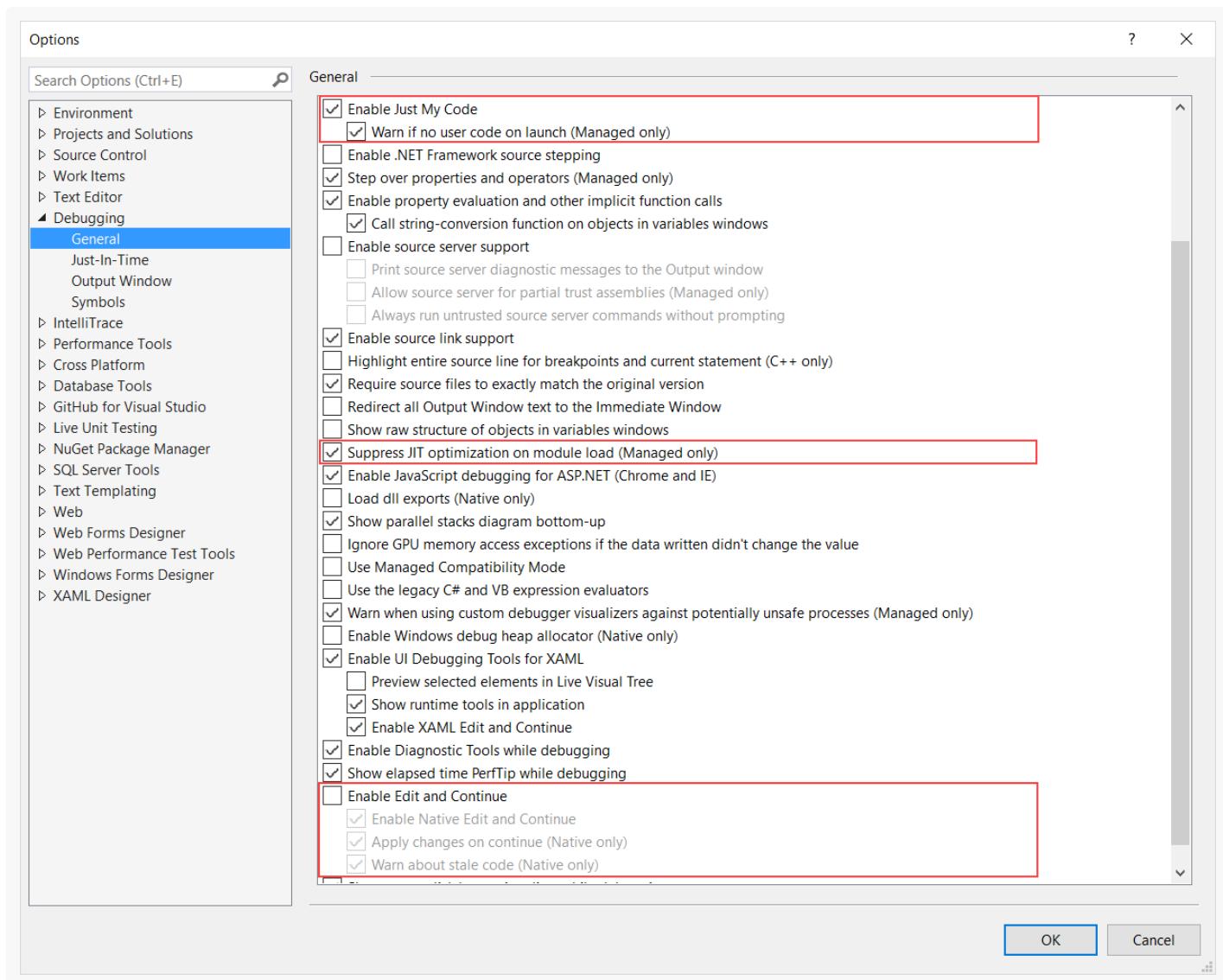
f. Для всех команд установите признак [*Use Output window*].



Настроить отладку в Visual Studio

Выполните команду меню [Debug] —> [Options...] и в появившемся диалоговом окне измените следующие **опции**:

- Чтобы отладчик не пытался зайти в исходный код, которого нет в проекте, включите опцию [*Enable Just My Code*].
- Поскольку после компиляции конфигурации выполняется автоматический перезапуск приложения, то опция [*Enable Edit and Continue*] не поддерживается и ее следует отключить.
- Чтобы отладчик корректно останавливался на точках останова (BreakPoint), необходимо удостовериться, что включена опция [*Suppress JIT optimization on module load*].



Разработка C# кода в конфигурационном проекте

Конфигурационный проект `Terrasoft.Configuration.sln` — это предварительно настроенное решение Visual Studio, которое поставляется с приложением Creatio и находится в каталоге [Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration.

Преимущества разработки в конфигурационном проекте:

- Полностью настроенное решение для разработки сложной back-end функциональности.
- Предоставляет возможность использовать всю функциональность IDE — отладку, рефакторинг, автозавершение и т. д.

Ограничения разработки в конфигурационном проекте:

- Довольно низкая производительность, связанная с перекомпиляцией всей конфигурационной части Creatio (`Terrasoft.Configuration.dll`).
- Разработку C# кода в файловой системе можно выполнять только взаимодействуя с базой данных приложения, развернутого **on-site**.

Для начала разработки в файловой системе с использованием конфигурационного проекта запустите файл [путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Terrasoft.Configuration.sln в Visual Studio.

Структура конфигурационного проекта

Каталог	Назначение
Lib	Каталог, в который выгружаются сторонние библиотеки классов, привязанные к пакетам
Autogenerated\Src	Каталог, содержащий файлы с автоматически сгенерированным исходным кодом схем предустановленных пакетов. Эти файлы нельзя редактировать
Pkg	Каталог, в который из базы данных выгружаются пакеты для разработки в файловой системе
bin	Каталог выгрузки скомпилированной конфигурации и сторонних библиотек

Разработка C# кода в пользовательском проекте

Пользовательский проект — отдельный проект библиотеки классов, предварительно настроенный для работы с Creatio.

Для отладки и проверки результата разработки в пользовательском проекте можно использовать как локальную базу данных, подключившись к ней с помощью утилиты WorkspaceConsole, так и размещенное в облаке приложение, подключившись к нему с помощью специально разработанной утилиты — Executor.

Преимущества разработки в пользовательском проекте:

- Высокая скорость проверки изменений, компиляции и запуска на выполнение.
- Полноценное использование функциональности Visual Studio.
- Возможность использования любых инструментов для непрерывного цикла разработки ([Continuous Integration](#)), например, Unit-тестирования.
- Простота настройки проекта — не нужны исходные коды конфигурации.
- Можно использовать базу данных приложения, развернутого как локально, так и в облаке.

Ограничения разработки в пользовательском проекте:

- Можно использовать только для разработки и отладки отдельных классов или небольших блоков back-end функциональности.
- Необходимо отдельно подключать в зависимости проекта нужные библиотеки классов Creatio.

Разработка JavaScript кода

Для разработки front-end функциональности с использованием внешних IDE необходимо:

- Выполнить предварительные настройки приложения Creatio для разработки в файловой системе.

2. Создать, получить или обновить пакет из репозитория SVN.
3. В разделе [Конфигурация] ([Configuration]) создать клиентскую схему, в которой будет выполняться разработка.
4. Выгрузить схему из базы данных в файловую систему.
5. Выполнить разработку исходного кода схемы во внешней IDE.
Для выполнения разработки необходимо открыть файл с исходным кодом схемы в предпочтаемой IDE (или любом текстовом редакторе) и добавить нужный исходный код.
6. Сохранить схему и выполнить отладку созданного исходного кода.

При разработке JavaScript кода в файловой системе после внесения изменений в исходный код схемы необходимо каждый раз **обновлять страницу браузера**, на которой открыто приложение. Это существенно снижает производительность разработки.

Для устранения этого была разработана функциональность автоматической перезагрузки страницы браузера после внесения изменений в исходный код.

Настроить автоматическое отображение изменений JavaScript кода

При старте приложения создается объект, отслеживающий изменения *.js-файла с исходным кодом разрабатываемого модуля в файловой системе. Если изменения произошли, то отправляется сообщение в клиентское приложение (Creatio). В клиентском приложении специальный объект, подписанный на это сообщение, определяет зависимые объекты измененного модуля, разрушает связи, регистрирует новые пути к модулям и пытается заново загрузить измененный модуль. Это приводит к тому, что все проинициализированные модули запрашиваются браузером по новым путям и загружают изменения из файловой системы.

Преимущества использования автоматического отображения изменений JavaScript кода:

- Экономия времени на интерпретацию и загрузку других модулей. Отдельная страница разработки позволяет не загружать ряд вспомогательных модулей, например, левую и правую панели, панель уведомлений и т. д.
- Уменьшение количества запросов на сервер.
- Выявление избыточной связанности модулей. Этот подход к разработке отдельных модулей позволяет вовремя обнаружить ненужные зависимости и избавиться от них.

Ограничения использования автоматического отображения изменений JavaScript кода

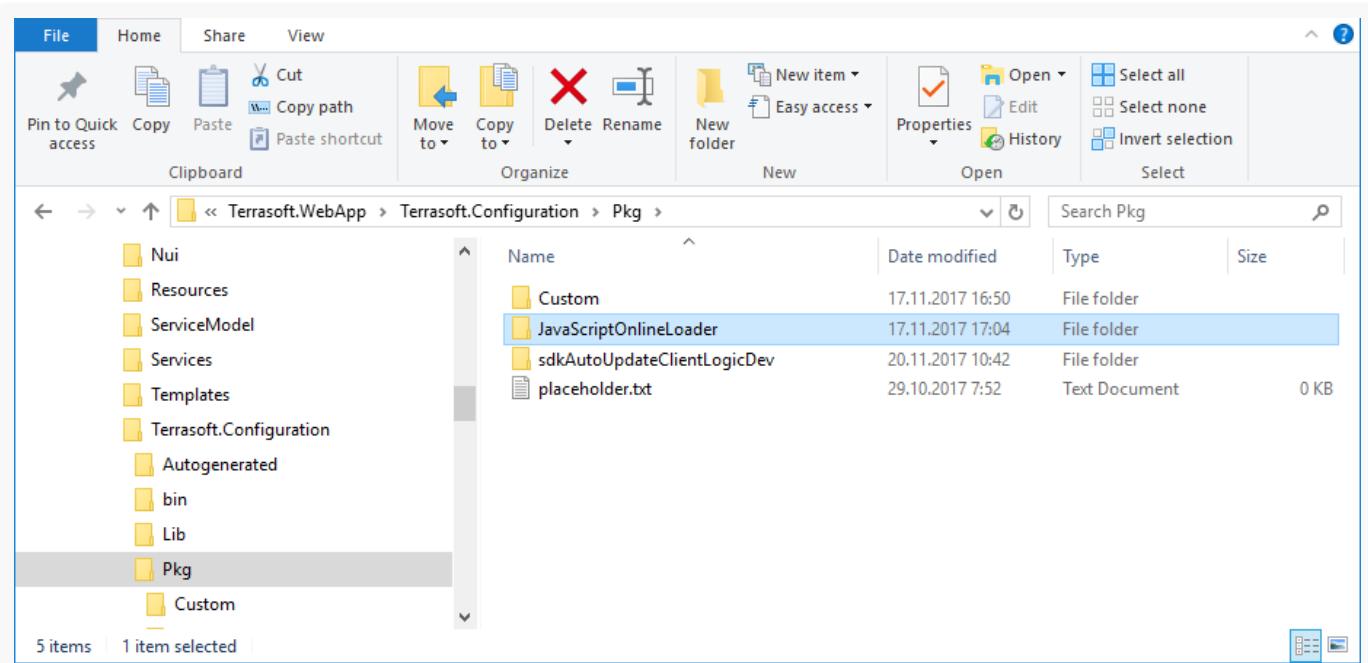
- Наличие синтаксических ошибок в коде. Если в исходном коде модуля допущена синтаксическая ошибка, то автоматическое обновление страницы не произойдет. Потребуется ее принудительное обновление (например, клавишей F5). При исправлении ошибки страница вернется к работоспособному состоянию.
- Эффект сильной связанности модулей. Не все модули Creatio могут загружаться отдельно.

Для использования автоматического отображения изменений при разработке JavaScript кода необходимо выполнить следующие действия:

1. Установить пакет **JavaScriptOnlineLoader**.

При включенном режиме разработки в файловой системе необходимо добавить каталог `JavaScriptOnlineLoader`, содержащий нужный пакет, в каталог

[Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Pkg .



Важно. Пакет доступен на [GitHub](#). Также архив с пакетом можно скачать по [ссылке](#).

Затем нужно загрузить пакет в конфигурацию, выполнив действие [*Обновить пакеты из файловой системы*] ([*Update packages from file system*]).

В результате пакет отобразится в области работы с пакетами.

2. Открыть в браузере страницу разрабатываемого модуля.

Для этого необходимо открыть страницу `ViewModule.aspx`, добавив к ней параметр

`?vm=DevViewModule#CardModuleV2/<Название модуля>`

Например, в пользовательский пакет добавлена замещенная схема `KnowledgeBasePageV2` — схема страницы редактирования записи раздела [*База знаний*].

URL страницы KnowledgeBasePageV2 с функциональностью автоматического отображения...

`http://localhost/creatio/0/Nui/ViewModule.aspx?vm=DevViewModule#CardModuleV2/KnowledgeBasePageV2`

Здесь `http://localhost/creatio` — URL приложения Creatio, развернутого локально.

После перехода по этому URL, отобразится страница `ViewModule.aspx` с загруженным модулем.

The screenshot shows a configuration management application interface. At the top, there are fields for 'Name' and 'Type' with red asterisks indicating required fields. Below these are 'Modified by' and 'Modified on' fields. A navigation bar at the top includes 'GENERAL INFORMATION', 'FILES', and 'CONNECTED TO' tabs, with 'GENERAL INFORMATION' being the active tab. Below the tabs is a toolbar with various icons for text editing, including bold, italic, underline, font size, and alignment. A large central area is available for editing, which appears to be a rich text editor. At the bottom, there is a search bar with the placeholder 'What are you working on?' and a settings gear icon.

3. Изменить исходный код разрабатываемой схемы.

Изменить исходный код разрабатываемой схемы можно в любом текстовом редакторе, например, в Блокноте. После сохранения изменений открытая в браузере страница будет автоматически обновлена.

Разработать C# код в конфигурационном проекте



Пример. Разработать код пользовательского веб-сервиса с использованием Visual Studio.

Важно. После успешной компиляции итоговая сборка `Terrasoft.Configuration.dll` будет помещена в каталог `Bin`, при этом IIS автоматически использует ее в приложении `Creatio`.

Алгоритм реализации примера

1. Выполнить предварительные настройки

[Настройте IDE](#) для разработки в файловой системе.

2. Создать, получить или обновить пакет из репозитория SVN

Создайте в разделе [Конфигурация] ([Configuration]) пользовательский пакет `sdkPackageInFileSystem` [с использованием](#) или [без использования SVN](#).

На заметку. При разработке в файловой системе вместо встроенных возможностей Creatio удобнее использовать клиентские приложения для работы с хранилищами систем контроля версий, например, [Tortoise SVN](#) или [Git](#).

3. Создать схему [Исходный код]

В пакете [создайте схему \[Исходный код \]](#):

- [Код] ([Code]) — "UsrGreetingService";
- [Заголовок] ([Title]) — "UsrGreetingService";
- [Пакет] ([Package]) — "sdkPackageInFileSystem";

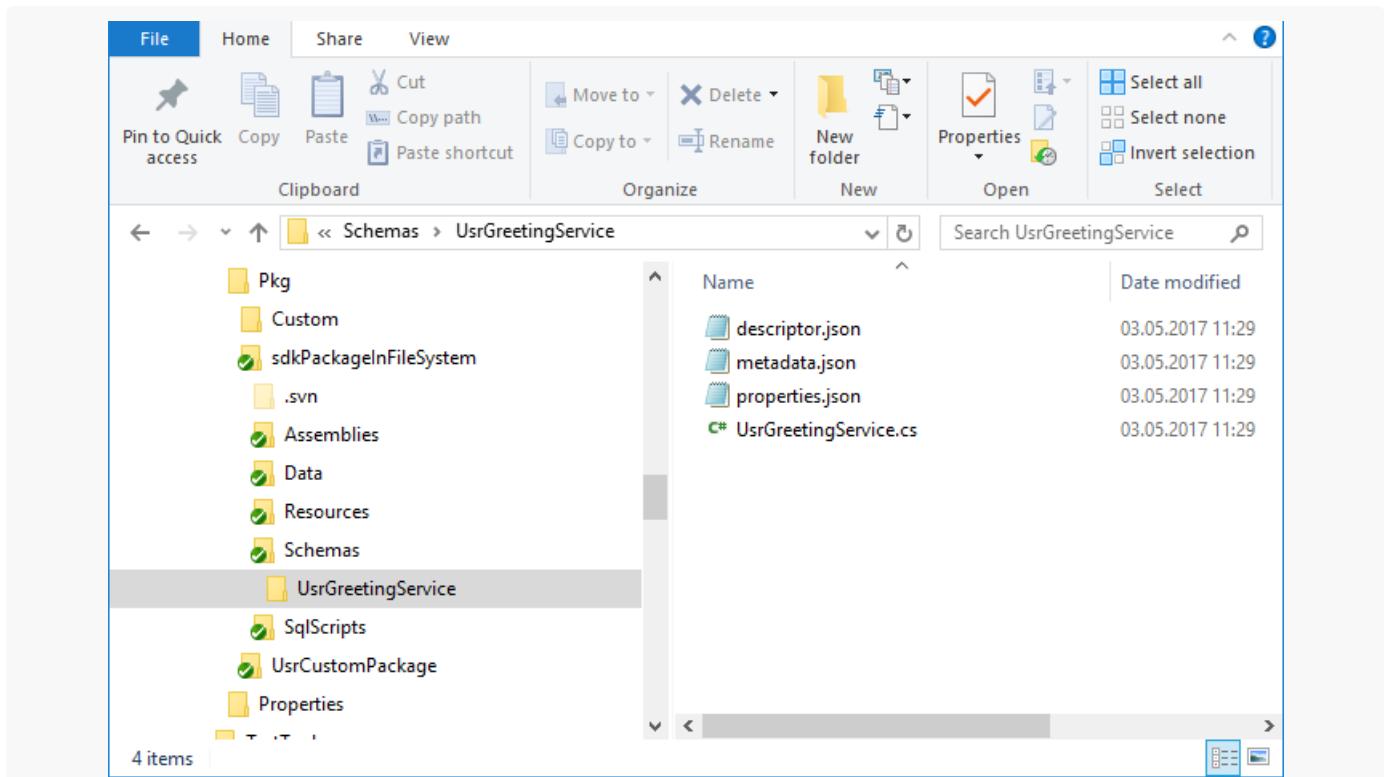
4. Выполнить разработку решения в Visual Studio

1. Выгрузите существующие схемы из базы данных в файловую систему.

Для этого в выпадающем списке [Действия] ([Actions]) на панели инструментов раздела [Конфигурация] ([Configuration]) выполните действие [Выгрузить пакеты в файловую систему] ([Download packages to file system]).

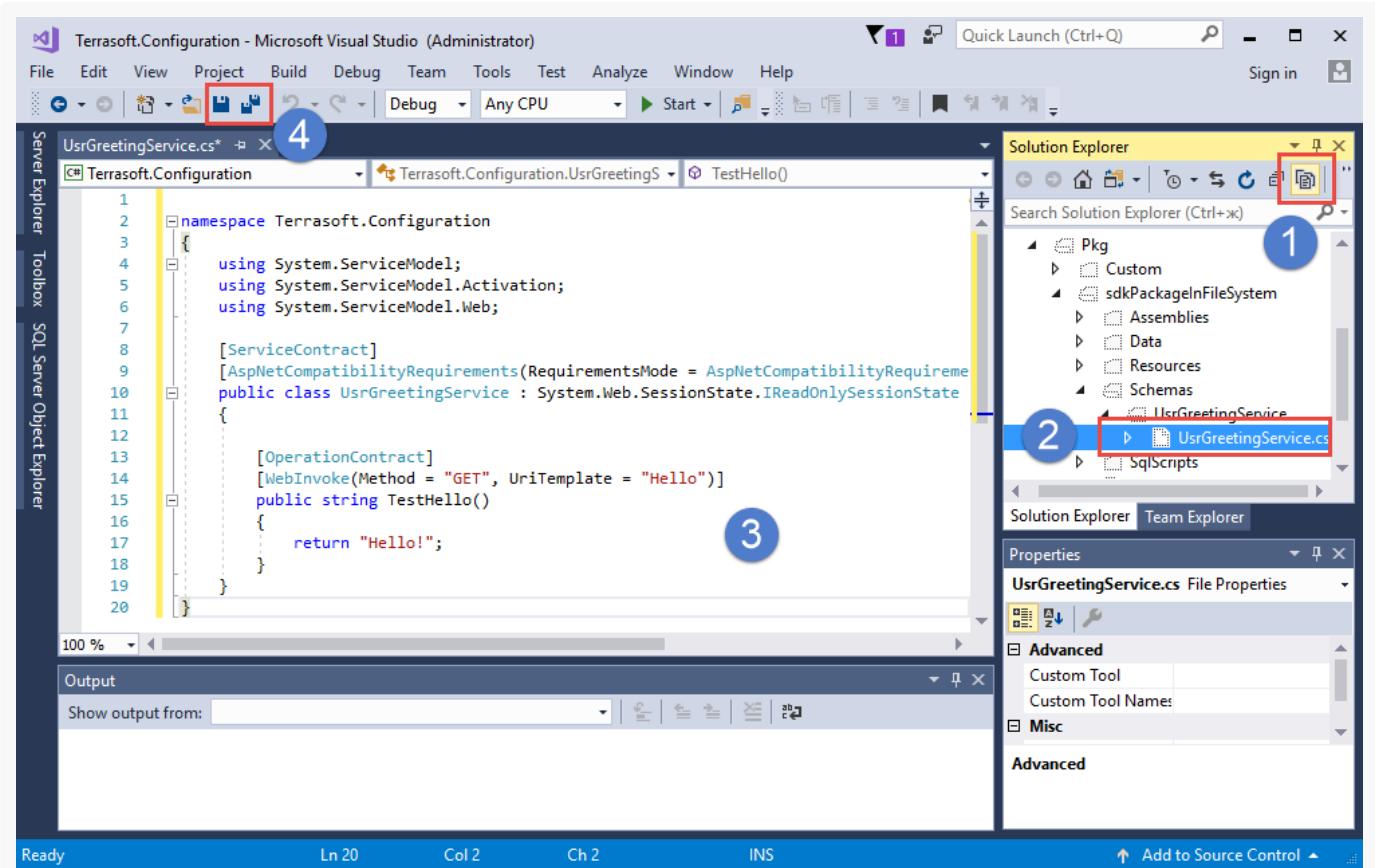
The screenshot shows the 'Configuration' application interface. At the top, there are buttons for 'CLOSE' and 'COMPILE'. To the right of these is a 'SEARCH by package' input field. Below it is a tree view of packages under 'All packages': Custom, TestPackage, ActionsDashboard, AnalyticsDashboard, BPMS_ENU, Base, BaseProcessDesigner, BaseScoring, Base_ENU, Base_Obsolete, BpmonlineCloudIntegr, BulkEmail, and CTIBase. On the right side, a vertical 'ACTIONS' dropdown menu is open. It is divided into sections: 'FILE SYSTEM DEVELOPMENT MODE', 'SVN REPOSITORIES', 'ACTUALIZE ITEMS', and 'SOURCE CODE'. The 'FILE SYSTEM DEVELOPMENT MODE' section contains two items: 'Download packages to file system' (which is highlighted with a red border) and 'Update packages from file system'. The other sections contain three items each: 'Install package from repository', 'Open list of repositories', and 'Restore from repository' for SVN; 'Update DB structure where it is needed', 'Install SQL scripts where it is needed', and 'Install data where it is needed' for ACTUALIZE ITEMS; and 'Generate for modified schemas', 'Generate where it is needed', and 'Generate for all schemas' for SOURCE CODE.

В результате в файловой системе в каталоге `Pkg\ sdkPackageInFileSystem\Schemas\` появится файл исходного кода схемы `UsrGreetingService.cs`. При этом в каталог `Autogenerated\Src` будет помещен сгенерированный системой файл схемы `UsrGreetingServiceSchema.sdkPackageInFileSystem_Entity.cs`.



На заметку. Чтобы добавить схему в SVN, необходимо добавить весь каталог `UsrGreetingService`, включая JSON-файлы.

2. Для выполнения разработки в Visual Studio откройте решение `Terrasoft.Configuration.sln`.
3. В проводнике решения Visual Studio включите отображение всех типов файлов (1), откройте файл `UsrGreetingService.cs` (2) и добавьте нужный исходный код (3).



Пример исходного кода, который нужно добавить в содержимое файла `UsrServiceGreeting.cs` для реализации [пользовательского web-сервиса](#).

```

UsrServiceGreeting.cs

namespace Terrasoft.Configuration
{
    using System.ServiceModel;
    using System.ServiceModel.Activation;
    using System.ServiceModel.Web;
    // Класс, реализующий конфигурационный сервис.
    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.R
    public class UsrGreetingService : System.Web.SessionState.IReadOnlySessionState
    {
        // Операция сервиса.
        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "Hello")]
        public string TestHello()
        {
            return "Hello!";
        }
    }
}

```

5. Сохранить, скомпилировать и отладить созданный исходный код

- После того как исходный код был изменен, прежде чем выполнить его компиляцию и отладку, **сохраните** его (4).

Обычно это выполняет Visual Studio автоматически, но поскольку компилятор Visual Studio не используется, эту операцию разработчику нужно выполнять самостоятельно.

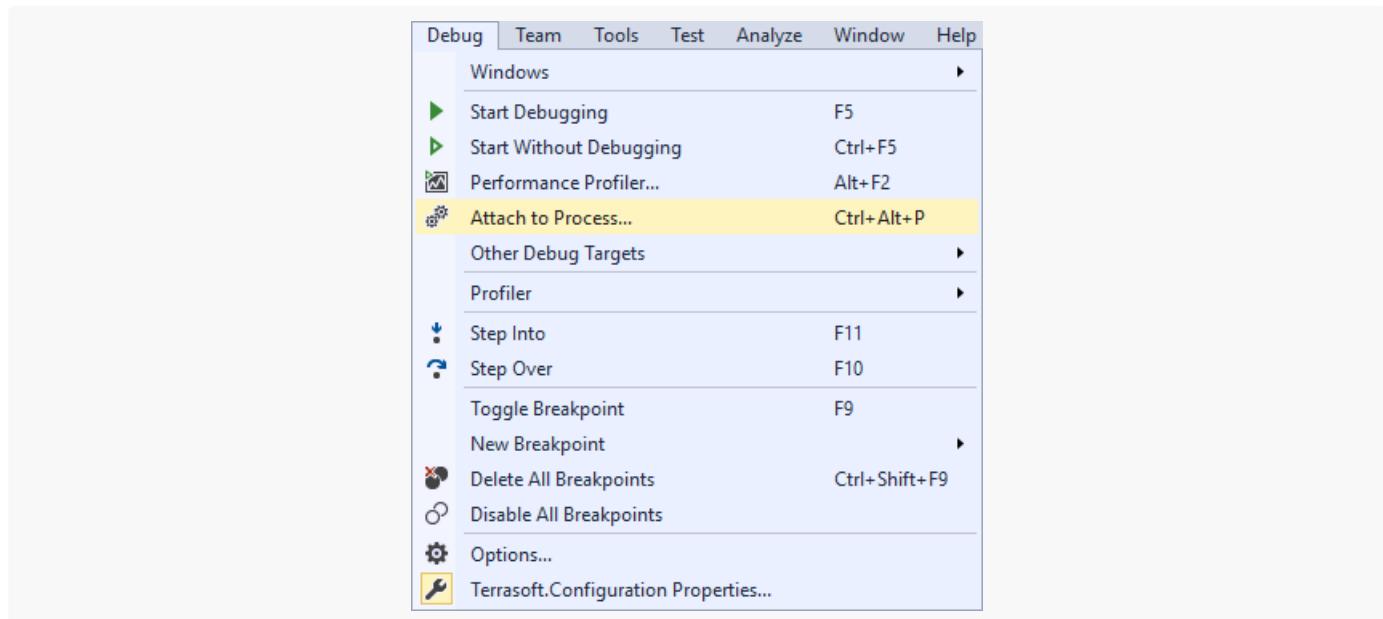
- После сохранения [скомпилируйте](#) измененный исходный код командой `Build Workspace` или `Rebuild Workspace`. В случае успешной компиляции разработанный веб-сервис становится доступным.

Адрес сервиса

`http://[URL приложения]/rest/UsrGreetingService>Hello`



- Для того чтобы начать **отладку**, присоединитесь к процессу сервера IIS, в котором запущено приложение. Для этого выполните команду меню `Debug —> Attach to process`.



В открывшемся окне в списке процессов выберите рабочий процесс IIS, в котором запущен пул приложения Creatio.

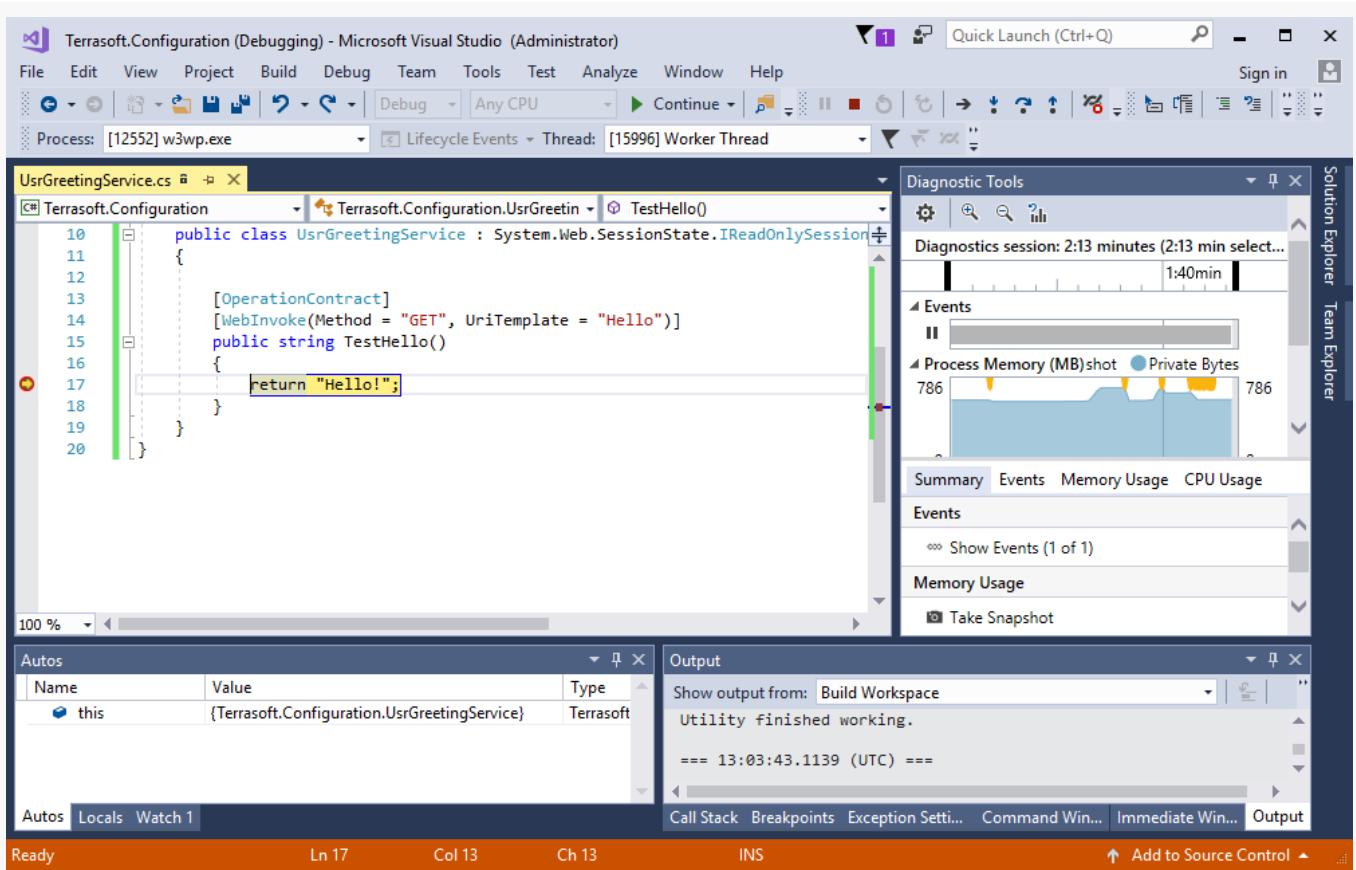
На заметку. Название рабочего процесса может различаться в зависимости от конфигурации используемого сервера IIS. Так, для полнофункционального IIS процесс называется `w3wp.exe`, для IIS Express — `iisexpress.exe`.

По умолчанию рабочий процесс IIS запущен под учетной записью, имя которой совпадает с именем папки приложения. Для того чтобы отобразить процессы всех пользователей, а не только текущего, необходимо установить признак [*Show processes from all users*].

После присоединения к рабочему процессу IIS выполните **повторную компиляцию**.

Далее можно приступить к процессу отладки средствами отладчика Visual Studio: можно установить точки останова, просматривать значения переменных, стек вызовов и т.д.

Например, после установки точки останова на строке возврата из метода `TestHello()`, повторной компиляции приложения и выполнения запроса к сервису отладчик остановит выполнение программы на точке останова.



Разработать C# код в пользовательском проекте

Сложный

Предварительные настройки

Для подключения библиотек классов Creatio, развертывания локальной базы данных из архивной копии, а также для работы с утилитой WorkspaceConsole рекомендуется использовать дистрибутив Creatio, распакованный на локальный диск компьютера разработчика. Во всех примерах этой статьи используется дистрибутив Creatio, распакованный в локальный каталог `C:\Creatio`.

Для примера работы с облачным приложением Creatio используется утилита Executor, находящаяся в каталоге `C:\Executor`. Скачать утилиту, настроенную для выполнения примера, можно по [ссылке](#).

Разработка C# кода для on-site приложения

1. Восстановить базу данных из резервной копии (при необходимости)

[Разверните](#) базу данных Creatio из резервной копии. Архивная копия базы данных приложения размещена в каталоге `C:\Creatio\db`.

2. Настроить утилиту WorkspaceConsole

Для работы с развернутой базой данных [настройте утилиту WorkspaceConsole](#), используя файлы приложения:

- Перейдите в каталог `C:\Creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole`.
- Запустите один из пакетных файлов `PrepareWorkspaceConsole.x64.bat` или `PrepareWorkspaceConsole.x86.bat`, в зависимости от версии Windows.

На заметку. После отработки пакетного файла команд удостоверьтесь, что в каталог `Terrasoft.WebApp\DesktopBin\WorkspaceConsole` также скопировались файлы `SharpPlink-xxx.svnExe` и `SharpSvn-DB44-20-xxx.svnD11` из соответствующей папки (x64 или x86).

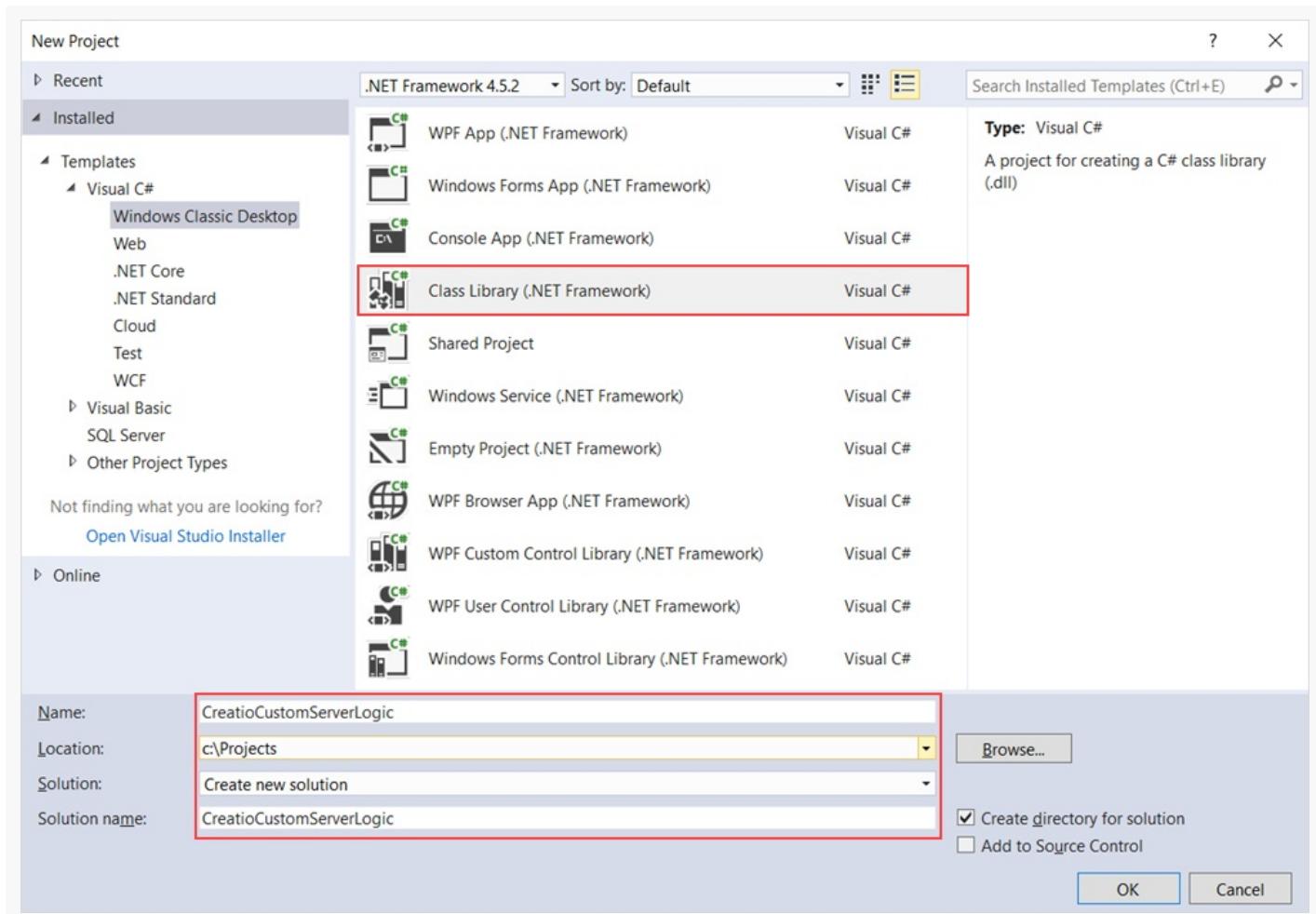
- Укажите параметры подключения к базе данных в файле `Terrasoft.Tools.WorkspaceConsole.exe.config`, размещенном в каталоге `C:\Creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole`). Например, если на сервере `dbserver` развернута база данных **CreatioDB**, то строка подключения будет иметь следующий вид:

Строка подключения базы данных **CreatioDB**

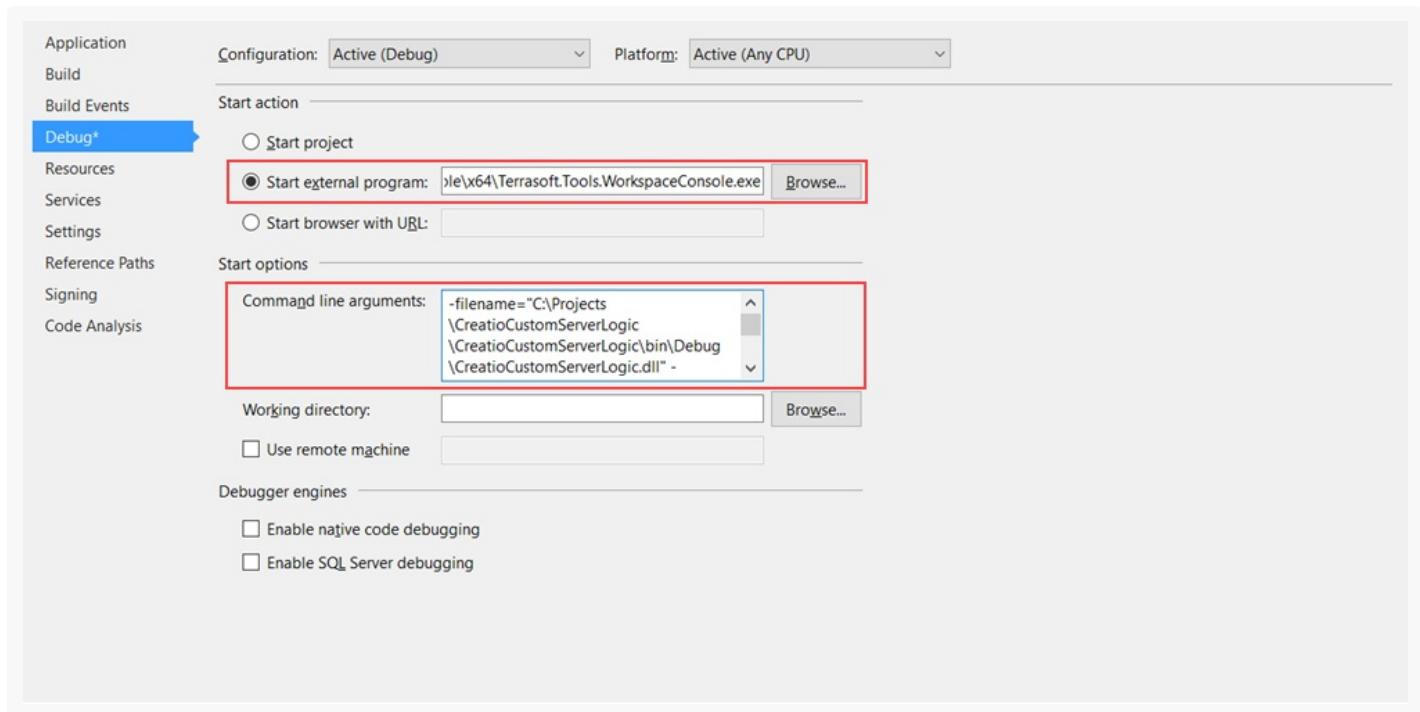
```
<connectionStrings>
  <add name="db" connectionString="Data Source=dbserver; Initial Catalog=CreatioDB; Persist Se
</connectionStrings>
```

3. Создать и настроить проект Visual Studio

Создайте в Visual Studio обычный проект библиотеки классов.



На вкладке [Debug] окна свойств созданного проекта библиотеки классов укажите в свойстве [Start external program] полный путь к настроенной утилите WorkspaceConsole. Утилита WorkspaceConsole используется как внешнее приложение для отладки разрабатываемой программной логики.



В свойстве [*Command line arguments*] укажите следующие [параметры запуска WorkspaceConsole](#):

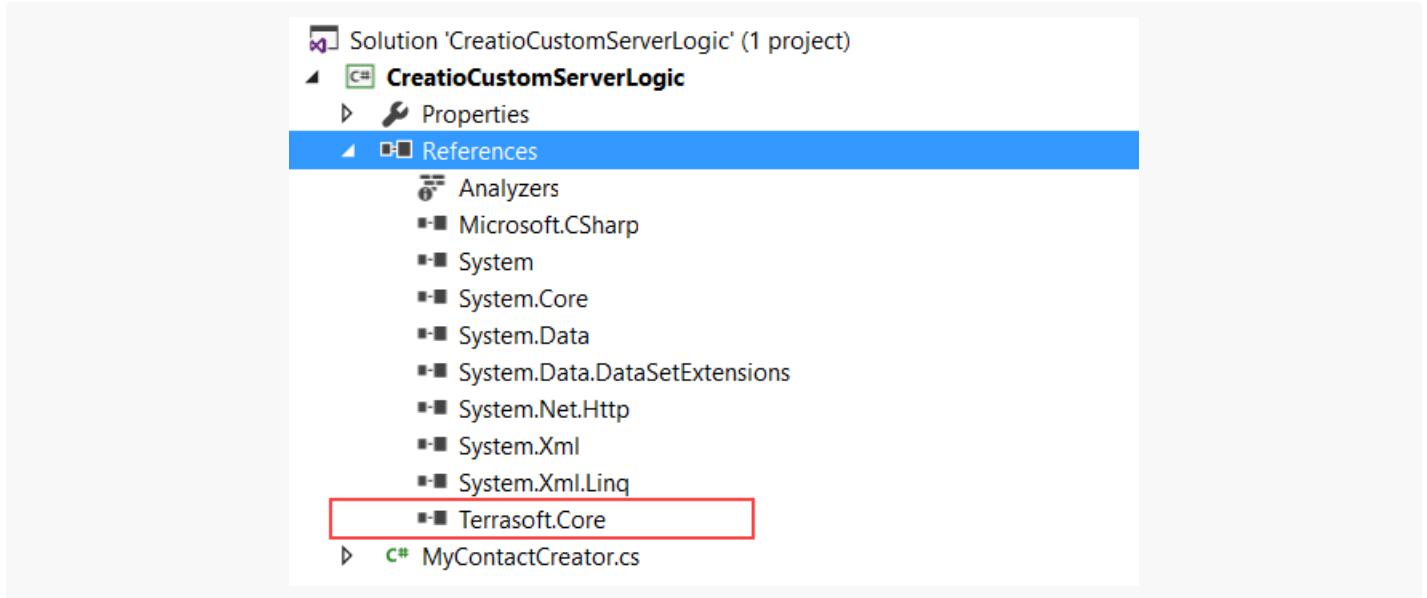
- `-filename` — полный путь к отладочной версии разрабатываемой библиотеки классов ("C:\Projects\CreatioCustomServerLogic\CreatioCustomServerLogic\bin\Debug\CreatioCustomServerLogic.dll");
- `-typeName` — полное имя класса, в котором реализуется разрабатываемая программная логика, включая названия всех пространств имен ("CreatioCustomServerLogic.MyContactCreator");
- `-operation` — операция WorkspaceConsole ("ExecuteScript");
- `-workspaceName` — название рабочего пространства ("Default");
- `-confRuntimeParentDirectory` — путь к [родительскому каталогу](#) для директории `conf` ("C:\creatio\Terrasoft.WebApp").

Пример строки аргументов запуска WorkspaceConsole

```
-filename="C:\Projects\CreatioCustomServerLogic\CreatioCustomServerLogic\bin\Debug\CreatioCustomServerLogic.dll" -typeName="MyContactCreator" -operation="ExecuteScript" -workspaceName="Default" -confRuntimeParentDirectory="C:\creatio\Terrasoft.WebApp"
```

Важно. В свойствах проектов Visual Studio, работающих с версией приложения 7.11.0 и выше, нужно указывать версию .NET Framework 4.7 (вкладка [*Application*], свойство [*Target framework*]).

Для работы с классами серверной части ядра Creatio в созданном проекте установите зависимости от нужных библиотек классов Creatio. Например, добавьте зависимость от библиотеки `Terrasoft.Core.dll`.



Библиотеки классов пространства имен `Terrasoft` можно найти в каталоге

`Terrasoft.WebApp\DesktopBin\WorkspaceConsole` дистрибутива приложения.

На заметку. Библиотеки классов копируются в каталог `Terrasoft.WebApp\DesktopBin\WorkspaceConsole` во время [выполнения пакетных файлов](#).

4. Выполнить разработку функциональности

В созданный проект библиотеки классов добавьте класс, полное название которого должно совпадать с названием, указанным в аргументе `-typeName` строки аргументов запуска утилиты `WorkspaceConsole` ("`CreatioCustomServerLogic.MyContactCreator`"). Класс должен реализовывать интерфейс `Terrasoft.Core.IExecutor`.

MyContactCreator.cs

```
using System;
using Terrasoft.Core;

namespace CreatioCustomServerLogic
{
    public class MyContactCreator : IExecutor
    {
        public void Execute(UserConnection userConnection)
        {
            // Получение экземпляра схемы [Контакты].
            var schema = userConnection.EntitySchemaManager.GetInstanceByName("Contact");
            var length = 10;
            for (int i = 0; i < length; i++)
            {

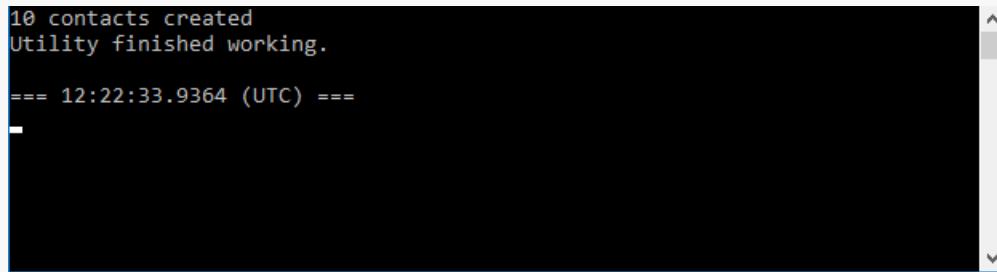
```

```

// Создание нового контакта.
var entity = schema.CreateEntity(userConnection);
// Установка свойств контакта.
entity.SetColumnValue("Name", string.Format("Name {0}", i));
entity.SetDefColumnValues();
// Сохранение контакта в базу данных.
entity.Save(false);
}
// Вывод сообщения в консоль.
Console.WriteLine($"{length} contacts created");
}
}
}

```

После запуска проекта на выполнение (клавиша F5) появится окно утилиты WorkspaceConsole с соответствующим сообщением.



```

10 contacts created
Utility finished working.

== 12:22:33.9364 (UTC) ==
-
```

Также можно установить точку останова на любой строке исходного кода и во время выполнения программы посмотреть текущие значения переменных, т. е. выполнить отладку.

Результат выполнения приведенного выше программного кода можно посмотреть в разделе [Контакты] приложения Creatio, либо выполнив запрос в базу данных.

Добавленные контакты

Name	Last Activity
Name 0	3/30/2020 6:16 PM
Name 1	3/30/2020 6:16 PM
Name 2	3/30/2020 6:16 PM
Name 3	3/30/2020 6:16 PM
Name 4	3/30/2020 6:16 PM
Name 5	3/30/2020 6:16 PM
Name 6	3/30/2020 6:17 PM
Name 7	3/30/2020 6:17 PM

Запрос к таблице контактов базы данных

```

1 select Name from Contact
2 where Name like 'Name%'

```

Results

	Name
1	Name 0
2	Name 1
3	Name 2
4	Name 3
5	Name 4
6	Name 5
7	Name 6

Разработка C# кода для cloud приложения

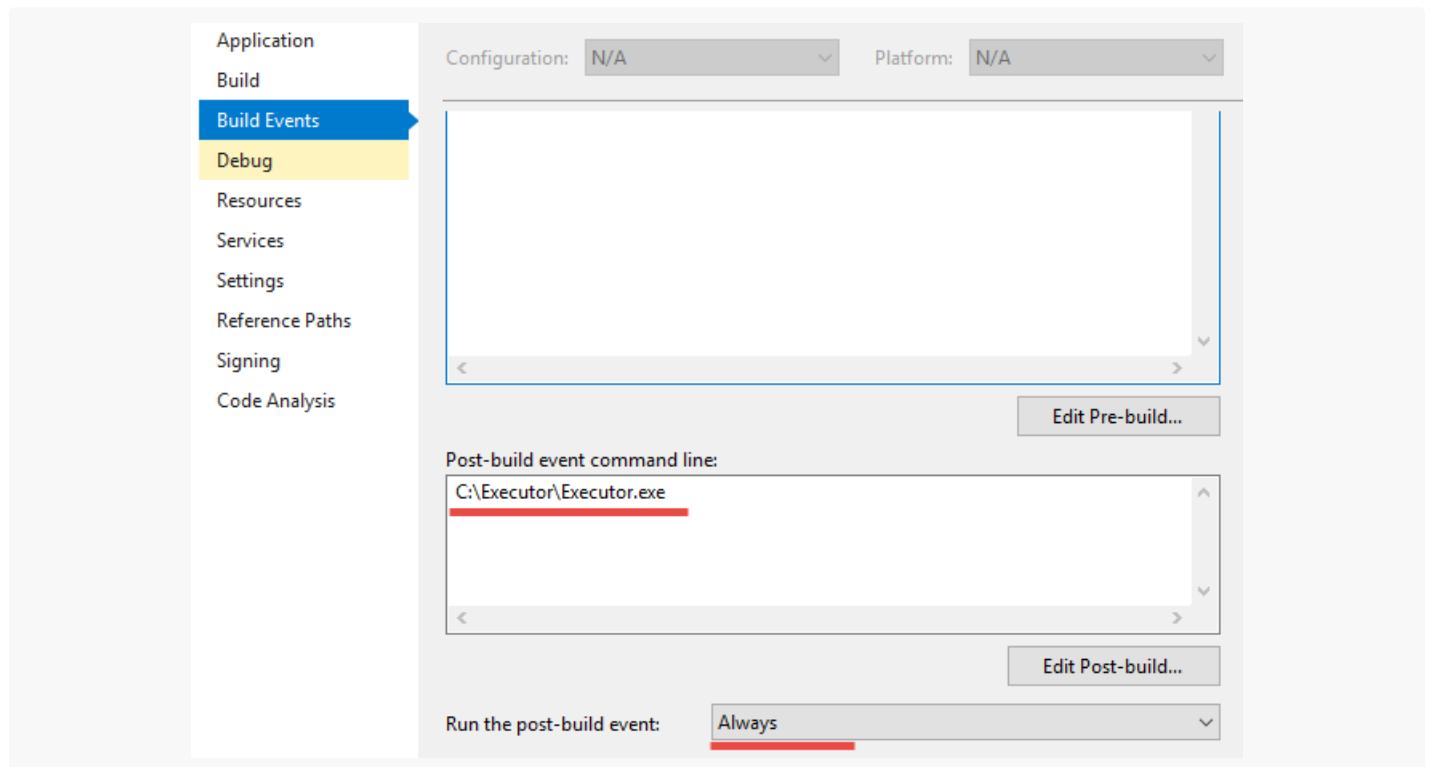
1. Создать проект библиотеки классов

Создайте в Visual Studio обычный проект библиотеки классов. В качестве имени проекта укажите, например, значение "CreatioCustomServerLogic.Cloud".

Для работы с классами серверной части ядра Creatio в созданном проекте установите зависимости от нужных библиотек классов Creatio. Например, добавить зависимость от библиотеки `Terrasoft.Core.dll`.

На вкладке [BuildEvents] окна свойств созданного проекта библиотеки классов укажите в свойстве [Post-build event command line] полный путь к настроенной утилите Executor ("C:\Executor\Executor.exe") и выберите условие запуска события сборки библиотеки ("Always").

Свойства вкладки [Build Events]



2. Выполнить разработку функциональности

В созданный проект библиотеки классов добавьте класс, который должен реализовывать интерфейс `Terrasoft.Core.IExecutor`.

MyContactReader.cs

```
using System;
using System.Web;
using Terrasoft.Core;
using Terrasoft.Core.Entities;

namespace CreatioCustomServerLogic.Cloud
{
    public class MyContactReader : IExecutor
    {
        public void Execute(UserConnection userConnection)
        {
            // Получение экземпляра схемы [Контакты].
            var entitySchema = userConnection.EntitySchemaManager.GetInstanceByName("Contact");
            // Создание экземпляра класса запроса.
            var esq = new EntitySchemaQuery(entitySchema);
            // Добавление в запрос всех колонок схемы.
            esq.AddAllSchemaColumns();
            // Получение коллекции записей раздела [Контакты].
            var collection = esq.GetEntityCollection(userConnection);
        }
    }
}
```

```
        foreach (var entity in collection)
    {
        // Вывод в http-ответ запроса от утилиты Executor необходимых значений.
        HttpContext.Current.Response.Write(entity.GetTypedColumnValue<string>("Name"));
        HttpContext.Current.Response.Write(Environment.NewLine);
    }
}
```

3. Настроить утилиту Executor

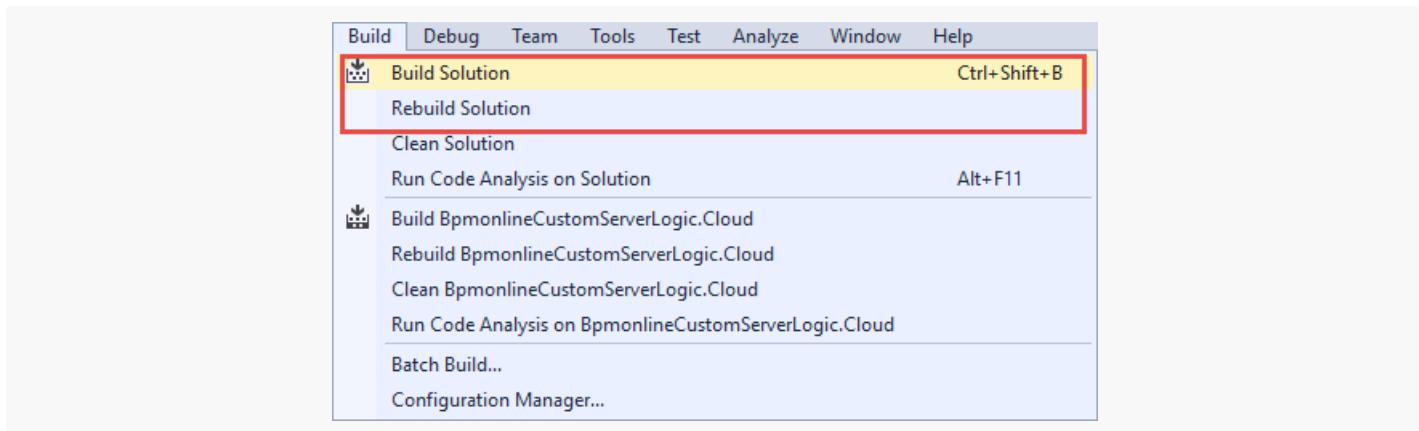
На заметку. Скачать утилиту, настроенную для выполнения этого примера, можно по [ссылке](#).

Перейдите в каталог с установленной утилитой Executor (`C:\Executor`). Затем в конфигурационном файле укажите значения для следующих элементов настройки:

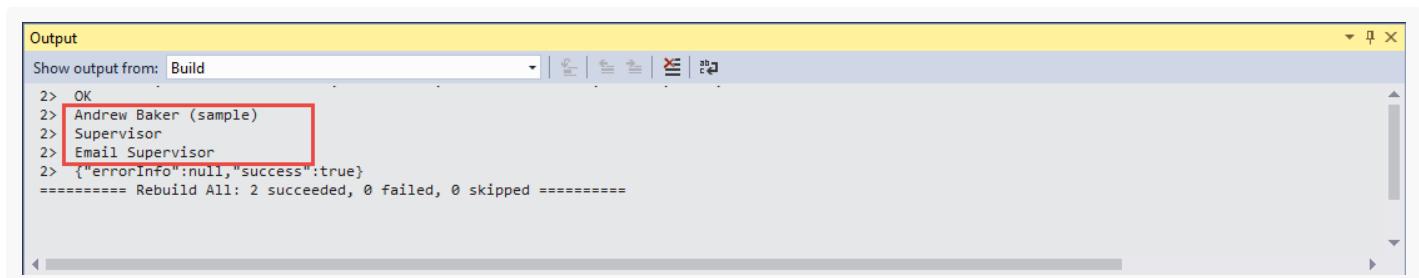
- `Loader` — URL загрузчика приложения Creatio. Как правило, это URL сайта Creatio, например "<https://mycloudapp.creatio.com>".
 - `WebApp` — URL приложения Creatio. Как правило, это путь к конфигурации по умолчанию приложения Creatio, например "<https://mycloudapp.creatio.com/0>".
 - `Login` — имя пользователя Creatio, например, "Supervisor".
 - `Password` — пароль пользователя Creatio.
 - `LibraryOriginalPath` — путь к исходной копии библиотеки классов. Как правило, это путь, по которому создается библиотека классов после компиляции в Visual Studio, например, "C:\Projects\CreatioCustomServerLogic\CreatioCustomServerLogic.Cloud\bin\Debug\CreatioCustomServerLogic.Cloud.dll".
 - `LibraryCopyPath` — путь, по которому будет создана копия библиотеки классов для работы с удаленным сервером. Это может быть любая временная папка или каталог, содержащий утилиту Executor, например, "C:\Executor\CreatioCustomServerLogic.Cloud.dll".
 - `LibraryType` — полное имя класса, в котором реализуется разрабатываемая программная логика, включая названия всех пространств имен. Например, "CreatioCustomServerLogic.Cloud.MyContactReader".
 - `LibraryName` — название библиотеки классов, например, "CreatioCustomServerLogic.Cloud.dll".

4. Выполнить разработанный программный код

Для запуска процесса сборки воспользуйтесь командами меню [*Build Solution*] и [*Rebuild Solution*].



Результат выполнения разработанного программного кода можно увидеть в окне [*Output*] Visual Studio после каждой успешной сборки библиотеки классов.



Разработать клиентский код



Для удобства разработчика предусмотрена возможность выгрузки исходного кода клиентских схем из базы данных в `*.js`-файлы и LESS-стилей модулей в `*.less`-файлы для работы с ними в интегрированной среде разработки (Integrated Development Environment, IDE), например, WebStorm, Visual Studio Code, Sublime Text и т.п.

Последовательность действий при разработке исходного хода клиентских схем в файловой системе

1. Выполнить предварительные настройки приложения Creatio

[Настроить](#) приложение для разработки в файловой системе.

2. Создать, получить или обновить пакет из репозитория SVN

Создать пользовательский пакет [с использованием](#) и [без использования SVN](#). [Установить](#) и, при необходимости, [обновить](#) пакет из системы контроля версий.

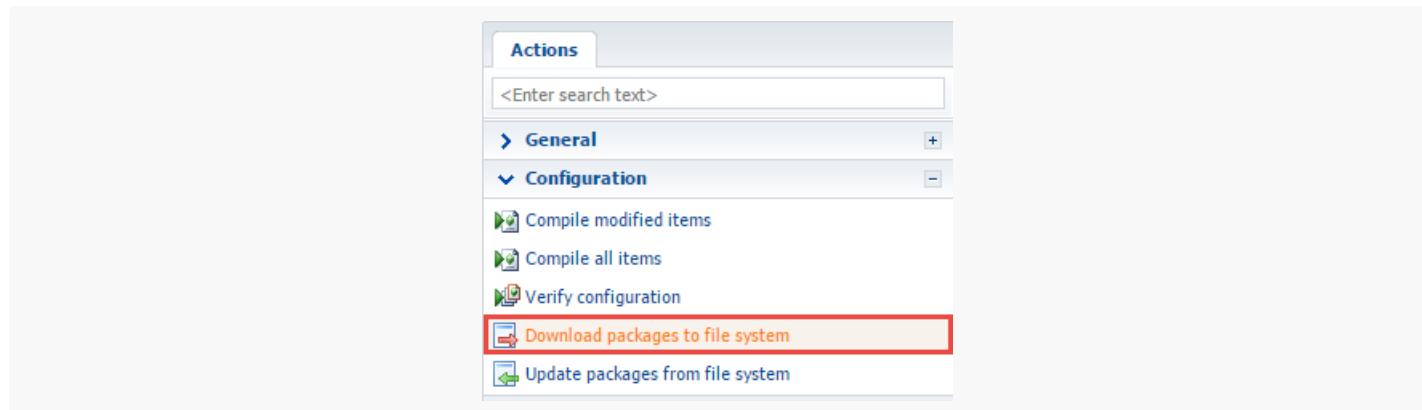
На заметку. При разработке в файловой системе вместо встроенных возможностей Creatio удобнее использовать клиентские приложения для работы с хранилищами систем контроля версий, например, [Tortoise SVN](#) или [Git](#).

3. Создать клиентскую схему, в которой будет выполняться разработка

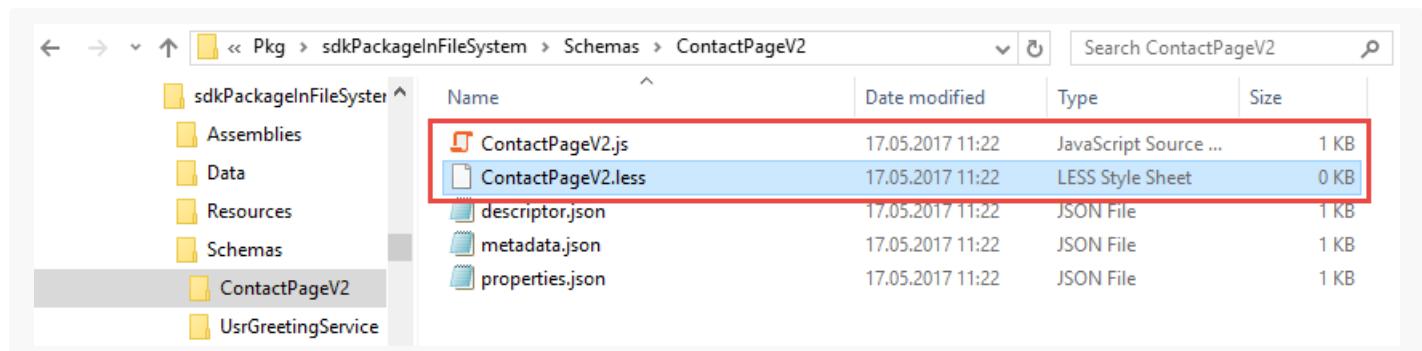
[Создать клиентскую схему](#) в пользовательском пакете.

4. Выгрузить схему из базы данных в файловую систему

Для этого нужно в разделе [Конфигурация] ([Configuration]) выполнить действие [Выгрузить пакеты в файловую систему] ([Download packages to file system]).



Так, например, если в пользовательском пакете `sdkPackageInFileSystem` была создана замещающая схема [Схема отображения карточки контакта] ([Display schema - Contact card]) с именем `ContactPageV2`, то в файловой системе в каталоге `Pkg\sdkPackageInFileSystem\schemas>ContactPageV2` появятся файлы исходного кода схемы `ContactPageV2.js` и стилем `ContactPageV2.less`.



5. Выполнить разработку исходного кода схемы в IDE.

Для выполнения разработки необходимо открыть файл с исходным кодом схемы в предпочтаемой IDE (или любом текстовом редакторе) и добавить нужный исходный код.

```

define("ContactPageV2", [],
    function() {
        return {
            entitySchemaName: "Contact",
            diff: /**SCHEMA_DIFF*/[
                {
                    "operation": "remove",
                    "name": "JobTitleProfile"
                }
            ]/**SCHEMA_DIFF*/
        };
    });

```

The screenshot shows the Visual Studio Code interface with the title bar "ContactPageV2.js - Terrasoft.Configuration - Visual Studio Code". The left sidebar is the Explorer view, showing a tree structure of files and folders. Under "TERRASOFT.CONFIGURATION", there is a "ContactPageV2" folder containing "ContactPageV2.js", "ContactPageV2.less", "descriptor.json", "metadata.json", and "properties.json". The "ContactPageV2.js" file is selected and shown in the main editor area. The code defines a schema named "Contact" with a single entry in the "diff" array that removes the "JobTitleProfile" field.

Например, чтобы скрыть со страницы контакта поле [Полное название должности] ([*Full job title*]), в файл `ContactPageV2.js` нужно добавить следующий исходный код:

ContactPageV2.js

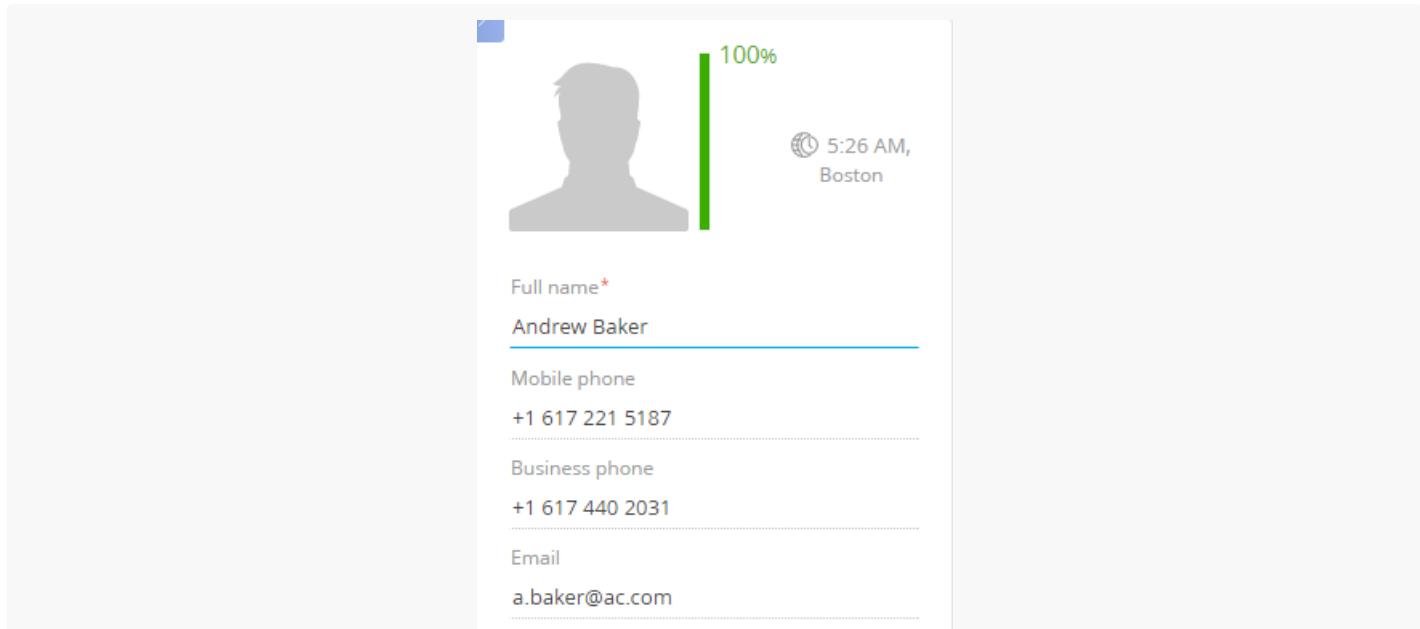
```

define("ContactPageV2", [],
    function() {
        return {
            entitySchemaName: "Contact",
            diff: /**SCHEMA_DIFF*/[
                {
                    "operation": "remove",
                    "name": "JobTitleProfile"
                }
            ]/**SCHEMA_DIFF*/
        };
    });

```

6. Сохранить схему и выполнить отладку созданного исходного кода

После сохранения файла `ContactPageV2.js` и обновления страницы браузера поле [Полное название должности] ([*Full job title*]) будет скрыто со страницы контакта.



Если при редактировании клиентского исходного кода были допущены ошибки, необходимо выполнить его [отладку](#).

Важно. Чтобы вернуться к разработке с помощью встроенных средств Creatio, необходимо:

1. Выполнить действие [Обновить пакеты из файловой системы] ([*Update packages from file system*]).
2. [Выключить](#) режим разработки в файловой системе, установив значение атрибута `enabled="false"` элемента `fileDesignMode` конфигурационного файла `Web.config` .

Автоматическое отображение изменений клиентского кода

При разработке клиентского кода в файловой системе после внесения изменений в исходный код схемы необходимо каждый раз обновлять страницу браузера, на которой открыто приложение. Это существенно снижает производительность разработки.

Для устранения этого была разработана функциональность автоматической перезагрузки страницы браузера после внесения изменений в исходный код. Она работает следующим образом.

При старте приложения создается объект, отслеживающий изменения `*.js`-файла с исходным кодом разрабатываемого модуля в файловой системе. Если изменения произошли, то отправляется сообщение в клиентское приложение (Creatio). В клиентском приложении специальный объект, подписанный на это сообщение, определяет зависимые объекты измененного модуля, разрушает их, регистрирует новые пути к модулям и пытается заново загрузить измененный модуль. Это приводит к тому, что все проинициализированные модули запрашиваются браузером по новым путям и загружают изменения из файловой системы. При этом не тратится время на интерпретацию и загрузку других модулей. Отдельная страница разработки позволяет не загружать ряд вспомогательных модулей, например, левую и правую панели, панель уведомлений и т. д. Это приводит к уменьшению количества запросов на сервер.

Также такой подход к разработке отдельных модулей очень хорошо выявляет связанность модулей, позволяя вовремя обнаружить ненужные зависимости и избавиться от них.

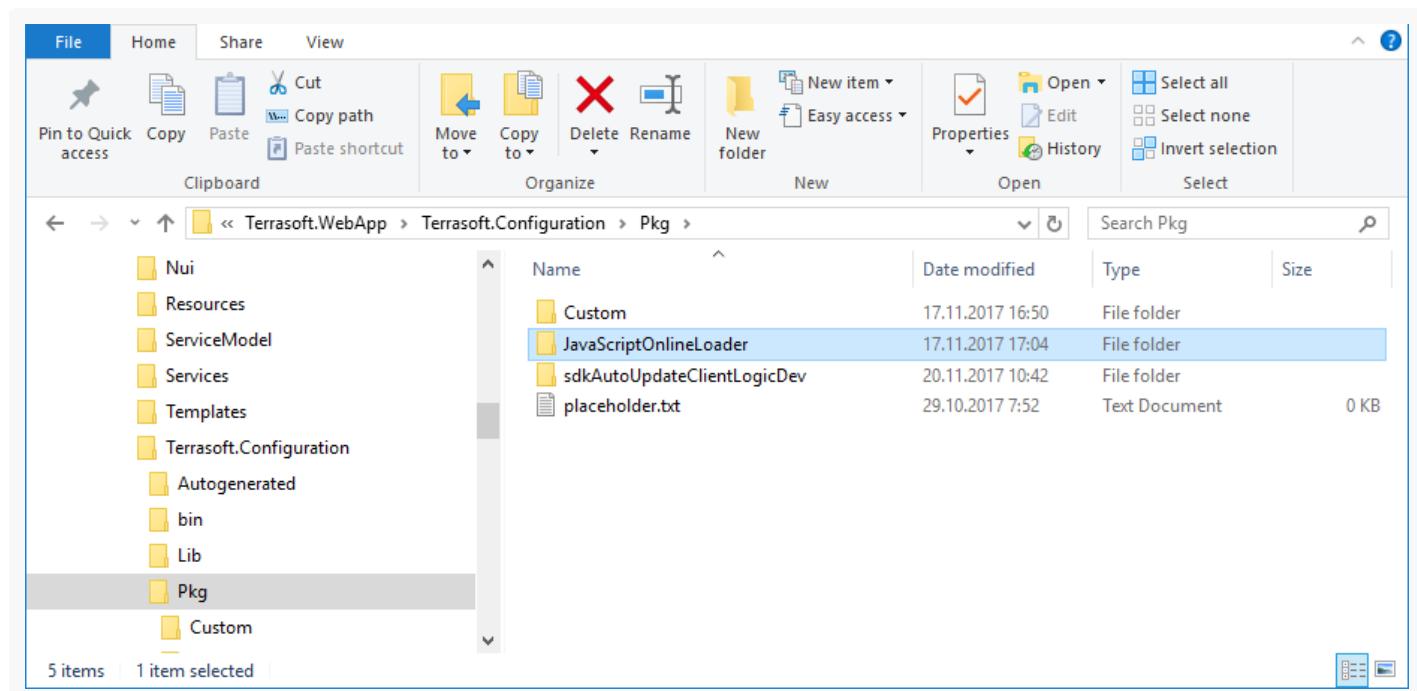
Известные проблемы

- Если в исходном коде модуля допущена синтаксическая ошибка, то автоматическое обновление страницы не произойдет. Потребуется ее принудительное обновление (например, клавишей F5). При исправлении ошибки страница вернется к работоспособному состоянию.
- Не все модули Creatio могут загружаться отдельно. Основная причина — эффект сильной [связанности модулей](#).

Последовательность настройки автоматического отображения изменений

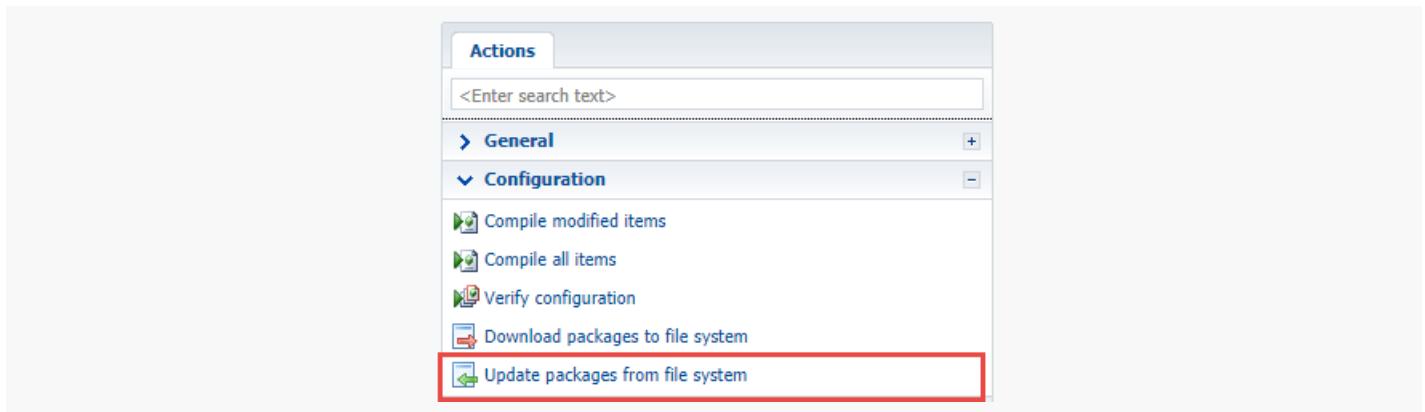
1. Установить пакет JavaScriptOnlineLoader

При включенном режиме разработки в файловой системе необходимо добавить каталог `JavaScriptOnlineLoader`, содержащий нужный пакет, в каталог `[Путь к установленному приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Pkg`.

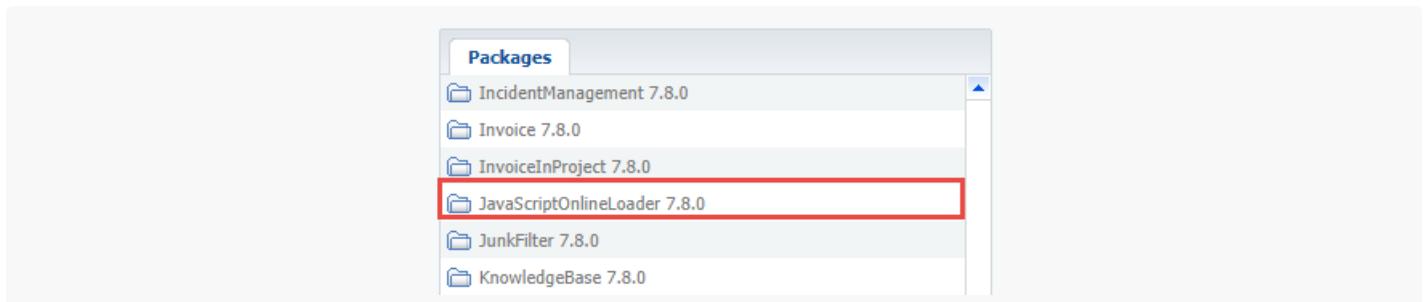


Важно. Пакет доступен на [GitHub](#). Также архив с пакетом можно скачать по [ссылке](#).

Затем нужно загрузить пакет в конфигурацию, выполнив действие [*Обновить пакеты из файловой системы*] ([*Update packages from file system*]).



В результате пакет отобразится на вкладке [Пакеты]([Packages]).



2. Открыть в браузере страницу разрабатываемого модуля

Для этого необходимо открыть страницу `ViewModule.aspx`, добавив к ней параметр.

Формат параметра для открытия страницы `ViewModule.aspx`

```
?vm=DevViewModule#CardModuleV2/<Название модуля>
```

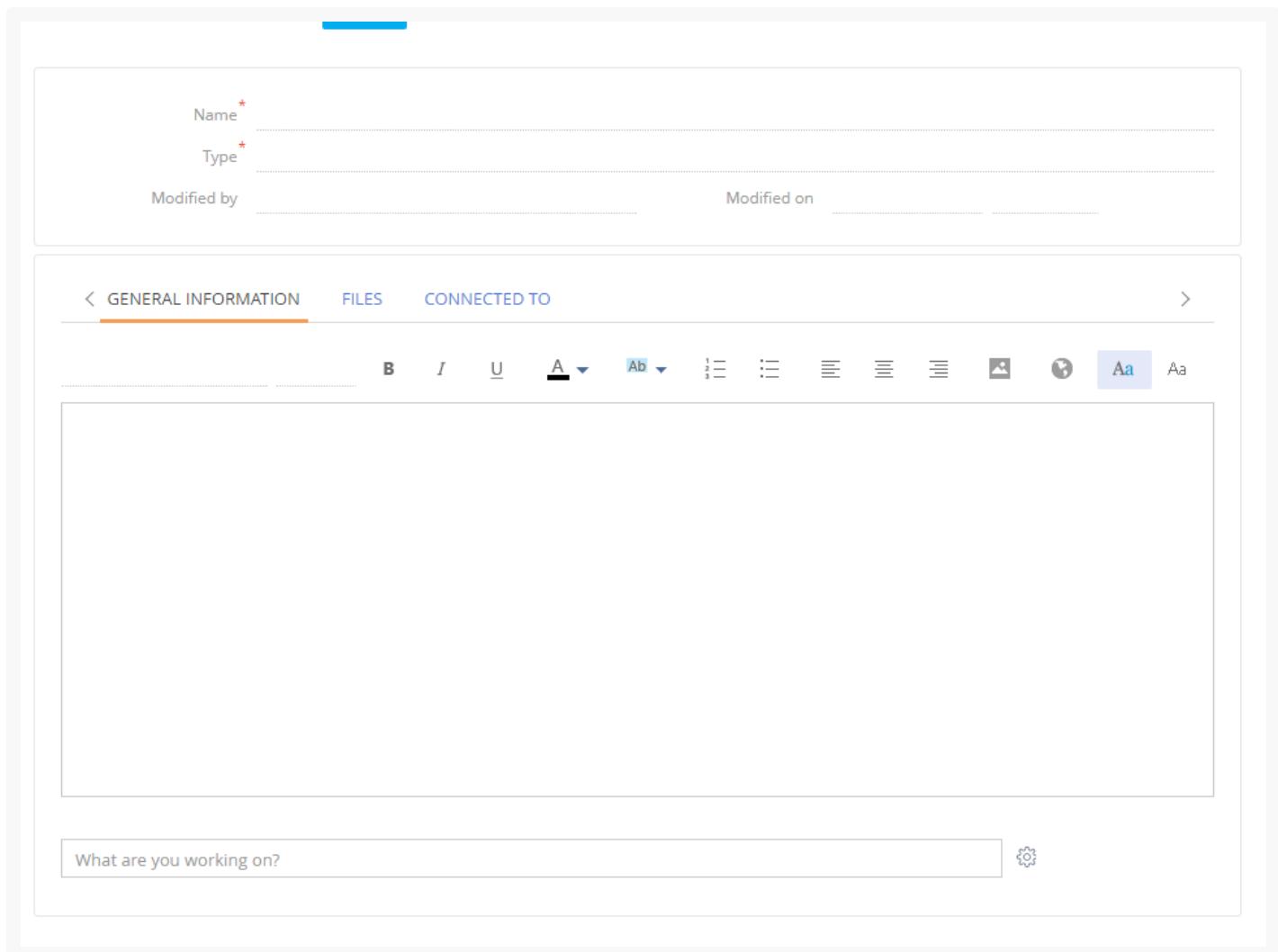
Например, в пользовательский пакет добавлена замещенная схема `KnowledgeBasePageV2` — схема страницы записи раздела [База знаний].

URL страницы `KnowledgeBasePageV2` с функциональностью автоматического отображения из..

```
http://localhost/creatio/0/Nui/ViewModule.aspx?vm=DevViewModule#CardModuleV2/KnowledgeBasePageV2
```

Здесь `http://localhost/creatio` — URL приложения Creatio, развернутого локально.

После перехода по этому URL, отобразится страница `ViewModule.aspx` с загруженным модулем.



3. Изменить исходный код разрабатываемой схемы

Изменить исходный код разрабатываемой схемы можно в любом текстовом редакторе, например, в Блокноте. После сохранения изменений открытая в браузере страница будет автоматически обновлена.

Например, в пользовательский пакет `sdkAutoUpdateClientLogicDev` добавлена замещенная схема `KnowledgeBasePageV2` — схема страницы записи раздела [База знаний]. После выгрузки в файловую систему исходный код схемы будет доступен в каталоге `..\Pkg\sdkAutoUpdateClientLogicDev\Tables\KnowledgeBasePageV2`.

Если в файл `KnowledgeBasePageV2.js` добавить исходный код, приведенный ниже, и сохранить файл, то выполнится автоматическое обновление страницы в браузере. Результат изменений будет отображен сразу же.

```
KnowledgeBasePageV2.js
```

```
define("KnowledgeBasePageV2", [], function() { return { entitySchemaName: "KnowledgeBase", diff:
```

Страница с изменениями

Общие принципы работы с пакетами



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

Типы пакетов:

- 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.

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.

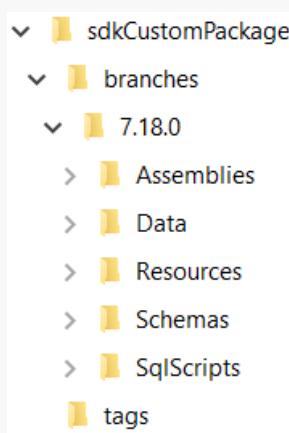
- Изменения выполняются при помощи мастеров или вручную, при этом объем изменений небольшой.
- Нет необходимости использовать систему контроля версий.

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.

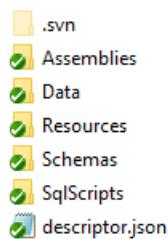
Рабочая копия пакета сохраняется локально в файловой системе. Путь для хранения пакетов задается в конфигурационном файле `ConnectionStrings.config` в атрибуте `connectionString` элемента `defPackagesWorkingCopyPath`.

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 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

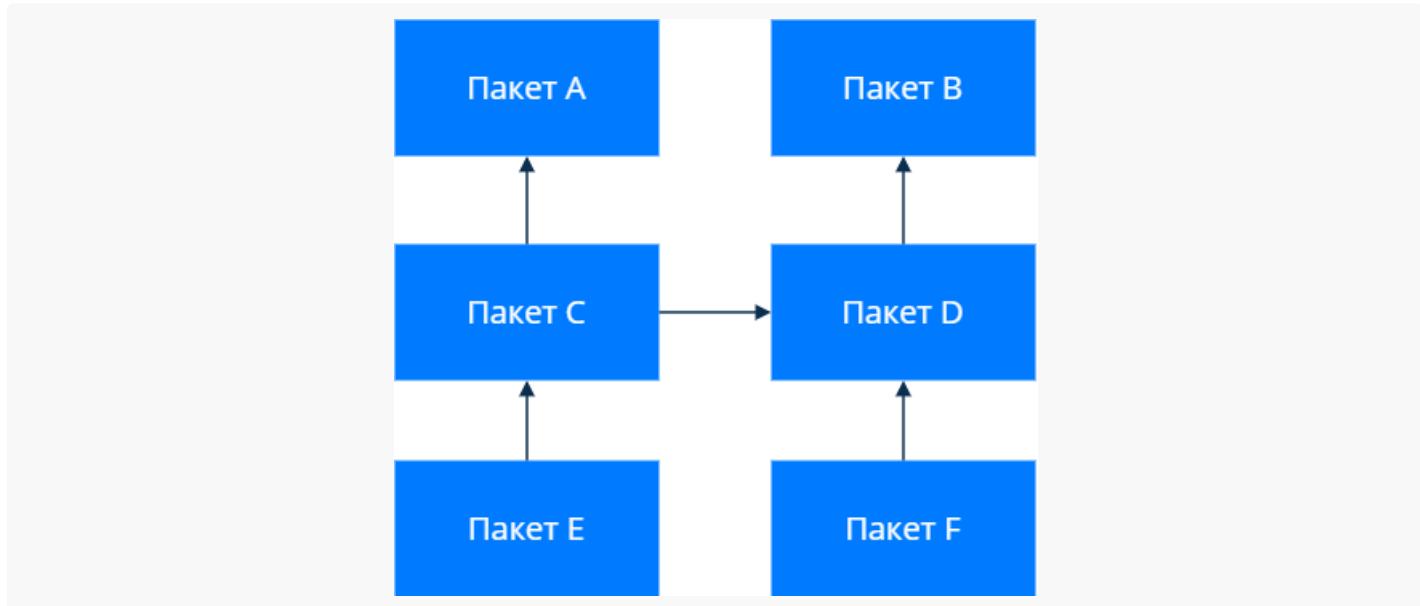
Разработка приложения Creatio базируется на основных принципах проектирования программного обеспечения, в частности, **принципа отсутствия повторений (DRY)**.

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 использована.

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 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.

Иерархия пакетов приложения

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

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

Configuration

CLOSE COMPILE ▾ ACTIONS ▾

Search by package

- All packages
- Custom •
- TryItPackage •

FILE SYSTEM DEVELOPMENT MODE

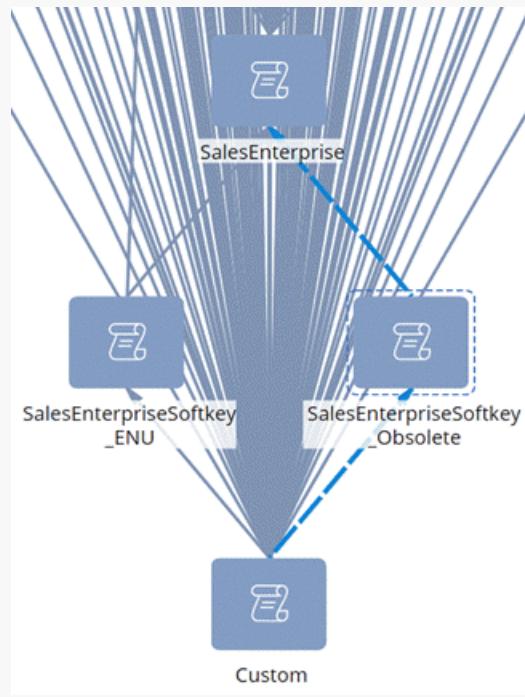
Download packages to file system
Update packages from file system

PACKAGES

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_Obslete] зависит только от пакета [SalesEnterprise] и всех его родительских пакетов. Также пакет [SalesEnterpriseSoftkey_Obslete] является родительским для пакета [Custom].



Добавление зависимостей пакета

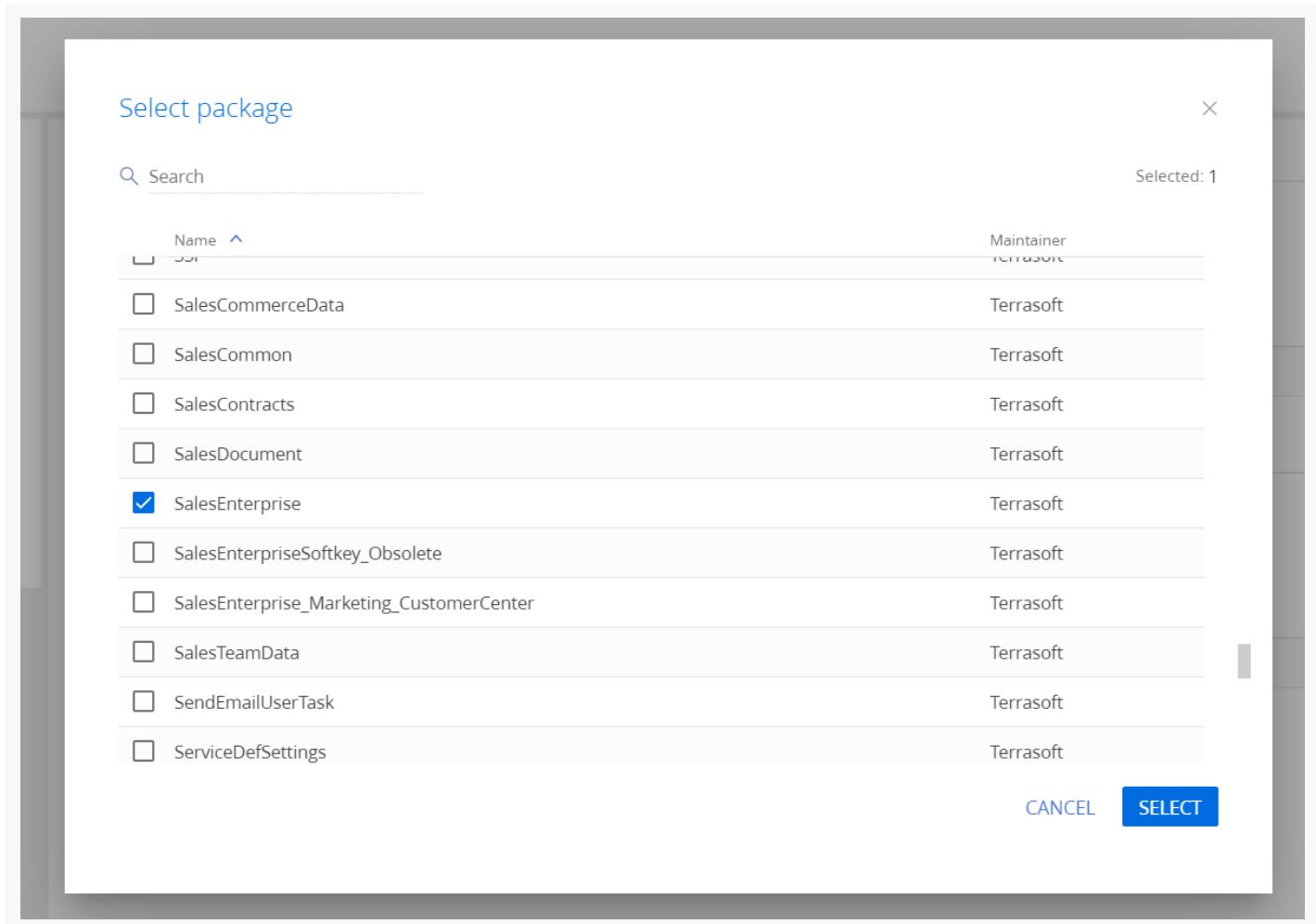
Зависимости можно добавить в пользовательский пакет при создании пакета или уже после него.

Чтобы **добавить зависимости**:

- Перейдите на страницу пакета.
- На вкладке [Зависимости] ([Dependencies]) на детали [Зависит от пакетов] ([Depends on packages])

нажмите кнопку [Добавить] ([Add]).

3. 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	DEPENDENCIES	SYSTEM INFORMATION
Repository address http://tscore-svn:8050/svn/tsfmdoc/SDKP...	Depends on Packages ⓘ	
Repository version SDKPackages	<input type="text"/> Search by package	
Package version 1.0.0	Name ⚡	SalesEnterpriseSoftkey_ENU
Maintainer Customer ⓘ	+ Add	
Description	<input type="text"/> Search by package	
	Name ⚡	Custom

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

Package properties

CLOSE ACTIONS ▾

Name TestPackage1	DEPENDENCIES	SYSTEM INFORMATION
Repository address http://tscore-svn:8050/svn/tsfmdoc/SDKP...	Depends on Packages ⓘ	
Repository version SDKPackages	<input type="text"/> Search by package Name ^ SalesEnterpriseSoftkey_ENU + Add	
Package version 1.0.0	Dependent Packages ⓘ	
Maintainer Customer ⓘ	<input type="text"/> Search by package Name ^ Custom	
Description		

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": {
    "UIId": "51b3ed42-678c-4da3-bd16-8596b95c0546",
    "PackageVersion": "7.18.0",
    "Name": "UsrDependentPackage",
    "ModifiedOnUtc": "\/Date(1522653150000)\/",
    "Dependencies": [
      {
        "Name": "SalesEnterpriseSoftkey_ENU"
      }
    ]
  }
}
```

```

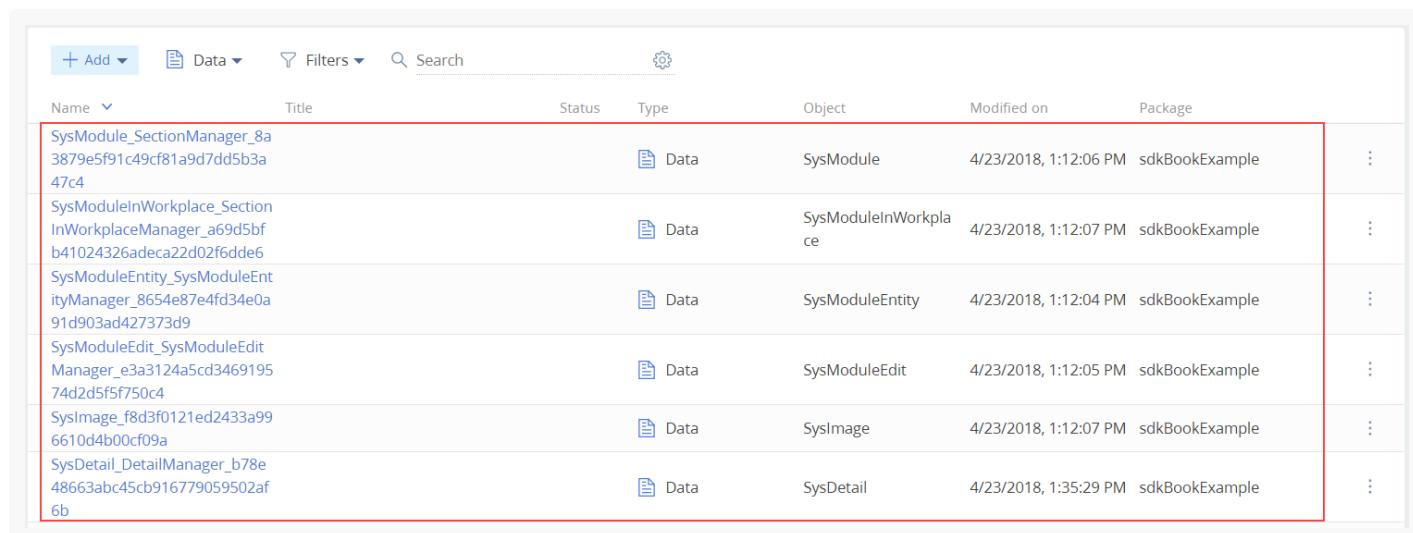
    "Maintainer": "Customer",
    "DependsOn": [
    {
        "UIId": "e14dcfb1-e53c-4439-a876-af7f97083ed9",
        "PackageVersion": "7.18.0",
        "Name": "SalesEnterprise"
    }
]
}
}

```

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.



Name	Title	Status	Type	Object	Modified on	Package	⋮
SysModule_SectionManager_8a 3879ef91c49cf81a9d7dd5b3a 47c4			Data	SysModule	4/23/2018, 1:12:06 PM	sdkBookExample	⋮
SysModuleInWorkplace_Section InWorkplaceManager_a69d5bf b4102432badeca22d02fdde6			Data	SysModuleInWorkpla ce	4/23/2018, 1:12:07 PM	sdkBookExample	⋮
SysModuleEntity_SysModuleEnt ityManager_8654e87e4fd34e0a 91d903ad427373d9			Data	SysModuleEntity	4/23/2018, 1:12:04 PM	sdkBookExample	⋮
SysModuleEdit_SysModuleEdit Manager_e3a3124a5cd3469195 74d2d5f5f750c4			Data	SysModuleEdit	4/23/2018, 1:12:05 PM	sdkBookExample	⋮
SysImage_f8d3f0121ed2433a99 6610d4b00cf09a			Data	SysImage	4/23/2018, 1:12:07 PM	sdkBookExample	⋮
SysDetail_DetailManager_b78e 48663abc45cb916779059502af 6b			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

- Перейдите в дизайнер системы по кнопке .
- 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

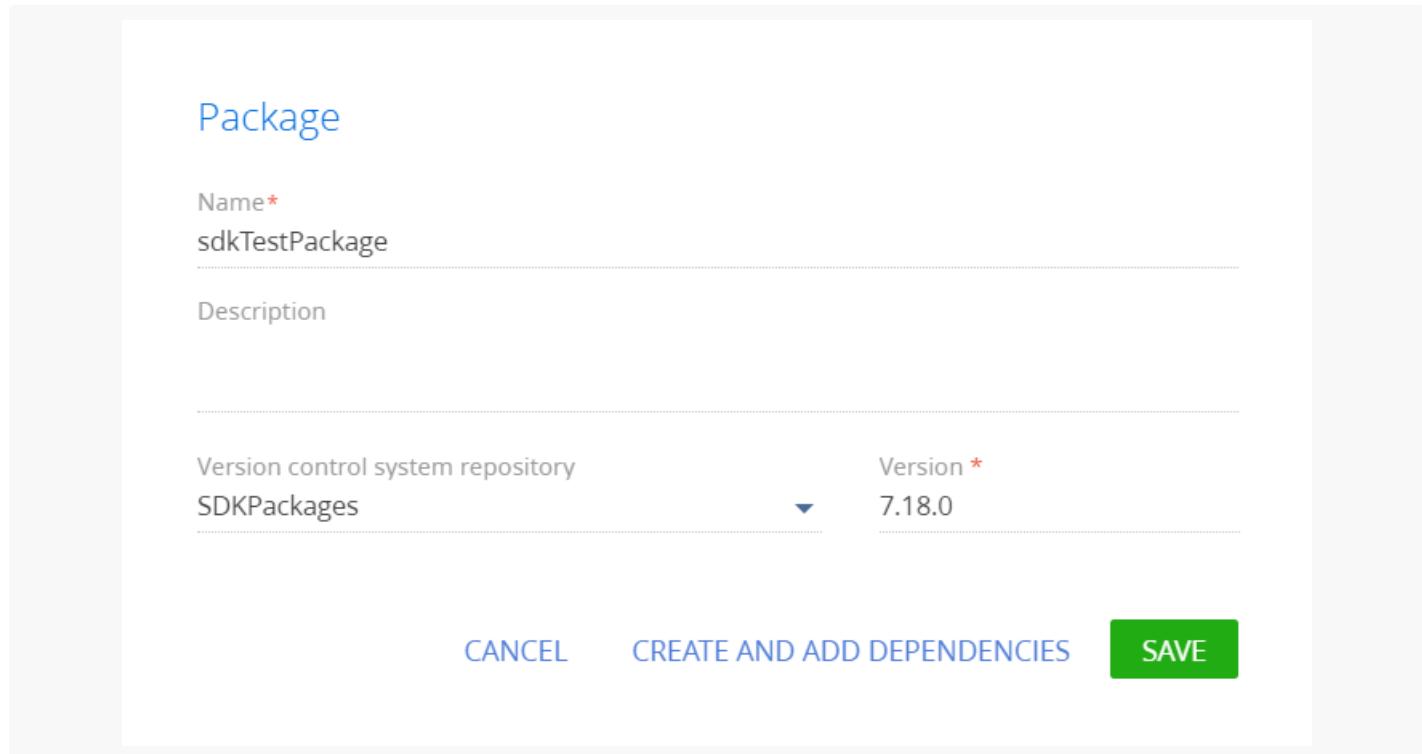
При нажатии на кнопку будет отображена карточка пакета, в которой необходимо заполнить свойства пакета.

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": {
    "UIId": "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*].

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



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

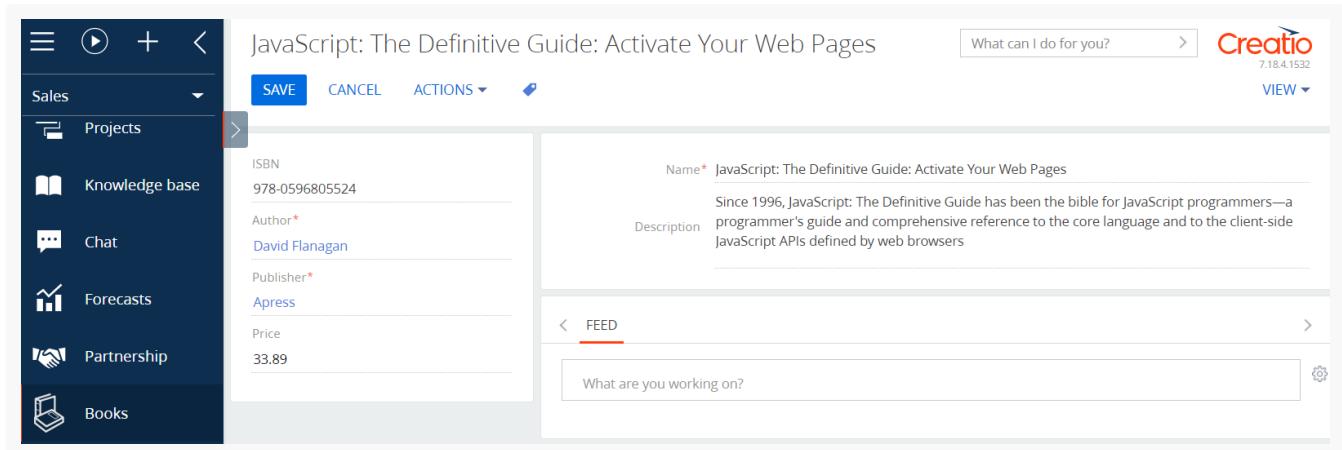
[Заголовок] ([Title])	[Код] ([Code])	Тип данных	Обязательность поля
[Название] ([Name])	UserName	Строка (String)	Обязательное поле
[ISBN]	UsrISBN	Строка (String)	
[Автор] ([Author])	UsrAuthor	Справочник (Lookup) [Контакт] ([Contact])	Обязательное поле
[Издатель] ([Publisher])	UsrPublisher	Справочник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

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

2. Добавить в раздел демонстрационные записи

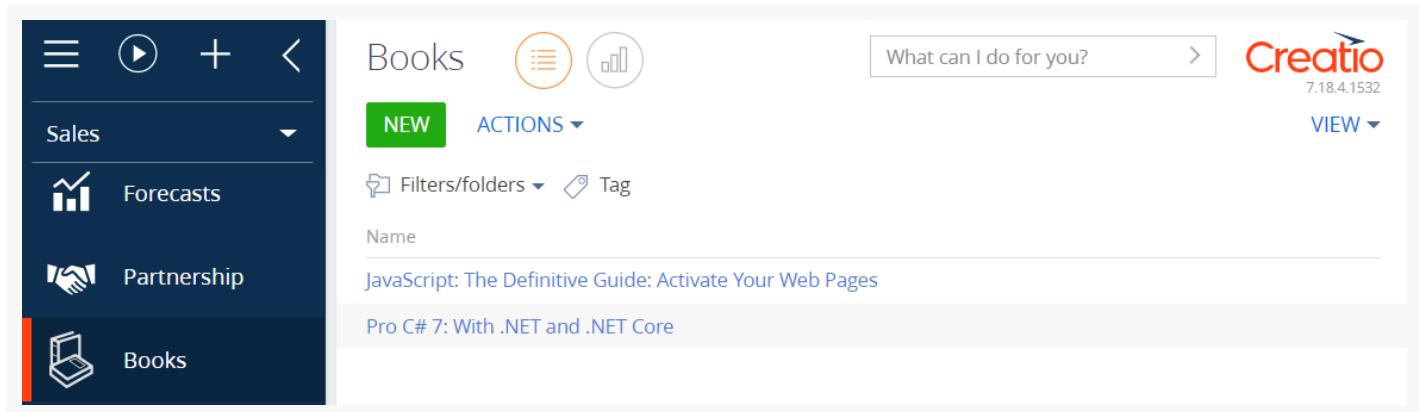
Чтобы **добавить записи** в реестр раздела [Книги] ([Books]):

- В разделе [Контакты] ([Contacts]) добавьте запись и заполните поле [ФИО] ([Full name]) значением "David Flanagan".
- В разделе [Контакты] ([Contacts]) добавьте запись и заполните поле [ФИО] ([Full name]) значением "Andrew Troelsen".
- В разделе [Контрагенты] ([Accounts]) добавьте запись и заполните поле [Название] ([Name]) значением "Apress".
- Добавьте книгу** JavaScript: The Definitive Guide: Activate Your Web Pages :
 - Перейдите в раздел [Книги] ([Books]).
 - Нажмите [Добавить] ([New]).
 - 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]).

The screenshot shows the 'Configuration' interface. On the left, there's a tree view of packages under 'Custom'. One package, 'sdkBookExamplePackage', is selected and highlighted with a red box. On the right, a list of objects is shown with a 'Multi actions' dropdown menu open. The 'Data' option in this menu is also highlighted with a red box.

Status	Type	Object	Modified on	Package
	Data	Account	4/20/2021, 9:03:47 AM	sdkBookE Package
	Data	UsrBooks	4/20/2021, 9:09:29 AM	sdkBookE Package
	Data	Contact	4/20/2021, 9:05:44 AM	sdkBookE Package
	Data	SysDetail	4/19/2021, 4:12:56 PM	sdkBookE Package
	Data	SysDetail	4/19/2021, 4:16:34 PM	sdkBookE Package
	Data	SysModuleEdit	4/19/2021, 4:16:34 PM	sdkBookE Package
	Data	SysModuleEdit	4/19/2021, 4:12:55 PM	sdkBookE Package

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".

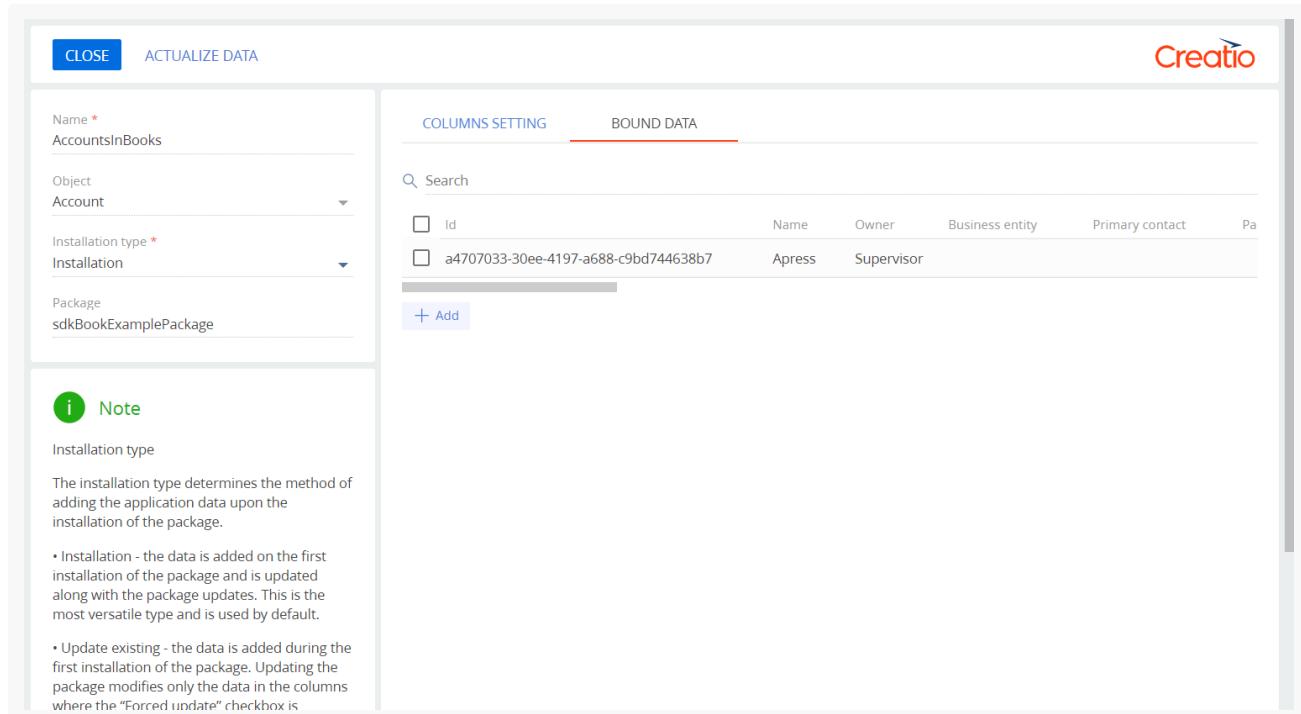
The screenshot shows the 'ACTUALIZE DATA' configuration page. On the left, there are fields for 'Name' (set to 'ContactsInBooks'), 'Object' (set to 'Contact'), 'Installation type' (set to 'Installation'), and 'Package' (set to 'sdkBookExamplePackage'). Below these fields is a note about installation type. On the right, the 'BOUND DATA' tab is selected, showing a table of contacts with columns for Id, Full name, Owner, Recipient's name, Title, and Gender. Two contacts are listed: Andrew Troelsen and David Flanagan, both marked as Supervisor.

	Id	Full name	Owner	Recipient's name	Title	Gender
<input type="checkbox"/>	e35e6588-dd4e-4cf3-9d46-5e0779269d20	Andrew Troelsen	Supervisor			
<input type="checkbox"/>	da9dd20c-5a79-44e3-8057-5e598294fea6	David Flanagan	Supervisor			

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".



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".

Id	Created on	Created by	Modified on	Modified by	Active processes	Name
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
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

- e. Сохраните данные.

4. Проверить привязки данных

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

Name	Title	Status	Type	Object	Modified on	Package
AccountsInBooks *			Data	Account	4/20/2021, 9:03:47 AM	sdkBookE
Books *			Data	UsrBooks	4/20/2021, 9:09:29 AM	sdkBookE
ContactsInBooks *			Data	Contact	4/20/2021, 9:05:44 AM	sdkBookE
SysDetail_DetailManage r_987f2dae9a1e4eb79e e8861dc9428a1e *			Data	SysDetail	4/19/2021, 4:12:56 PM	sdkBookE
SysDetail_DetailManage r_a61ff163c0f146f1be72 40bb57345693 *			Data	SysDetail	4/19/2021, 4:16:34 PM	sdkBookE
SysModuleEdit_SysMod uleEditManager_bdf314 3752da45e4ab98f2ef77 67d57b *			Data	SysModuleEdit	4/19/2021, 4:16:34 PM	sdkBookE
SysModuleEdit_SysMod uleEditManager_c3f22fb b9ee54c148c352bf835e d4e2c *			Data	SysModuleEdit	4/19/2021, 4:12:55 PM	sdkBookE
SysModuleEntity_SysMo duleEntityManager_62ac					4/19/2021	sdkBookF

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.

Контроль версий в Git



Эта статья о том, как начать работу с Git в Creatio. Вначале изучим основы Git, затем перейдем к особенностям использования Git в Creatio и настройке Git. В конце статьи вы уже будете знать, что такое Git и почему им следует пользоваться, а также получите окончательно настроенную для работы систему.

Git — распределенная система управления версиями. Основное **отличие Git** от других систем контроля версий — это подход к работе с данными. Подход Git к **хранению данных** похож на набор снимков файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Если файлы не были изменены, то Git не запоминает эти файлы еще раз, а создает ссылку на предыдущую версию идентичного сохраненного файла. Git представляет данные как поток снимков. Большинство других систем (CVS, Subversion, Perforce, Bazaar и т. д.) хранят информацию в виде набора файлов и перечня изменений в файлах по времени (обычно это называют контролем версий, основанным на различиях).

Состояния файлов в Git:

- Зафиксированное (`committed`) — файл уже сохранен в локальной базе.
- Измененное (`modified`) — файл был изменен, но еще не зафиксирован в локальной базе.
- Подготовленное (`staged`) — файл был изменен и отмечен для включения в следующий коммит.

Инструкция по установке и работе с системой контроля версий Git содержится в официальной [документации Git](#).

Для работы можно использовать различные GUI, предоставляющие удобный интерфейс для работы с Git, например [Sourcetree](#).

Особенности работы с Git в Creatio

Creatio IDE предоставляет инструменты для работы с системой контроля версий Subversion. При включенном режиме разработки в файловой системе встроенный механизм интеграции с системой контроля версий отключен. Это позволяет использовать любую систему контроля версий. Мы рекомендуем использовать Git.

Система контроля версий Git **рекомендуется к использованию** для:

- Приложений, в которых разработка ведется в файловой системе.
- On-site приложений.

Для cloud-приложений рекомендуется использовать SVN. Работа с системой контроля версий SVN описана в статье [Контроль версий в Subversion](#).

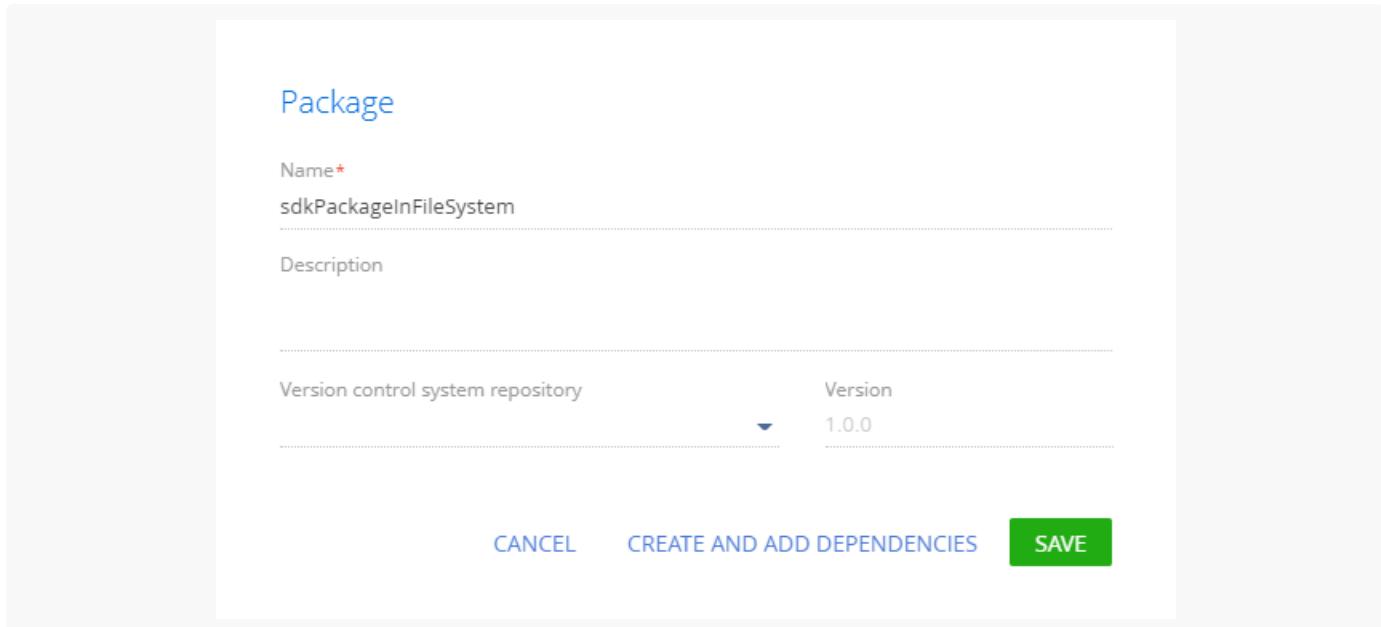
Система контроля версий Git **рекомендуема к использованию** для on-site приложений Creatio.

Общая последовательность работы в Git

1. Создать пакет

1. Перейдите в дизайнер системы Creatio 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. Заполните **свойства пакета**:
 - [Название] ([Name]) — "sdkPackageInFileSystem".

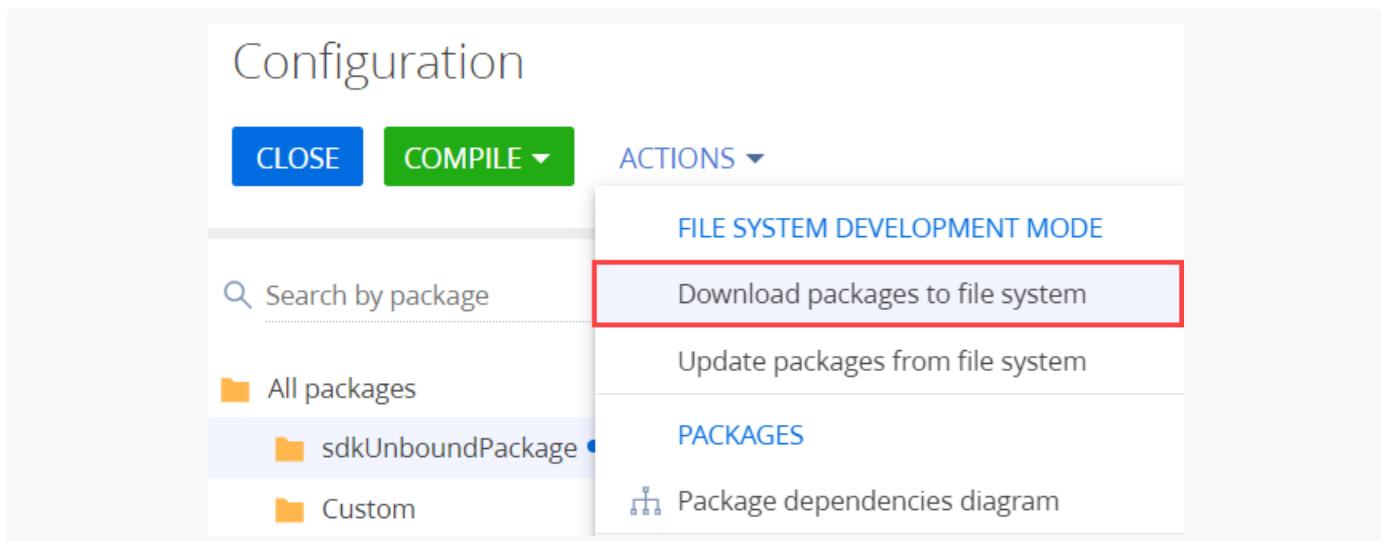
Пакет создайте без привязки к хранилищу.



- Нажмите кнопку [*Создать и добавить зависимости*] ([*Create and add dependencies*]) и установите [зависимости пакета](#).
- В пользовательском пакете создайте конфигурационные элементы. Конфигурационные элементы подробно описаны в статье [Разработка конфигурационных элементов](#).

2. Выгрузить пакет в файловую систему

- Настройте Creatio для работы в файловой системе. Настройка описана в статье [Внешние IDE](#).
- На панели инструментов в группе действий [*Разработка в файловой системе*] ([*File system development mode*]) выберите [*Выгрузить все пакеты в файловую систему*] ([*Download packages to file system*]).



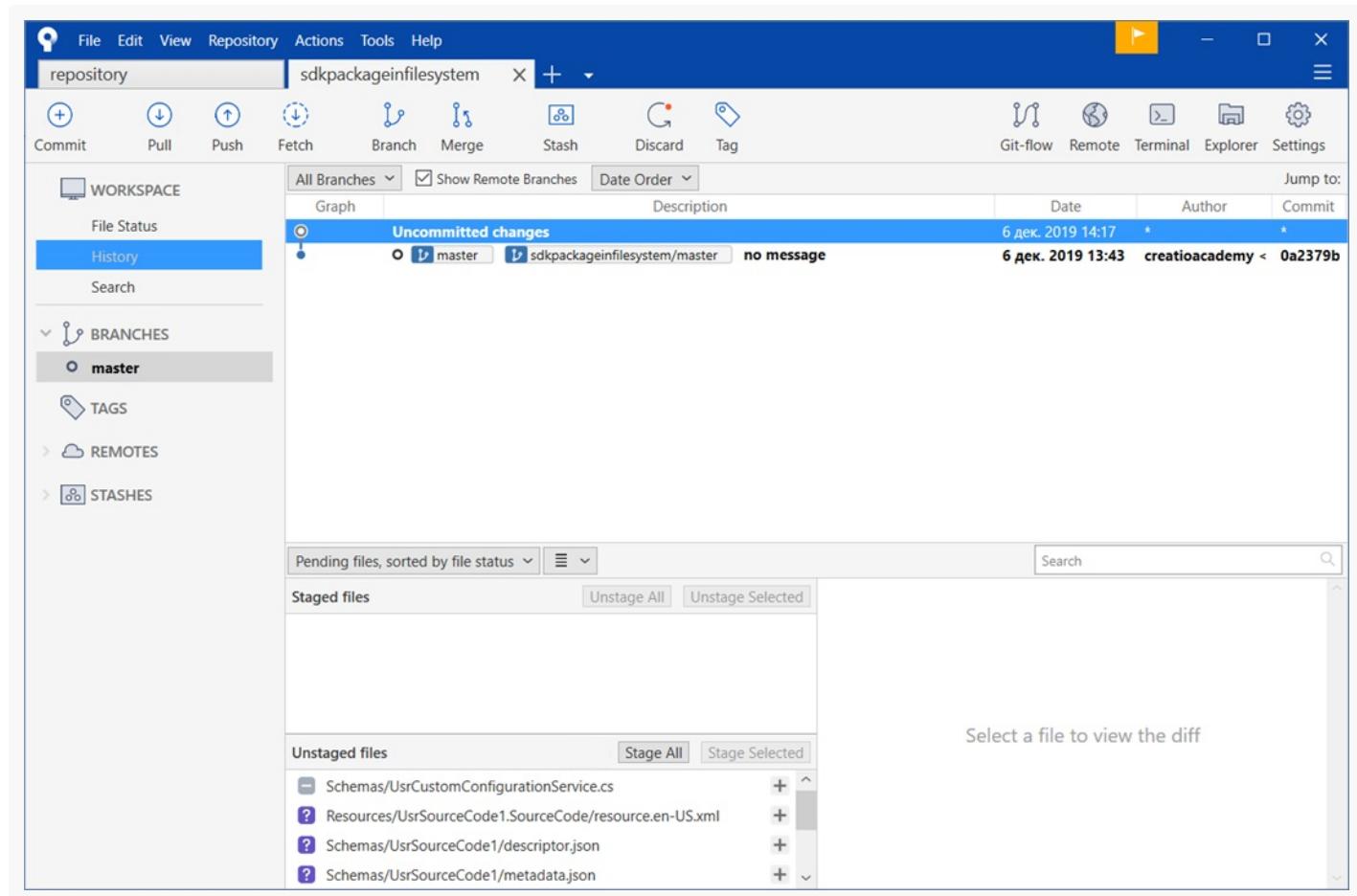
В результате все пакеты будут выгружены по пути `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в каталог с соответствующим названием пакета.

3. Добавить исходный код

Для работы с исходным кодом клиентских или серверных схем используйте [внешнюю IDE](#).

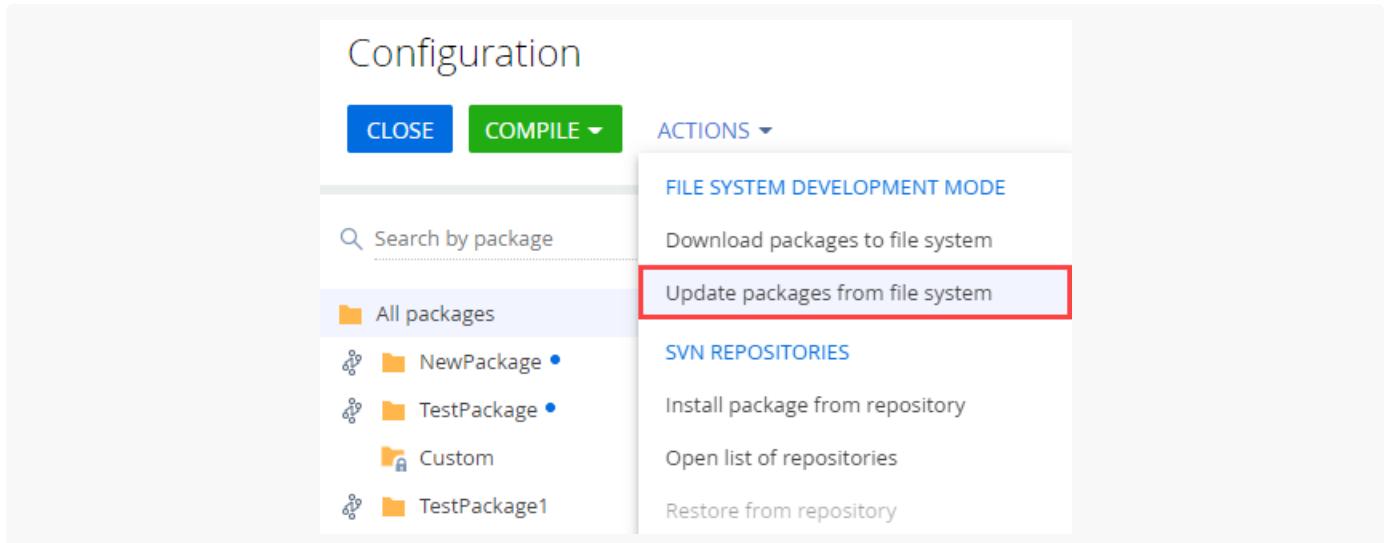
4. Зафиксировать изменения в хранилище Git

- Нажмите [*Stage All*] и выберите файлы, которые необходимо зафиксировать.
- Нажмите [*Pull*] и выгрузите изменения из глобального хранилища, сделанные другими пользователями.
- Нажмите [*Commit*] и зафиксируйте изменения в локальном хранилище.
- Нажмите [*Push*] и зафиксируйте изменения в глобальном хранилище.



5. Установить пакет в приложение

- Перейдите в дизайнер системы Creatio по кнопке
- В блоке [Конфигурирование разработчиком] ([*Admin area*]) перейдите по ссылке [Управление конфигурацией] ([*Advanced settings*]).
- На панели инструментов в группе действий [Разработка в файловой системе] ([*File system development mode*]) выберите [Обновить пакеты из файловой системы] ([*Update packages from file system*]).



В результате пакет будет добавлен в приложение.

6. Выполнить генерацию исходных кодов

Чтобы **выполнить генерацию исходных кодов**, на панели инструментов Creatio в группе действий [Исходный код] ([*Source code*]) выберите [Сгенерировать для требующих генерации] ([*Generate where it is needed*]).

Configuration

CLOSE COMPILE ▾ ACTIONS ▾

Search by package

- All packages
 - Custom
 - TestForSupport
 - sdkBookExample
 - sdkCreateDetailWithEc...
 - sdkExampleUtilsModul...**
 - sdkLimitedIntegerEditF...
 - sdkMixinPackage
 - sdkStandardModulePa...
 - sdkTestPackage
 - sdkWebFormPackage
 - ActionsDashboard
 - AnalyticsDashboard
 - BPMS_ENU
 - Base
 - BaseProcessDesigner
 - BaseScoring

FILE SYSTEM DEVELOPMENT MODE

- Download packages to file system
- Update packages from file system

PACKAGES

- Package dependencies diagram

SVN REPOSITORIES

- Install package from repository
- Open list of repositories
- Restore from repository

ACTUALIZE ITEMS

- Update DB structure where it is needed
- Install SQL scripts where it is needed
- Install data where it is needed

SOURCE CODE

- Generate for modified schemas
- Generate where it is needed**
- Generate for all schemas

7. Скомпилировать изменения

Чтобы **скомпилировать изменения**, на панели инструментов Creatio нажмите [Компилировать] ([*Compile*]).

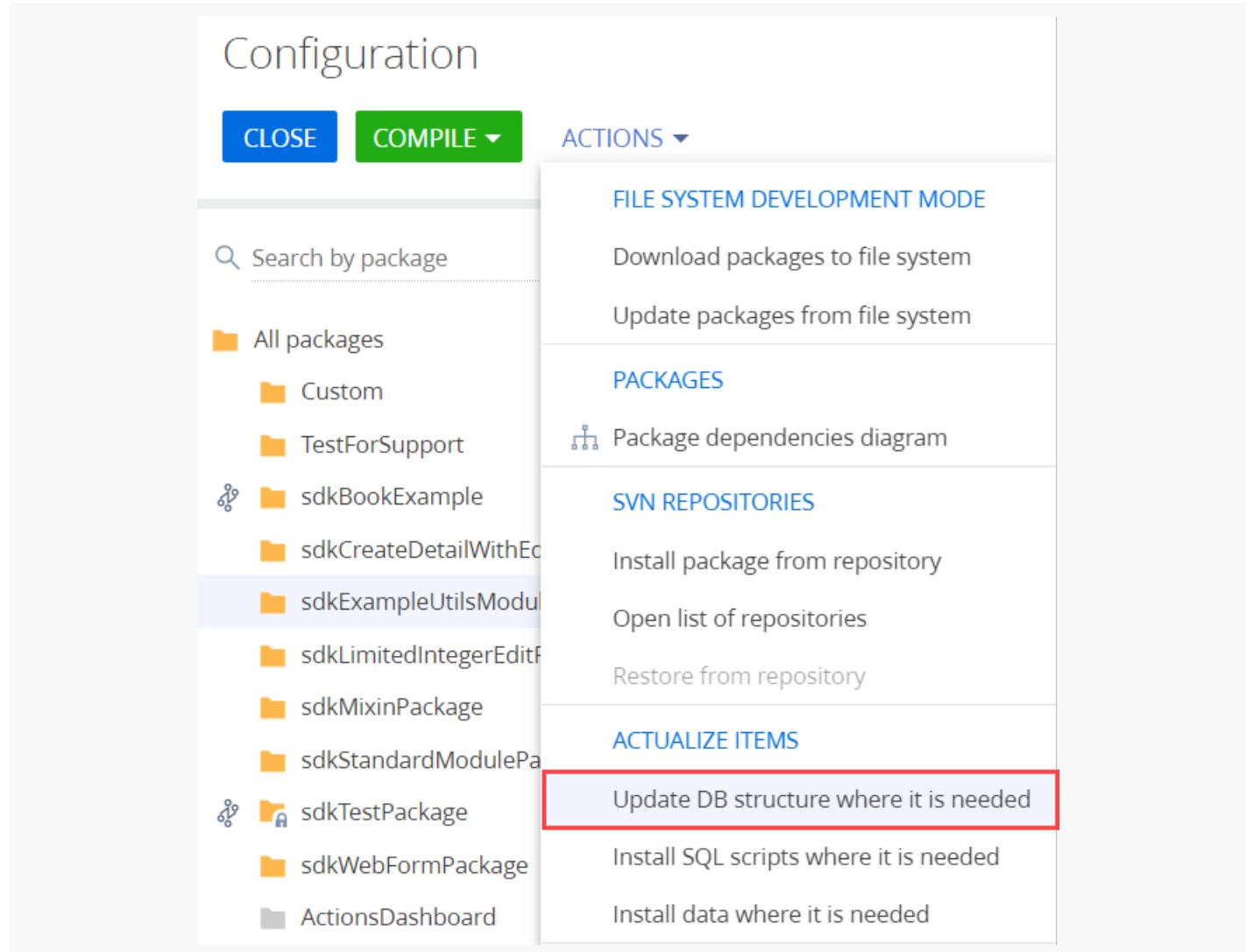
Configuration

CLOSE COMPILE ▾ ACTIONS ▾

Необходимость обновления структуры базы данных, установки SQL-скриптов и привязанных данных отображается в колонке [Статус] ([Status]) рабочей области раздела [Конфигурация] ([Configuration]).

8. Обновить структуру базы данных

Чтобы **обновить структуру базы данных**, на панели инструментов Creatio в группе действий [Актуализировать элементы] ([Actualize items]) выберите [Обновить структуру БД для требующих обновления] ([Update DB structure where it is needed]).



9. Установить SQL-сценарии и привязанные данные (опционально)

Если пакет содержит привязанные SQL-сценарии или данные, то необходимо выполнить соответствующие действия для их выполнения или установки.

После установки в приложении станет доступной реализованная в пакете функциональность.

Для отображения примененных изменений может понадобиться обновление страницы с очисткой кэша.

УПРАВЛЕНИЕ ПОСТАВКАМИ В CREATIO IDE



Легкий

Creatio предоставляет различные инструменты поставок функциональности.

Инструменты управления поставками, которые предоставляет Creatio:

- Creatio IDE.
- Утилита WorkspaceConsole.

В этой статье будет рассмотрено управление поставками с использованием Creatio IDE.

Creatio IDE предоставляет различные способы переноса функциональности в зависимости от этапа [процесса поставок функциональности](#).

Использование Creatio IDE в качестве инструмента переноса решений позволяет:

- Переносить пакеты в виде архивов — на этапе переноса функциональности между рабочими средами.
- Переносить пакеты с использованием системы контроля версий SVN — на этапе разработки новой функциональности или в процессе доработок.
- Переносить схемы конфигурационных элементов — на этапе разработки новой функциональности или в процессе доработок.

Перенос пакета

Перенос пользовательского пакета — это процесс экспорта разработанной функциональности, представленной в виде пакета, из одной рабочей среды и импорта в другую рабочую среду. При многопользовательской разработке также можно выполнять перенос между средами разработки.

Экспорт пакетов выполняется в разделе [Конфигурация] ([Configuration]) дизайнера системы. Пакет скачивается в виде *.zip-архива, который содержит *.gz-архив пакета.

Импорт пакетов выполняется в разделе [Установка и удаление приложений] ([Installed applications]) дизайнера системы. Для выполнения импорта пакетов воспользуйтесь инструкцией [Перенос первого приложения](#).

Важно. Пакет `Custom` нельзя переносить между приложениями.

Также *.gz-архив пакета можно [выгрузить из базы данных](#) или [из хранилища SVN](#) с помощью [утилиты WorkspaceConsole](#).

Не допускается создавать пакеты в промышленной среде, затем на основе промышленной среды создавать среду разработки, дорабатывать функциональность пакетов и переносить их обратно на промышленную среду. Подробнее о рекомендуемой последовательности разработки описано в статье [Процесс управления поставками](#).

Перенос пакета с использованием SVN

Система управления версиями является опциональным компонентом приложения Creatio. Используется когда параллельно с эксплуатацией системы необходимо на платформе организовать разработку

пользовательской конфигурации. Для **многопользовательской разработки** применение системы контроля версий является обязательным.

Creatio IDE настроена на работу с [системой контроля версий SVN](#). Одной из функций SVN является перенос изменений между средами разработки.

Особенности использования SVN для переноса изменений:

- Возможность переноса между средами разработки пакетов и схем конфигурационных элементов.
- Возможность переноса привязанных к пакету данных, например, наполнение справочников, новые системные настройки, демонстрационные записи раздела и т. д.
- Автоматическая установка из SVN пакетов-зависимостей.
- Независимость от службы поддержки при переносе изменений для on-site приложений.

Важно. Систему контроля версий SVN разрешено использовать для переноса изменений только между [средами разработки](#). Запрещено использовать SVN на [предпромышленной](#) и [промышленной](#) среде, поскольку это может привести к неработоспособности или ухудшению производительности приложения. Подробнее читайте в статье [Контроль версий в Subversion](#).

Перед обновлением приложения с помощью SVN необходимо выполнить резервное копирование базы данных. Если приложение развернуто в облаке, то необходимо обратиться в службу поддержки.

Перенос схемы конфигурационного элемента

Перенос схем — это процесс экспорта схемы из среды разработки и импорта в предпромышленную среду. При многопользовательской разработке также можно выполнять перенос между средами разработки.

Перенос схем конфигурационных элементов допускается использовать при:

- Передаче незавершенной разрабатываемой схемы от одного разработчика другому, поскольку заливка незавершенной функциональности в систему контроля версий является нецелесообразной.
- Сохранении промежуточных результатов разработки, если система контроля версий не используется, или при возникновении проблем с системой контроля версий.
- Быстрым переносе схемы между средами разработки.

Особенности использования механизма переноса схем конфигурационных элементов:

- Возможность заменить содержимое одной схемы.
- Механизм позволяет экспортить и импортировать только схемы. Для переноса пакетов необходимо использовать процедуру, описанную в статье [Перенести пакеты](#).
- Требуются права доступа администратора в приложении.
- Нет возможности загрузить несколько схем одновременно.

Перенести пакеты

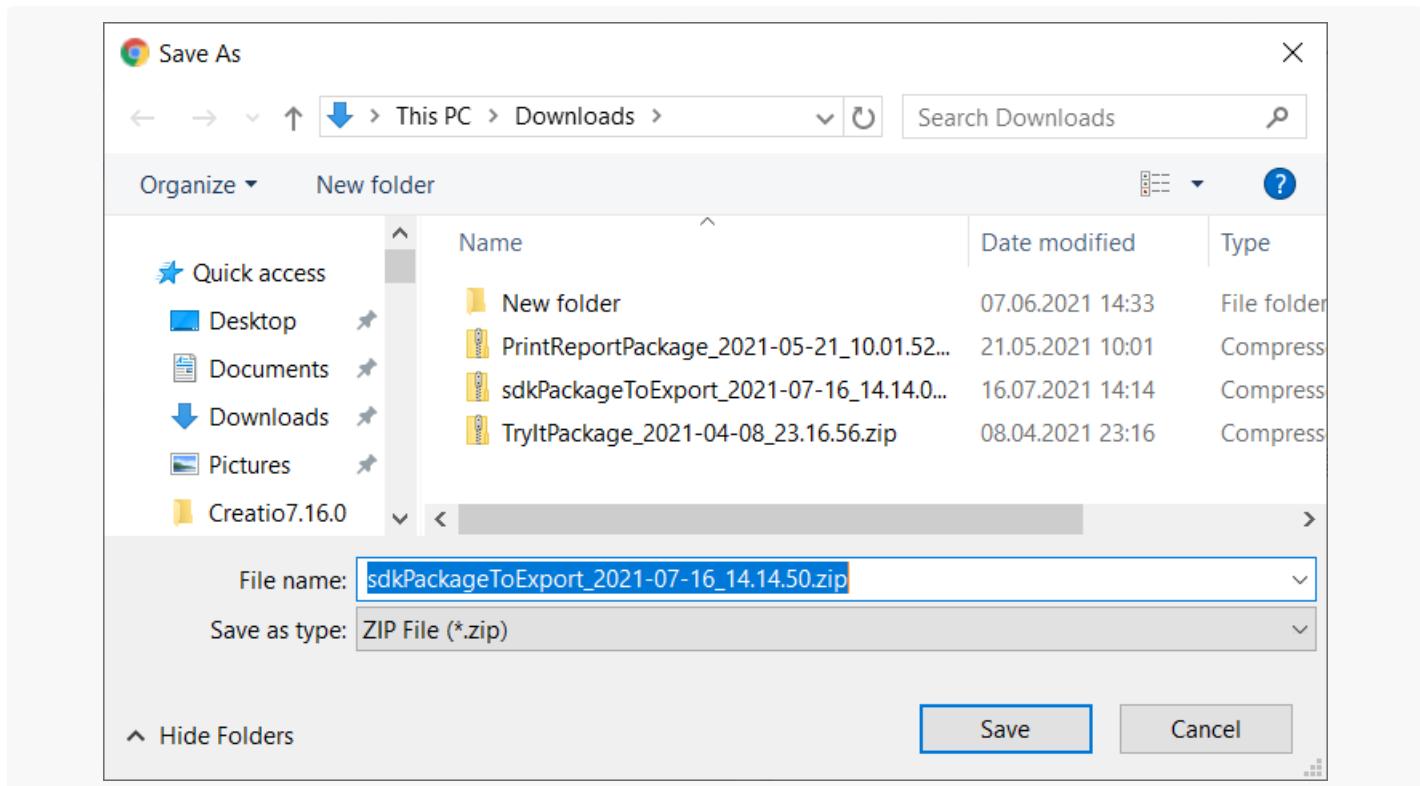


Экспортировать пакет

1. Перейдите в раздел [Конфигурация] ([Configuration]) и выберите пользовательский [пакет](#), который необходимо экспорттировать.
2. В меню свойств пакета нажмите [Экспортировать] ([Export]).

The screenshot shows a list of packages in a software interface. A search bar at the top left says 'Search by package'. Below it is a tree view of packages under 'All packages': 'Custom', 'TestPackage', 'TryItPackage', 'sdkPackageToExport' (which is selected and highlighted in blue), 'tsaWebData', 'sdkCustomPackage', and 'sdkDependentPackage'. To the right of the list is a context menu with several options: 'Add', 'Multi-select', 'Export' (which is highlighted with a red box), 'Move all elements', and 'Delete'. The 'Export' option has a small upward arrow icon next to it.

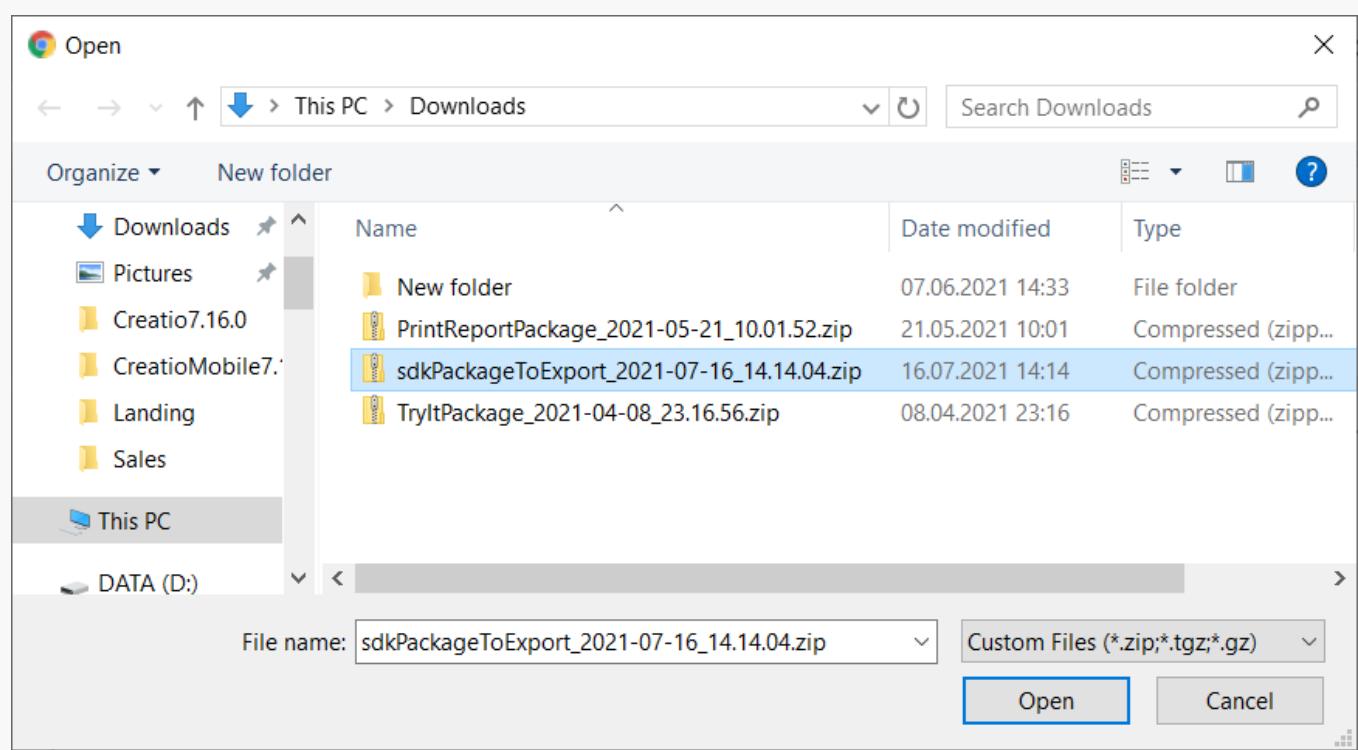
В зависимости от настроек браузера *.zip-архив с *.gz-архивом пакета будет выгружен в каталог по умолчанию или браузер отобразит диалоговое окно для выбора каталога, в который необходимо сохранить архив.



После этого пакет можно импортировать в другую рабочую среду.

Импортировать пакет

1. Перейдите в дизайнер системы по кнопке
2. В блоке [Приложения] ([Applications]) перейдите по ссылке [Установка и удаление приложений] ([Installed applications]).
3. На странице мастера установки приложений Marketplace нажмите [Добавить приложение] —> [Установить из файла] ([Add application] —> [Install from file]).
4. Нажмите [Выбрать файл] ([Select file]) и укажите путь к *.zip-архиву, который содержит *.gz-архив пакета.



После этого выполнится создание резервной копии базы данных приложения. Процесс может занять несколько минут.

После успешной установки пакета в приложение Creatio отобразится соответствующее сообщение.

Application installation

[CLOSE](#)



Application installed successfully.

You can close the page and continue working with the Creatio.

[DOWNLOAD INSTALLATION LOG](#)

Attention!

Ensure that the application you are installing is compatible with your version of Creatio. Description of the Marketplace application contains a list of compatible Creatio products.

The vendor guarantees the compatibility of these products. Do not install an application if your Creatio version is not present in the list.

If you install a Marketplace application to become familiar with its capabilities, it is better to install it on your sandbox version of the system (support service can provide it) or install it on 14-days trial version which is available on Creatio website.

Only after that, we recommend to install the product on your Creatio production environment.

Если установка пакета выполнилась неуспешно также отобразится соответствующее сообщение.

Application installation

CLOSE

Attention!

Application installation failed.
Restore configuration from backup?

You can close this page and continue with the Creatio.

[RESTORE PACKAGES FROM BACKUP](#) [DOWNLOAD INSTALLATION LOG](#)

Ensure that the application you are installing is compatible with your version of Creatio. Description of the Marketplace application contains a list of compatible Creatio products.

The vendor guarantees the compatibility of these products. Do not install an application if your Creatio version is not present in the list.

If you install a Marketplace application to become familiar with its capabilities, it is better to install it on your sandbox version of the system (support service can provide it) or install it on 14-days trial version which is available on Creatio website.

Only after that, we recommend to install the product on your Creatio production environment.

Чтобы посмотреть причину неуспешной установки пакета, нажмите [*Скачать лог установки*] ([*Download installation log*]).

Чтобы восстановить предыдущее состояние конфигурации, нажмите [*Восстановить пакеты из резервной копии*] ([*Restore packages from backup*]). После восстановления резервной копии отобразится соответствующее сообщение.

Application installation

[CLOSE](#)

The configuration has been successfully restored from backup.

You can close this page and continue with the Creatio.

[DOWNLOAD INSTALLATION LOG](#)



Attention!

Ensure that the application you are installing is compatible with your version of Creatio. Description of the Marketplace application contains a list of compatible Creatio products.

The vendor guarantees the compatibility of these products. Do not install an application if your Creatio version is not present in the list.

If you install a Marketplace application to become familiar with its capabilities, it is better to install it on your sandbox version of the system (support service can provide it) or install it on 14-days trial version which is available on Creatio website.

Only after that, we recommend to install the product on your Creatio production environment.

Перенести пакеты с использованием SVN



На заметку. При включенном [режиме разработки в файловой системе](#) механизм интеграции с SVN отключен.

1. Подключить хранилище системы контроля версий SVN

Если к приложению подключено хранилище системы контроля версий SVN, переходите к пункту [2. Включить автоматическое применение изменений](#).

Чтобы **подключить хранилище системы контроля версий SVN** к приложению Creatio:

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]).
2. В меню действий в группе [Хранилища SVN] ([SVN repositories]) выберите [Открыть список хранилищ] ([Open list of repositories]).
3. Нажмите [Добавить] ([Add]) и заполните **свойства хранилища**:
 - [Название] ([Name]) — название хранилища (обязательное свойство).

- [Адрес SVN хранилища] ([*SVN storage address*]) — адрес хранилища SVN (обязательное свойство).
- [Логин] ([*Login*]) — логин пользователя хранилища SVN.
- [Пароль] ([*Password*]) — пароль пользователя хранилища SVN.

The screenshot shows a configuration dialog titled "SVN storage". It contains the following fields:

- Name ***: SDKPackages
- SVN storage address ***: http://.../SDKPackages
- Login**: (empty)
- Password**: (empty)
- Active**: checked

At the bottom of the dialog are three buttons: CANCEL, AUTHENTICATE, and ADD (highlighted in green).

4. Нажмите [Аутентифицироваться] ([*Authenticate*]) и авторизуйтесь в хранилище SVN.

В результате к приложению будет подключено хранилище системы контроля версий SVN. Подробнее о работе с системой контроля версий SVN с использованием встроенных средств Creatio читайте в статье [Контроль версий в Creatio IDE](#).

2. Включить автоматическое применение изменений

Чтобы изменения применились после выполнения переноса, необходимо включить автоматическое применение изменений.

Чтобы **включить автоматическое применение изменений**, в файле `..\Terrasoft.WebApp\Web.config` установите значение `true` для ключей элемента `<appSettings>`:

- `AutoUpdateOnCommit` — ключ отвечает за обновление пакетов из хранилища SVN перед их заливкой. Если для этого ключа установлено значение `false`, то перед заливкой в хранилище SVN приложение предупредит пользователя о необходимости обновления, если схемы пакета были изменены.
- `AutoUpdateDBStructure` — ключ отвечает за обновление структуры базы данных.
- `AutoInstallSqlScript` — ключ отвечает за установку SQL-схемариев.
- `AutoInstallPackageData` — ключ отвечает за установку привязанных данных.

`..\Terrasoft.WebApp\Web.config`

```
<appSettings>
  ...
  <add key="AutoUpdateOnCommit" value="true" />
  <add key="AutoUpdateDBStructure" value="true" />
  <add key="AutoInstallSqlScript" value="true" />
  <add key="AutoInstallPackageData" value="true" />
</appSettings>
```

3. Проверить привязки данных

Перед переносом пакета необходимо проверить, что все данные, которые необходимо перенести, привязаны к пакету. При создании раздела с помощью мастера к пакету автоматически привязываются данные, необходимые для регистрации и корректной работы раздела.

Описание привязки данных к пакету содержится в статье [Общие принципы работы с пакетами](#).

4. Проверить возможность переноса зависимостей пакета

Во время разработки к пользовательскому пакету могут быть добавлены зависимости от других пакетов. Если пакеты-зависимости разработаны сторонними разработчиками, то необходимо проверить, что они уже установлены в приложение, в которое будет осуществляться перенос пользовательского пакета. Если пакеты-зависимости находятся в доступном хранилище SVN, то они будут установлены автоматически.

Описание зависимостей пакетов содержится в статье [Общие принципы работы с пакетами](#).

5. Установить пакет из хранилища SVN

Описание установки пакет из хранилища SVN содержится в статье [Контроль версий в Creatio IDE](#).

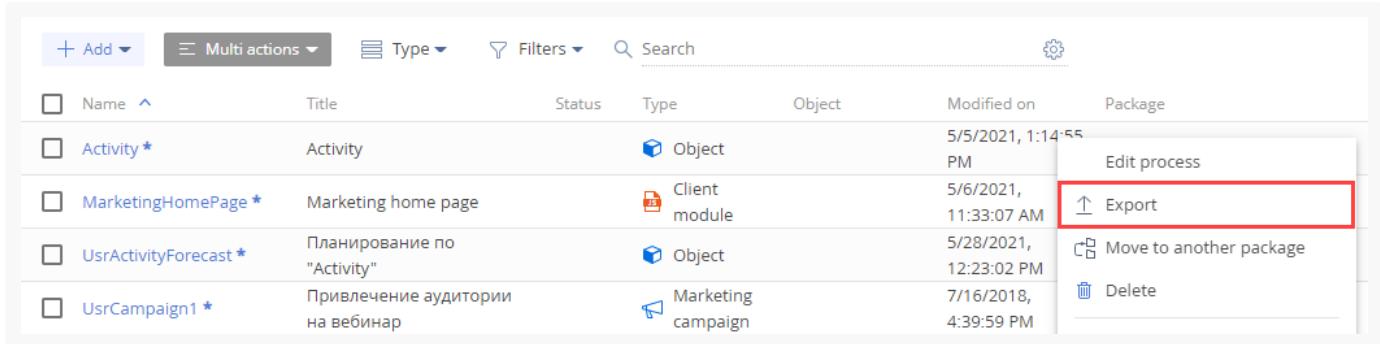
Перенести схемы конфигурационных элементов



Легкий

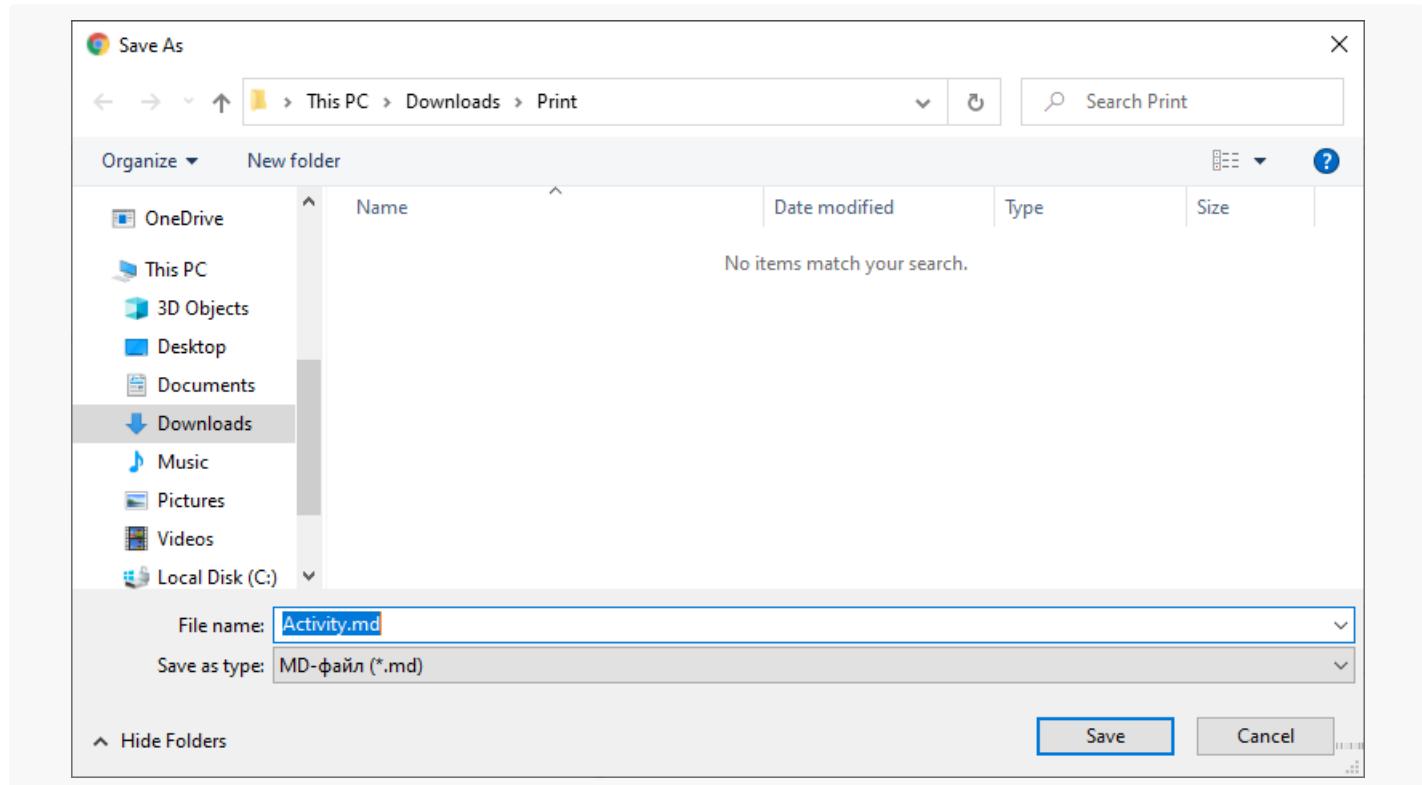
Экспортировать схему конфигурационного элемента

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и в пользовательском [пакете](#) выберите схему конфигурационного элемента, которую необходимо экспорттировать.
2. В меню свойств конфигурационного элемента нажмите —> [Экспортировать] ([Export]).



Name	Title	Status	Type	Object	Modified on	Package
Activity*	Activity		Object		5/5/2021, 1:14:55 PM	Edit process
MarketingHomePage*	Marketing home page		Client module		5/6/2021, 11:33:07 AM	<input type="button" value="Export"/> <input type="button" value="Move to another package"/> <input type="button" value="Delete"/>
UsrActivityForecast*	Планирование по "Activity"		Object		5/28/2021, 12:23:02 PM	
UsrCampaign1*	Привлечение аудитории на вебинар		Marketing campaign		7/16/2018, 4:39:59 PM	

В зависимости от настроек браузера *.md-файл схемы конфигурационного элемента будет выгружен в каталог по умолчанию или браузер отобразит диалоговое окно для выбора каталога, в который необходимо сохранить файл.



Импортировать схему конфигурационного элемента

- Перейдите в раздел [Конфигурация] ([Configuration]) и выберите пользовательский [пакет](#), в который необходимо импортировать схему конфигурационного элемента.
- На панели инструментов реестра раздела нажмите [Добавить] —> [Импорт] ([Add] —> [Import]).

The screenshot shows the Terrasoft configuration interface. On the left, there's a tree view of packages: 'All packages', 'Custom', 'TestForSupport', 'sdkBookExample', 'sdkCreateDetailWithEditable...', 'sdkExampleUtilsModulePacka...', 'sdkInsertQueryPackage', 'sdkLimitedIntegerEditPackage', 'sdkMixinPackage', 'sdkPackageInFileSystem', 'sdkSelectQueryPackage' (which is selected and highlighted in blue), 'sdkStandardModulePackage', 'sdkTestPackage', 'sdkUnboundPackage', 'sdkWebFormPackage', and 'ActionsDashboard'. On the right, there's a list of item types with a 'Type' dropdown and a 'Filters' dropdown. The items listed are: Object, Replacing object, Source code, Module, Page view model, Section view model, Detail (list) view model, Detail (fields) view model, Replacing view model, Business process, User task, Web service, SQL script, Data, and Import. The 'Import' item is highlighted with a red rectangle.

3. Выберите ранее экспортенный *.md-файл схемы конфигурационного элемента.
4. На панели инструментов нажмите [Компилировать] —> [Перекомпилировать все] ([Compile] —> [Compile all]) и скомпилируйте конфигурацию.

Важно. При импорте нескольких схем необходимо учитывать их зависимости друг от друга. Сначала необходимо импортировать все схемы-зависимости, после этого импортировать схему, зависящую от них. Например, сначала необходимо импортировать схему объекта, а затем схему модели представления страницы, зависящую от данного объекта.

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-файлами 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 = {
    "MyPackage1descriptor.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 пулам 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 перегенерацию файлового статического контента. Остальные действия в системе выполняют перегенерацию 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](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.ImageUrlParserBuilder` (`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



Сложный

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`:

- 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 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.getButtonItem({ "Type": "Terrasoft.MenuSeparator" }));
                actionMenuItems.addItem(this.getButtonItem({
                    "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



Сложный

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**:

- Проверьте наличие среды выполнения `Node.js` в вашей операционной системе.
Скачать инсталлятор можно на сайте <https://nodejs.org>.
- 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.

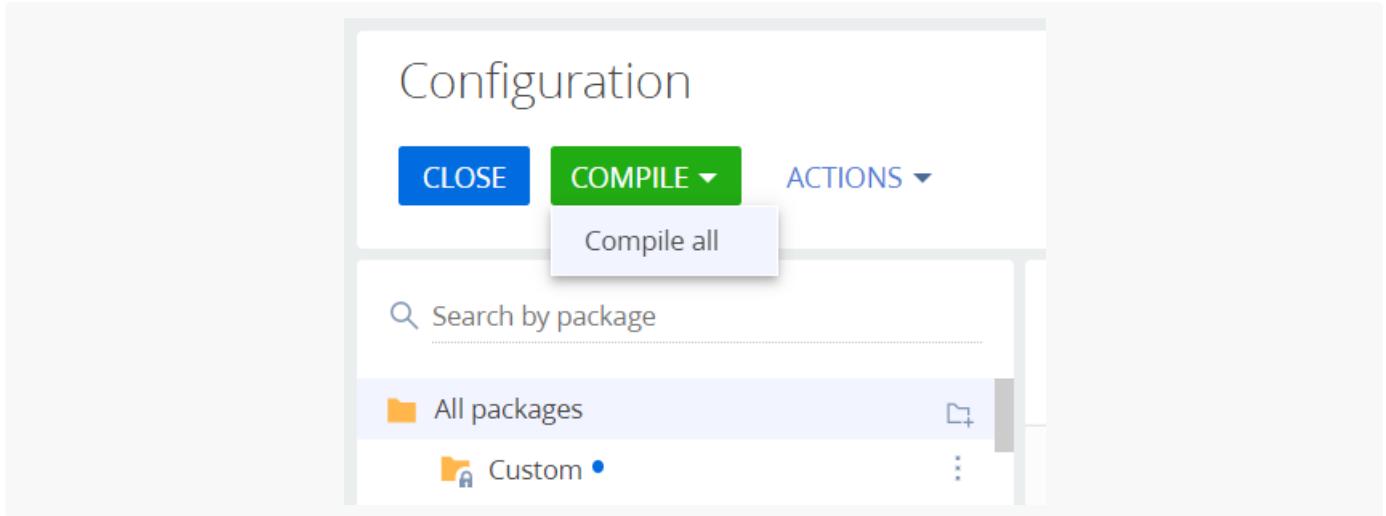
В файле `Web.config`, который находится в корневом каталоге приложения, установите значение `false` для флага `useStaticFileContent`.

Web.config

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

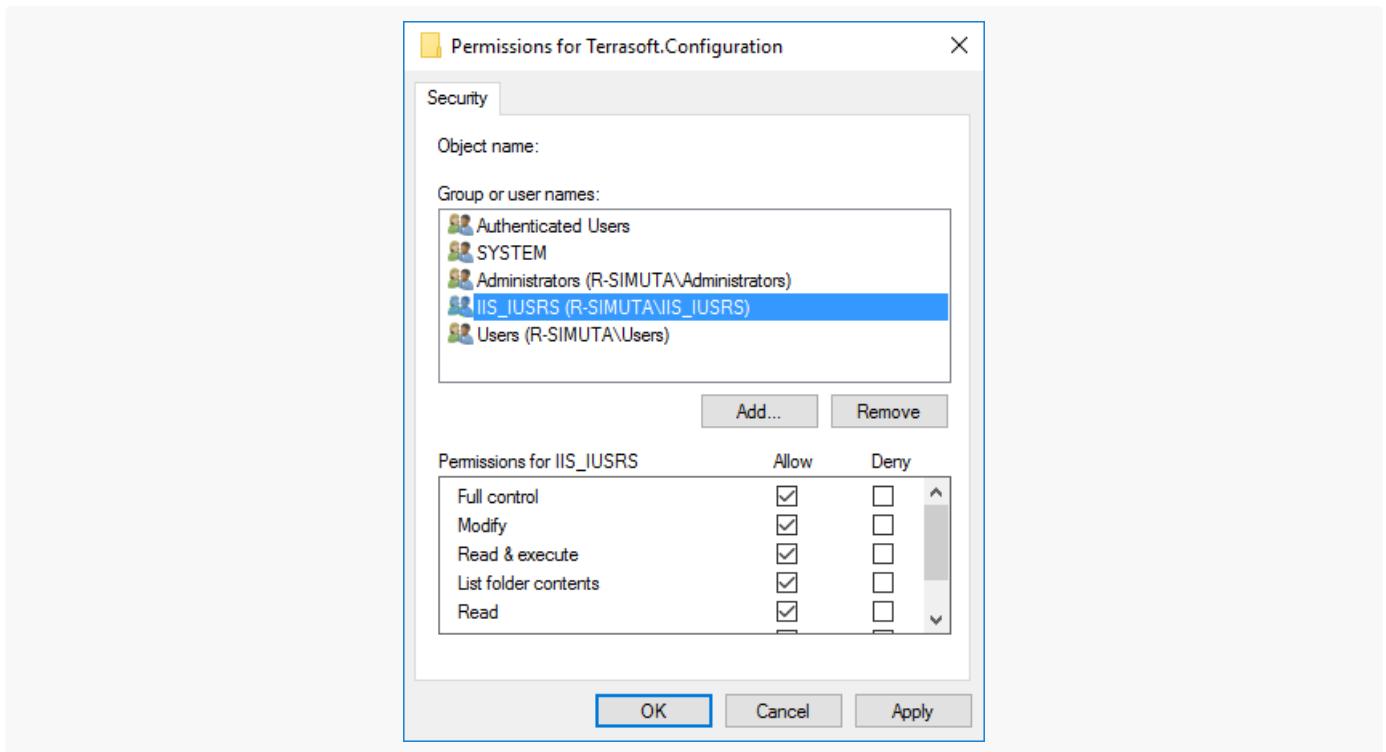
3. Скомпилируйте приложение.

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 .

2. В каталоге `Files` создайте вложенный каталог `src`.
3. В каталоге `src` создайте вложенный каталог `js`.
4. В каталоге `Files` создайте файл `descriptor.json`.

descriptor.json

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

5. 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**:

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

Validation.ts

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

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

LattersOnlyValidator.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`.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\creatio\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\ sdkTypeScript\Files\src\js>tsc

C:\creatio\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\ sdkTypeScript\Files\src\js>
```

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.

Pkg > sdkTypeScript > Files > src > js		
Name	Date modified	Type
bootstrap.js	09.05.2018 11:26	JavaScript File
LettersOnlyValidator.js	09.05.2018 14:12	JavaScript File
LettersOnlyValidator.js.map	09.05.2018 14:12	Linker Address Map
LettersOnlyValidator.ts	09.05.2018 13:58	TS File
tsconfig.json	08.05.2018 16:31	JSON File
Validation.js	09.05.2018 14:12	JavaScript File
Validation.js.map	09.05.2018 14:12	Linker Address Map
Validation.ts	08.05.2018 16:27	TS File

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

```
LettersOnlyValidator.js

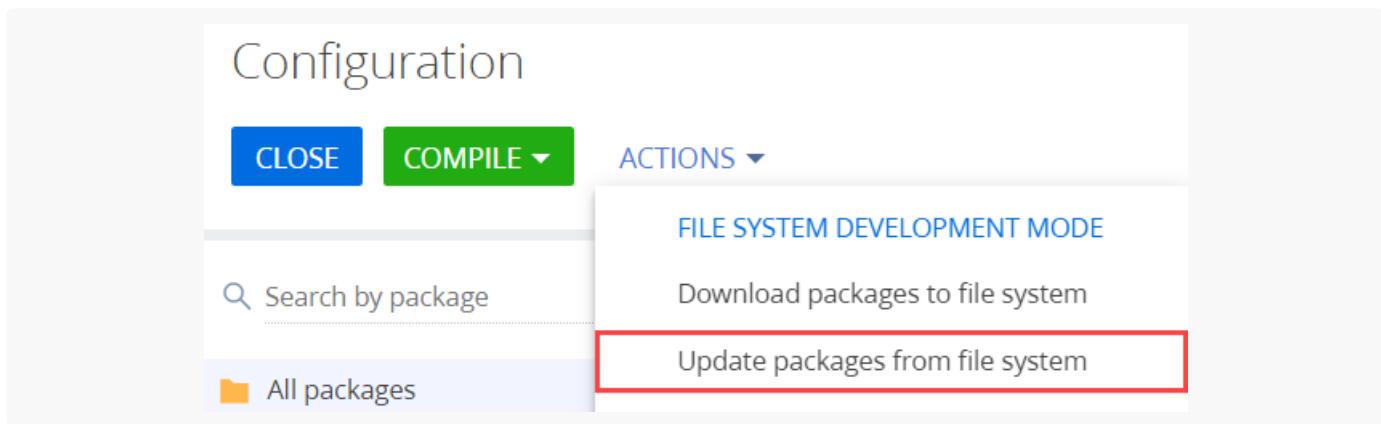
define(["require", "exports"], function (require, exports) {
    "use strict";
    var Terrasoft;
    (function (Terrasoft) {
        var LettersOnlyValidator = /** @class */ (function () {
            function LettersOnlyValidator() {
                this.lettersRegexp = /^[A-Za-z]+$/;
            }
            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
```

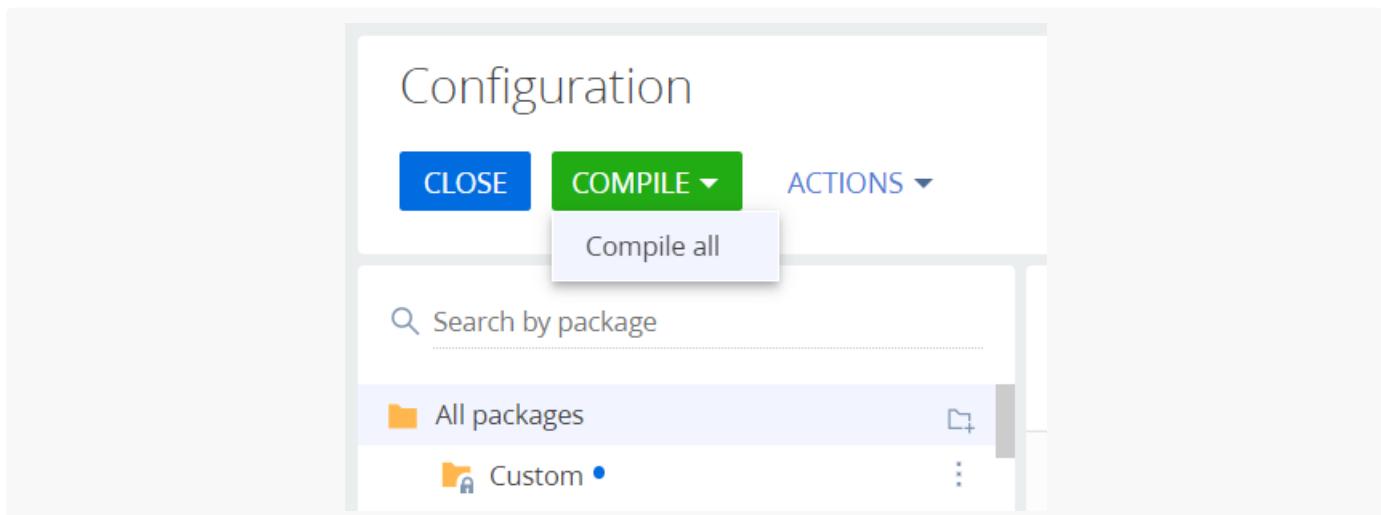
6. Выполнить генерацию вспомогательных файлов

Чтобы **выполнить генерацию вспомогательных файлов** `_FileContentBootsraps.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])).

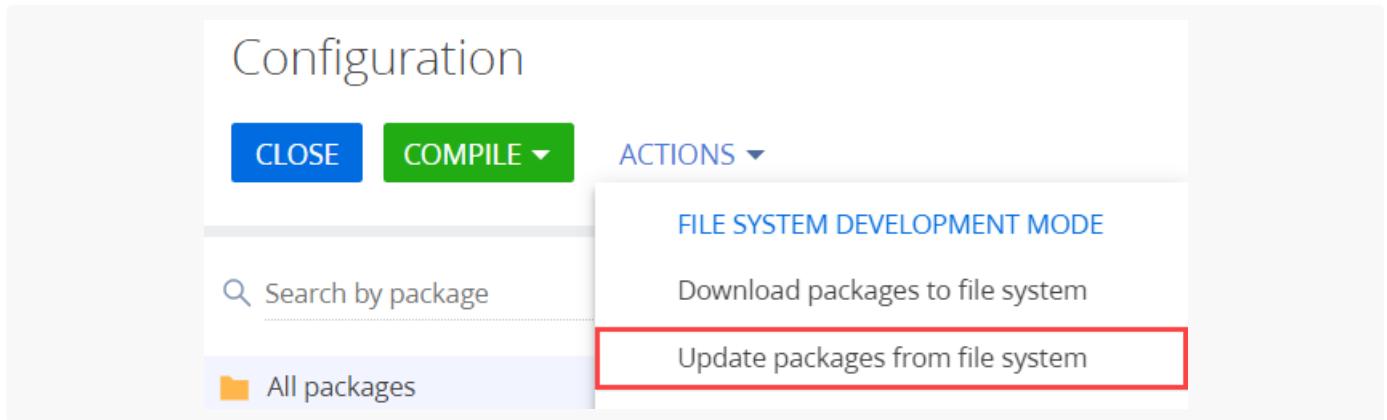


7. Проверить результат выполнения примера

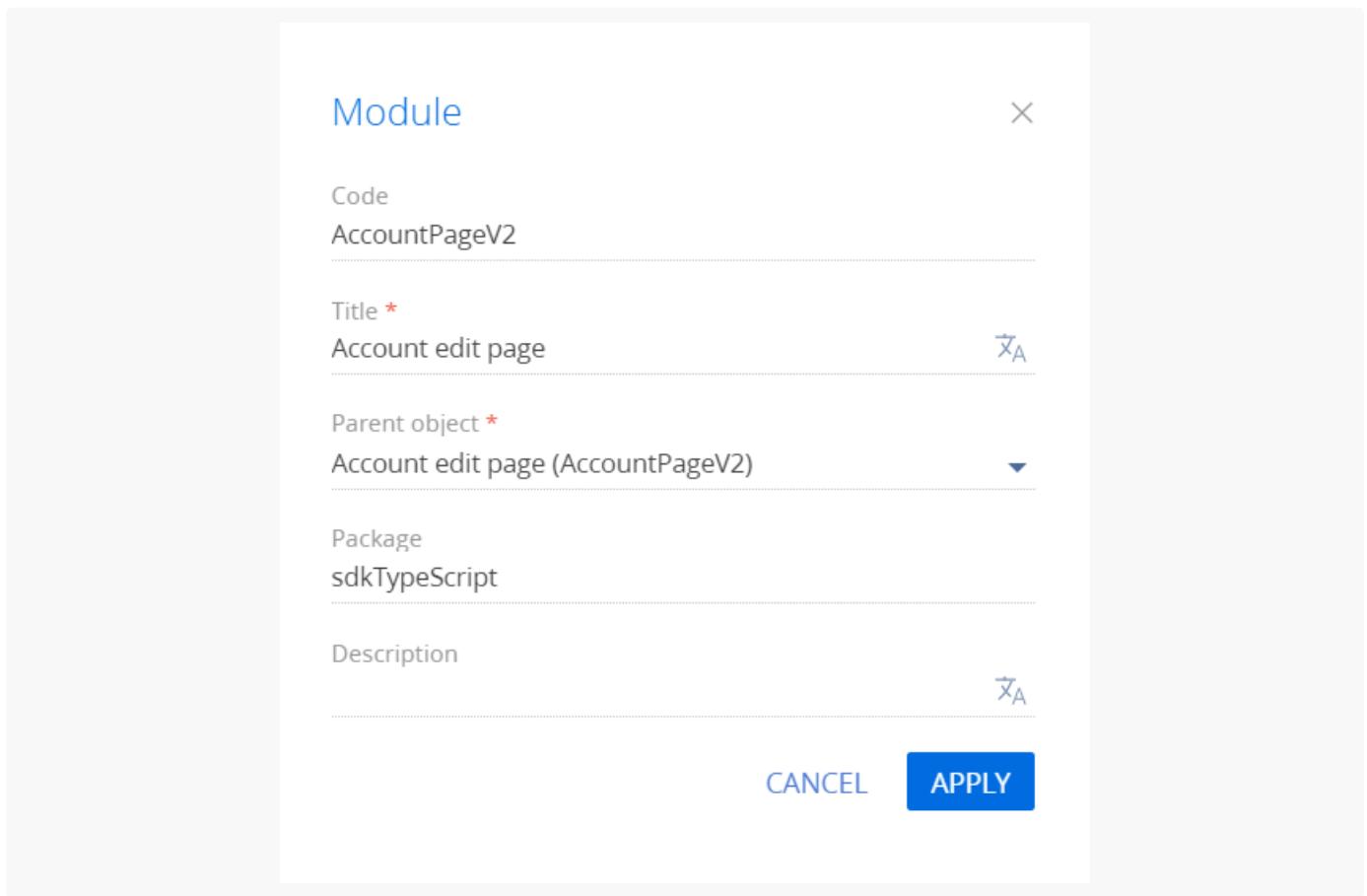
Чтобы **использовать валидацию**:

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

The screenshot shows the Creatio application interface for managing accounts. On the left, there's a sidebar with navigation links: General, Dashboards, Employees, Contacts, Accounts (which is selected), Activities, Feed, Group sections, and Books. The main workspace displays an account record for 'Accom (sample)' with the following details:

- Name***: Accom (sample)
- Type**: Customer
- Owner**: Supervisor
- Web**: ac.com
- Primary phone**: +1 617 440 2498
- Category**: B
- Industry**: Business services

A modal dialog box is centered on the screen, displaying the message: "Is 'Also known as' field valid: false". Below the message is an "OK" button. The background shows a summary section with metrics like "NEXT STEPS (0)" and "You don't have any tasks yet". To the right, there are tabs for STRUCTURE, MAINTENANCE, and TIMELINE, along with some contact information and communication options.

Сообщение о корректно заполненном поле

The screenshot shows the Creatio application interface. On the left is a sidebar with navigation links: General, Dashboards, Employees, Contacts, Accounts (which is selected), Activities, Feed, Group sections, and Books. The main area displays a record for 'Accom (sample)' with a progress bar at 95%. A modal window is open, displaying the message: 'Is 'Also known as' field valid: true' with an 'OK' button. The right side of the screen shows sections for Segmentation, Communication options, and contact details like web and phone numbers.

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`:

- Добавьте импорт модуля `createCustomElement`.
- 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 генерируются 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 загрузку 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 выборок 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 скрытие элемента внутри Shadow DOM компонента.

К сведению. Примеры использования методов можно найти в классе

`BPMonline.pages.ForecastTabUIV2`.

Управление поставками в WorkspaceConsole



Creatio предоставляет различные инструменты поставок функциональности.

Инструменты управления поставками, которые предоставляет Creatio:

- Creatio IDE.
- Утилита WorskpaceConsole.

В этой статье будет рассмотрено управление поставками с использованием утилиты WorkspaceConsole.

WorkspaceConsole — это утилита, которая предназначена для выполнения операций с [пакетами](#) и [схемами конфигурационных элементов](#) Creatio.

Использование утилиты WorkspaceConsole в качестве инструмента переноса решений позволяет:

- Переносить [пакеты](#) и [схемы конфигурационных элементов](#) между [рабочими средами](#) и конфигурациями.
- Устанавливать новые пакеты при обновлении или при экспорте из среды разработки.
- Переносить привязанные к пакету данные, например, наполнение справочников, новые системные настройки, демонстрационные записи раздела и т. д.
- Переносить ресурсы для локализации.
- Создавать и переносить рабочие пространства между рабочими средами.

Перед использованием утилиты WorkspaceConsole необходимо выполнить ее настройку.

Настройка утилиты WorkspaceConsole

1. Узнайте значение используемой строки подключения.

Для этого откройте файл

`..\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe.config`. Стока подключения содержится в атрибуте `connectionStringName` XML-элемента `<db>`.

Файл `Terrasoft.Tools.WorkspaceConsole.exe.config`

```

<terrasoft>
    ...
    <db>
        <general connectionStringName="db" securityEngineType="Terrasoft.DB.MSSql.MSSqlSecuri
    </db>
    ...
</terrasoft>

```

2. Отредактируйте строку подключения.

Для этого откройте файл `ConnectionStrings.config`, который находится в корневом каталоге приложения. Стока подключения содержится в атрибуте `name` XML-элемента `<connectionStrings>`. Значение атрибута `name` файла `ConnectionStrings.config` должно совпадать со значением атрибута `connectionStringName` файла `Terrasoft.Tools.WorkspaceConsole.exe.config`.

Файл ConnectionStrings.config

```
<connectionStrings>
    <add name="db" connectionString="Data Source=dbserver\MSSQL2016; Initial Catalog=YourDBNa
    <add name="dbOracle" connectionString="Data Source=(DESCRIPTION = (ADDRESS_LIST = (ADDRES
</connectionStrings>
```

На заметку. Как правило, в конфигурационном файле по умолчанию настроено только две строки подключения. Стока с названием `"db"` используется для подключения к базе данных MS SQL Server, а строка с названием `"dbOracle"` — к базе данных Oracle.

Если с помощью WorkspaceConsole необходимо выполнить разовую операцию, то утилиту можно запустить с параметром `-webApplicationPath`. В этом параметре необходимо указать путь к каталогу с установленным приложением. В таком случае утилита самостоятельно определит все необходимые параметры подключения к базе данных из файла `ConnectionStrings.config`. При этом параметры подключения из файла `Terrasoft.Tools.WorkspaceConsole.exe.config` будут проигнорированы.

3. В файле `Terrasoft.Tools.WorkspaceConsole.exe.config` установите значение `true` для атрибута `enabled` элемента `loadFromRemoteSources`.

```
<loadFromRemoteSources enabled="true" />
```

4. Установите утилиту WorkspaceConsole.

Для этого с правами администратора запустите на выполнение предустановленный пакетный файл команд, который находится по пути `..\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\`. Это необходимо для установки версии исполняемого файла утилиты и библиотек, которые используются утилитой.

Пакетные файлы команд утилиты WorkspaceConsole:

- Для **32-битной операционной системы** необходимо запустить файл `PrepareWorkspaceConsole.x86.bat`.
- Для **64-битной операционной системы** необходимо запустить файл `PrepareWorkspaceConsole.x64.bat`.

5. Настройте утилиту на выполнение операций с хранилищем SVN (опционально).

Для этого скопируйте файлы `SharpPlink-x64.svnExe`, `SharpSvn.dll` и `SharpSvn-DB44-20-x64.svnD11` из соответствующего каталога в каталог `..\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\`.

- Для **32-битной операционной системы** скопируйте файлы из каталога
`...\\Terrasoft.WebApp\\DesktopBin\\WorkspaceConsole\\x86` .
- Для **64-битной операционной системы** скопируйте файлы из каталога
`...\\Terrasoft.WebApp\\DesktopBin\\WorkspaceConsole\\x64` .

Использование утилиты WorkspaceConsole

Исполняемый файл утилиты `Terrasoft.Tools.WorkspaceConsole.exe` расположен в каталоге
`..\\Terrasoft.WebApp\\DesktopBin\\WorkspaceConsole`, а версия утилиты совпадает с версией приложения.

Важно. Версия утилиты `WorkspaceConsole` должна совпадать с версией приложения `Creatio`.

Например, если текущая версия приложения 7.18.1.1794, а пакеты необходимо обновить до версии 7.18.2.1658, то используйте утилиту версии 7.18.2.1658. Чтобы получить утилиту требуемой версии, обратитесь в службу поддержки.

Утилита `WorkspaceConsole` работает напрямую с базой данных приложения. Поэтому для корректной работы утилиты необходимо внести информацию о базе данных в конфигурационный файл `Terrasoft.Tools.WorkspaceConsole.exe.config` утилиты. Если приложение развернуто в облаке, то работать с утилитой могут только сотрудники отдела облачных сервисов. В таком случае для переноса изменений необходимо обратиться в службу поддержки.

Команды для `WorkspaceConsole` рекомендуется формировать в пакетном файле (*.bat или *.cmd), созданном в текстовом редакторе.

Чтобы **выполнить перенос решений с помощью утилиты `WorkspaceConsole`**:

1. Проверьте привязки данных.
2. Выполните резервное копирование базы данных.
3. Экспортируйте пакеты.
4. Импортируйте пакеты.
5. Перезапустите приложение в IIS.

1. Проверить привязки данных

Перед экспортом пакета проверьте правильность [привязки данных](#) к пакету. К привязанным данным относятся наполнение справочников, новые системные настройки, демонстрационные записи раздела и т. д.

Если раздел был создан при помощи мастера, то данные, необходимые для работы раздела, автоматически привязываются мастером раздела. Чтобы после импорта раздел отобразился в рабочем месте, необходимо привязать соответствующее значение объекта `SysModuleInWorkplace` .

2. Выполнить резервное копирование базы данных

Перед внесением изменений в приложение с помощью утилиты `WorkspaceConsole` выполните резервное копирование базы данных. Это позволит восстановить приложение при некорректном использовании

команд и параметров утилиты.

3. Экспортировать пакеты

Утилита WorkspaceConsole позволяет экспортить пакет из базы данных или из хранилища SVN.

Экспортировать пакеты из базы данных

1. Сформируйте команду для экспорта пакетов.

Параметры WorkspaceConsole для экспорта пакетов из базы данных

Параметр	Значение	Описание
<code>-operation</code>	<code>SaveDBContent</code>	Сохраняет содержимое базы данных в файловую систему. Тип содержимого определяется значением параметра <code>-contentTypes</code> . Место в файловой системе, куда будет экспортировано содержимое, определяется параметром <code>-destinationPath</code> . Требует указания одного из параметров <code>-webApplicationPath</code> или <code>-configurationPath</code> .
<code>-contentTypes</code>	<code>Repository</code>	Тип содержимого, экспортируемого из базы данных на диск. Значение <code>Repository</code> определяет экспорт рабочего пространства, имя которого задается значением параметра <code>-workspaceName</code> , в каталог, путь к которому задается значением параметра <code>-destinationPath</code> .
<code>-workspaceName</code>	[Название рабочего пространства]	Название рабочего пространства (конфигурации), в котором выполняется операция. По умолчанию все пользователи работают в рабочем пространстве <code>Default</code> .
<code>-destinationPath</code>	[Путь к локальному каталогу]	Путь к локальному каталогу на диске. В этот каталог будут экспортированы *.gz-архивы пакетов.
<code>-webApplicationPath</code>	[Путь к локальному каталогу]	Путь к каталогу на диске, в который установлено приложение Creatio. По этому пути из файла <code>ConnectionStrings.config</code> будет считана информация по соединению с базой данных. Если параметр не указан, то будет установлено соединение с базой данных, указанной в строке соединения в конфигурационном файле утилиты.
<code>-configurationPath</code>	[Путь к локальному каталогу]	Путь к каталогу <code>..\Terrasoft.WebApp\Terrasoft.Configuration</code> . В этот каталог экспортируются исходные коды и ресурсы схем пользовательских пакетов в режиме разработки в файловой системе .

Сигнатура команды, которую необходимо выполнить в интерпретаторе команд (консоли) Windows, для выполнения операции экспортации пакетов из базы данных:

```
[Путь к WorkspaceConsole]\Terrasoft.Tools.WorkspaceConsole.exe -operation=SaveDBContent -cont
```

2. Запустите утилиту.

На заметку. Во время выполнения операции экспорта пакетов, экспортируются все пакеты рабочего пространства. Процесс может занять несколько десятков минут.

В результате выполнения архивы пакетов указанного рабочего пространства будут экспортаны в локальный каталог.

Экспортировать пакеты из хранилища SVN

1. Сформируйте команду для экспорта пакетов.

Параметры WorkspaceConsole для экспорта пакетов из SVN

Параметр	Значение	Описание
<code>-operation</code>	<code>SaveVersionSvnContent</code>	Выгружает иерархию пакетов в виде *.zip-архивов. Место в файловой системе, куда будет экспортано содержимое, определяется параметром <code>-destinationPath</code> . SVN-хранилища определяются параметром <code>-sourcePath</code> .
<code>-workspaceName</code>	[Название рабочего пространства]	Название рабочего пространства (конфигурации), в котором выполняется операция. По умолчанию все пользователи работают в рабочем пространстве <code>Default</code> .
<code>-destinationPath</code>	[Путь к локальному каталогу]	Путь к локальному каталогу на диске. В этот каталог будут экспортаны *.gz-архивы пакетов.
<code>-workingCopyPath</code>	[Путь к локальному каталогу]	Локальный каталог для рабочей копии пакетов, которые хранятся в SVN.
<code>-sourcePath</code>	[Путь к SVN-хранилищу]	Адрес хранилища SVN для хранения структуры и метаданных пакетов. Может принимать несколько значений, указанных через запятую.
<code>-packageName</code>	[Имя пакета]	Имя пакета в SVN-хранилище, которое будет использоваться для экспорта. Все пакеты, от которых зависит

текущий пакет, также будут задействованы.

-packageVersion	[Версия пакета]	Версия пакета в SVN-хранилище, которое будет использоваться для экспорта.
-sourceControlLogin	[Имя пользователя SVN]	Логин пользователя хранилища SVN.
-sourceControlPassword	[Пароль пользователя SVN]	Пароль пользователя хранилища SVN.
-cultureName	[Языковая культура]	Код языковой культуры. Например, en-US .
-excludeDependentPackages	true ИЛИ false	Признак необходимости экспорта пакетов, от которых зависит пакет, указанный в параметре -packageName .
-logPath	[Путь к локальному каталогу]	Путь к каталогу, в который будет сохранен файл с логом операции. Название файла состоит из даты и времени запуска операции. Необязательный параметр.

Сигнатура команды, которую необходимо выполнить в интерпретаторе команд (консоли) Windows, для выполнения операции экспорта пакетов из SVN-хранилища:

```
[Путь к WorkspaceConsole]\Terrasoft.Tools.WorkspaceConsole.exe -operation=SaveVersionSvnConte
```

2. Запустите утилиту.

В результате выполнения архивы пакетов указанного рабочего пространства будут экспортированы в локальный каталог.

4. Импортировать пакеты

1. Сформируйте команду для импорта пакетов.

Параметры WorkspaceConsole для импорта пакетов в базу данных

Параметр	Значение	Описание
-operation	InstallFromRepository	Импортирует в конфигурацию содержимое и метаданные пакетов из *.zip-архивов. При необходимости

		выполняются привязанные SQL-скрипты, перегенерация исходных кодов, установка привязанных данных. Работает только с измененными или новыми пакетами и их элементами. Требует указания одного из параметров <code>-webApplicationPath</code> или <code>-configurationPath</code> . Требует указания параметра <code>-confRuntimeParentDirectory</code> .
<code>-packageName</code>	[Имя пакета]	Имя пакета в конфигурации, которая указана в параметре <code>-workspaceName</code> . Все пакеты, от которых зависит текущий пакет, также будут задействованы. Если параметр не указан, то будут задействованы все пакеты конфигурации.
<code>-workspaceName</code>	[Название рабочего пространства]	Название рабочего пространства (конфигурации), в котором выполняется операция. По умолчанию все пользователи работают в рабочем пространстве <code>Default</code> .
<code>-sourcePath</code>	[Путь к локальному каталогу]	Путь к локальному каталогу на диске. В этом каталоге находятся *.gz-архивы пакетов, которые необходимо установить.
<code>-destinationPath</code>	[Путь к локальному каталогу]	Путь к локальному каталогу на диске. В этот каталог будут экспортированы *.gz-архивы пакетов, которые определены в параметре <code>-sourcePath</code> .
<code>-skipConstraints</code>	<code>false</code>	Пропустить создание внешних ключей в таблицах базы данных. Принимает значения <code>true</code> или <code>false</code> .
<code>-skipValidateActions</code>	<code>true</code>	Пропустить проверку возможности создания индексов таблиц при обновлении структуры базы данных. Принимает значения <code>true</code> или <code>false</code> .
<code>-regenerateSchemaSources</code>	<code>true</code>	Указывает на необходимость

<code>-updateDBStructure</code>	<code>true</code>	перегенерации исходных кодов после сохранения пакетов в базе данных. Принимает значения <code>true</code> или <code>false</code> . По умолчанию — <code>true</code> .
<code>-updateSystemDBStructure</code>	<code>true</code>	Указывает на необходимость обновления структуры базы данных после сохранения пакетов. Принимает значения <code>true</code> ИЛИ <code>false</code> . По умолчанию — <code>true</code> .
<code>-installPackageSqlScript</code>	<code>true</code>	Указывает на необходимость изменения структуры базы данных системных схем перед выполнением установки пакетов. Также создает все отсутствующие индексы в системных таблицах. Принимает значения <code>true</code> ИЛИ <code>false</code> .
<code>-installPackageData</code>	<code>true</code>	Указывает на необходимость выполнения SQL скриптов до и после сохранения пакетов. Принимает значения <code>true</code> ИЛИ <code>false</code> . По умолчанию — <code>true</code> .
<code>-continueIfError</code>	<code>true</code>	Указывает на необходимость прервать выполнение процесса установки при получении первой ошибки. Если значение параметра — <code>true</code> , то процесс установки пройдет до конца, а пользователь получит список всех возникших ошибок. Принимает значения <code>true</code> ИЛИ <code>false</code> . По умолчанию — <code>false</code> .
<code>-webApplicationPath</code>	[Путь к локальному каталогу]	Путь к каталогу на диске, в который установлено приложение Creatio. По этому пути из файла <code>.ConnectionStrings.config</code> будет считана информация по соединению с базой данных. Если параметр не

		указан, то будет установлено соединение с базой данных, указанной в строке соединения в конфигурационном файле утилиты.
-confRuntimeParentDirectory	[Путь к локальному каталогу]	Путь к родительскому каталогу директории <code>..\Terrasoft.WebApp\conf</code> .
-logPath	[Путь к локальному каталогу]	Путь к каталогу, в который будет сохранен файл с логом операции. Название файла состоит из даты и времени запуска операции. Необязательный параметр.
-configurationPath	[Путь к локальному каталогу]	Путь к каталогу <code>..\Terrasoft.WebApp\Terrasoft.Configuration</code> . В этот каталог экспортируются исходные коды и ресурсы схем пользовательских пакетов в режиме разработки в файловой системе .

Сигнатура команды, которую необходимо выполнить в интерпретаторе команд (консоли) Windows, для выполнения операции импорта пакетов в базу данных:

```
[Путь к WorkspaceConsole]\Terrasoft.Tools.WorkspaceConsole.exe -operation=InstallFromReposito
```

2. Запустите утилиту.
3. Сформируйте команду для генерации статического контента в файловую систему.

Параметры WorkspaceConsole для компиляции конфигурации

Параметр	Значение	Описание
-operation	<code>BuildConfiguration</code>	Выполняет генерацию статического контента в файловую систему . Требует указания одного из параметров <code>-webApplicationPath</code> или <code>-configurationPath</code> .
-workspaceName	[Название рабочего пространства]	Название рабочего пространства (конфигурации), в котором определены экспортируемые пакеты. По умолчанию все пользователи работают в рабочем пространстве <code>Default</code> .
-destinationPath	[Путь к	Путь к локальному каталогу на диске. В

	[Путь к локальному каталогу]	этот каталог будут экспортированы *.gz-архивы пакетов, которые определены в параметре <code>-sourcePath</code> .
<code>-webApplicationPath</code>	[Путь к локальному каталогу]	Путь к каталогу на диске, в который установлено приложение Creatio. По этому пути из файла <code>ConnectionStrings.config</code> будет считана информация по соединению с базой данных. Если параметр не указан, то будет установлено соединение с базой данных, указанной в строке соединения в конфигурационном файле утилиты.
<code>-confRuntimeParentDirectory</code>	[Путь к локальному каталогу]	Путь к родительскому каталогу директории <code>..\Terrasoft.WebApp\conf</code> .
<code>-logPath</code>	[Путь к локальному каталогу]	Путь к каталогу, в который будет сохранен файл с логом операции. Название файла состоит из даты и времени запуска операции. Необязательный параметр.
<code>-force</code>	<code>true</code> или <code>false</code>	Задает условия генерации файлового контента. Если значение параметра равно <code>true</code> , то генерация файлового контента выполняется по всем схемам. Если значение параметра равно <code>false</code> , то генерация файлового контента выполняется по измененным схемам. По умолчанию — <code>false</code> . Требует указания одного из параметров <code>-webApplicationPath</code> или <code>-configurationPath</code> .
<code>-configurationPath</code>	[Путь к локальному каталогу]	Путь к каталогу <code>..\Terrasoft.WebApp\Terrasoft.Configuration</code> . В этот каталог экспортируются исходные коды и ресурсы схем пользовательских пакетов в режиме разработки в файловой системе .

Сигнатура команды, которую необходимо выполнить в интерпретаторе команд (консоли) Windows, для выполнения операции генерации статического контента в файловую систему:

```
[Путь к WorkspaceConsole]\Terrasoft.Tools.WorkspaceConsole.exe -operation=BuildConfiguration
```

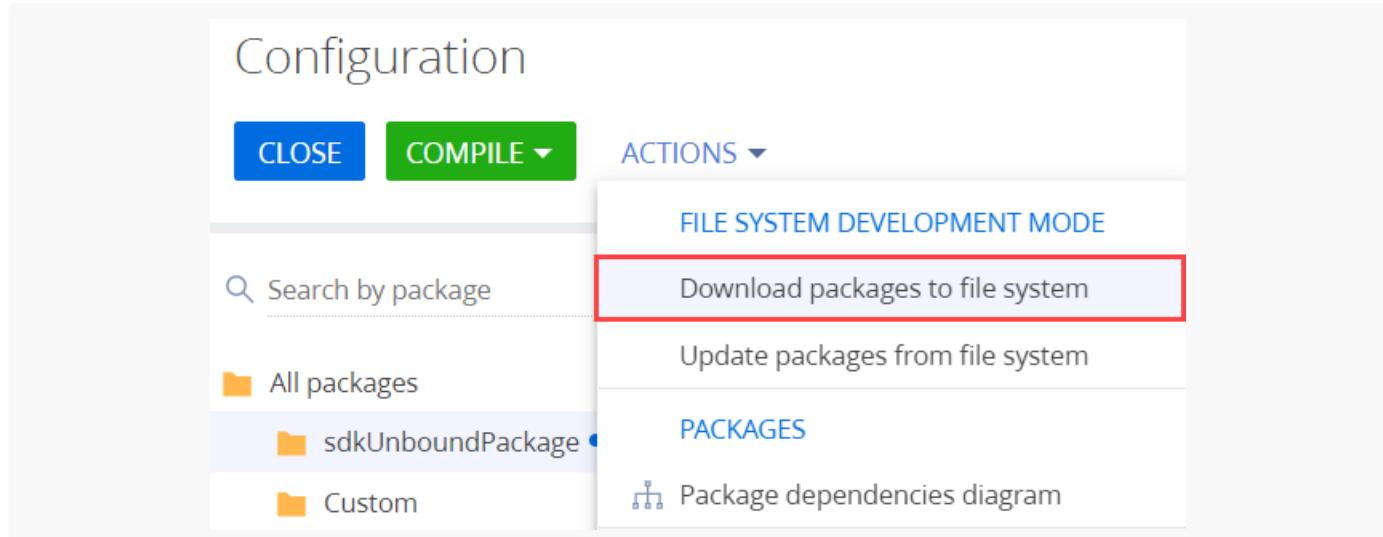
4. Запустите утилиту.

На заметку. Пакеты, импортируемые в приложение с помощью WorkspaceConsole, считаются предустановленными и недоступны для изменения.

Не рекомендуется использовать утилиту WorkspaceConsole для импорта пакетов в базу данных при **включенном режиме разработки в файловой системе**. Если утилита будет использована, то исходный код измененных схем будет изменен в базе данных, но останется без изменений в файловой системе. Т. е. при открытии схемы конфигурационного элемента в Creatio IDE отобразится неизмененный код из файловой системы. При этом дата модификации схемы конфигурационного элемента будет обновлена. Это приводит к ложному ощущению, что перенос схемы прошел корректно.

Чтобы импортировать пакеты в базу данных при **включенном режиме разработки в файловой системе**:

1. Воспользуйтесь [инструкцией](#) и импортируйте пакеты в базу данных с помощью утилиты WorkspaceConsole.
2. На панели инструментов в группе действий [Разработка в файловой системе] ([File system development mode]) выберите [Выгрузить все пакеты в файловую систему] ([Download packages to file system]).



В результате все пакеты будут выгружены по пути `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в каталог с соответствующим названием пакета.

5. Перезапустить приложение в IIS

Для применения изменений перезапустите приложение в IIS. Это необходимо, поскольку утилита WorkspaceConsole вносит изменения напрямую в базу данных.

Экспортировать пакеты из базы данных

ЭКСПОРТИРОВАТЬ ПАКЕТЫ ИЗ БАЗЫ ДАННЫХ



Средний

Пример. Экспортировать в каталог все пакеты рабочего пространства `Default`.

- `C:\creatio` — каталог с установленным приложением.
- `C:\SavedPackages` — каталог для экспорта пакетов.
- `C:\Logs` — каталог для экспорта файла с логом операции.

1. Сформировать команду для экспорта пакета из базы данных

1. Создайте файл пакетных команд Windows (*.bat или *.cmd), используя текстовый редактор.
2. В созданный файл добавьте команду для запуска утилиты.

Команда для запуска утилиты

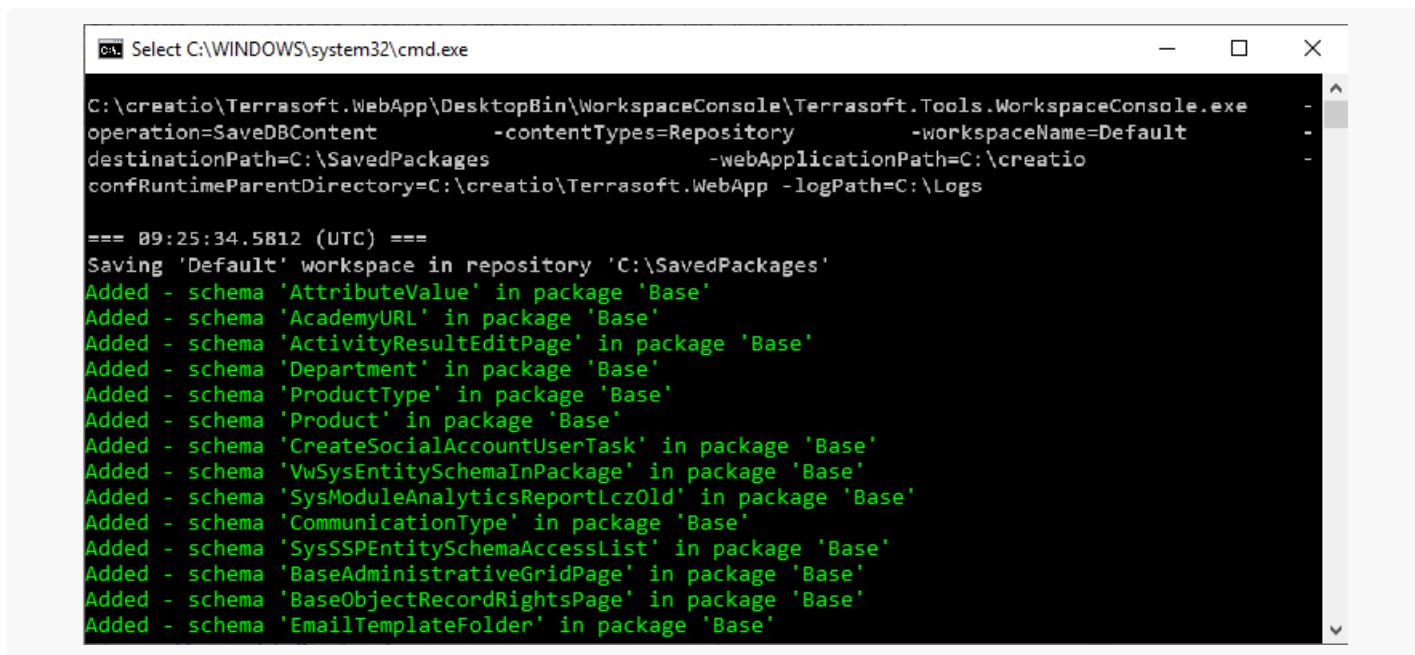
```
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe
pause
```

Сохраните пакетный файл.

2. Экспортировать пакет из базы данных

Чтобы **экспортировать пакет из базы данных** дважды кликните по имени пакетного файла.

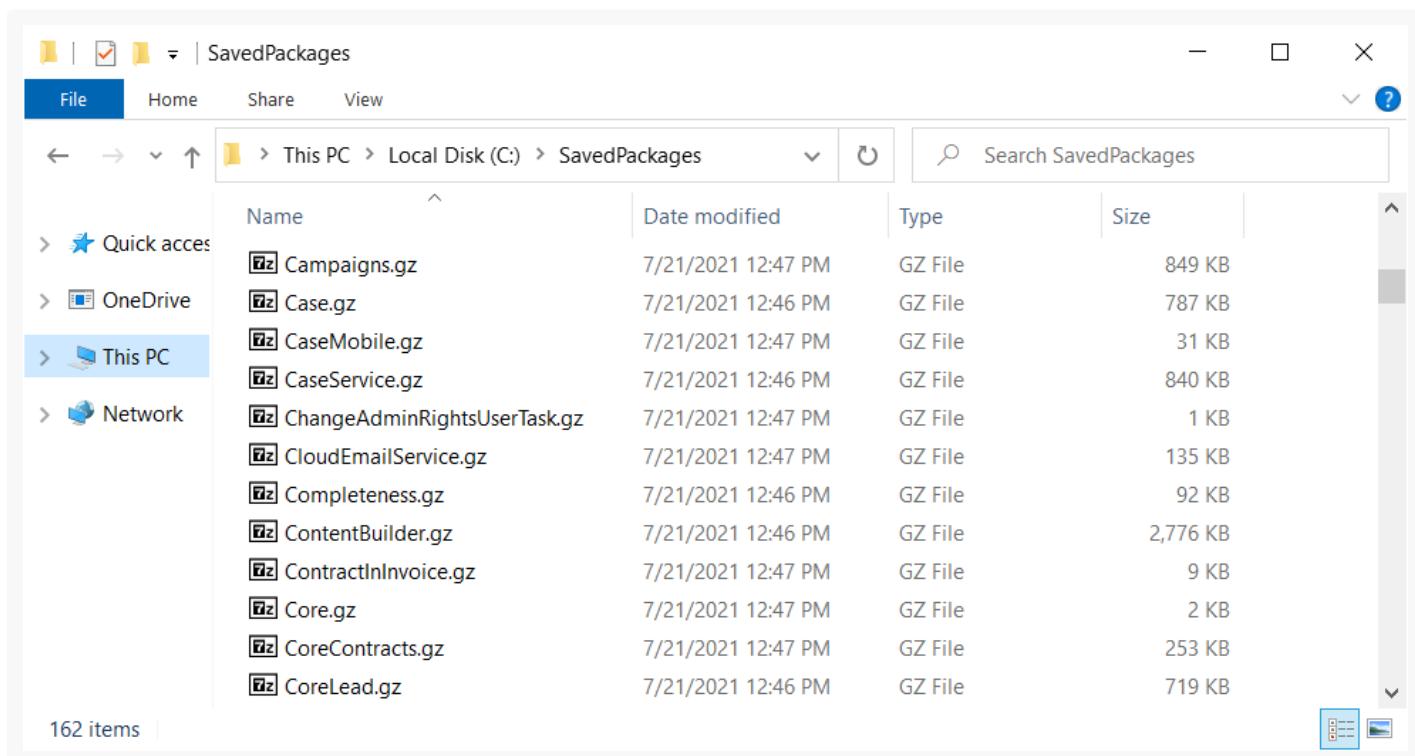
После этого появится консольное окно, в котором будет отображаться процесс выполнения операции, указанной в соответствующей команде `WorkspaceConsole`.



```
cmd Select C:\WINDOWS\system32\cmd.exe
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe
operation=SaveDBContent -contentTypes=Repository -workspaceName=Default
destinationPath=C:\SavedPackages -webApplicationPath=C:\creatio
confRuntimeParentDirectory=C:\creatio\Terrasoft.WebApp -logPath=C:\Logs

== 09:25:34.5812 (UTC) ==
Saving 'Default' workspace in repository 'C:\SavedPackages'
Added - schema 'AttributeValue' in package 'Base'
Added - schema 'AcademyURL' in package 'Base'
Added - schema 'ActivityResultEditPage' in package 'Base'
Added - schema 'Department' in package 'Base'
Added - schema 'ProductType' in package 'Base'
Added - schema 'Product' in package 'Base'
Added - schema 'CreateSocialAccountUserTask' in package 'Base'
Added - schema 'VwSysEntitySchemaInPackage' in package 'Base'
Added - schema 'SysModuleAnalyticsReportLczOld' in package 'Base'
Added - schema 'CommunicationType' in package 'Base'
Added - schema 'SysSSPEntitySchemaAccessList' in package 'Base'
Added - schema 'BaseAdministrativeGridPage' in package 'Base'
Added - schema 'BaseObjectRecordRightsPage' in package 'Base'
Added - schema 'EmailTemplateFolder' in package 'Base'
```

В результате выполнения команды в каталог `C:\SavedPackages` будут экспортированы *gz-архивы всех пакетов конфигурации `Default`.



Экспортировать пакет из SVN



Средний

Пример. Экспортировать пакет из SVN-хранилища в каталог рабочего пространства `Default`.

- `C:\creatio` — каталог с установленным приложением.
- `C:\SavedPackages` — каталог для экспорта *gz-архива пакета.
- `C:\WorkingCopy` — каталог для экспорта структуры пакета из SVN-хранилища.
- `http://server-svn:8050/Packages` — адрес SVN-хранилища.
- `sdkTestPackage` — пакет для экспорта из SVN-хранилища.
- `7.18.1` — версия пакета для экспорта из SVN-хранилища.
- "User" — логин пользователя хранилища SVN.
- "Password" — пароль пользователя хранилища SVN.
- `ru-RU` — языковая культура.
- `c:\Logs` — каталог для экспорта файла с логом операции.

1. Сформировать команду для экспорта пакета из SVN-хранилища

1. Создайте файл пакетных команд Windows (*.bat или *.cmd), используя текстовый редактор.
2. В созданный файл добавьте команду для запуска утилиты.

Команда для запуска утилиты

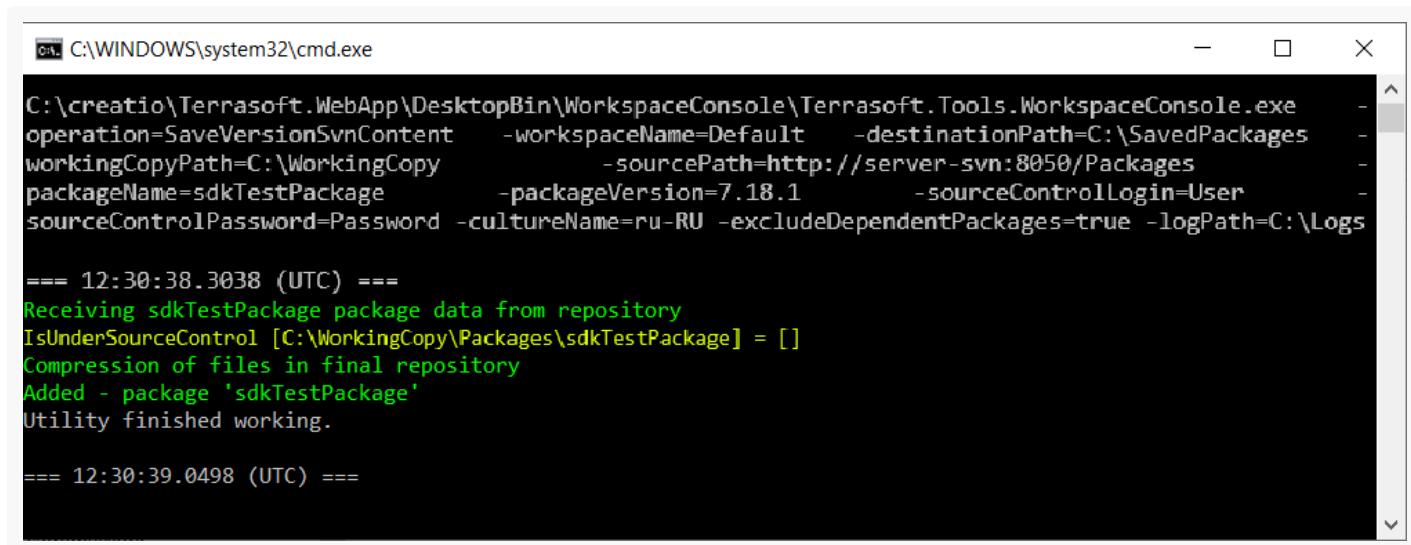
```
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe
pause
```

Сохраните пакетный файл.

2. Экспортировать пакет из SVN

Чтобы **экспортировать пакет из SVN** дважды кликните по имени пакетного файла.

После этого появится консольное окно, в котором будет отображаться процесс выполнения операции, указанной в соответствующей команде WorkspaceConsole.

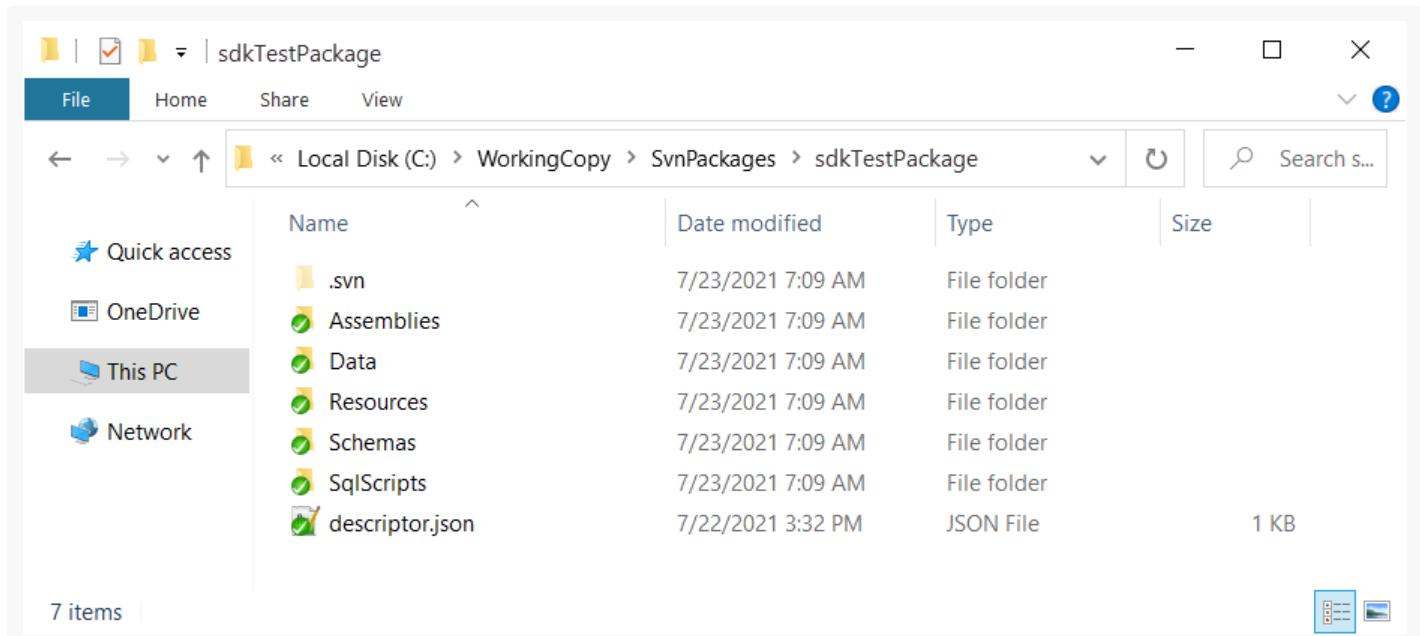


```
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe
operation=SaveVersionSvnContent -workspaceName=Default -destinationPath=C:\SavedPackages
workingCopyPath=C:\WorkingCopy -sourcePath=http://server-svn:8050/Packages
packageName=sdkTestPackage -packageVersion=7.18.1 -sourceControlLogin=User
sourceControlPassword=Password -cultureName=ru-RU -excludeDependentPackages=true -logPath=C:\Logs

--- 12:30:38.3038 (UTC) ---
Receiving sdkTestPackage package data from repository
IsUnderSourceControl [C:\WorkingCopy\Packages\sdkTestPackage] = []
Compression of files in final repository
Added - package 'sdkTestPackage'
Utility finished working.

--- 12:30:39.0498 (UTC) ---
```

В результате выполнения команды в каталог `C:\SavedPackages` будет экспортирован пакет `sdkTestPackage` конфигурации `Default`. Структура папки с именем пакета описана в статье [Общие принципы работы с пакетами](#).



Импортировать пакет в базу данных

 Средний

Пример. Импортировать пакет из каталога в рабочее пространство `Default`.

- `C:\creatio` — каталог с установленным приложением.
- `sdkTestPackage` — пакет для импорта в приложение, который находится по пути `C:\SavedPackages`

- `C:\SavedPackages` — каталог с импортированными пакетами.
- `C:\TempPackages` — каталог для импорта пакета.
- `C:\Logs` — каталог для экспорта файла с логом операции.

1. Сформировать команду для импорта пакета в базу данных

1. Создайте файл пакетных команд Windows (*.bat или *.cmd), используя текстовый редактор.
2. В созданный файл добавьте команду для запуска утилиты.

Команда для запуска утилиты

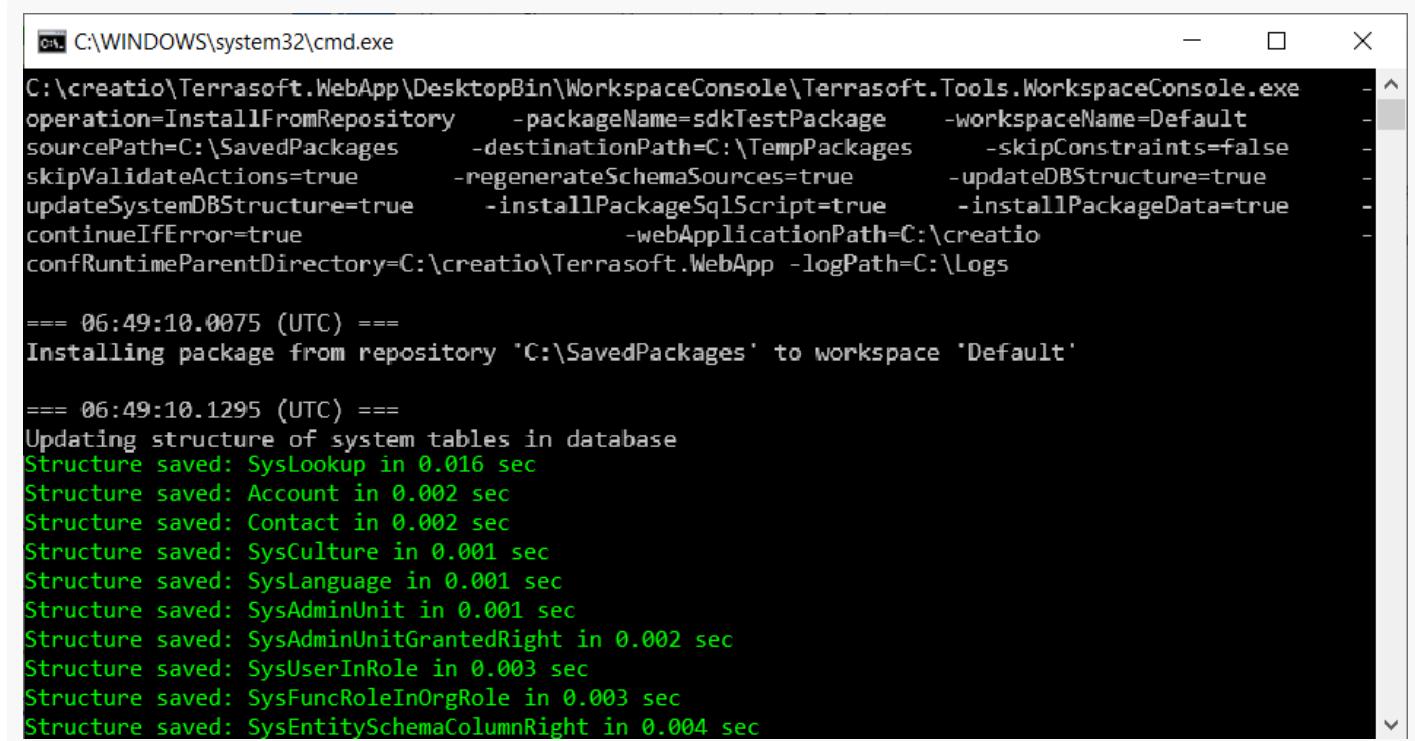
```
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe
pause
```

Сохраните пакетный файл.

2. Импортировать пакет в базу данных

Чтобы **импортировать пакет в базу данных** дважды кликните по имени пакетного файла.

После этого появится консольное окно, в котором будет отображаться процесс выполнения операции, указанной в соответствующей команде WorkspaceConsole.

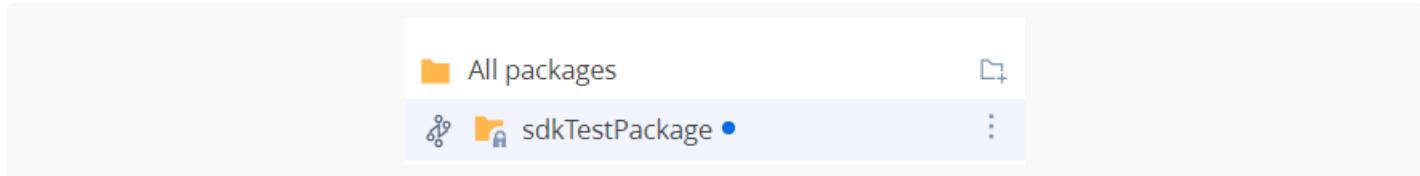


```
C:\WINDOWS\system32\cmd.exe
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe
operation=InstallFromRepository -packageName=sdkTestPackage -workspaceName=Default
sourcePath=C:\SavedPackages -destinationPath=C:\TempPackages -skipConstraints=false
skipValidateActions=true -regenerateSchemaSources=true -updateDBStructure=true
updateSystemDBStructure=true -installPackageSqlScript=true -installPackageData=true
continueIfError=true -webApplicationPath=C:\creatio
confRuntimeParentDirectory=C:\creatio\Terrasoft.WebApp -logPath=C:\Logs

== 06:49:10.0075 (UTC) ==
Installing package from repository 'C:\SavedPackages' to workspace 'Default'

== 06:49:10.1295 (UTC) ==
Updating structure of system tables in database
Structure saved: SysLookup in 0.016 sec
Structure saved: Account in 0.002 sec
Structure saved: Contact in 0.002 sec
Structure saved: SysCulture in 0.001 sec
Structure saved: SysLanguage in 0.001 sec
Structure saved: SysAdminUnit in 0.001 sec
Structure saved: SysAdminUnitGrantedRight in 0.002 sec
Structure saved: SysUserInRole in 0.003 sec
Structure saved: SysFuncRoleInOrgRole in 0.003 sec
Structure saved: SysEntitySchemaColumnRight in 0.004 sec
```

В результате выполнения команды в конфигурацию `Default` будет импортирован пакет `sdkTestPackage`.



3. Сформировать команду для генерации статического контента в файловую систему

1. Создайте файл пакетных команд Windows (*.bat или *.cmd), используя текстовый редактор.
2. В созданный файл добавьте команду для запуска утилиты.

Команда для запуска утилиты

```
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe  
pause
```

Сохраните пакетный файл.

4. Сгенерировать статический контент в файловую систему

Чтобы **сгенерировать статический контент в файловую систему** дважды кликните по имени пакетного файла.

После этого появится консольное окно, в котором будет отображаться процесс выполнения операции, указанной в соответствующей команде `WorkspaceConsole`.

```
C:\creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasoft.Tools.WorkspaceConsole.exe
operation=BuildConfiguration -workspaceName=Default -destinationPath=C:\creatio\Terrasoft.WebApp
webApplicationPath=C:\creatio -confRuntimeParentDirectory=C:\creatio\Terrasoft.WebApp
configurationPath=C:\creatio\Terrasoft.WebApp\Terrasoft.Configuration -logPath=C:\Logs

--- 07:35:34.6758 (UTC) ---
Configuration build started
Operation [BuildChanged] started
Operation [GenerateFileContentDescriptors] started
Operation [GenerateFileContentDescriptors] took 5 s 270 ms
Operation [GenerateFileContentBootstraps] started
Operation [GenerateFileContentBootstraps] took 0 s 39 ms
Ordered by CPU time
[ClientUnitSchemaManager.AccountPageV2] - Total: 5899 ms, GetInstance: 564 ms, Less: 1 ms, Images: 1559 ms,
SourceCode: 3610 ms, Resources: 165 ms
[ClientUnitSchemaManager.UsrBook1Section] - Total: 4952 ms, GetInstance: 611 ms, Less: 55 ms, Images: 3886 m
s, SourceCode: 77 ms, Resources: 323 ms
[ClientUnitSchemaManager.AccountSectionV2] - Total: 4318 ms, GetInstance: 321 ms, Less: 1 ms, Images: 0 ms,
SourceCode: 3277 ms, Resources: 719 ms
```

В результате выполнения команды в файловую систему будет сгенерирован статический контент измененных схем.

Параметры утилиты WorkspaceConsole CLI



Средний

`-help`

Справка с полным перечнем параметров утилиты и их кратким описанием. Если указать другие параметры, то они будут проигнорированы.

`-operation`

Название операции, которую необходимо выполнить. Обязательный параметр. Значение по умолчанию — `LoadLicResponse`.

Возможные значения

`LoadLicResponse`

Загружает лицензии в базу данных, указанную в строке подключения. Единственная операция, которая не требует указания параметра `-workspaceName`.

`SaveRepositoryContent`

Сохраняет указанное в параметре `-contentTypes` содержимое *.zip-архивов пакетов из каталога, указанного в параметре `-sourcePath`, в каталог, указанный в параметре `-destinationPath`.

SaveDBContent	Сохраняет содержимое базы данных в файловую систему. Тип содержимого определяется значением параметра <code>-contentTypes</code> . Место в файловой системе, куда будет экспортировано содержимое, определяется параметром <code>-destinationPath</code> . Требует указания одного из параметров <code>-webApplicationPath</code> ИЛИ <code>-configurationPath</code> .
SaveVersionSvnContent	Выгружает иерархию пакетов в виде *.zip-архивов. Место в файловой системе, куда будет экспортировано содержимое, определяется параметром <code>-destinationPath</code> . SVN-хранилища определяются параметром <code>-sourcePath</code> .
RegenerateSchemaSources	Выполняет перегенерацию исходных кодов и их компиляцию. Требует указания параметра <code>-confRuntimeParentDirectory</code> .
InstallFromRepository	Импортирует в конфигурацию содержимое и метаданные пакетов из *.zip-архивов. При необходимости выполняются привязанные SQL-скрипты, перегенерация исходных кодов, установка привязанных данных. Работает только с измененными или новыми пакетами и их элементами. Требует указания одного из параметров <code>-webApplicationPath</code> ИЛИ <code>-configurationPath</code> . Требует указания параметра <code>-confRuntimeParentDirectory</code> .
InstallBundlePackages	Устанавливает набор пакетов, перечисленных через запятую в параметре <code>-packageName</code> , в рабочее пространство, указанное в параметре <code>-workspaceName</code> . Требует указания одного из параметров <code>-webApplicationPath</code> ИЛИ <code>-configurationPath</code> . Требует указания параметра <code>-confRuntimeParentDirectory</code> .
PrevalidateInstallFromRepository	Проверяет возможность установки пакетов из *.zip-архивов.
ConcatRepositories	Слияние нескольких репозиториев.
ConcatSVNRepositories	Слияние нескольких хранилищ SVN.
ExecuteProcess	Выполняет запуск бизнес-процесса в конфигурации, если он будет найден. Требует указания параметра <code>-confRuntimeParentDirectory</code> .
UpdatePackages	В базе данных выполняет обновление пакетов (параметр

	-packageName), которые присутствуют в иерархии пакета продукта (параметр -productPackageName).
	Требует указания одного из параметров -webApplicationPath ИЛИ -configurationPath .
	Требует указания параметра -confRuntimeParentDirectory .
BuildWorkspace	Компилирует рабочее пространство (конфигурацию). Используется при разработке схем с помощью Visual Studio . Требует указания параметра -confRuntimeParentDirectory .
ReBuildWorkspace	Перекомпилирует рабочее пространство (конфигурацию). Используется при разработке схем с помощью Visual Studio . Требует указания параметра -confRuntimeParentDirectory .
UpdateWorkspaceSolution	Обновляет решение и файлы проекта Visual Studio .
BuildConfiguration	Выполняет генерацию статического контента в файловую систему . Используемые параметры: <ul style="list-style-type: none"> • -workspaceName • -destinationPath • -webApplicationPath • -logPath • -force • -configurationPath Требует указания одного из параметров -webApplicationPath ИЛИ -configurationPath .

Важно. Для корректной работы приложения после выполнения операций `InstallFromRepository`, `InstallBundlePackages`, `UpdatePackages` необходимо выполнить операцию `BuildConfiguration`.

-user

Имя пользователя для выполнения авторизации. Указывается, если эта информация отсутствует в конфигурационном файле утилиты или необходимо выполнить операцию от имени другого пользователя.

-password

Пароль пользователя для выполнения авторизации. Указывается, если эта информация отсутствует в конфигурационном файле утилиты или необходимо выполнить операцию от имени другого пользователя.

-workspaceName

Название рабочего пространства (конфигурации), в котором определены экспортируемые пакеты. По умолчанию все пользователи работают в рабочем пространстве `Default`.

-autoExit

Указывает, завершать ли автоматически процесс утилиты после выполнения операции. Принимает значения `true` или `false`. По умолчанию — `false`.

-processName

Имя процесса, который необходимо запустить.

-repositoryUri

Адрес хранилища SVN для хранения структуры и метаданных пакетов. Необязательный параметр. При указании параметра его значение используется вместо значения параметра, указанного в параметре

`-workspaceName`.

-sourceControlLogin

Логин пользователя хранилища SVN.

-sourceControlPassword

Пароль пользователя хранилища SVN.

-workingCopyPath

Локальный каталог для рабочей копии пакетов, которые хранятся в SVN.

-contentTypes

Тип содержимого, экспортируемого из базы данных на диск.

Возможные значения

SystemData	Данные системных схем в формате JSON. Выгружаются все системные схемы и их колонки, кроме указанных в параметре <code>-excludedSchemas</code> .
ConfigurationData	Данные схем конфигурационных элементов в формате JSON. Выгружаются все схемы, кроме указанных в параметре <code>-excludedSchemas</code> .
Resources	Ресурсы схем конфигурационных элементов для локализации в формате XML.
LocalizableData	Наполнение схем конфигурационных элементов для локализации в формате XML. Выгружаются только текстовые колонки. Дополнительные ограничения указываются в параметрах <code>-excludedSchemas</code> И <code>-excludedSchemaColumns</code> .
Repository	Данные рабочего пространства в формате zip.
SqlScripts	SQL-скрипты, привязанные к пакетам.
Data	Данные системных схем и схем конфигурационных элементов в формате JSON. Комбинация значений <code>SystemData</code> И <code>ConfigurationData</code> .
LocalizableSchemaData	Данные локализуемых объектов.
All	Все типы содержимого.

-sourcePath

Путь к локальному каталогу на диске, из которого необходимо забрать данные (например, пакеты, схемы или ресурсы). Для операций `ConcatRepositories` И `SaveVersionSvnContent` может принимать несколько значений, указанных через запятую.

-destinationPath

Путь к локальному каталогу на диске, в который необходимо сохранить данные (например, пакеты, схемы или ресурсы).

-webApplicationPath

Путь к каталогу на диске, в который установлено приложение Creatio. По этому пути из файла `ConnectionStrings.config` будет считана информация по соединению с базой данных. Если параметр не указан, то будет установлено соединение с базой данных, указанной в строке соединения в конфигурационном файле утилиты.

Для операций `BuildWorkspace`, `ReBuildWorkspace` И `UpdateWorkspaceSolution` параметр `-webApplicationPath` должен указывать на путь к каталогу `Terrasoft.WebApp`.

`-configurationPath`

Путь к каталогу `..\Terrasoft.WebApp\Terrasoft.Configuration`. В этот каталог экспортируются исходные коды и ресурсы схем пользовательских пакетов в режиме разработки в [файловой системе](#).

`-filename`

Имя файла. Обязательный параметр для операции `LoadLicResponse`.

`-excludedSchemas`

Названия схем конфигурационных элементов, которые необходимо исключить при выполнении операции.

`-excludedSchemaColumns`

Названия колонок схем конфигурационных элементов, которые необходимо исключить при выполнении операции.

`-excludedWorkspaceNames`

Названия рабочих пространств (конфигураций), которые необходимо исключить при выполнении операции.

`-includedSchemas`

Названия схем конфигурационных элементов, которые принудительно используются при выполнении операции.

`-includedSchemaColumns`

Названия колонок схем конфигурационных элементов, которые принудительно используются при выполнении операции.

`-cultureName`

Код языковой культуры. Обязательный параметр, если используются значения `Resources` И/ИЛИ `LocalizableData` для параметра `-contentTypes`.

`-schemaManagerNames`

Имена менеджеров схем конфигурационных элементов, которые используются для операций. Значение по умолчанию — `EntitySchemaManager`.

`-packageName`

Имя пакета в конфигурации, которая указана в параметре `-workspaceName`. Все пакеты, от которых зависит текущий пакет, также будут задействованы. Если параметр не указан, то будут задействованы все пакеты конфигурации.

`-clearWorkspace`

Указывает на необходимость очистки рабочего пространства перед обновлением. Принимает значения `true` или `false`. По умолчанию — `false`.

`-installPackageSqlScript`

Указывает на необходимость выполнения SQL скриптов до и после сохранения пакетов. Принимает значения `true` или `false`. По умолчанию — `true`.

`-installPackageData`

Указывает на необходимость установки привязанных к пакету данных после сохранения пакетов. Принимает значения `true` или `false`. По умолчанию — `true`.

`-updateDBStructure`

Указывает на необходимость обновления структуры базы данных после сохранения пакетов. Принимает значения `true` или `false`. По умолчанию — `true`.

`-regenerateSchemaSources`

Указывает на необходимость перегенерации исходных кодов после сохранения пакетов в базе данных. Принимает значения `true` или `false`. По умолчанию — `true`.

`-continueIfError`

Указывает на необходимость прервать выполнение процесса установки при получении первой ошибки. Если значение параметра — `true`, то процесс установки пройдет до конца, а пользователь получит список всех возникших ошибок. Принимает значения `true` или `false`. По умолчанию — `false`.

Для операций `InstallFromSvn` и `InstallFromRepository` необходимо использовать параметр `-continueIfError=false`. Поскольку эти операции работают с измененными или новыми пакетами и их элементами. Информация об измененных элементах получается на основании сравнения новой и существующей структур пакетов. По этой причине, если пользователь выполняет команду без указания

ключа `-continueOnError=true` и получает ошибку, то повторный запуск команды для той же конфигурации выполнится без ошибок, но не внесет изменения в базу данных. Причина — предыдущая операция синхронизировала структуры пакетов указанной конфигурации и хранилища, и измененных элементов пакетов в этой операции нет.

`-skipCompile`

Указывает на необходимость выполнения компиляции. Работает, если для параметра `-updateDBStructure` установлено значение `false`. Принимает значения `true` или `false`. По умолчанию — `false`.

`-autoUpdateConfigurationVersion`

Обновляет значение версии конфигурации в базе данных до версии приложения Creatio. Принимает значения `true` или `false`. По умолчанию — `false`.

`-warningsOnly`

При обнаружении ошибки выполнения операции утилита информирует об ошибке. Принимает значения `true` или `false`. По умолчанию — `false`.

`-confRuntimeParentDirectory`

Путь к родительскому каталогу директории `.. \Terrasoft.WebApp\conf`. Параметр необходимо использовать в командах, которые вызывают компиляцию приложения или генерацию статического контента:

- `RegenerateSchemaSources`
- `InstallFromRepository`
- `InstallBundlePackages`
- `UpdatePackages`
- `BuildWorkspace`
- `RebuildWorkspace`
- `ExecuteProcess`

NUnit



Сложный

Unit-тестирование ([модульное тестирование](#)) — процесс в программировании, позволяющий проверить работоспособность изолированных частей программы. Как правило, тесты пишутся разработчиками для каждого нетривиального метода разрабатываемого класса. Это позволяет обнаружить **регрессию исходного кода** — появление ошибок в уже протестированных частях программы.

Одним из фреймворков Unit-тестирования .NET-приложений является [NUnit](#) — среда Unit-тестирования с

открытым исходным кодом. Для ее интеграции с Visual Studio разработан специальный **адаптер**.

Варианты установки адаптера:

- Расширение к Visual Studio.
- NuGet-пакет проекта, в котором реализованы Unit-тесты.

Особенности использования адаптера NUnit **как расширения для Visual Studio**:

- Доступность для любого проекта тестов, поскольку адаптер становится частью IDE.
- Автоматическое обновление расширения.
- Необходимость установки для каждого участника команды, работающего над проектом тестов.

Особенности использования адаптера NUnit **как пакета NuGet**:

- Пакет является частью проекта Visual Studio и доступен для всех разработчиков, использующих проект.
- Необходимость установки для всех проектов Unit-тестов.

Работа с NUnit описана в официальной [документации NUnit](#).

Чтобы **создать Unit-тесты** для методов или свойств класса пользовательского пакета необходимо:

1. Установить адаптер NUnit для Visual Studio.
2. Перейти в [режим разработки в файловой системе](#).
3. Настроить проект Unit-тестов.
4. Создать тесты.
5. Выполнить тестирование.

Добавить тест для пользовательского класса



Сложный

Пример. Добавить тесты для пользовательского класса, реализованного в схеме типа [Исходный код] ([*Source code*]) `UsrNUnitSourceCode` пользовательского пакета `sdkNUnit`.

1. Установить адаптер NUnit для Visual Studio

Установить адаптер NUnit как расширение Visual Studio

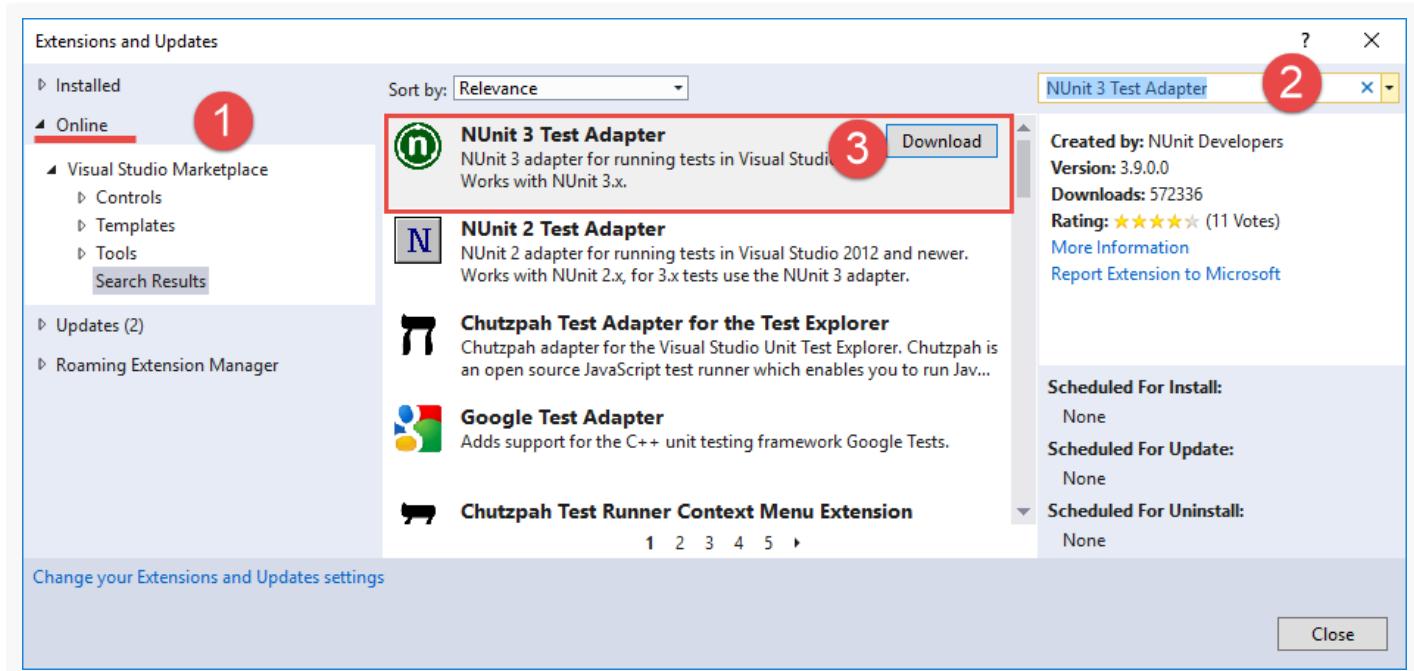
Способ 1:

1. [Скачайте расширение](#) из Visual Studio Marketplace.
2. Двойным кликом по *.vsix-файлу запустите установку.

3. Во время установки выберите необходимые версии Visual Studio.

Способ 2:

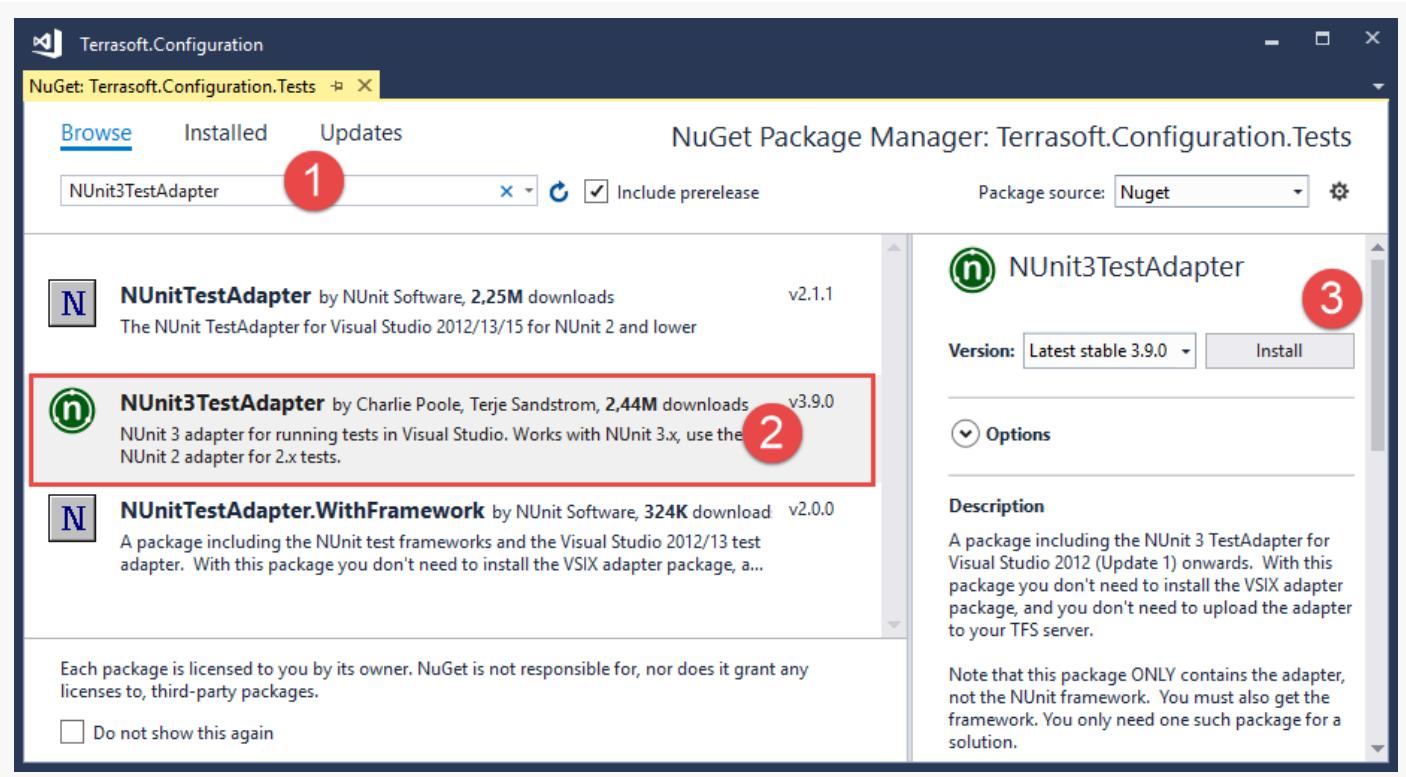
1. В Visual Studio нажмите [*Tools*] —> [*Extensions and Updates*].
2. Выберите фильтр [*Online*] (1).
3. В строке поиска укажите "NUnit 3 Test Adapter" (2).
4. В результатах поиска выберите расширение **NUnit 3 Test Adapter** (3).
5. Для установки расширения нажмите кнопку [*Download*].



Установить адаптер NUnit как пакет NuGet

1. Кликните правой кнопкой мыши по названию проекта тестов (например, `Terrasoft.Configuration.Tests.csproj`) и выберите [*Manage NuGet Packages...*].
2. В строке поиска укажите "NUnit3TestAdapter" (1).
3. В результатах поиска выберите пакет **NUnit 3 Test Adapter** (2).
4. Для установки пакета нажмите кнопку [*Install*] (3).

Установка пакета NuGet подробно описана в официальной [документации Microsoft](#).



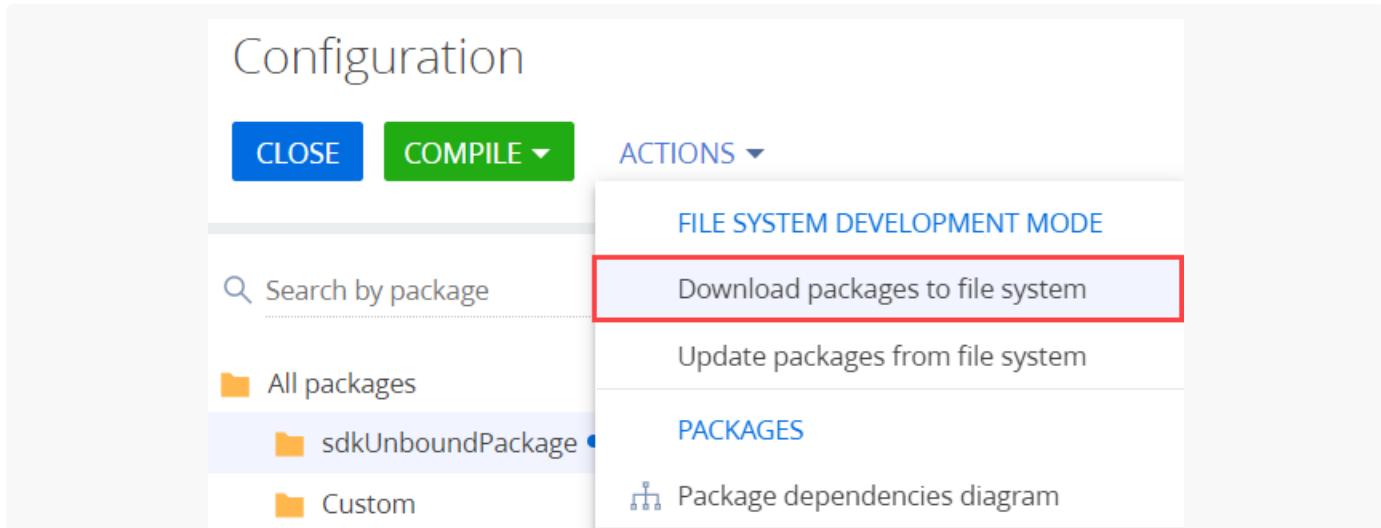
2. Выгрузить пакет в файловую систему

Создание Unit-тестов для .NET классов, реализованных в пакетах Creatio, возможно только в [режиме разработки в файловой системе](#).

В нашем примере используется пользовательский пакет `sdkNUnit`.

Чтобы **выгрузить пакет** в файловую систему:

1. Настройте Creatio для работы в файловой системе. Настройка описана в статье [Внешние IDE](#).
2. На панели инструментов в группе действий [*Разработка в файловой системе*] ([*File system development mode*]) выберите [*Выгрузить все пакеты в файловую систему*] ([*Download packages to file system*]).



В результате все пакеты будут выгружены по пути `..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в каталог с соответствующим названием пакета. Пакет `sdkNUnit` содержит схему типа [Исходный код] ([Source code]) `UsrNUnitSourceCode`.

В исходном коде схемы `UsrNUnitSourceCode` реализован класс `UsrNUnitSourceCode`, содержащий методы, для которых необходимо написать тесты.

UsrNUnitSourceCode

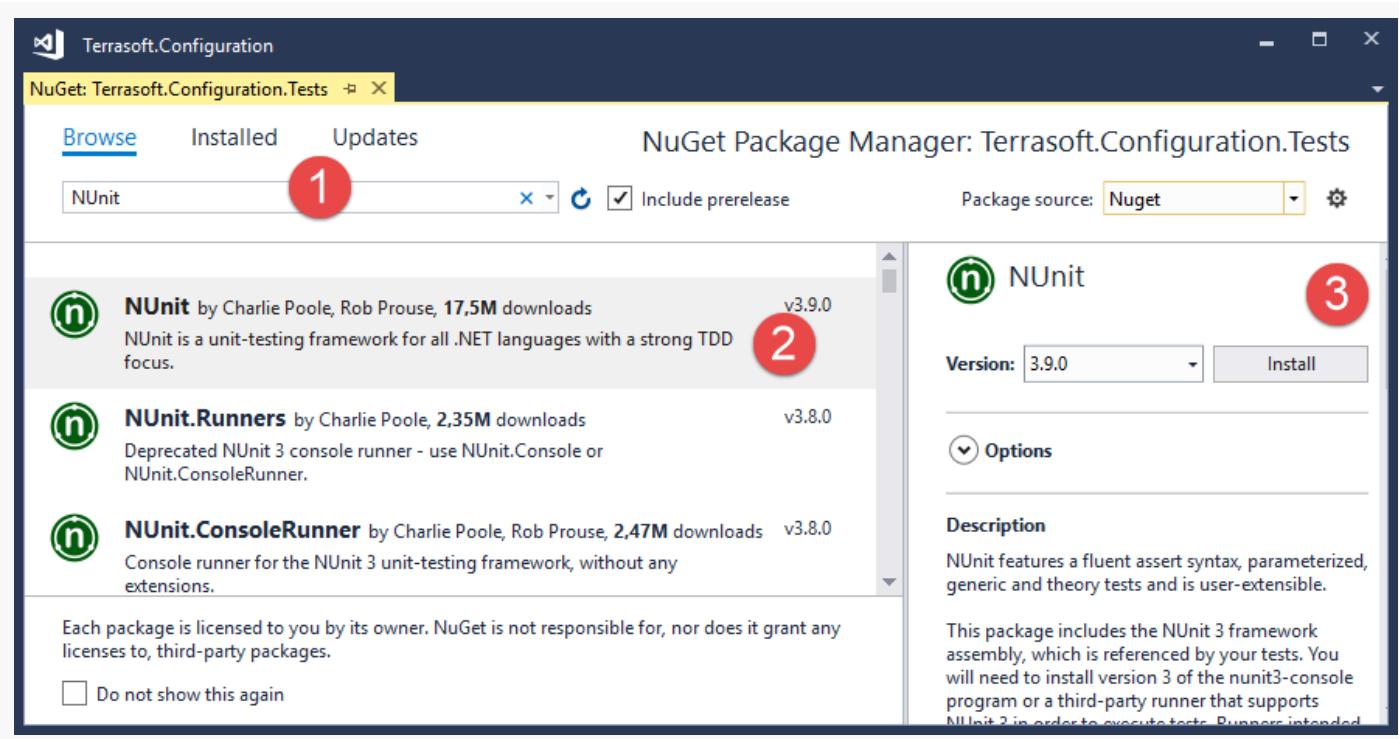
```
namespace Terrasoft.Configuration
{
    public class UsrNUnitSourceCode
    {
        // Строковое свойство.
        public string StringToTest
        {
            get
            {
                return "String to test";
            }
        }
        // Метод, проверяющий равенство двух строк.
        public bool AreStringsEqual(string str1, string str2)
        {
            return str1 == str2;
        }
    }
}
```

3. Настроить проект Unit-тестов

В нашем примере для создания Unit-тестов используется предварительно настроенный [проект](#) `Terrasoft.Configuration.Tests.csproj`, поставляемый вместе с решением `Terrasoft.Configuration.sln`.

Чтобы использовать в проекте `Terrasoft.Configuration.Tests.csproj` фреймворк NUnit для создания тестов, необходимо **добавить NuGet-пакет NUnit в зависимости проекта**. Для этого:

1. В [*Solution Explore*] кликните правой кнопкой мыши по названию проекта тестов `Terrasoft.Configuration.Tests`.
2. Выберите команду [*Manage NuGet Packages...*].
3. В строке поиска укажите "NUnit" (1).
4. В результатах поиска выберите пакет `NUnit` (2).
5. Для установки пакета нажмите кнопку [*Install*] (3).

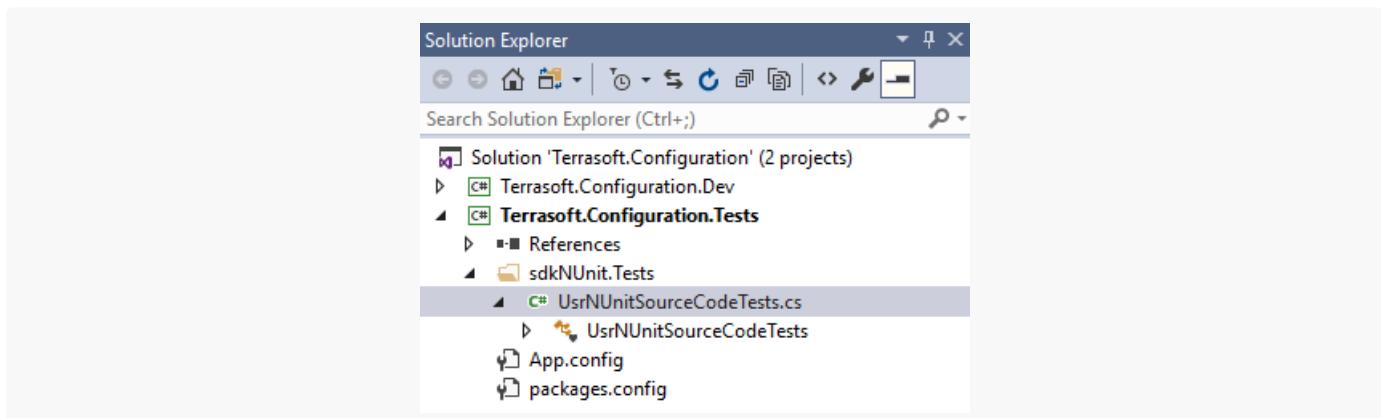


4. Создать тесты

Имя содержащего тесты класса должно состоять из имени тестируемого класса с добавлением окончания "Tests". Так же для **группировки тестов в проекте** удобно помещать их в каталог, название которого совпадает с названием тестируемого пакета с добавлением окончания ".Tests".

Чтобы **создать тесты**:

1. В проекте `Terrasoft.Configuration.Tests.csproj` создайте каталог `sdkNUnit.Tests`.
2. В каталоге `sdkNUnit.Tests` создайте новый класс `UsrNUnitSourceCodeTests`. Исходный код этого класса будет сохранен в файле `UsrNUnitSourceCodeTests.cs`.



3. В класс `UsrNUnitSourceCodeTests` добавьте методы, реализующие тесты.

```
UsrNUnitSourceCodeTests
```

```

using NUnit.Framework;

namespace Terrasoft.Configuration.Tests.sdkNUnitTests
{
    [TestFixture]
    class UsrNUnitSourceCodeTests
    {
        // Экземпляр тестируемого класса.
        UsrNUnitSourceCode objToTest = new UsrNUnitSourceCode();
        // Стока для тестирования.
        string str = "String to test";

        [Test]
        public void ClassReturnsCorrectStringProperty()
        {
            // Тестирование значения строкового свойства.
            // Значение должно быть не пустым и совпадать с требуемым.
            string res = objToTest.StringToTest;
            Assert.That(res, Is.Not.Null.And.EqualTo(str));
        }

        [Test]
        public void StringsMustBeEqual()
        {
            // Тестирование на равенство значений двух строк.
            bool res = objToTest.AreStringsEqual(str, "String to test");
            Assert.That(res, Is.True);
        }

        [Test]
        public void StringsMustBeNotEqual()
        {
            // Тестирование на неравенство значений двух строк.
            // Этот тест будет провален, т.к. значения равны.
            bool res = objToTest.AreStringsEqual(str, "String to test");
            Assert.That(res, Is.False);
        }
    }
}

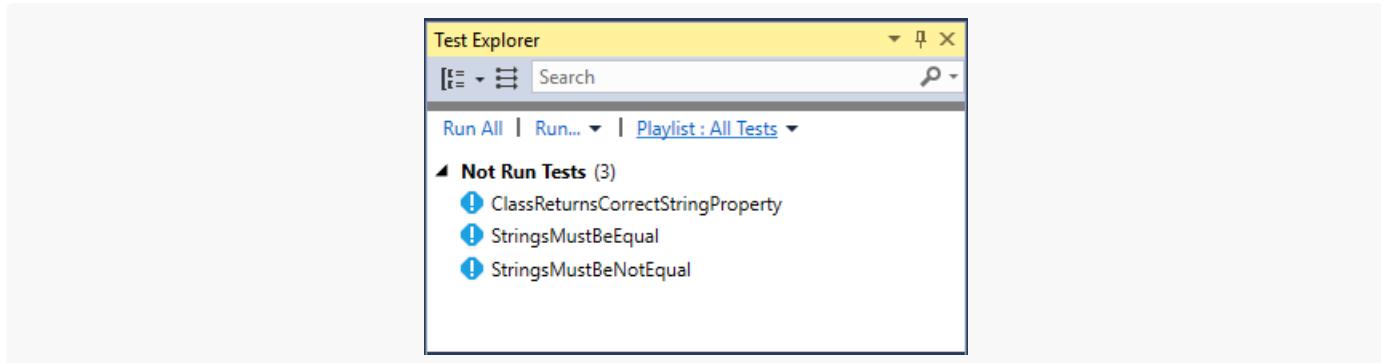
```

Атрибут `[TestFixture]` показывает, что класс `UsrNUnitSourceCodeTests` содержит тесты. Для каждого метода, тестирующего определенную функциональность этого класса, необходимо добавить атрибут `[Test]`. Атрибуты фреймворка NUnit описаны в [документации NUnit](#).

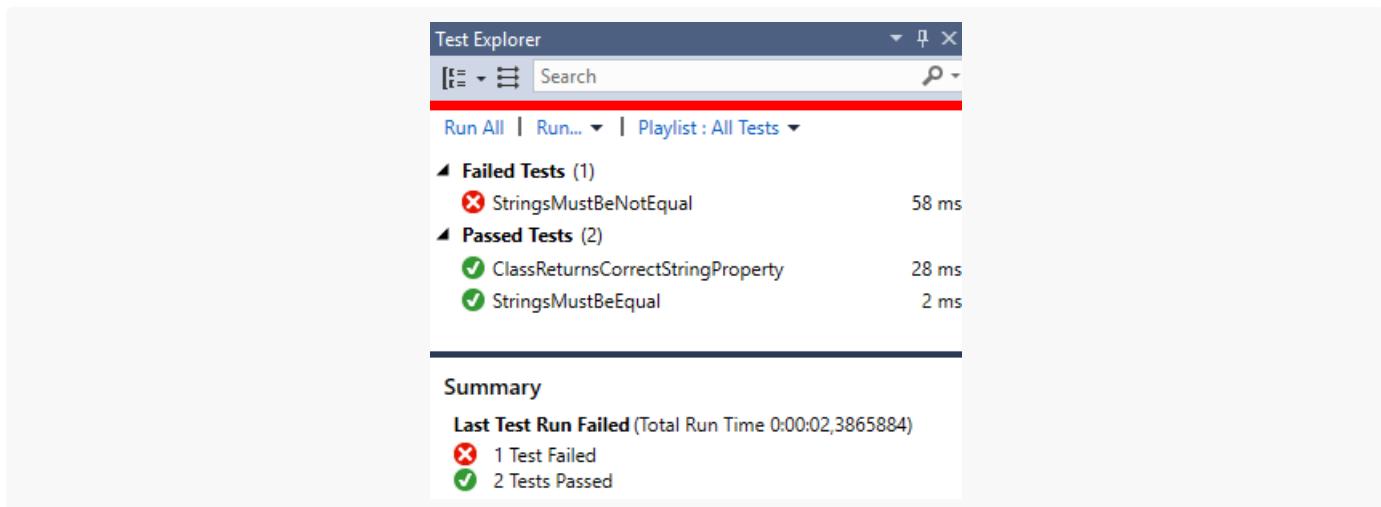
Тестирование выполняется с помощью метода `Assert.That()`, который принимает тестируемое значение. В качестве аргументов используются объекты, ограничивающие тестируемое значение. Модель ограничений описана в [документации NUNIT](#).

5. Выполнить тестирование

1. В Visual Studio нажмите [Test] —> [Windows] —> [Test Explorer].



2. Для запуска тестов выполните команду [Run All]. Успешно пройденные тесты будут перемещены в группу [Passed Test], а непройденные тесты — в группу [Failed Test].



Функциональность окна [Test Explorer] описана в [документации Visual Studio](#).

Логирование. log4net

Средний

Важно. Начиная с версии 7.17.2 **log4net** более не доступен в продукте, и для логирования необходимо использовать [NLog](#).

Логирование чаще всего используется при возникновении сбоев в работе приложения для их локализации, а также при тестировании новой функциональности. Чтобы избежать ухудшения производительности, рекомендуется включать логирование только при тестировании и отладке приложения.

В Creatio выполняется логирование всех основных операций.

Для логирования используется решение [log4net](#). Этот инструмент позволяет выполнять логирование параметров из разных архитектурных компонентов приложения в отдельные файлы логов.

Хранение логов

Местоположение файлов логов зависит от значения системных переменных Windows.

Путь к файлам с логами загрузчика (по умолчанию)

```
[TEMP]\Creatio\Site_[{SiteId}]\[{ApplicationName}]\Log\[{DateTime.Today}]
```

Пример пути

```
C:\Windows\Temp\Creatio\Site_1\creatio7121\Log\2018_05_22
```

Путь к файлам с логами конфигурации Default

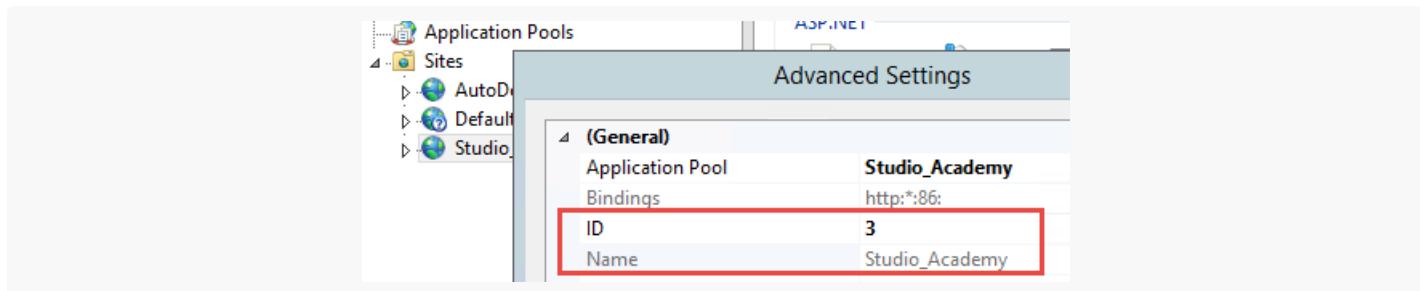
```
[TEMP]\Creatio\Site_[{SiteId}]\[{ApplicationName}]\[ConfigurationNumber]\Log\[{DateTime.Today}]
```

Пример пути

```
C:\Windows\Temp\Creatio\Site_1\creatio7121\0\Log\2018_05_22
```

Здесь в квадратных скобках указаны следующие переменные:

- [TEMP] — базовый каталог. По умолчанию для IIS используется каталог C:\Windows\Temp , а для Visual Studio (IIS Express) — C:\Users\{Имя пользователя}\AppData\Local\Temp .
- [{SiteId}] — номер сайта. Для IIS указан в расширенных настройках сайта. Для Visual Studio номер содержит значение 2.
- [{ApplicationName}] — название приложения.
- [ConfigurationNumber] — номер конфигурации. Конфигурация Default , как правило, имеет номер 0.
- [{DateTime.Today}] — дата логирования.



Уровни логирования

По умолчанию уровень логирования для всех компонентов Creatio установлен таким образом, чтобы обеспечить максимальную производительность приложения.

Уровни логирования в порядке возрастания приоритета:

- **ALL** — логирование всех событий. Существенно уменьшает производительность приложения.
- **DEBUG** — логирование всех событий при отладке.
- **INFO** — логирование ошибок, предупреждений и сообщений.
- **WARN** — логирование ошибок и предупреждений.
- **ERROR** — логирование всех ошибок.
- **FATAL** — логирование только ошибок, приводящих к прекращению работы компонента, для которого ведется логирование.
- **OFF** — логирование отключено.

Настройка логирования

Логирование ведется отдельно для загрузчика приложения и для конфигурации `Default`. Настройки логирования выполняются в конфигурационном файле `..\Terrasoft.WebApp\log4net.config`.

Установить максимальный уровень логирования для всех компонентов

Для этого необходимо в XML-элементе `<root>` файла `..\Terrasoft.WebApp\log4net.config` указать уровень "ALL".

```
..\Terrasoft.WebApp\log4net.config
```

```
<root>
    <level value="ALL" />
    <appender-ref ref="commonAppender" />
</root>
```

При работе с SVN вести логирование только ошибок

Для этого необходимо в XML-элементе `<logger name="Svn">` файла `..\Terrasoft.WebApp\log4net.config` указать уровень "ERROR".

`..\Terrasoft.WebApp\log4net.config`

```
<logger name="Svn" >
    <level value="ERROR" />
    <appender-ref ref="SvnAppender" />
</logger>
```