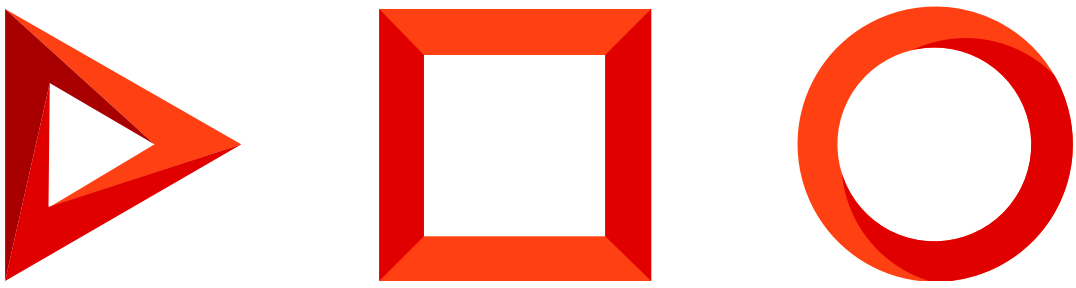
Creatio Academy

Отладка

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

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 поозволяет 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:

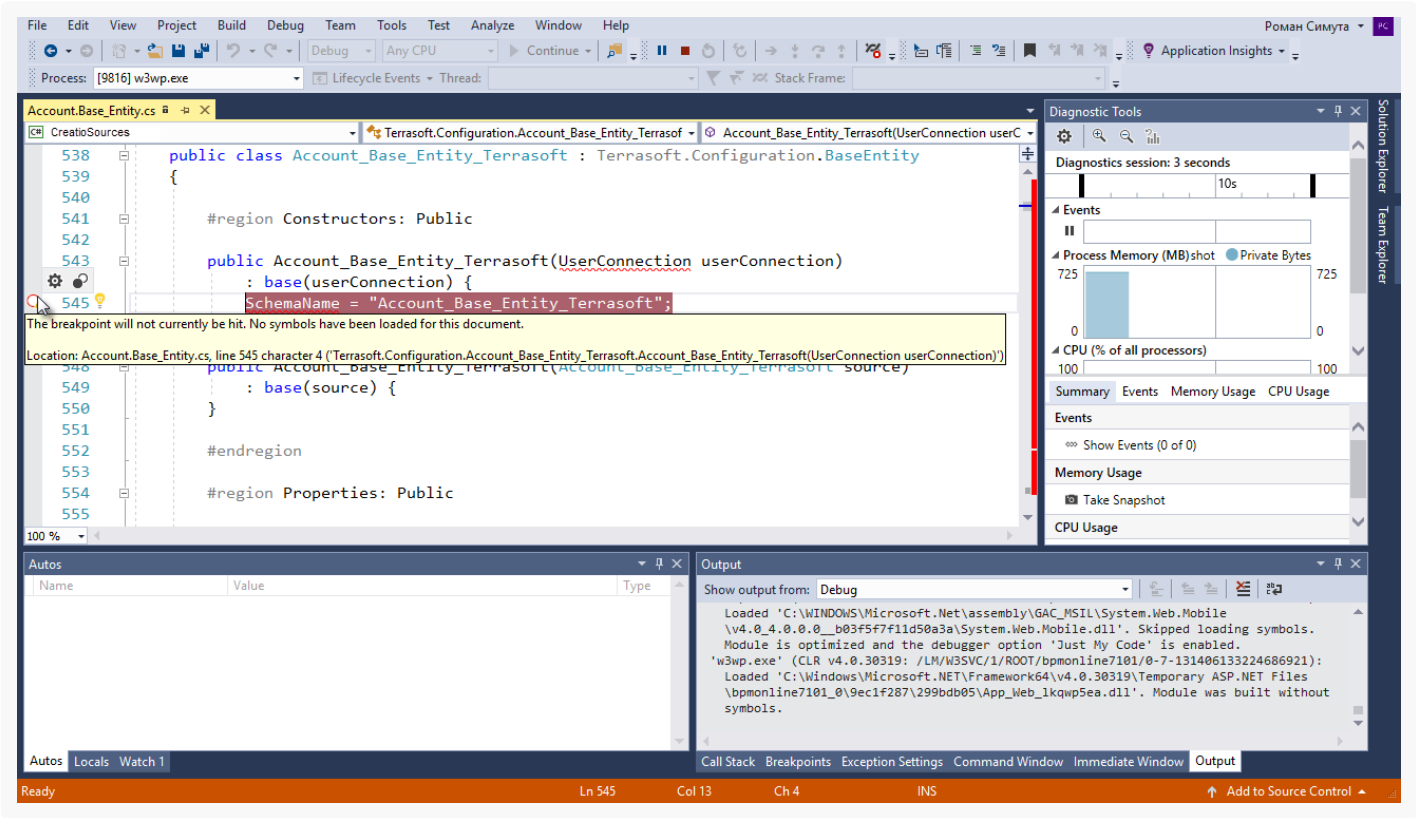
1. Выгрузить исходные коды конфигурации Creatio в файлы локального каталога.
2. Создать новый проект в Visual Studio, в котором будет выполняться отладка.
3. Добавить в проект Visual Studio выгруженные файлы с исходным кодом.
4. Подключить проект к рабочему процессу сервера IIS и начать процесс отладки.

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.

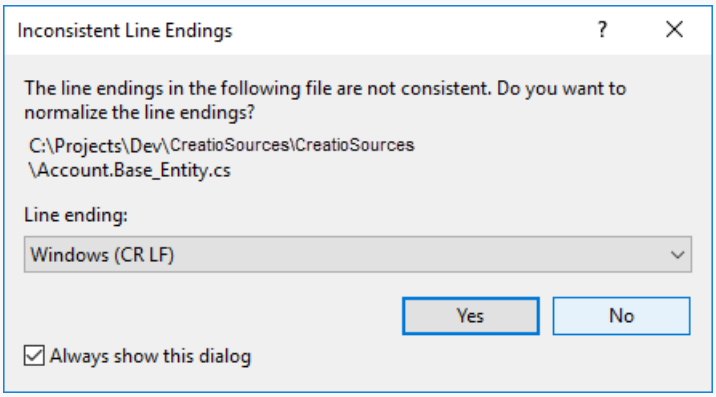


Решение:

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 неединообразных символах в конце строк.

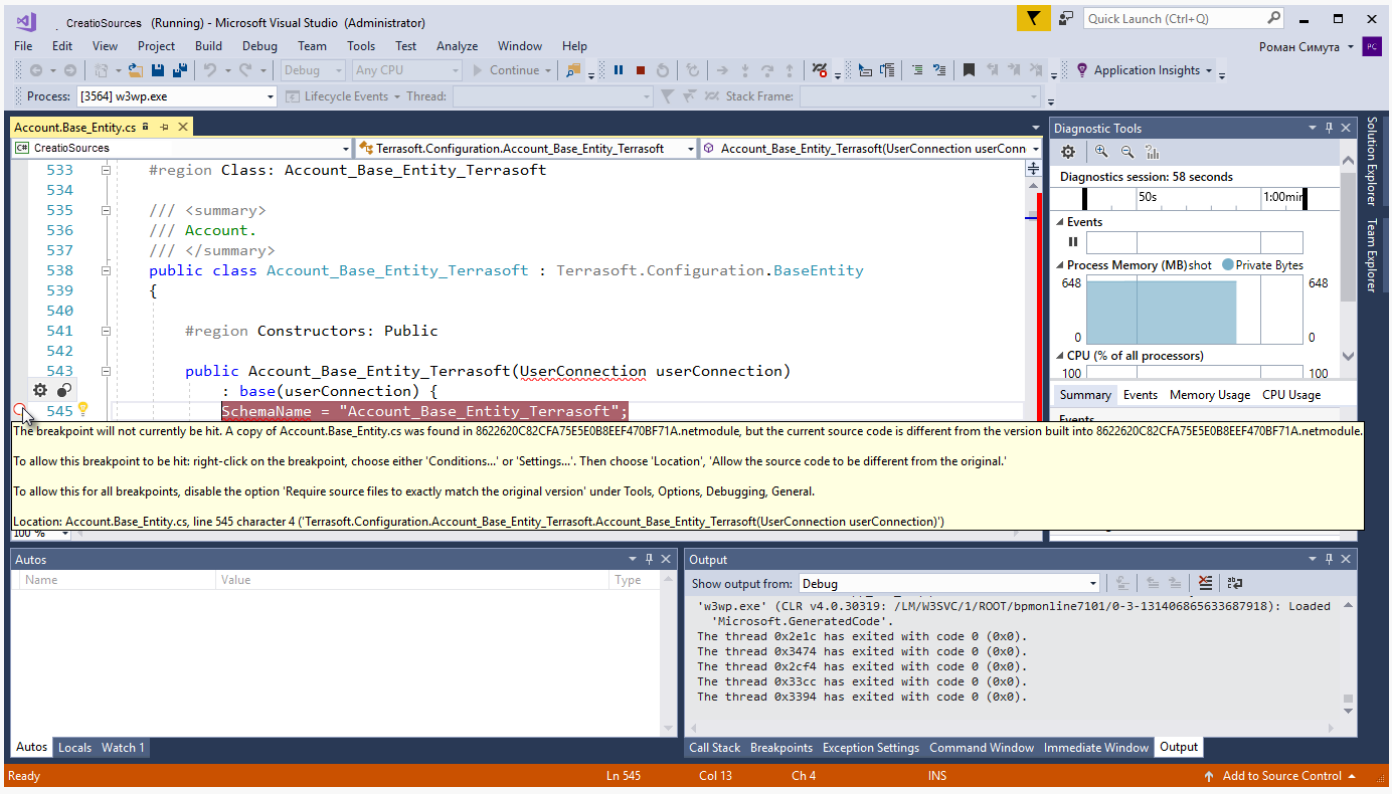


Решение:

1. 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.

2. 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**:

1. 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.

2. 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).

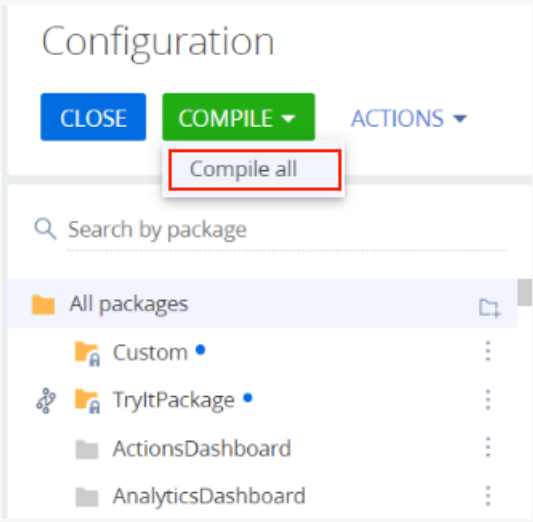
   `Web.config`

   ```
   <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`.

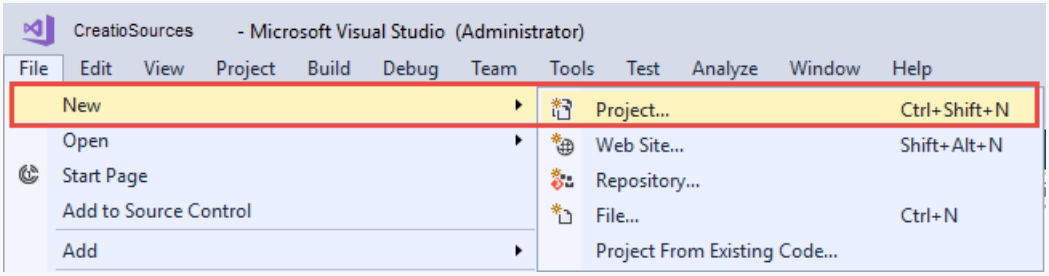
Например: `Contact.Base_Entity.cs`, `ContractReport.Base_Report.cs`.

3. Создать проект Visual Studio для отладки

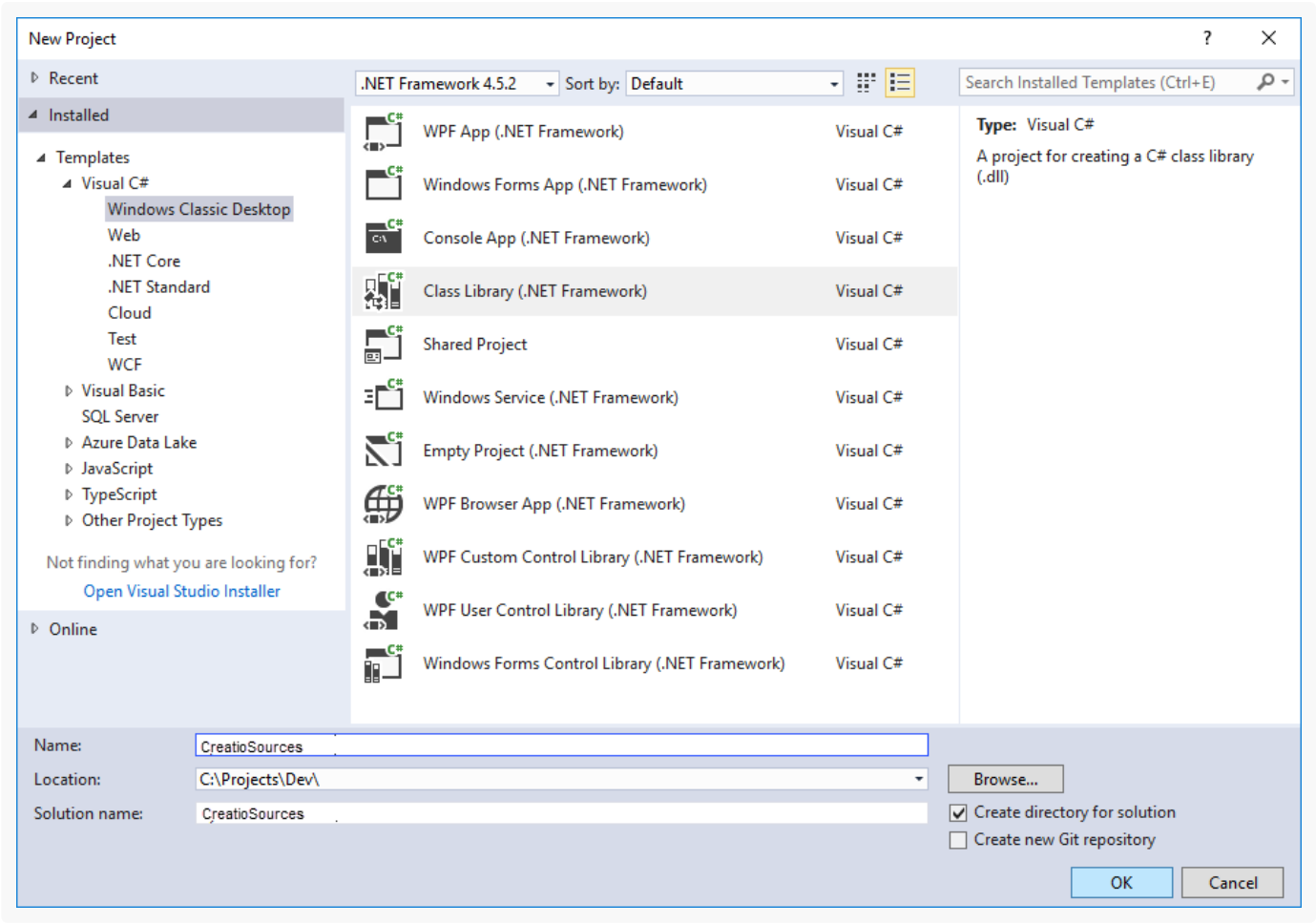
Важно. Для отладки исходного кода создавать проект Visual Studio не обязательно. Достаточно открыть в Visual Studio нужные файлы. Однако если отладка выполняется часто или необходимо работать с большим количеством файлов одновременно, то создание проекта сделает работу более удобной.

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

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



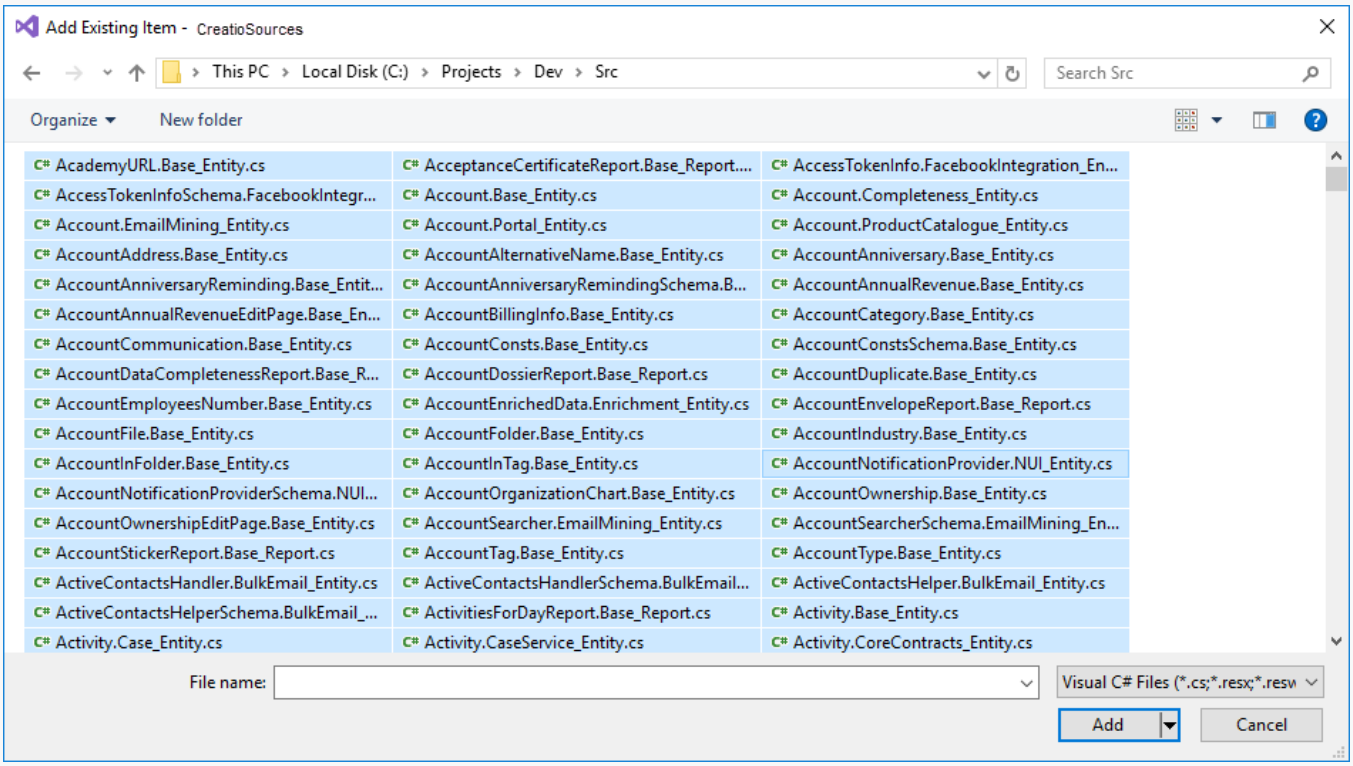
2. В окне свойств создаваемого проекта выберите тип проекта [*Class Library (.NET Framework)*], укажите название и расположение проекта.



3. 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

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

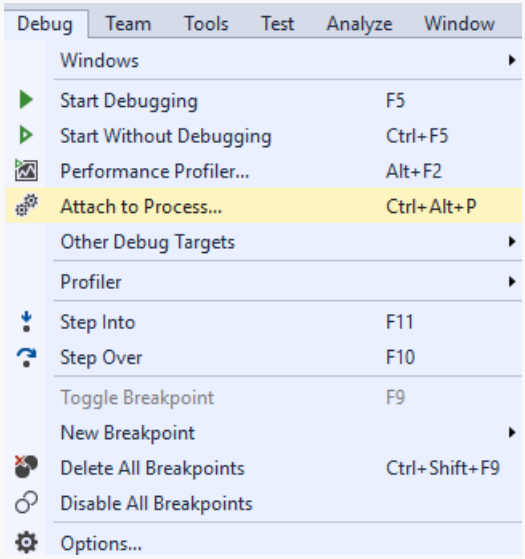


На заметку. В проект Visual Studio можно добавлять только необходимые для отладки файлы. Но тогда переход между методами при отладке будет ограничен только методами классов, реализованных в добавленных в проект файлах.

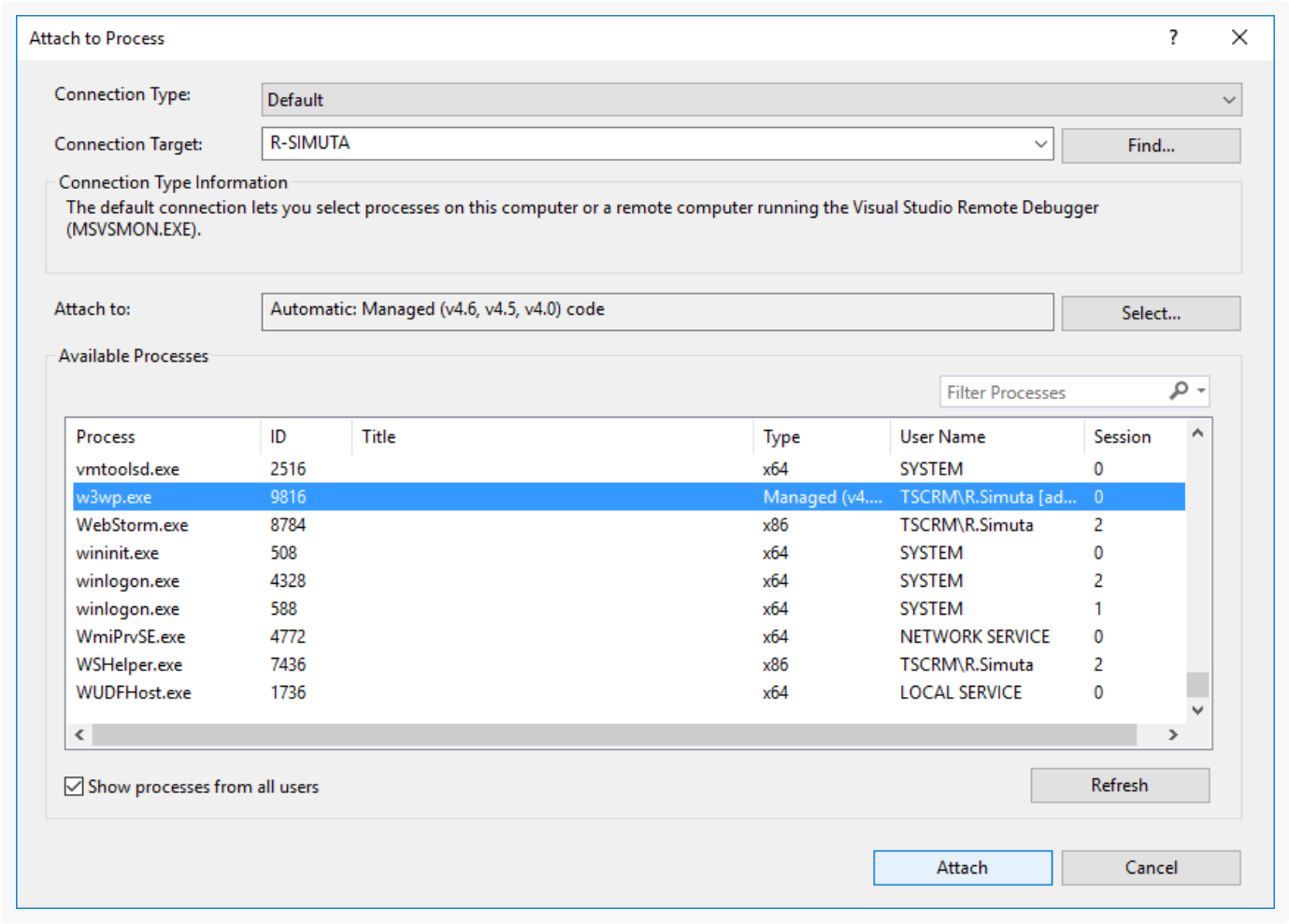
3. После добавления файлов сохраните проект.

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.

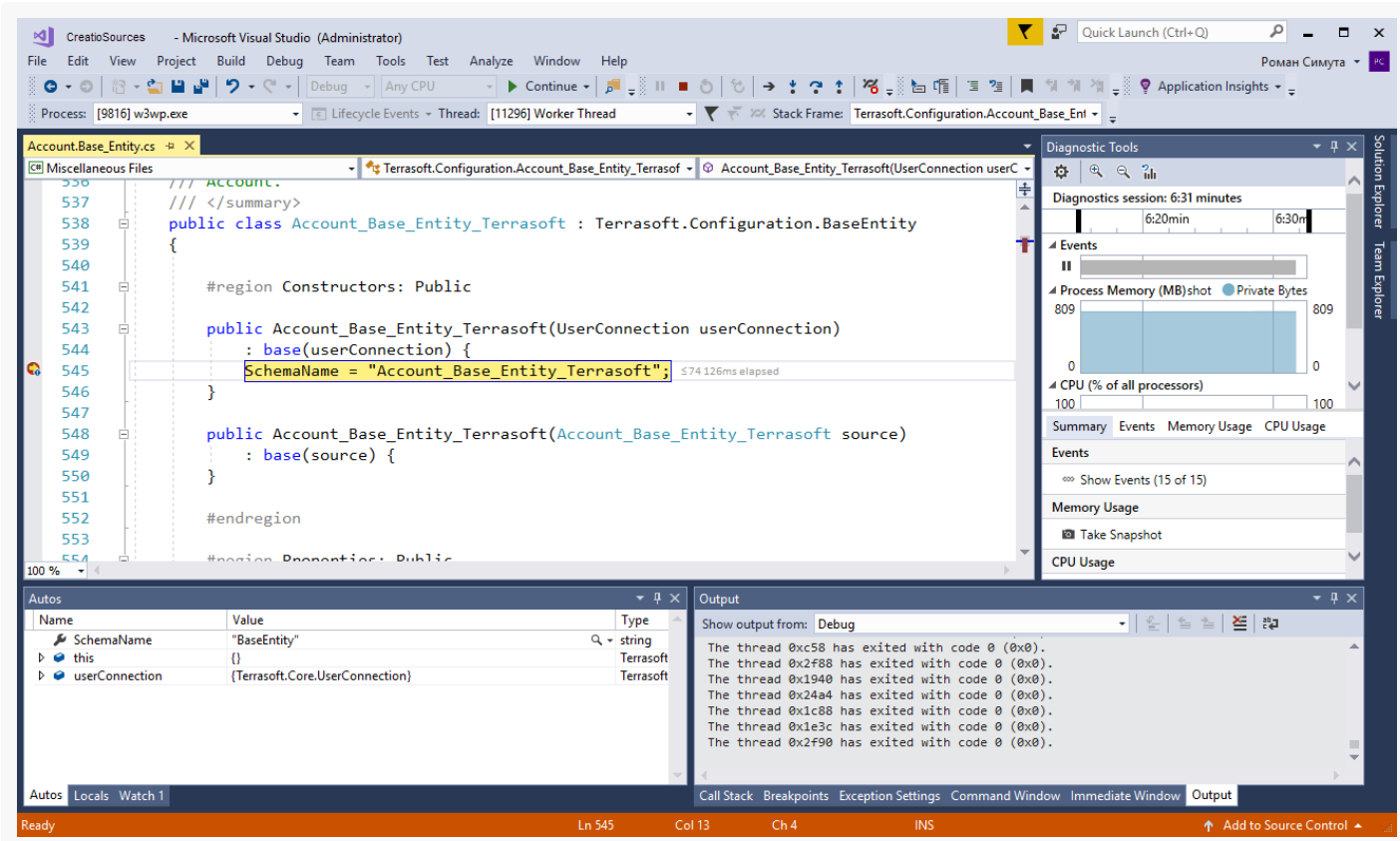


Важно. Название рабочего процесса может различаться в зависимости от конфигурации используемого сервера IIS. Для полнофункционального IIS Web Server процесс называется `w3wp.exe`, для IIS Express — `iisexpress.exe`.

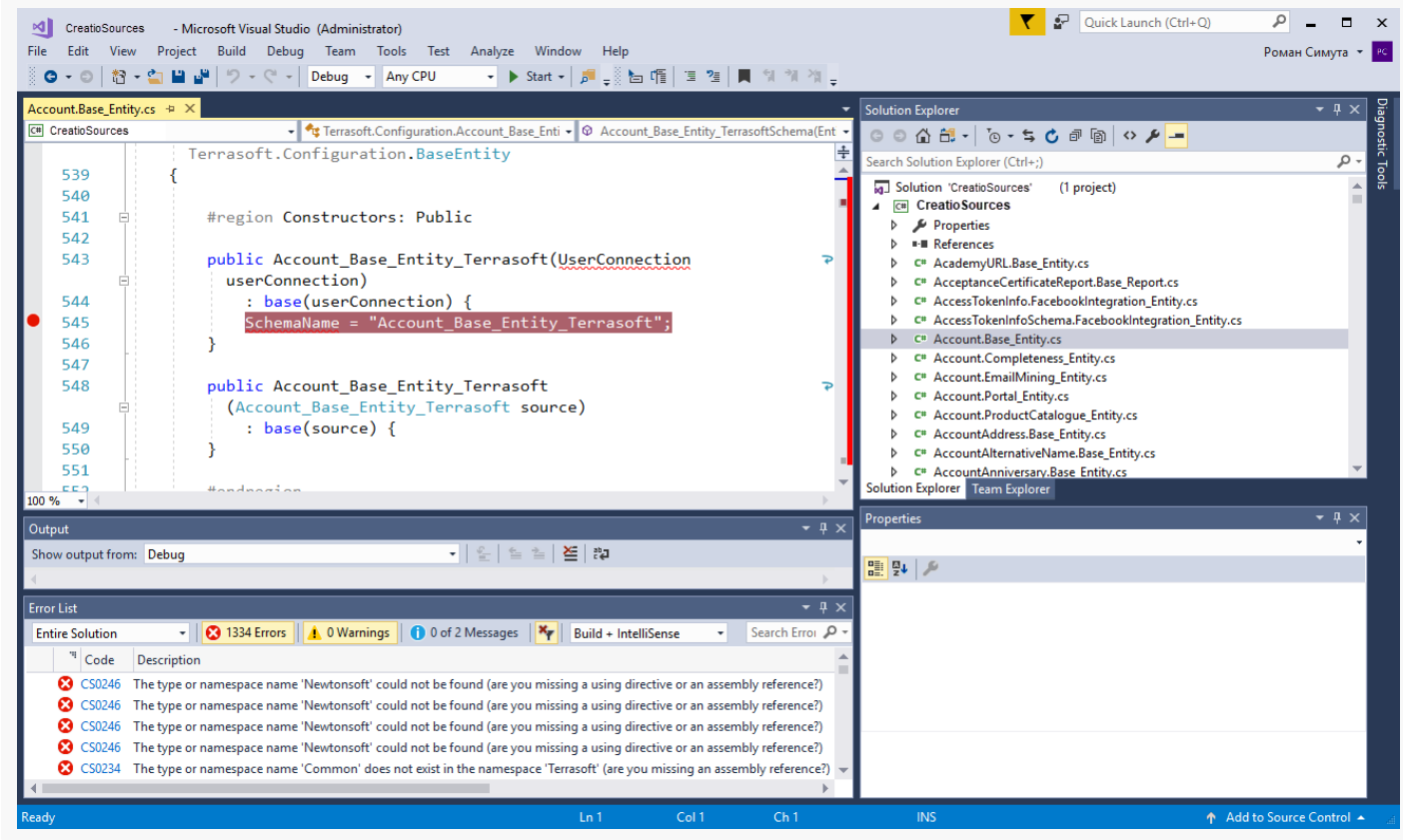
По умолчанию рабочий процесс IIS запущен под учетной записью, имя которой совпадает с именем пула приложения. Чтобы отобразить процессы всех пользователей, а не только текущего, необходимо установить признак [*Show processes from all users*].

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.



Front-end отладка

Сложный

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

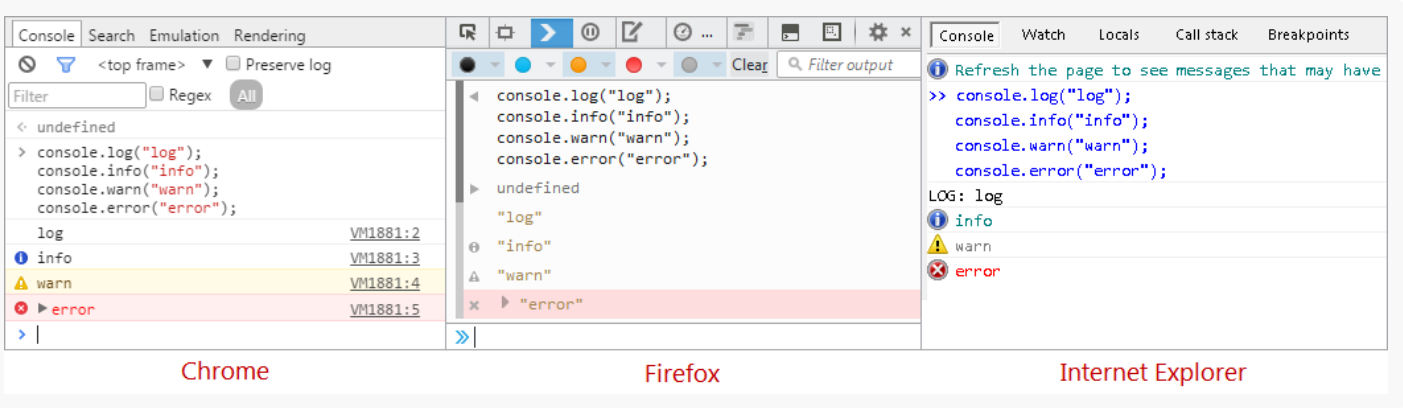
В консоли можно выводить отладочную информацию различного характера:

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

Для этого используются соответствующие методы объекта `console`.

Метод	Описание	Chrome	Firefox	Internet Explorer
`console.log(object [, object, ...])`	Выводит в консоль аргументы, разделяя их запятыми. Используется для вывода различных сообщений общего назначения.	+	+	+
`console.info(object [, object, ...])`	Аналогичен методу `log()`, но выводит сообщения в другом стиле, за счет чего позволяет акцентировать внимание на их важности.	+	+	+
`console.warn(object [, object, ...])`	Выводит в консоль сообщение предупреждающего характера.	+	+	+
`console.error(object [, object, ...])`	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
`%s`	Строка	`console.log("%s – один из флагманских продуктов компании %s", "Creatio sales", "Terrasoft");`
`%d`, `%i`	Число	`console.log("Платформа %s впервые была выпущена в %d году", "Creatio", 2011);`
`%f`	Число с плавающей точкой	`console.log("Число Пи равно %f", Math.PI);`
`%o`	DOM-элемент (не поддерживается IE)	`console.log("DOM-представление элемента <body/>: %o", document.getElementsByTagName("body")[0]);`
`%O`	Объект Java Script (не поддерживается IE, Firefox)	`console.log("Объект: %O", {a:1, b:2});`
`%c`	Стиль CSS (не поддерживается IE)	`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");`

Трассировка и проверки

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

Метод	Описание	Chrome	Firefox	Internet Explorer
`console.trace()`	Выводит стек вызовов из точки кода, откуда был вызван метод. Стек вызовов включает в себя имена файлов, номера строк, а также счетчик вызовов метода `trace()` из одной и той же точки.	+	+	+ (11+)
`console.assert(expression[, object, ...])`	Выполняет проверку выражения, переданного в качестве параметра `expression`. Если выражение ложно, то выводит в консоль ошибку вместе со стеком вызовов (`console.error()`), 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
`console.time(label)`	Включает счетчик миллисекунд с меткой `label`.	+	+	+ (11+)
`console.timeEnd(label)`	Останавливает счетчик миллисекунд с меткой `label` 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
`console.profile(label)`	Запускает профайлер Java Script, затем показывает результаты под меткой `label`.	+	+ (при открытой панели DevTools)	+ (10+)
`console.profileEnd(label)`	Останавливает профайлер Java Script.	+	+ (при открытой панели DevTools)	+ (10+)

Результаты профилирования отображаются в браузерах:

- Chrome — на вкладке [*Profiles*];

- Firefox — на вкладке [*Perfomance*];
- 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.

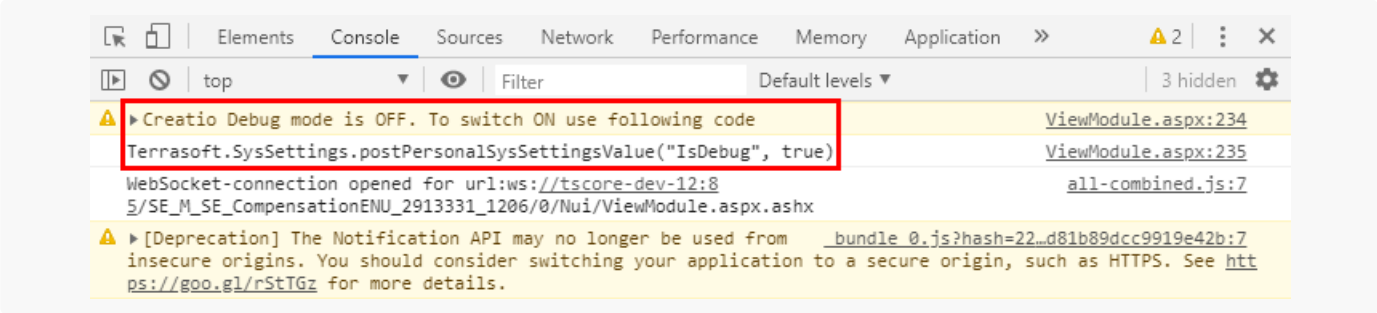
В обычном режиме работы в браузере выполняется минификация кода. Это означает что сборка клиентских скриптов осуществляется в файл `all-combined.js`. Файл собирается в момент создания сборки и содержит всю функциональность. Включение режима `isDebug` отключает сборку и минификацию *.js-файлов и позволяет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



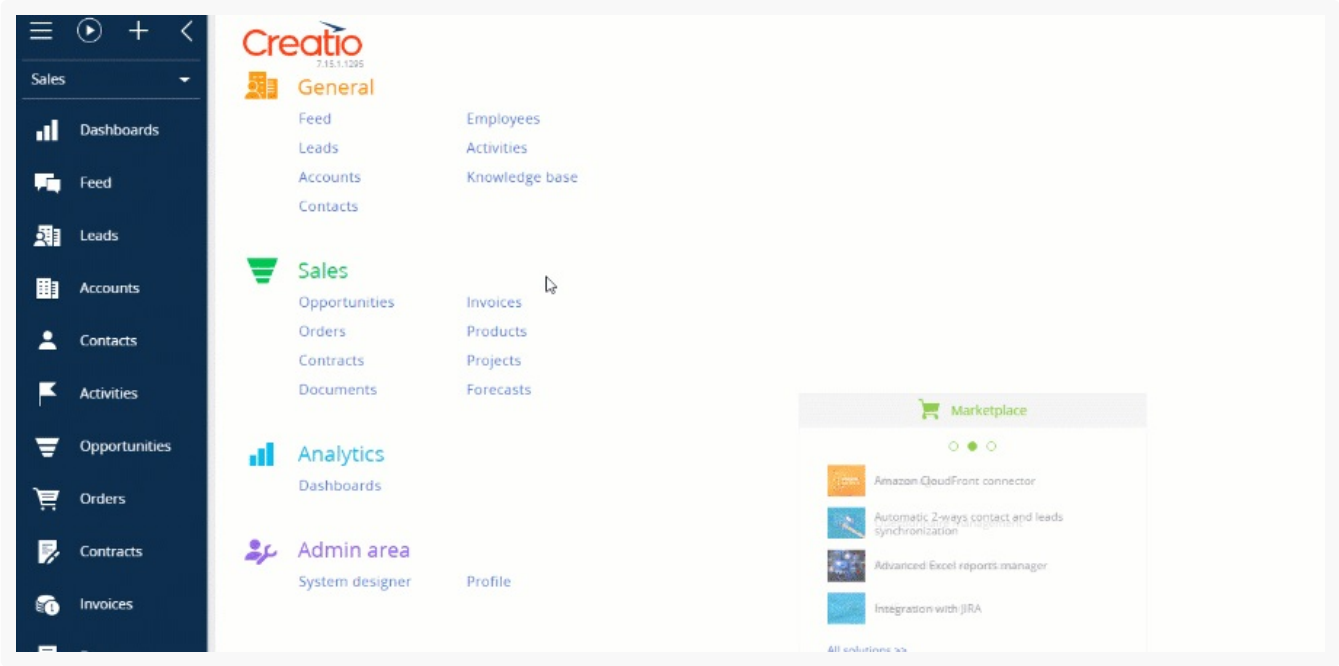
2. **Включите режим клиентской отладки**

   Это можно сделать следующими **способами**:

   - Выполнить в консоли браузера код:

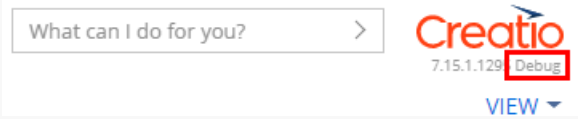
     ```
     Terrasoft.SysSettings.postPersonalSysSettingsValue("IsDebug", true)
     ```

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



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

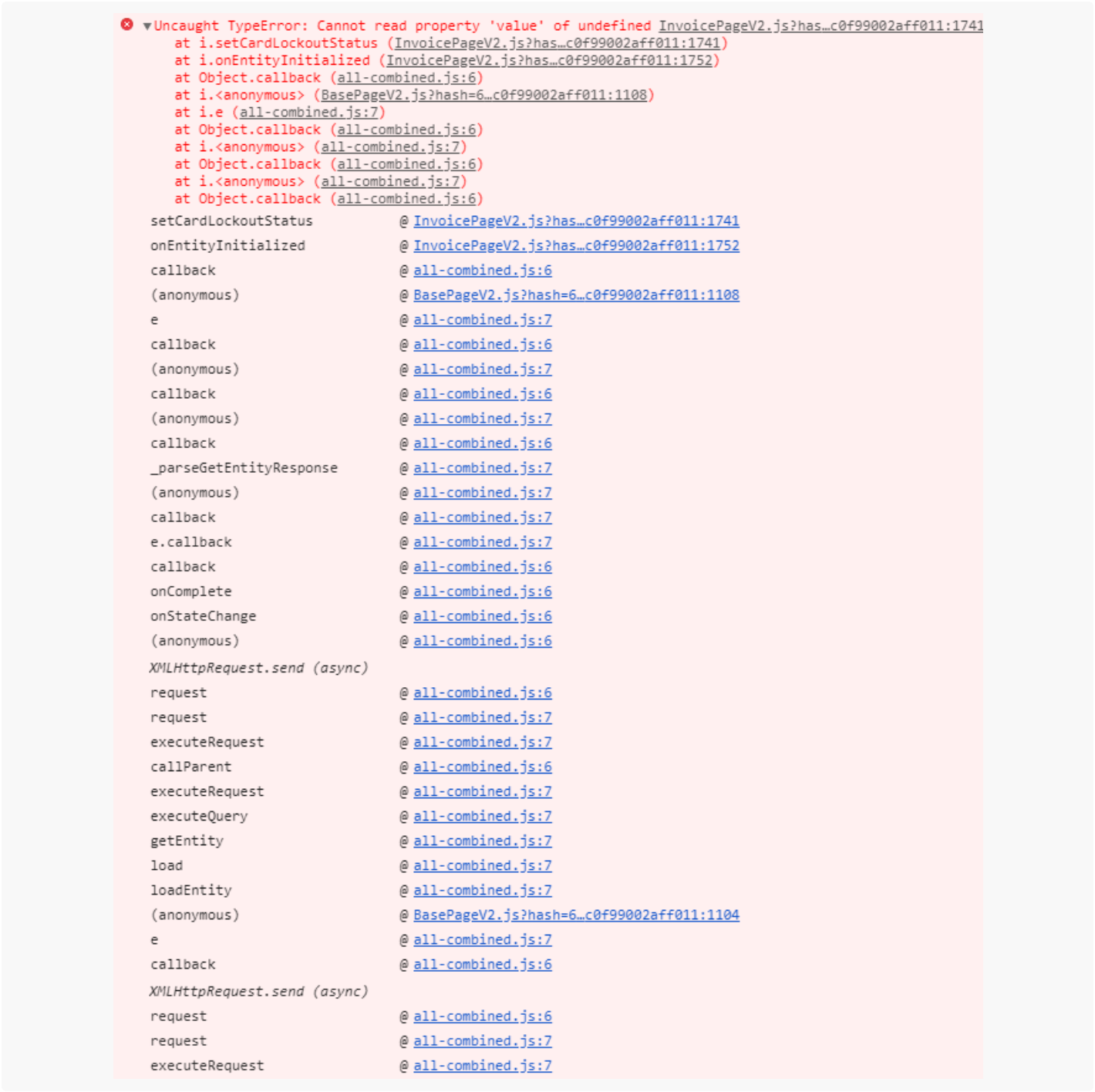
   После включения режима клиентской отладки возле номера версии сайта отобразится **индикатор** `Debug`.



> **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 `isDebug`.

Отображение информации об ошибке (`isDebug` 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
```