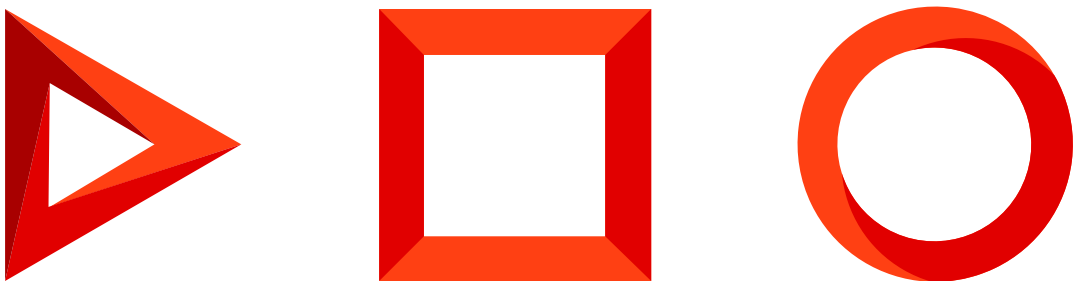
Creatio Academy

Внешние IDE

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

Внешние IDE

Средний

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 РФС необходимо отключить получение статического клиентского контента из файловой системы. В файле `Web.config`, который находится в корневом каталоге установленого приложения, установите значение `false` для флага `UseStaticFileContent`.

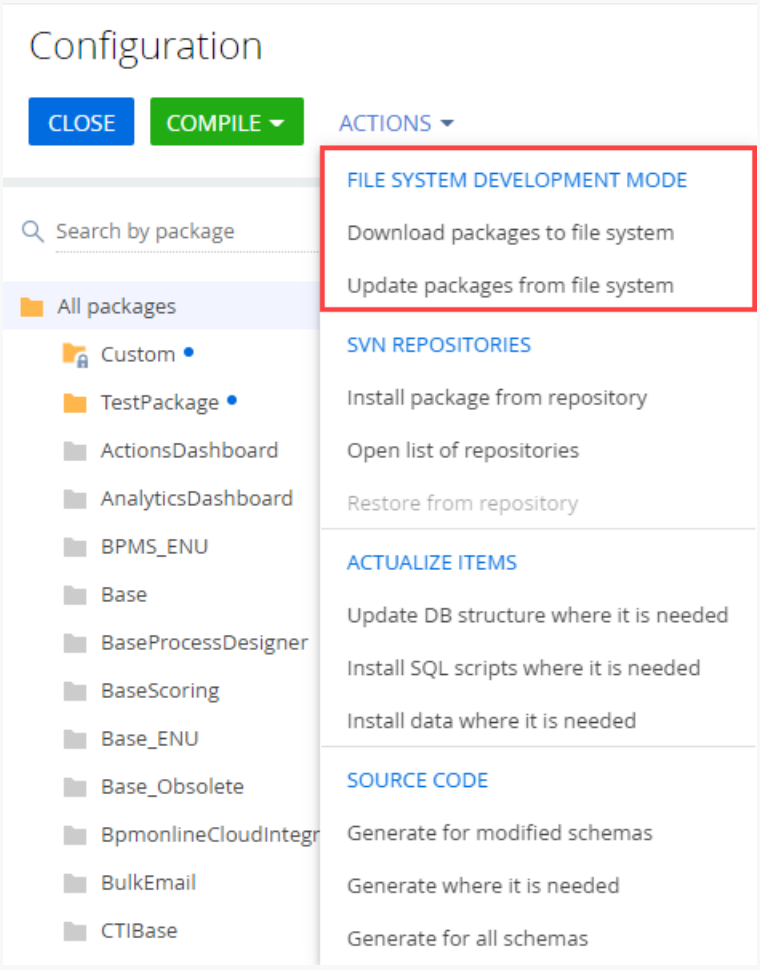
   Web.config

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

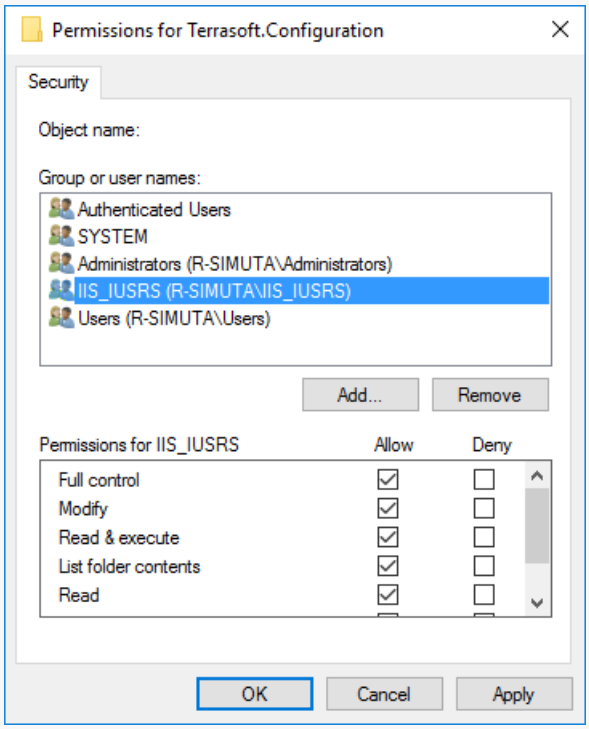
3. **Скомпилировать приложение.**
   Выполните действие [*Перекомпилировать все*] ([*Compile all*]) панели инструментов раздела [*Конфигурация*] ([*Configuration*]).

   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]`.

5. 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.

6. 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

1. С помощью **встроенного компилятора Visual Studio**. Однако он может не учесть зависимости и позиции пакетов.
2. 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

1. **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" />
```

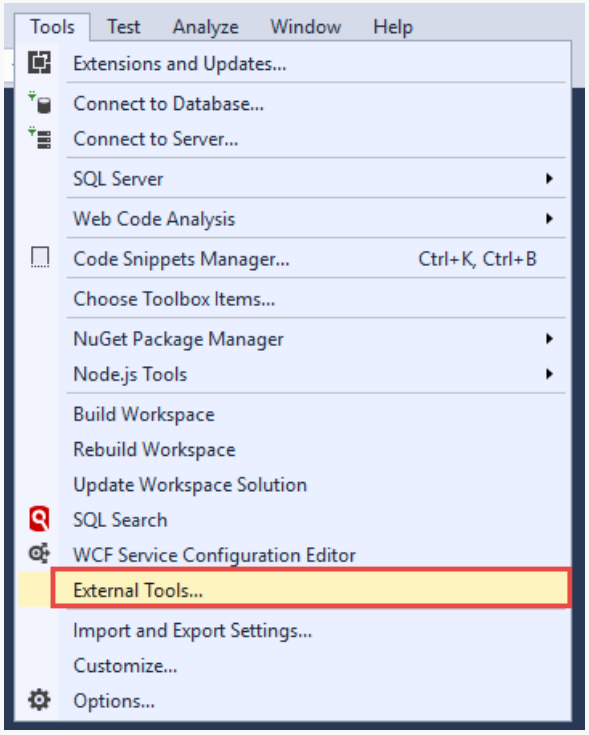
Для ускорения компиляции конфигурационную сборку необходимо разбивать на независимые компилируемые модули. В таком случае необходимо установить значение `true` для настройки `CompileByManagerDependencies` во "внутреннем" Web.config, находящемся в каталоге `Terrasoft.WebApp`, и в файле настроек `Terrasoft.Tools.WorkspaceConsole.exe.config`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

Настройка "внутреннего" Web.Config

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

2. **Настроить интеграцию WorkspaceConsole в Microsoft Visual Studio**.

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



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

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

Свойства команды `Build Workspace`

Command	Структура
	`[Путь к каталогу с установленным приложением]\Terrasoft.WebApp\Desk`
	
	Пример
	`C:\creatio713\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasof`

Arguments	Структура `--operation=BuildWorkspace --workspaceName=Default --webApplicatio` Пример `--operation=BuildWorkspace --workspaceName=Default --webApplicatio`

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

Свойства команды `Rebuild Workspace`

Command	Структура
	`[Путь к каталогу с установленным приложением]\Terrasoft.WebApp\Desk`
	
	Пример
	`C:\creatio713\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasof`

Arguments	Структура `--operation=RebuildWorkspace --workspaceName=Default --webApplicati` Пример `--operation=RebuildWorkspace --workspaceName=Default --webApplicati`

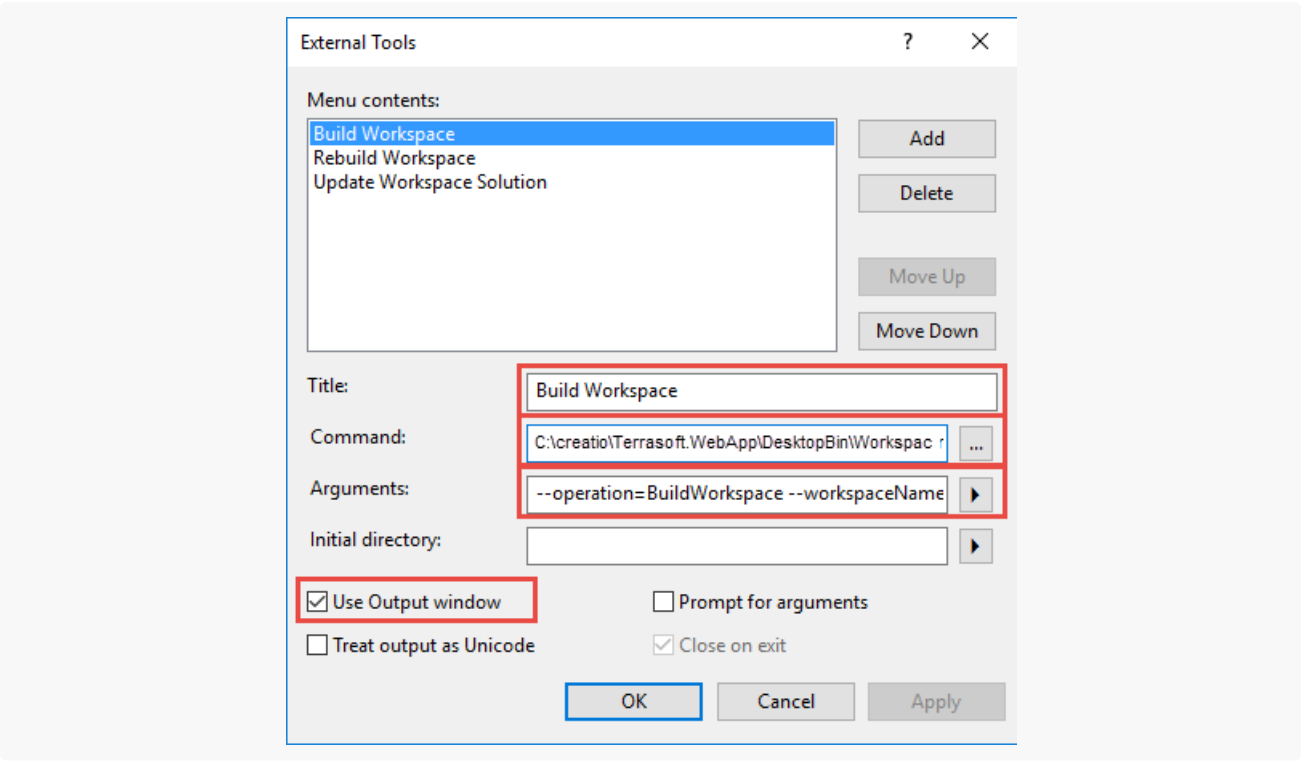
- **Update Workspace Solution** — обновление решения Visual Studio конфигурационного проекта из базы данных приложения. Команда применяет все изменения, внесенные пользователем через приложение.

Свойства команды `Update Workspace Solution`

Command	Структура
	`[Путь к каталогу с установленным приложением]\Terrasoft.WebApp\Desk`
	
	Пример
	`C:\creatio713\Terrasoft.WebApp\DesktopBin\WorkspaceConsole\Terrasof`

Arguments	Структура `--operation=UpdateWorkspaceSolution --workspaceName=Default --webAp` Пример `--operation=UpdateWorkspaceSolution --workspaceName=Default --webAp`

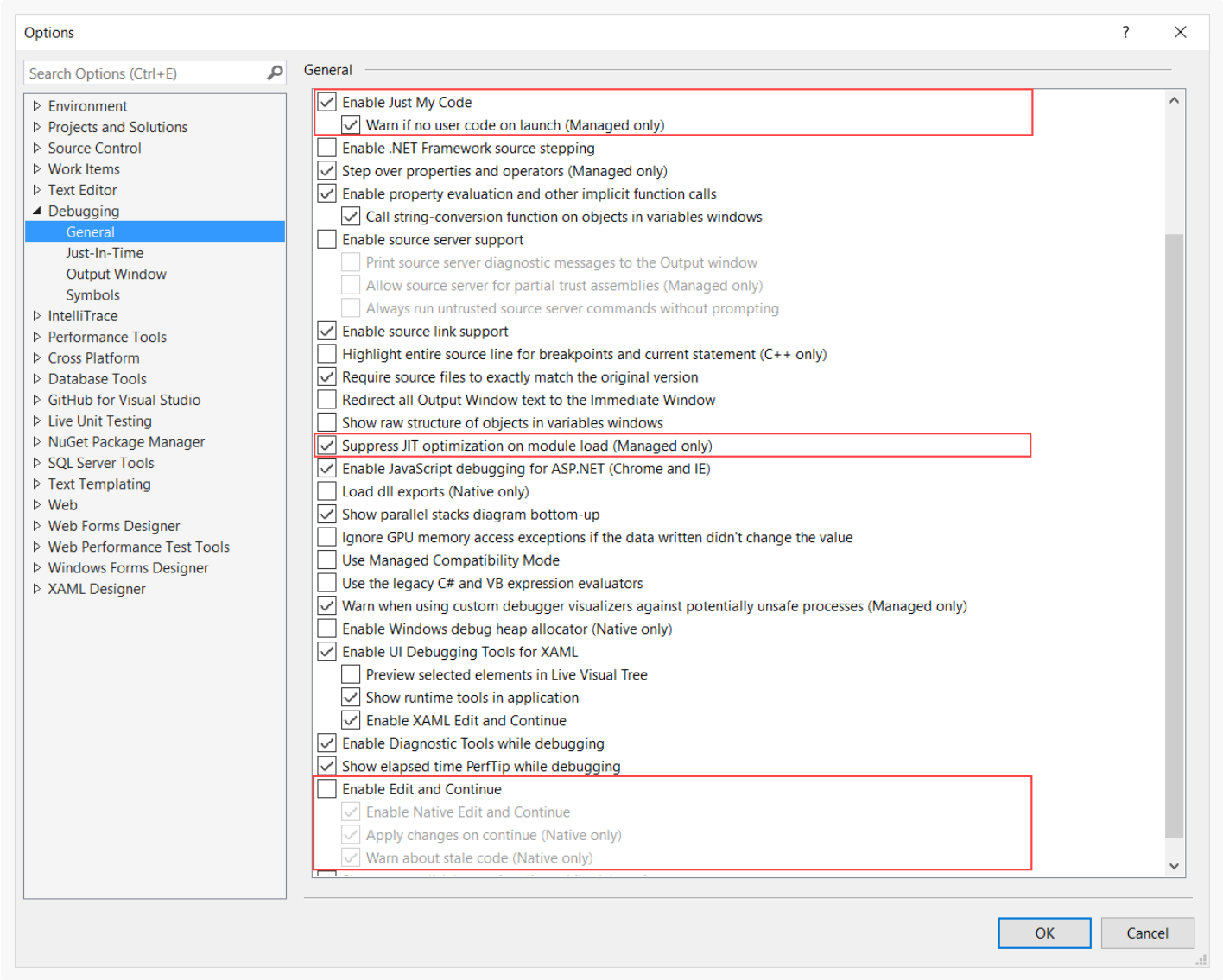
f. Для всех команд установите признак [*Use Output window*].



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:

1. 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 предпочитаемой 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

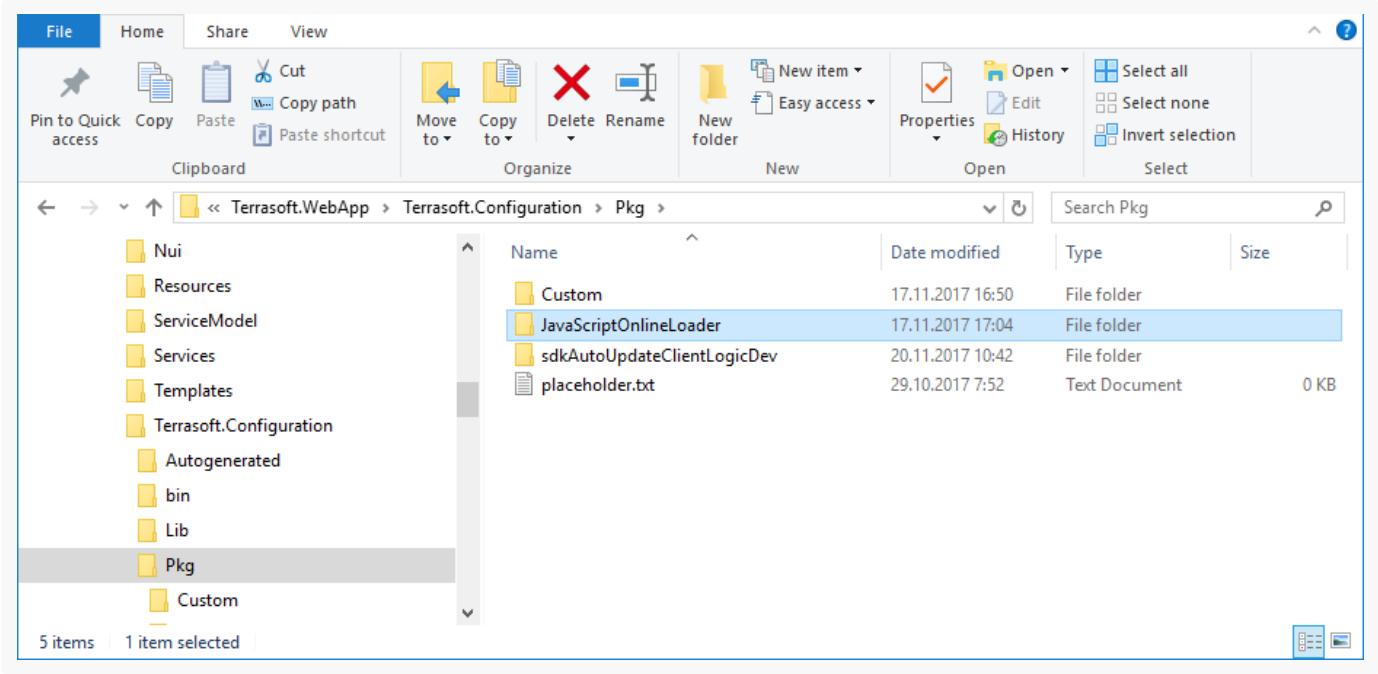
- Наличие синтаксических ошибок в коде. Если в исходном коде модуля допущена синтаксическая ошибка, то автоматическое обновление страницы не произойдет. Потребуется ее принудительное обновление (например, клавишей F5). При исправлении ошибки страница вернется к работоспособному состоянию.
- Эффект сильной 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>`

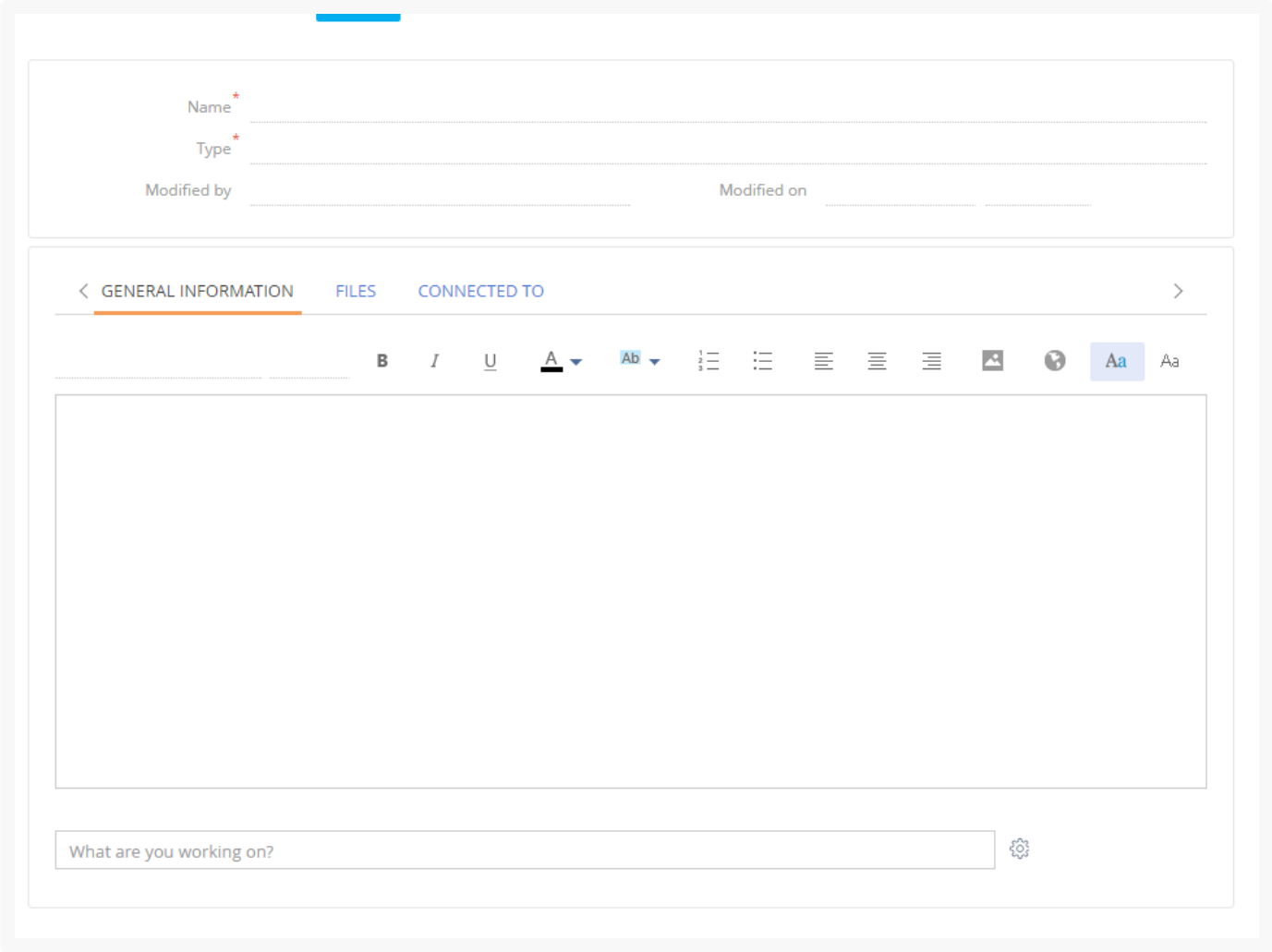
   Например, в пользовательский пакет добавлена замещенная схема `KnowledgeBasePageV2` — схема страницы редактирования записи раздела [*База знаний*].

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

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

   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.

Разработать C# код в конфигурационном проекте

Сложный

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.

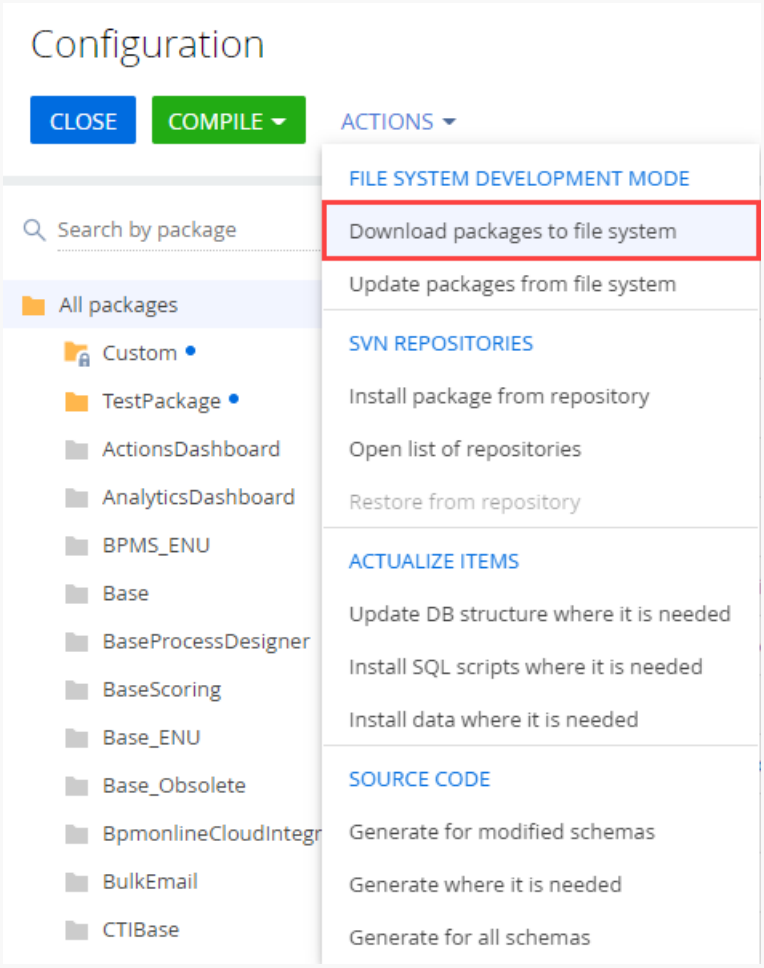
3. Создать схему [*Исходный код*]

В пакете создайте схему [*Исходный код*]:

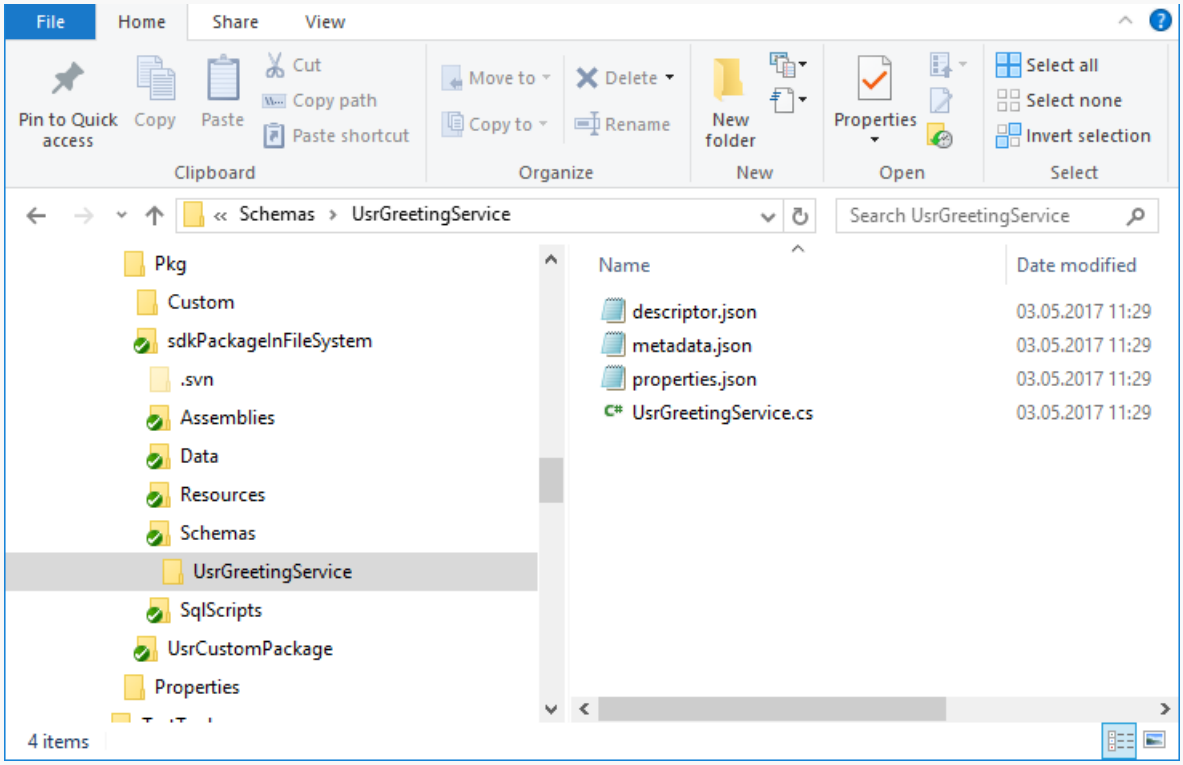
- [*Код*] ([*Code*]) — "`UsrGreetingService`";
- [*Заголовок*] ([*Title*]) — "`UsrGreetingService`";
- [*Пакет*] ([*Package*]) — "`sdkPackageInFileSystem`";

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*]).

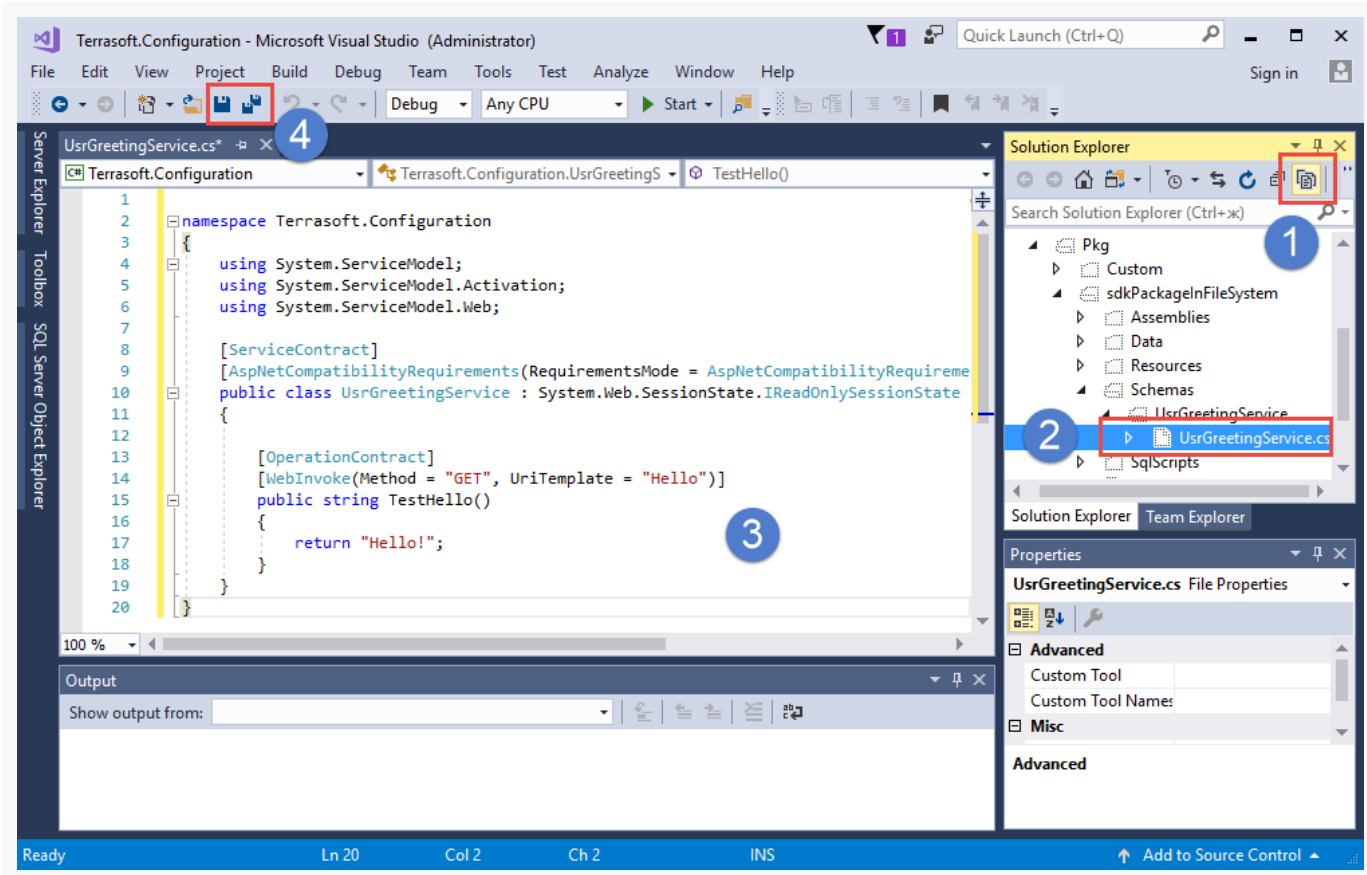


В результате в файловой системе в каталоге `Pkg\sdkPackageInFileSystem\Schemas\` появится файл исходного кода схемы `UsrGreetingService.cs`. При этом в каталог `Autogenerated\Src` будет помещен сгенерированный системой файл схемы `UsrGreetingServiceSchema.sdkPackageInFileSystem_Entity.cs`.



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!";
        }
    }
}
```

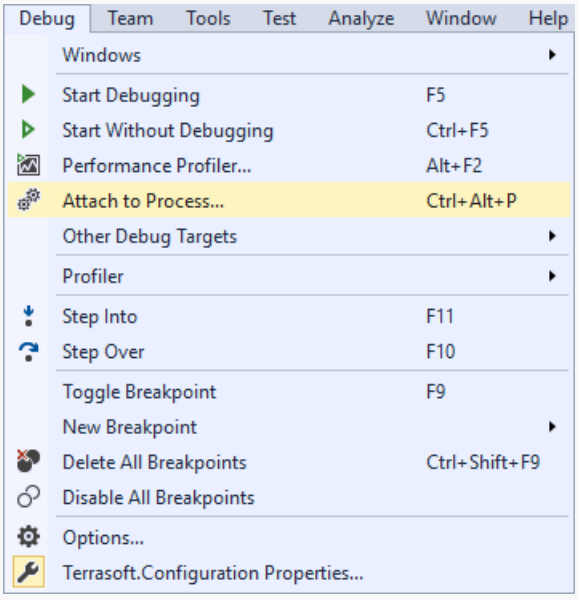
5. Сохранить, скомпилировать и отладить созданный исходный код

1. 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.
2. 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/0/rest/UsrGreetingService/Hello
```

3. 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.

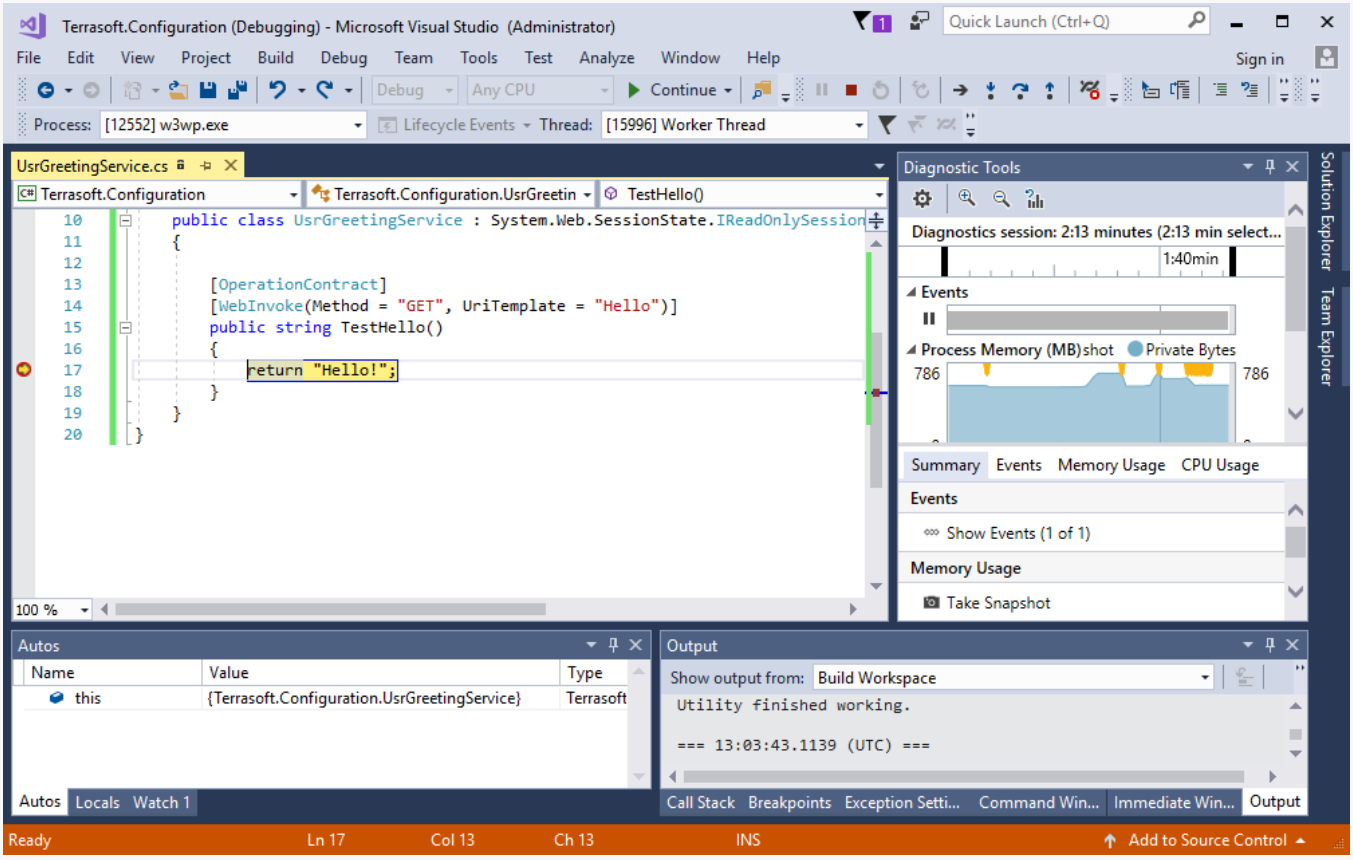
На заметку. Название рабочего процесса может различаться в зависимости от конфигурации используемого сервера IIS. Так, для полнофункционального IIS процесс называется `w3wp.exe`, для IIS Express — `iisexpress.exe`.

По умолчанию рабочий процесс IIS запущен под учетной записью, имя которой совпадает с именем пула 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:

1. Перейдите в каталог `C:\Creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole`.
2. 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.svnDll` из соответствующей папки (x64 или x86).

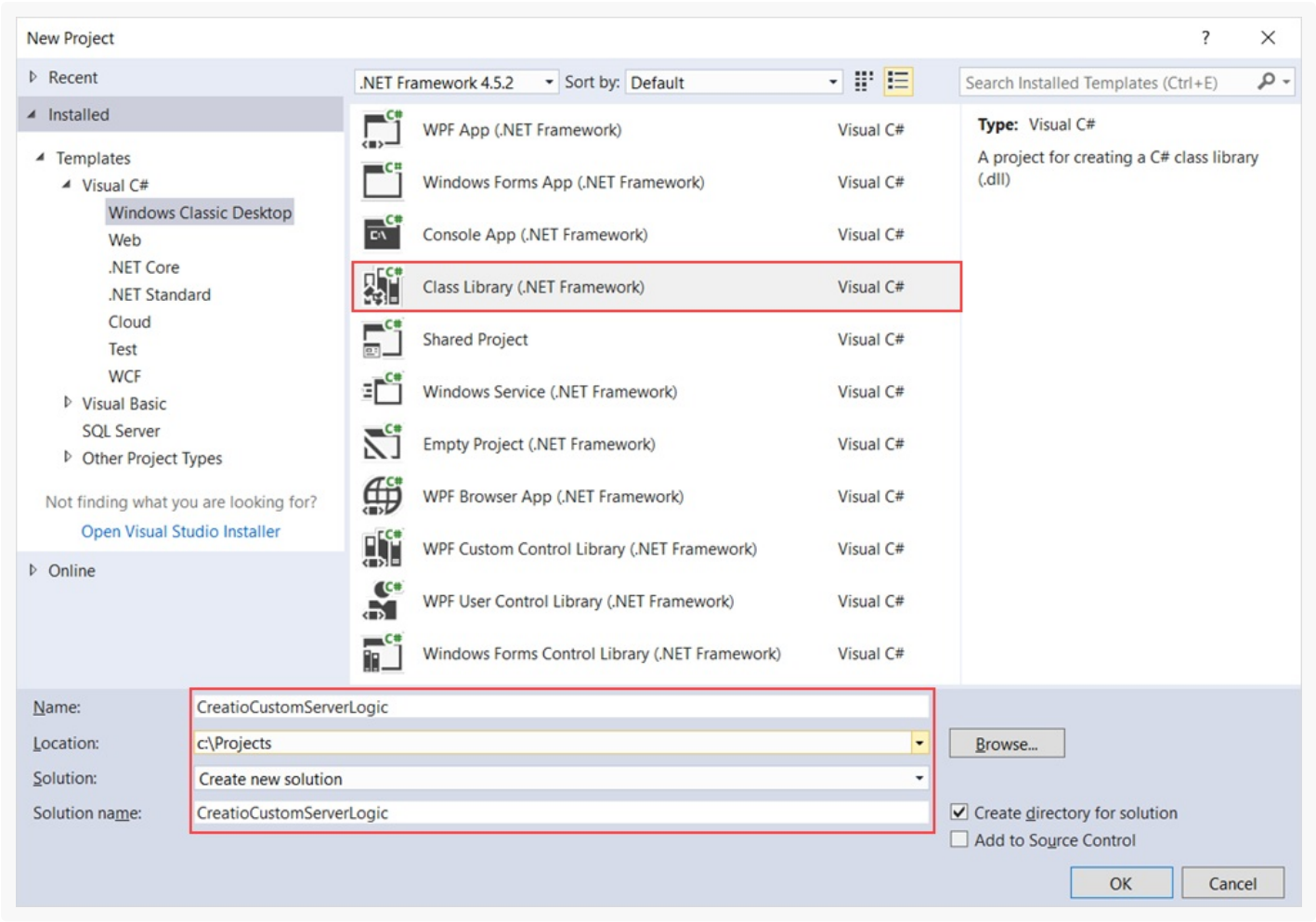
1. Укажите параметры подключения к базе данных в файле `Terrasoft.Tools.WorkspaceConsole.exe.config`, размещенном в каталоге `C:\Creatio\Terrasoft.WebApp\DesktopBin\WorkspaceConsole`). Например, если на сервере `dbserver` развернута база данных **CreatioDB**, то строка подключения будет иметь следующий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

Строка подключения базы данных CreatioDB

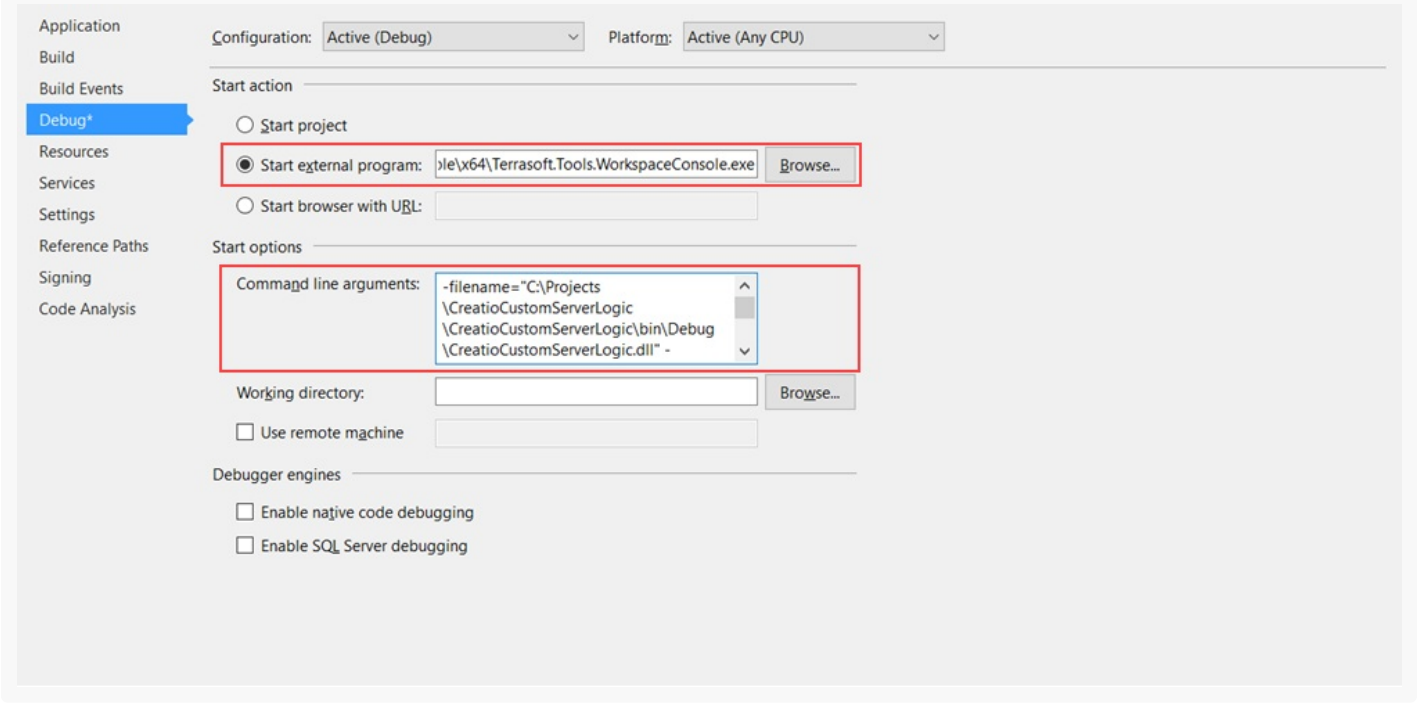
```
<connectionStrings>
    <add name="db" connectionString="Data Source=dbserver; Initial Catalog=CreatioDB; Persist Se
</connectionStrings>
```

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:

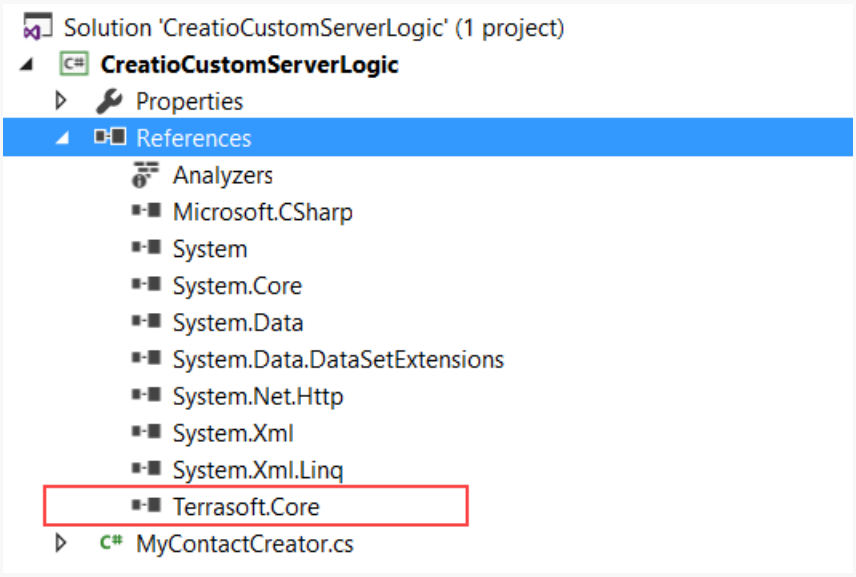
- `-filename` — полный путь к отладочной версии разрабатываемой библиотеки классов ("C:\Projects\CreatioCustomServerLogic\CreatioCustomServerLogic\bin\Debug\CreatioCustomServerLogic.dll");
- `-typeName` — полное имя класса, в котором реализуется разрабатываемая программная логика, включая названия всех 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\CreatioCustom
```

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++)
            {
```

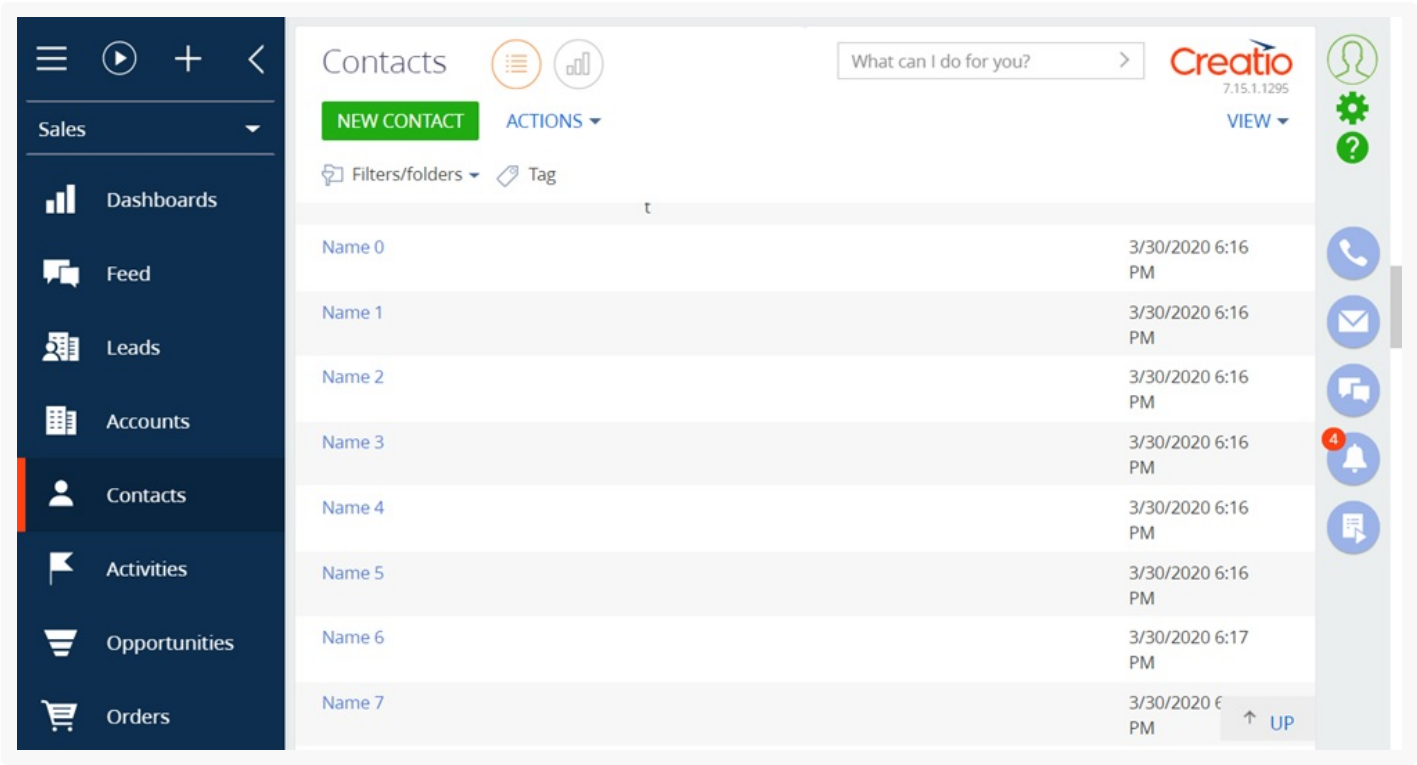
```
                // Создание нового контакта.
                var entity = schema.CreateEntity(userConnection);
                // Установка свойств контакта.
                entity.SetColumnValue("Name", string.Format("Name {0}", i));
                entity.SetDefColumnValues();
                // 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.

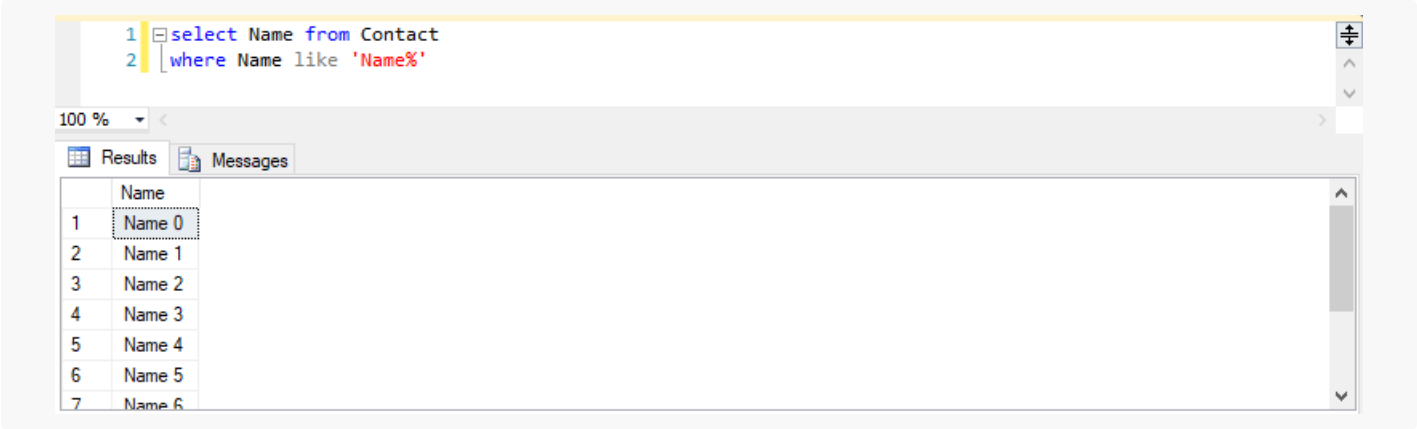
Также можно установить точку останова на любой строке исходного кода и во время выполнения программы посмотреть текущие значения переменных, т. е. выполнить отладку.

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

Добавленные контакты



Запрос к таблице контактов базы данных



Разработка C# кода для cloud приложения

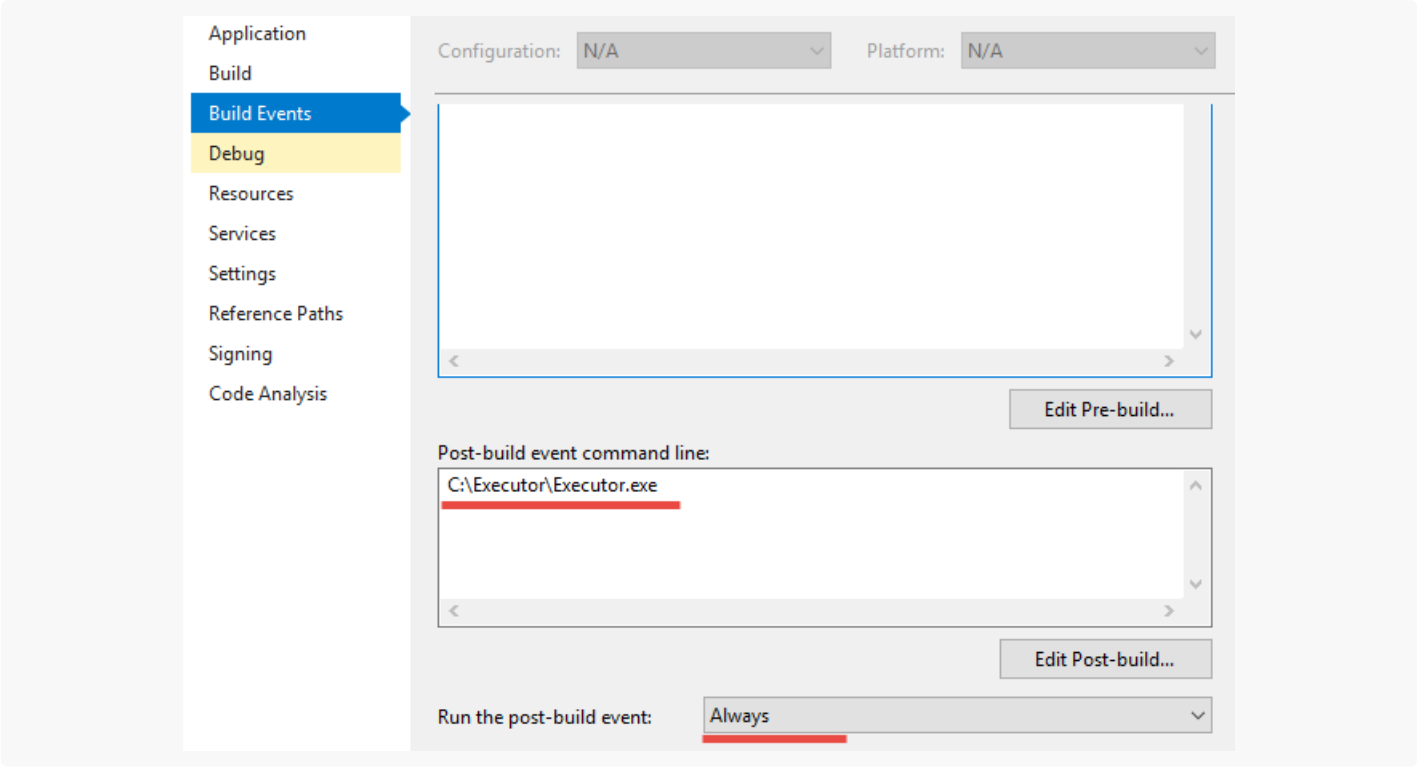
1. Создать проект библиотеки классов

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 указажите 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

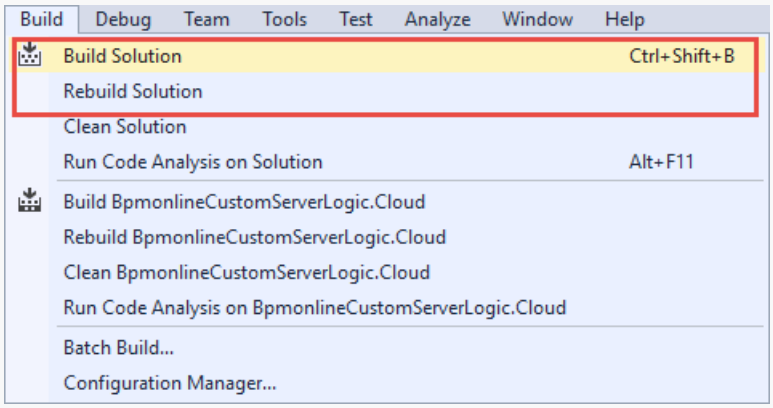
На заметку. Скачать утилиту, настроенную для выполнения этого примера, можно по ссылке.

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

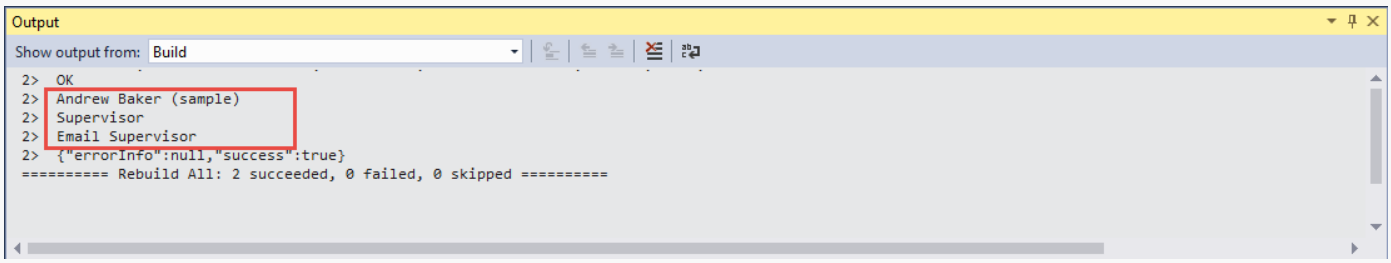
- `Loader` — URL загрузчика приложения Creatio. Как правило, это URL сайта Creatio, например "`https://mycloudapp.creatio.com`".
- `WebApp` — URL приложения Creatio. Как правило, это путь к конфигурации по умолчанию приложения Creatio, например "`https://mycloudapp.creatio.com/0`".
- `Login` — имя пользователя 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

Сложный

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.

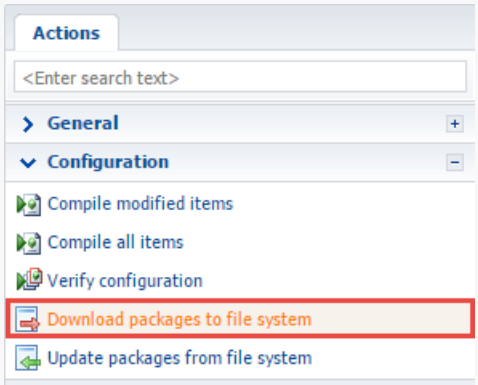
На заметку. При разработке в файловой системе вместо встроенных возможностей Creatio удобнее использовать клиентские приложения для работы с хранилищами систем контроля версий, например, Tortoise SVN или Git.

3. Создать клиентскую схему, в которой будет выполняться разработка

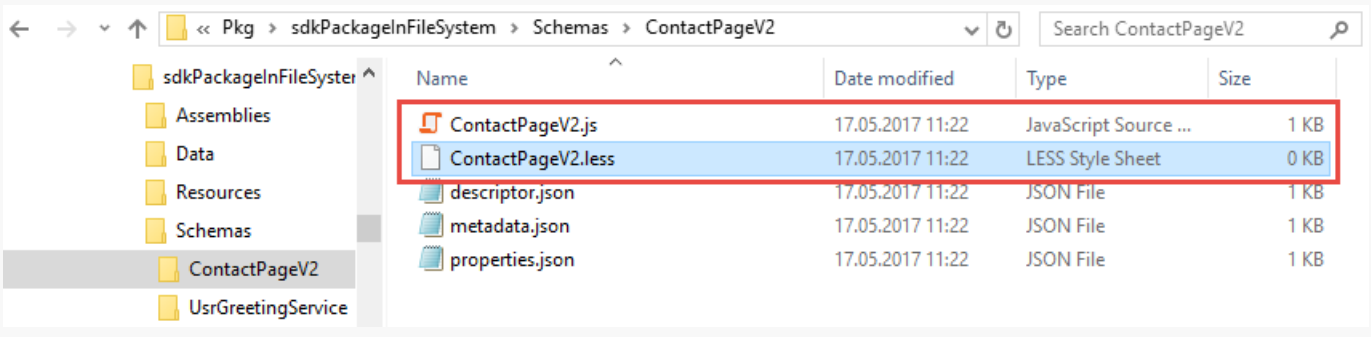
Создать клиентскую схему в пользовательском пакете.

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*]).

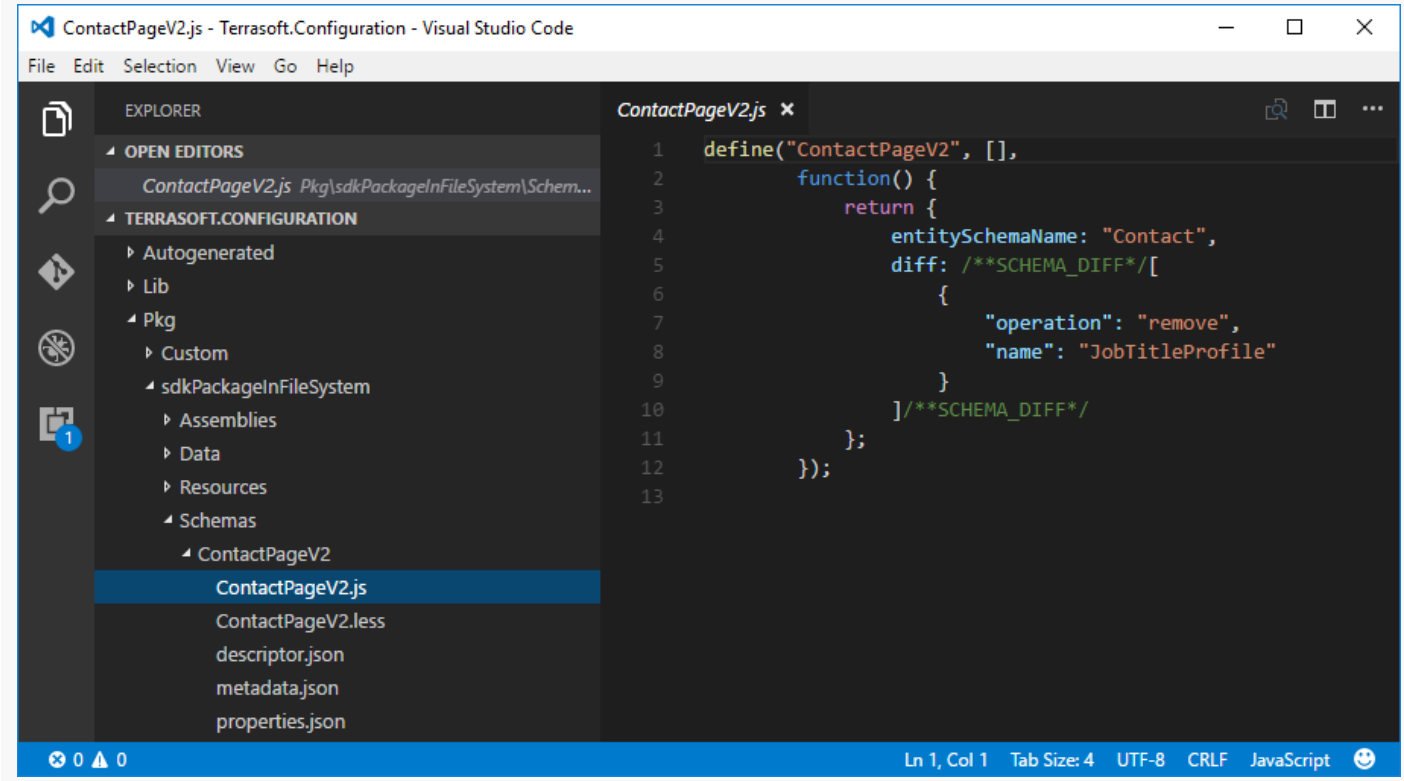


Так, например, если в пользовательском пакете `sdkPackageInFileSystem` была создана замещающая схема [*Схема отображения карточки контакта*] ([*Display schema - Contact card*]) с именем `ContactPageV2`, то в файловой системе в каталоге `Pkg\sdkPackageInFileSystem\Schemas\ContactPageV2` появятся файлы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



5. Выполнить разработку исходного кода схемы в IDE.

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



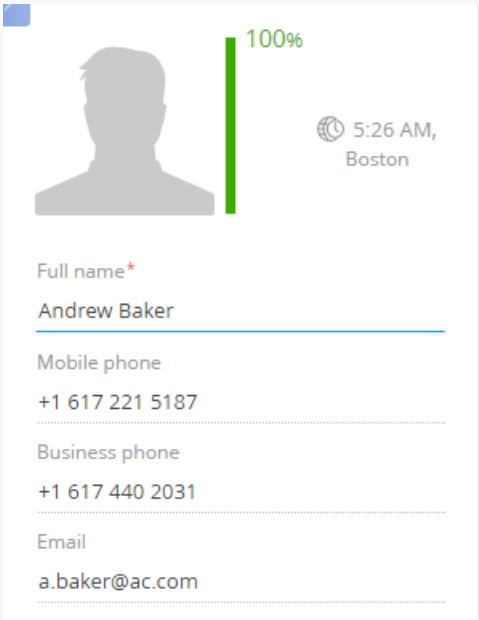
Например, чтобы скрыть со страницы контакта поле [*Полное название должности*] ([*Full job title*]), в файл `ContactPageV2.js` нужно добавить следующий исходный код:

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.

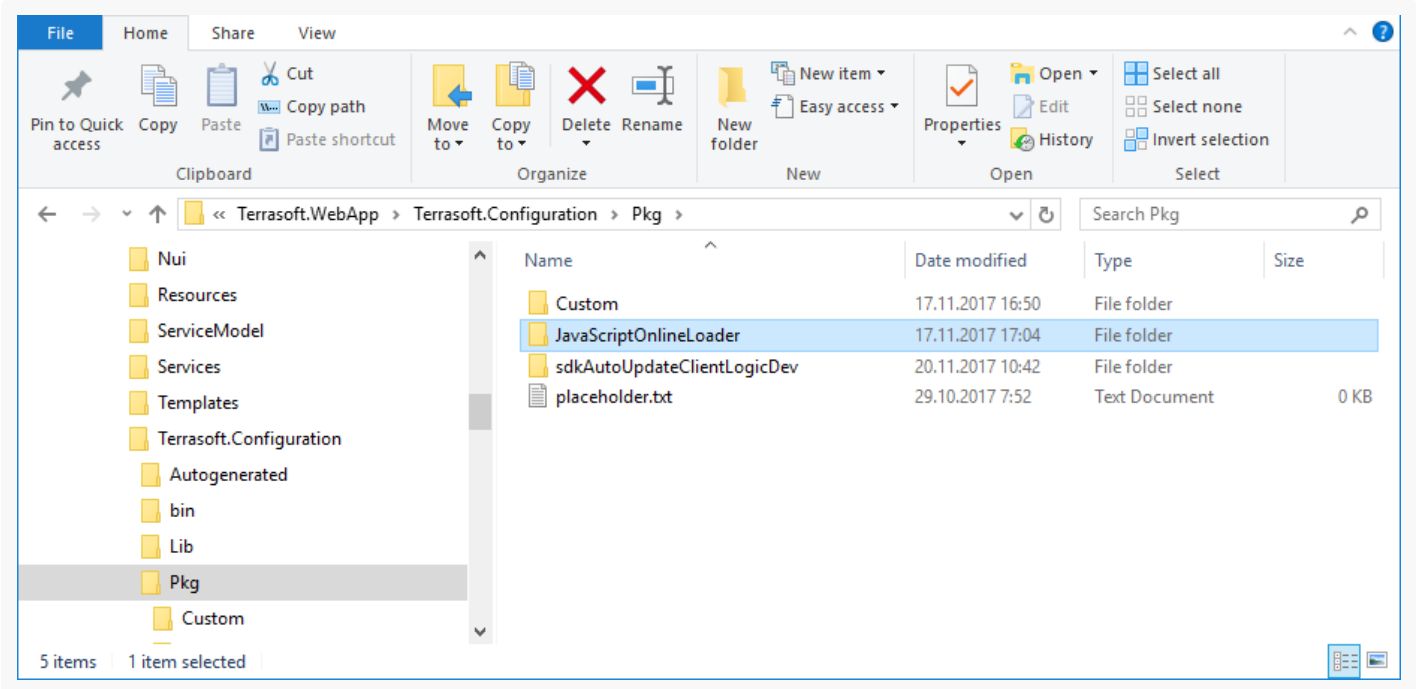
Известные проблемы

1. 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.
2. Не все модули Creatio могут загружаться отдельно. Основная причина — эффект сильной связанности модулей.

Последовательность настройки автоматического отображения изменений

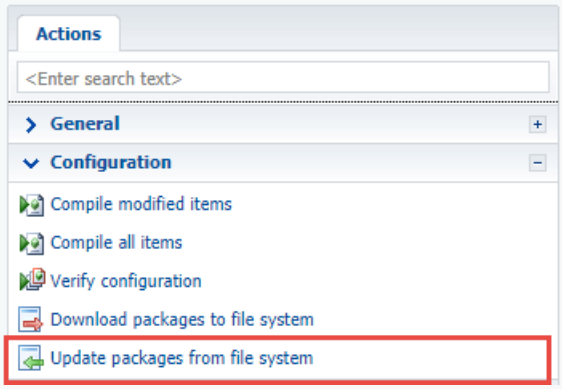
1. Установить пакет JavaScriptOnlineLoader

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`.

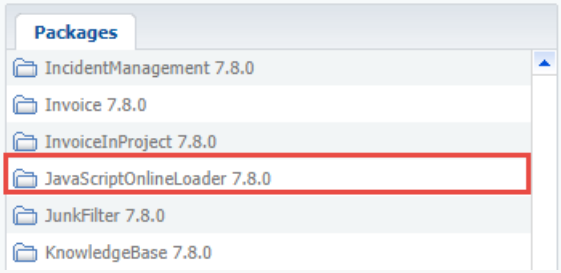


Важно. Пакет доступен на GitHub. Также архив с пакетом можно скачать по ссылке.

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>
```

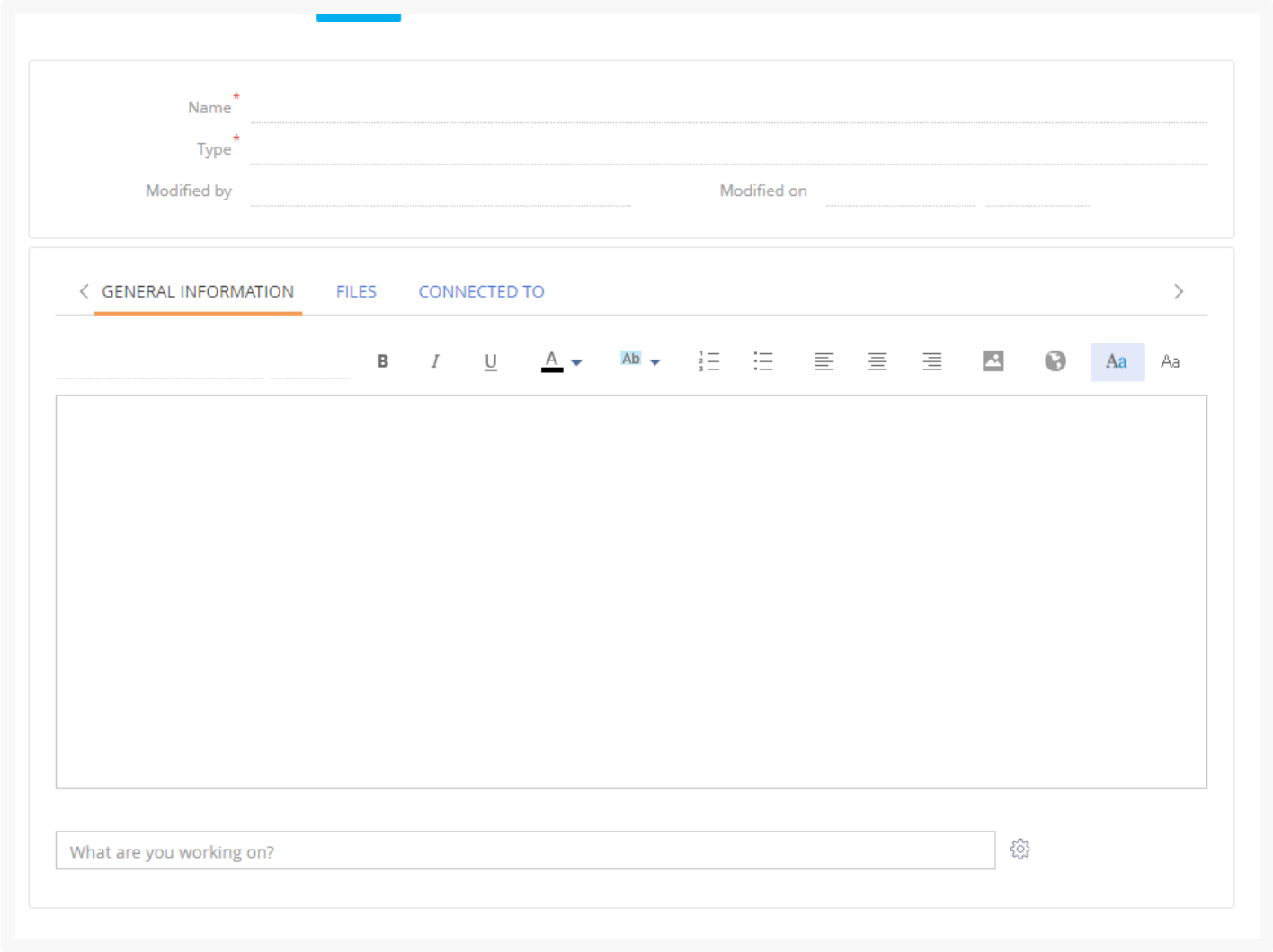
Например, в пользовательский пакет добавлена замещенная схема `KnowledgeBasePageV2` — схема страницы записи раздела [*База знаний*].

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\Schemas\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