

Отладка

Front-end 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

Front-end отладка	4
Интегрированные инструменты отладки	4
Скрипты и точки останова	4
Управление выполнением отладки	5
Использование консоли браузера	6
Режим клиентской отладки isDebug	11

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.

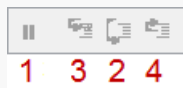
Команды для **пошаговой навигации** по коду в отладчиках браузеров:

- 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

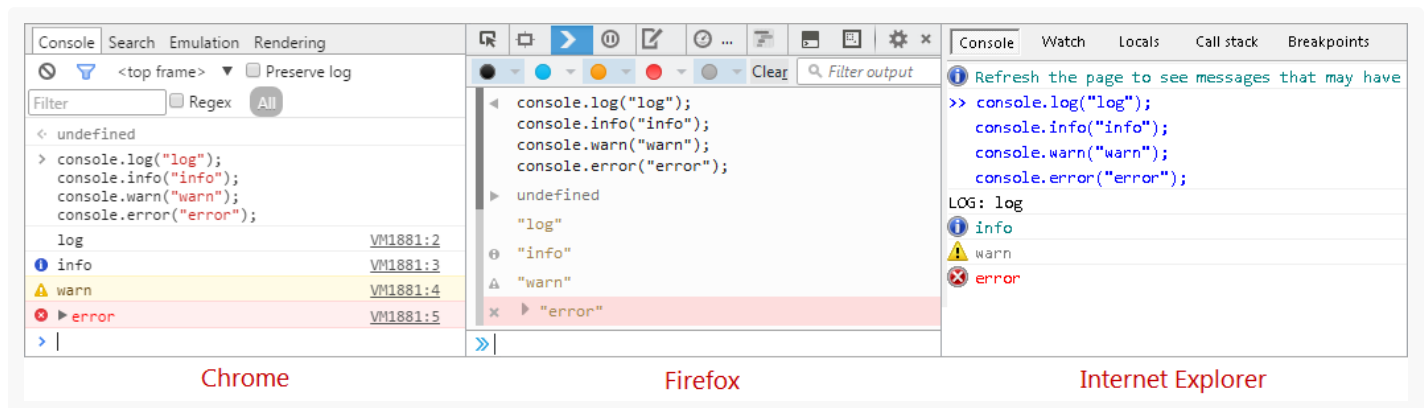
В консоли можно выводить отладочную информацию различного характера:

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

Для этого используются соответствующие методы объекта `console`.

Метод	Описание	Chrome	Firefox	Internet Explorer
<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 JavaScript (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.

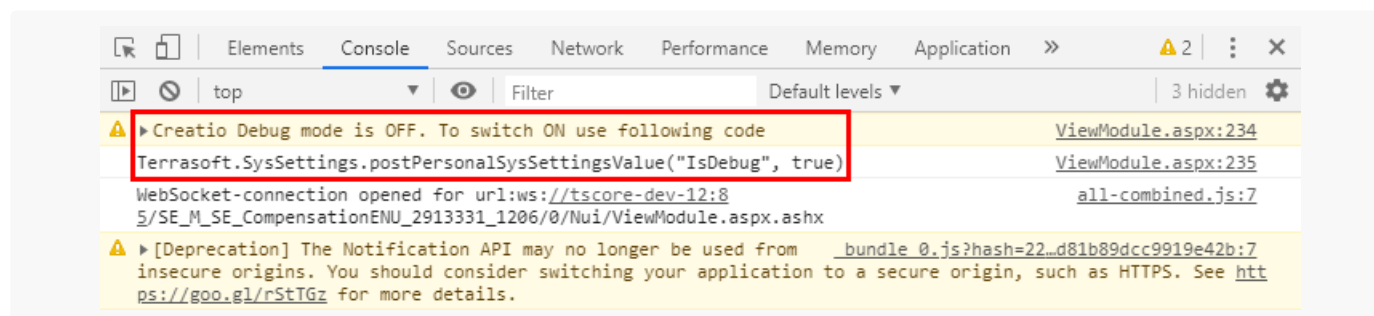
В обычном режиме работы в браузере выполняется [минификация кода](#). Это означает что сборка клиентских скриптов осуществляется в файл `all-combined.js`. Файл собирается в момент создания сборки и содержит всю функциональность. Включение режима `isDebug` отключает сборку и минификацию *.js-файлов и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



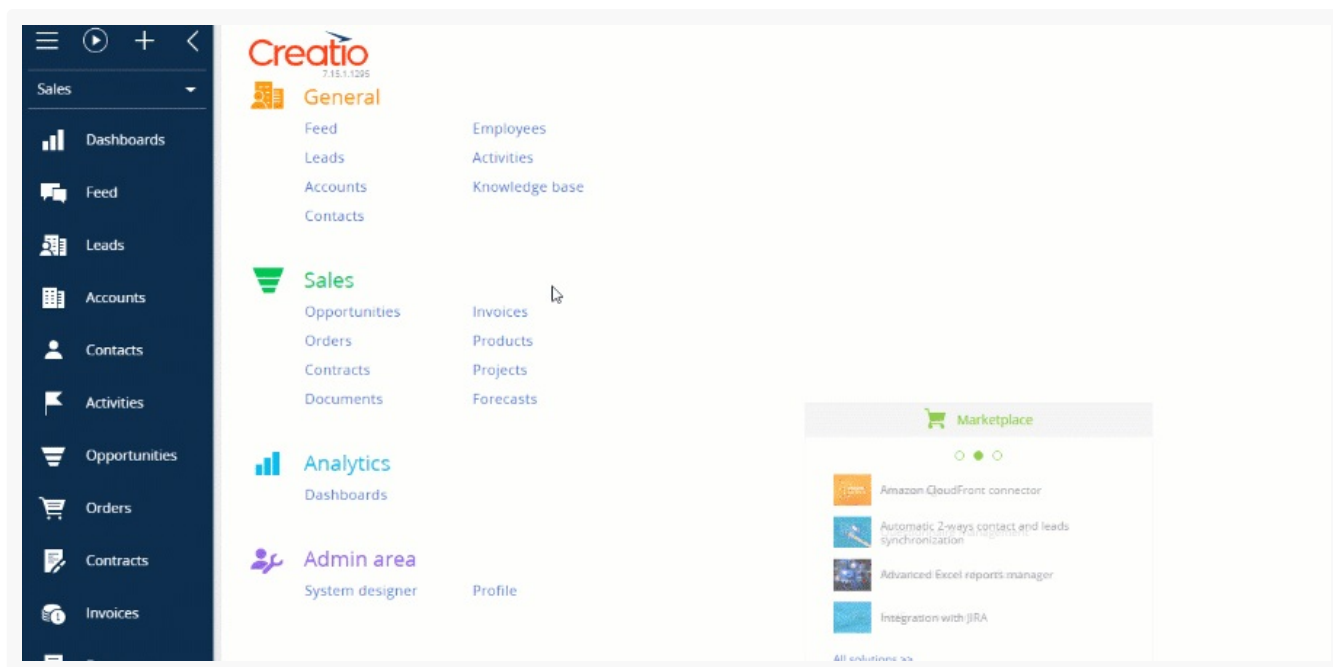
2. Включите режим клиентской отладки

Это можно сделать следующими **способами**:

- Выполнить в консоли браузера код:

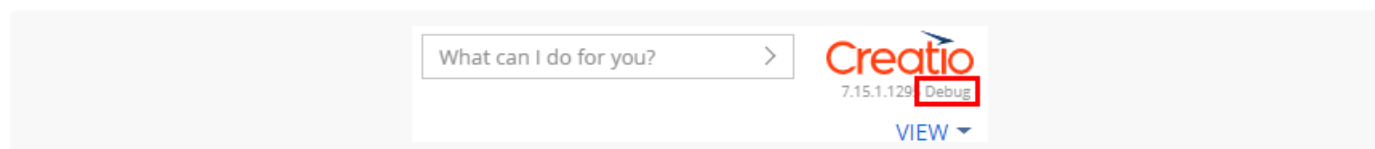
```
Terrasoft.SysSettings.postPersonalSysSettingsValue("IsDebug", true)
```

- 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 `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

```