

# Разработка решений в Creatio

Front-end (JS)

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 (JS)	4
Компоненты front-end ядра приложения	4
Асинхронное определение модулей	5
Модульная разработка в Creatio	6

# Front-end (JS)

## Основы

Front-end платформы — это набор модулей, которые определяются и загружаются асинхронно по мере необходимости.

Основой front-end Creatio является ядро.

Уровень ядра предоставляет:

- возможность использования объектно-ориентированного подхода и механизма наследования;
- инструменты для определения и асинхронной загрузки модулей и их зависимостей;
- функциональность базовых элементов системы;
- механизм взаимодействия модулей.

Front-end разработка в Creatio осуществляется на уровне конфигурации и представляет собой создание новых и расширение базовых визуальных и не визуальных модулей, схем моделей представления.

## Компоненты front-end ядра приложения



## Внешние JS-библиотеки

**ExtJS** — JavaScript-фреймворк для разработки веб-приложений и пользовательских интерфейсов. В

Creatio ExtJS используется как механизм создания структуры классов клиентской части ядра. Позволяет реализовать объектно-ориентированный подход, который в чистом виде не реализован в JavaScript. Предоставляет возможность создавать классы, реализовывать иерархию наследования, группировать классы в пространства имен.

**RequireJS** — библиотека, которая реализует подход [Asynchronous Module Definition \(AMD\)](#). Подход AMD декларирует механизм определения и асинхронной загрузки модулей и их зависимостей.

**Angular** — JavaScript-фреймворк для разработки одностраничных приложений. Его цель — расширение браузерных приложений на основе MVC-шаблона, а также упрощение тестирования и разработки. В Creatio реализована возможность встраивания кастомных Angular-компонентов с использованием единого ядра Angular.

## Базовые JS-классы

Неизменяемая часть системы. Эти классы:

- Обеспечивают работу клиентских модулей конфигурации.
- Определяют функциональность основных объектов системы, элементов управления, перечислений и констант.
- Хранятся в виде исполняемых файлов на диске в составе папок приложения.

## Механизм сообщений

**Sandbox** — компонент ядра, который служит диспетчером при взаимодействии модулей системы.

Sandbox предоставляет механизм обмена сообщениями между модулями (методы `sandbox.publish()` и `sandbox.subscribe()`) и загрузки модулей по требованию в интерфейс приложения (метод `sandbox.load()`).

## Асинхронное определение модулей

Front-end платформы имеет **модульную структуру**.

**Модуль** — инкапсулированный в обособленный блок набор функциональности, который в свою очередь может использовать другие модули в качестве зависимостей.

Создание модулей в JavaScript декларируется паттерном программирования "Модуль". Классическим приемом реализации этого паттерна является использование анонимных функций, возвращающих определенное значение (объект, функцию и т. д.), которое ассоциируется с модулем. При этом значение модуля экспортируется в глобальный объект.

Для управления большим количеством модулей в Creatio загрузка модулей и их зависимостей выполняется в соответствии с подходом **Asynchronous Module Definition (AMD)**.

Подход AMD декларирует механизм определения и асинхронной загрузки модулей и их зависимостей, который в процессе работы с системой позволяет подгружать только те данные, которые необходимы для работы в текущий момент. В Creatio для работы с модулями используется загрузчик **RequireJS**.

**Принципы** определения модулей в Creatio:

- Объявление модуля выполняется в специальной функции `define()`, которая регистрирует функцию-фабрику для инстанцирования модуля, но при этом не загружает его немедленно в момент вызова.
- Зависимости модуля передаются как массив строковых значений, а не через свойства глобального

объекта.

- Загрузчик выполняет загрузку всех модулей-зависимостей, переданных в качестве аргументов в `define()`. Модули загружаются асинхронно, при этом фактически порядок их загрузки определяется загрузчиком произвольно.
- После того как загрузчиком будут загружены все указанные зависимости модуля, будет вызвана функция-фабрика, которая вернет значение модуля. При этом в функцию-фабрику в качестве аргументов будут переданы загруженные модули-зависимости.

## Модульная разработка в Creatio

Реализация всей пользовательской функциональности выполняется в **клиентских модулях**.

Иерархия модулей в Creatio



Несмотря на некоторые функциональные различия, все клиентские модули Creatio имеют одинаковую структуру описания, которая соответствует формату описания модулей AMD.

### Общая структура описания клиентского модуля

```
define(
    "ModuleName",
    "dependencies",
    function(dependencies) {
        // someMethods...
        return { moduleObject };
    });
```

```
});
```

- `ModuleName` — имя модуля;
- `dependencies` — модули-зависимости, функциональность которых можно использовать в текущем модуле;
- `moduleObject` — конфигурационный объект созданного модуля.

**Виды** клиентских модулей в Creatio:

- Невизуальный модуль.
- Визуальный модуль.
- Схема модели представления.

## Невизуальный модуль

Содержит реализацию функциональности системы, которая не сопряжена с привязкой данных и отображением их в интерфейсе.

### Структура описания невидуального модуля

```
define(
  "ModuleName",
  "dependencies",
  function(dependencies) {
    // Методы, реализующие необходимую бизнес-логику.
    return;
  });
```

Примерами невидуальных модулей являются утилитные модули, которые реализуют служебные функции.

## Визуальный модуль

Используется для создания пользовательских визуальных элементов системы.

Реализует модель представления (ViewModel) согласно шаблону MVVM. Инкапсулирует в себе данные, которые отображаются в элементах управления графического интерфейса, а также методы работы с этими данными.

Визуальный модуль содержит **методы**:

- `init()` — метод инициализации модуля.  
Отвечает за инициализацию свойств объекта класса, а также за подписку на сообщения.
- `render(renderTo)` — метод отрисовки представления модуля в DOM.  
Должен возвращать представление.

Принимает единственный аргумент `renderTo` — элемент, в который будет вставлено представление объекта модуля.

- `destroy()` — метод, отвечающий за удаление представления модуля, удаление модели представления, отписку от ранее подписанных сообщений и уничтожение объекта класса модуля.

Для создания визуального модуля можно использовать базовые классы ядра. Например, создать в модуле класс-наследник `Terrasoft.configuration.BaseModule` или `Terrasoft.configuration.BaseSchemaModule`. Это базовые классы, в которых в необходимой степени уже реализованы необходимые методы визуального модуля — `init()`, `render(renderTo)` и `destroy()`.

### Структура описания визуального модуля (наследника базового класса Terrasoft.BaseMo...

```
define("ModuleName", "dependencies", function(dependencies) {
    // Определение класса модуля.
    Ext.define("Terrasoft..configuration.className") {
        alternateClassName: "Terrasoft.className"
        extend: "Terrasoft.BaseModule",
        ...
        // Свойства и методы.
        ...
    };
    // Создание объекта модуля.
    return Ext.create(Terrasoft.className)
});
```

Примерами визуальных модулей являются модули, реализующие функциональность элементов управления на странице приложения.

## Схема модели представления

Наиболее частыми задачами кастомизации приложения являются создание и доработка основных элементов интерфейса — разделов, страниц, деталей.

Модули этих элементов имеют **шаблонизированную структуру** и называются схемами моделей представления.

Схема модели представления является конфигурационным объектом для генерации представления и модели представления генераторами Creatio `ViewGenerator` и `ViewModelGenerator`.

Для большинства задач кастомизации в Creatio используется механизм замещения базовых схем. К таким схемам относятся все схемы из предустановленных пакетов конфигурации.

Основные базовые схемы моделей представления:

- `BasePageV2`
- `BaseSectionV2`
- `BaseDetailV2`



## Структура схемы модели представления

```
define("SchemaName", "dependencies",
  function(dependencies) {
    return {
      entitySchemaName: "ExampleEntity",
      mixins: {},
      attributes: {},
      messages: {},
      methods: {},
      rules: {},
      businessRules: {},
      modules: {},
      diff:
    };
  });
```

Примерами схем моделей представления являются схемы страниц, разделов и деталей.