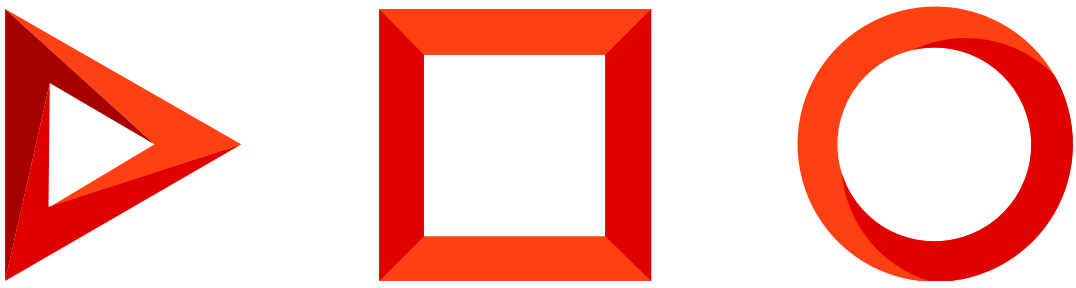
Creatio Academy

Front-end разработка

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.

© 2022 Terrasoft. Все права защищены.

Содержание

© 2022 Terrasoft. Все права защищены.

© 2022 Terrasoft. Все права защищены.

© 2022 Terrasoft. Все права защищены.

© 2022 Terrasoft. Все права защищены.

Понятие модуля

Сложный

Концепция AMD

Front-end часть приложения Creatio представляет собой набор блоков функциональности, каждый из которых реализован в отдельном **модуле**. Согласно **концепции** Asynchronous Module Definition (AMD), в процессе работы приложения выполняется асинхронная загрузка модулей и их зависимостей. Таким образом, концепция AMD позволяет подгружать только те данные, которые необходимы для работы в текущий момент.

Концепция AMD поддерживается различными JavaScript-фреймворками. В Creatio для работы с модулями используется **загрузчик** RequireJS.

Модульная разработка в Creatio

Модуль — фрагмент кода, инкапсулированный в обособленный блок, который может быть загружен и самостоятельно выполнен.

Создание модулей в специфике JavaScript декларируется **паттерном программирования** "Модуль". Классическая **реализация паттерна** — использование анонимных функций, возвращающих определенное значение (объект, функцию и т. д.), которое ассоциируется с модулем. При этом значение модуля экспортируется в глобальный объект.

Пример экспорта значения модуля в глобальный объект

```
// Немедленно вызываемое функциональное выражение (IIFE). Анонимная функция,
// которая инициализирует свойство myGlobalModule глобального объекта функцией,
// возвращающей значение модуля. Таким образом, фактически происходит загрузка модуля,
// к которому в дальнейшем можно обращаться через глобальное свойство myGlobalModule.
(function () {
    // Обращение к некоторому модулю, от которого зависит текущий модуль.
    // Этот модуль на момент обращения к нему должен быть загружен
    // в глобальную переменную SomeModuleDependency.
    // Контекст this в данном случае – глобальный объект.
    var moduleDependency = this.SomeModuleDependency;
    // Объявление в свойстве глобального объекта функции, возвращающей значение модуля.
    this.myGlobalModule = function () { return someModuleValue; };
}());
```

Интерпретатор, обнаруживая в коде такое функциональное выражение, сразу вычисляет его. В результате выполнения в свойство `myGlobalModule` глобального объекта будет помещена функция, которая будет возвращать значение модуля.

Особенности подхода:

© 2022 Terrasoft. Все права защищены.

- Сложность декларирования и использования модулей-зависимостей.
- В момент выполнения анонимной функции все зависимости модуля должны быть загружены.
- Загрузка модулей-зависимостей выполняется в заголовке страницы через HTML-элемент `<script>`. Обращение к модулям-зависимостям осуществляется через имена глобальных переменных. При этом разработчик должен четко представлять и реализовывать порядок загрузки модулей-зависимостей.
- Как следствие предыдущего пункта — модули загружаются до начала рендеринга страницы, поэтому в модулях нельзя обращаться к элементам управления страницы для реализации пользовательской логики.

Особенности использования подхода в Creatio:

- Отсутствие возможности динамической загрузки модулей.
- Применение дополнительной логики при загрузке модулей.
- Сложность управления большим количеством модулей со многими зависимостями, которые могут перекрывать друг друга.

Загрузчик RequireJS

Загрузчик RequireJS предоставляет механизм объявления и загрузки модулей, базирующийся на концепции AMD, и позволяющий избежать перечисленных выше недостатков.

Принципы работы механизма загрузчика RequireJS:

- Объявление модуля выполняется в функции `define()`, которая регистрирует функцию-фабрику для инстанцирования модуля, но при этом не загружает его в момент вызова.
- Зависимости модуля передаются как массив строковых значений, а не через свойства глобального объекта.
- Загрузчик выполняет загрузку модулей-зависимостей, переданных в качестве аргументов в функции `define()`. Модули загружаются асинхронно, при этом порядок их загрузки произвольно определяется загрузчиком.
- После загрузки указанных зависимостей модуля будет вызвана функция-фабрика, которая вернет значение модуля. При этом в нее в качестве аргументов будут переданы загруженные модули-зависимости.

Пример объявления модуля

Основы

Пример использования функции `define()` для объявления модуля `SumModule`

```
// Модуль с именем SumModule реализует функциональность суммирования двух чисел.
// SumModule не имеет зависимостей.
// Поэтому в качестве второго аргумента передается пустой массив, а
// анонимной функции-фабрике не передаются никакие параметры.
define("SumModule", [], function () {
```

© 2022 Terrasoft. Все права защищены.

```
    // Тело анонимной функции содержит внутреннюю реализацию функциональности модуля.
    var calculate = function (a, b) { return a + b; };
    // Возвращаемое функцией значение — объект, которым является модуль для системы.
    return {
        // Описание объекта. В данном случае модуль представляет собой объект со свойством summ.
        // Значение этого свойства — функция с двумя аргументами, которая возвращает сумму этих
        summ: calculate
    };
});
```

Функция define()

Основы

Назначение функции `define()` — объявление асинхронного модуля в исходном коде, с которым будет работать загрузчик.

Объявление модуля

```
define(
        ModuleName,
        [dependencies],
        function (dependencies) {
        }
);
```

Параметры

`ModuleName`

Строка с именем модуля. Необязательный параметр.

Если не указать параметр, загрузчик самостоятельно присвоит модулю имя в зависимости от его расположения в дереве скриптов приложения. Для обращения к модулю из других частей приложения (в том числе для асинхронной загрузки как зависимости другого модуля), имя модуля должно быть однозначно определено.

`dependencies`

Массив имен модулей, от которых зависит модуль. Необязательный параметр.

RequireJS выполняет асинхронную загрузку зависимостей, переданных в массиве. Порядок перечисления зависимостей в массиве `dependencies` должен соответствовать порядку перечисления параметров, передаваемых в фабричную функцию. Фабричная функция будет вызвана после загрузки зависимостей, перечисленных в `dependencies`.

© 2022 Terrasoft. Все права защищены.

`function(dependencies)`

Анонимная фабричная функция, которая инстанцирует модуль. Обязательный параметр.

В качестве параметров в функцию передаются объекты, которые ассоциируются загрузчиком с модулями–зависимостями, перечисленными в аргументе `dependencies`. Через эти аргументы осуществляется доступ к свойствам и методам модулей–зависимостей внутри создаваемого модуля. Порядок перечисления модулей в `dependencies` должен соответствовать порядку параметров фабричной функции.

Фабричная функция должна возвращать значение, которое загрузчик будет ассоциировать как экспортируемое значение создаваемого модуля.

Виды возвращаемого значения фабричной функции:

- Объект, которым является модуль для системы. Модуль сохраняется в кэше браузера после первичной загрузки на клиент. Если объявление модуля было изменено после загрузки на клиент (например, в процессе реализации конфигурационной логики), то необходимо очистить кэш и заново загрузить модуль.
- Функция-конструктор модуля. В качестве аргумента в конструктор передается объект контекста, в котором будет создаваться модуль. Загрузка модуля приведет к созданию на клиенте экземпляра модуля (**инстанцируемого модуля**). Повторная загрузка модуля на клиент функцией `require()` приведет к созданию еще одного экземпляра модуля. Эти экземпляры одного и того же модуля система будет воспринимать как два самостоятельных модуля. Примером объявления инстанцируемого модуля является модуль `CardModule` пакета `NUI`.

Клиентская схема

Средний

Клиентская схема модели представления — это схема визуального модуля, с помощью которой реализуется front-end часть приложения. Клиентская схема модели представления является конфигурационным объектом для генерации представления и модели представления генераторами Creatio `ViewGenerator` и `ViewModelGenerator`. Типы модулей и их особенности описаны в статье Виды клиентских модулей.

Разработка клиентской схемы

Способы разработки клиентских схем модели представления:

- Раздел [*Конфигурация*] ([*Configuration*]). Разработка с помощью раздела [*Конфигурация*] описана в статье Схема модели представления.
- Мастер разделов. Разработка раздела с помощью мастера разделов описана в статье Добавить новый раздел.
- Мастер деталей. Разработка деталей с помощью мастера деталей описана в статье Создать новую деталь.

Структурные элементы клиентской схемы:

© 2022 Terrasoft. Все права защищены.

- Автоматически сгенерированный код — содержит описание схемы, ее зависимостей, локализованных ресурсов и сообщений.
- Стили, используемые для визуализации (присутствуют не во всех типах клиентских схем).
- Исходный код схемы — синтаксически правильный код на языке JavaScript, который является определением модуля.

Маркерные комментарии необходимо использовать в исходном коде схемы для свойств `diff`, `modules`, `details` и `businessRules`.

Назначение маркерных комментариев — однозначное определение свойств клиентской схемы. При открытии мастера выполняется валидация наличия маркерных комментариев, которые представлены в таблице ниже.

Правила валидации маркерных комментариев в клиентских схемах

Тип схемы	Необходимые маркерные комментарии

Схема представления модели страницы записи `EditViewModelSchema`	`details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,`
	`modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,`
	`diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/,`
	`businessRules: /**SCHEMA_BUSINESS_RULES*/{}/**SCHEMA_BUSI`

Схема представления модели раздела `ModuleViewModelSchema`	`modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,` `diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/`
Схема представления модели детали с полями `EditControlsDetailViewModelSchema`	
Схема представления модели детали `DetailViewModelSchema`	
Схема представления модели детали с реестром `GridDetailViewModelSchema`	

Свойства клиентской схемы

Исходный код клиентских схем имеет общую структуру, которая представлена ниже.

Структура исходного кода клиентской схемы

© 2022 Terrasoft. Все права защищены.

```
define("ExampleSchema", [], function() {
    return {
        entitySchemaName: "ExampleEntity",
        mixins: {},
        attributes: {},
        messages: {},
        methods: {},
        rules: {},
        businessRules: /**SCHEMA_BUSINESS_RULES*/{}/**SCHEMA_BUSINESS_RULES*/,
        modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

После загрузки модуля выполняется вызов анонимной функции-фабрики, которая возвращает конфигурационный объект схемы. **Свойства** конфигурационного объекта схемы:

- `entitySchemaName` — имя схемы объекта, который используется текущей клиентской схемой.
- `mixins` — конфигурационный объект, который содержит объявление миксинов.
- `attributes` — конфигурационный объект, который содержит атрибуты схемы.
- `messages` — конфигурационный объект, который содержит сообщения схемы.
- `methods` — конфигурационный объект, который содержит методы схемы.
- `rules` — конфигурационный объект, который содержит бизнес-правила схемы.
- `businessRules` — конфигурационный объект, который содержит бизнес-правила схемы, созданные или измененные мастером разделов или мастером деталей. Маркерные комментарии `/**SCHEMA_BUSINESS_RULES*/` обязательны к использованию для работы мастеров.
- `modules` — конфигурационный объект, который содержит модули схемы. Маркерные комментарии `/**SCHEMA_MODULES*/` обязательны к использованию для работы мастеров.

  > **На заметку.** Для загрузки детали на страницу используется свойство `details`. Поскольку деталь является модулем, то правильным будет использование свойства `modules`.

- `diff` — массив конфигурационных объектов, который содержит описание представления схемы. Маркерные комментарии `/**SCHEMA_DIFF*/` обязательны к использованию для работы мастеров.
- `properties` — конфигурационный объект, который содержит свойства схемы модели представления.
- `$-свойства` — автоматически сгенерированные свойства для атрибутов схемы модели представления.

Имя схемы (`entitySchemaName`)

Для реализации имени схемы объекта необходимо использовать обязательное свойство `entitySchemaName`. Достаточно указать его в одной из схем иерархии наследования.

© 2022 Terrasoft. Все права защищены.

Пример объявления свойства `entitySchemaName`

```
define("ClientSchemaName", [], function () {
    return {
        /* Схема объекта (модель). */
        entitySchemaName: "EntityName",
        /* ... */
    };
});
```

Миксины (`mixins`)

Миксин — это класс-примесь, предназначенный для расширения функциональности других классов. В JavaScript нет множественного наследования, а наличие миксинов позволяет расширить функциональность схемы без дублирования логики, часто используемой в методах схемы. В разных клиентских схемах приложения Creatio может использоваться один и тот же набор действий. Чтобы не дублировать код в каждой схеме, необходимо создать миксин. **Особенность** миксинов в сравнении с другими модулями, подключаемыми в список зависимостей, — способ вызова методов из схемы модуля (к методам миксина можно обращаться напрямую, как к методам схемы). Для реализации миксинов необходимо использовать свойство `mixins`.

Алгоритм работы с миксинами:

1. Создайте миксин.
2. Присвойте миксину имя.
3. Подключите соответствующее пространство имен.
4. Реализуйте функциональность миксина.
5. Используйте миксин в клиентской схеме.

Создать миксин

Создание миксина аналогично созданию схемы объекта.

Присвоить миксину имя

При именовании миксинов в названии схемы необходимо использовать суффикс "`-able`" (например, `Serializable` — миксин, который добавляет компонентам способность сериализироваться). Если нет возможности сформулировать имя миксина в описанной выше форме, необходимо в название схемы добавить окончание "`Mixin`".

Важно. Использование слов `Utilities`, `Extension`, `Tools` и т. д. является недопустимым, поскольку не позволяет четко формализовать функциональность, которая скрывается за названием миксина.

© 2022 Terrasoft. Все права защищены.

Подключить пространство имен

Для миксина необходимо подключить соответствующее пространство имен (для конфигурации — `Terrasoft.configuration.mixins`, для ядра — `Terrasoft.core.mixins`).

Реализовать функциональность миксина

Миксины не должны зависеть от внутренней реализации схемы, к которой они будут применены. Это должен быть самодостаточный механизм, который принимает набор параметров, выполняет с ними действие и, при необходимости, возвращает результат. Миксины оформляются в виде модулей, которые необходимо подключить в список зависимостей схемы при ее объявлении функцией `define()`.

Структура миксина представлена ниже.

Структура миксина

```
define("ИмяМиксина", [], function() {
    Ext.define("Terrasoft.configuration.mixins.ИмяМиксина", {
        alternateClassName: "Terrasoft.ИмяМиксина",
        /* Функциональность миксина. */
    });
    return Ext.create(Terrasoft.ИмяМиксина);
})
```

Использовать миксин

Миксин реализует функциональность, необходимую в клиентской схеме. Чтобы получить набор действий миксина, необходимо указать его в свойстве `mixins` клиентской схемы.

Использование миксина в клиентской схеме

```
/* ИмяМиксина — модуль, в котором реализован класс миксина. */
define("ИмяКлиентскойСхемы", ["ИмяМодуля"], function () {
    return {
        /* ИмяСхемы — название схемы сущности. */
        entitySchemaName: "ИмяСхемы",
        mixins: {
            /* Подключение миксина. */
            ИмяМиксина: "Terrasoft.ПространствоИмен.ИмяМиксина"
        },
        attributes: {},
        messages: {},
        methods: {},
        rules: {},
        modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

© 2022 Terrasoft. Все права защищены.

После подключения можно пользоваться методами, атрибутами и полями миксина в клиентской схеме так, как будто они являются частью данной клиентской схемы. При этом вызовы методов получаются более краткими, чем при использовании отдельной схемы. Например, `getDefaultImageResource` — функция миксина. Для вызова функции миксина `getDefaultImageResource` в пользовательской схеме, к которой подключен миксин, необходимо записать `this.getDefaultImageResource();`.

На заметку. При переопределении функции из миксина необходимо в клиентской схеме создать функцию с аналогичным именем. В результате при вызове будет использоваться функция схемы, а не миксина.

Атрибуты (attributes)

Для реализации атрибутов необходимо использовать свойство `attributes`.

Сообщения (messages)

Назначение сообщений — организация обмена данными между модулями. Для реализации сообщений необходимо использовать свойство `messages`. Используя перечисление `Terrasoft.MessageMode`, можно задать **режим** сообщения.

Типы режимов сообщения

Режим сообщения	Описание	Подключение
Адресное	Адресные сообщения принимаются только последним подписанным подписчиком.	Для установки сообщения в адресный режим необходимо свойству `mode` присвоить значение `this.Terrasoft.MessageMode.PTP`.
Широковещательное	Широковещательные сообщения принимаются всеми подписчиками.	Для установки сообщения в широковещательный режим необходимо свойству `mode` присвоить значение `this.Terrasoft.MessageMode.BROADCAST`.

Кроме режимов, также можно задать **направление** сообщения.

© 2022 Terrasoft. Все права защищены.

Типы направлений сообщения

Направление сообщения	Описание	Подключение
Публикация	Сообщение может быть только опубликовано, т. е. является **исходящим**.	Для установки направления публикации сообщения необходимо свойству `direction` присвоить значение `this.Terrasoft.MessageDirectionType.PUBLISH`.
Подписка	На сообщение можно только подписаться, т. е. является **входящим**.	Для установки направления подписки на сообщение необходимо свойству `direction` присвоить значение `this.Terrasoft.MessageDirectionType.SUBSCRIBE`.
Двунаправленное	Позволяет публиковать и подписываться на одно и то же сообщение в разных экземплярах одного и того же класса или в рамках одной и той же иерархии наследования схем. В иерархии наследования схем не может быть объявлено одно и то же сообщение с разным направлением. Для таких случаев необходимо использовать двунаправленные сообщения, особенности которых описаны в статье Sandbox.	Соответствует значению перечисления `Terrasoft.MessageDirectionType.BIDIRECTIONAL`.

Публикация сообщения

В схеме, в которой необходимо осуществить публикацию сообщения, должно быть объявлено сообщение с направлением "Публикация".

Пример объявления сообщения с направлением "Публикация"

```
messages: {
    /* Имя сообщения. */
    "GetColumnsValues": {
```

© 2022 Terrasoft. Все права защищены.

```
        /* Режим сообщения — адресное. */
        mode: this.Terrasoft.MessageMode.PTP,
        /* Направление сообщения — публикация. */
        direction: this.Terrasoft.MessageDirectionType.PUBLISH
    }
}
```

Публикация осуществляется вызовом метода `publish` у экземпляра класса `sandbox`.

Пример публикации сообщения

```
/* Метод получения результата публикации сообщения GetColumnsValues. */
getColumnsValues: function(argument) {
/* Публикация сообщения.
GetColumnsValues — имя сообщения.
argument — аргумент, передаваемый в функцию-обработчик подписчика. Объект, содержащий параметры
key — массив тегов для фильтрации сообщений. */
return this.sandbox.publish("GetColumnsValues", argument, ["key"]);
}
```

Важно. Публикация сообщения может возвращать результат работы функции-обработчика только при установленном **адресном** режиме.

Подписка на сообщение

В схеме-подписчике должно быть объявлено сообщение с направлением "Подписка".

Пример объявления сообщения с направлением "Подписка"

```
messages: {
    /* Имя сообщения. */
    "GetColumnsValues": {
        /* Режим сообщения — адресное. */
        mode: this.Terrasoft.MessageMode.PTP,
        /* Направление сообщения — подписка. */
        direction: this.Terrasoft.MessageDirectionType.SUBSCRIBE
    }
}
```

Подписка осуществляется вызовом метода `subscribe` у экземпляра класса `sandbox`.

Пример подписки на сообщение

© 2022 Terrasoft. Все права защищены.

```
/* GetColumnsValues — имя сообщения.
messageHandler — функция-обработчик сообщения.
context — контекст выполнения функции-обработчика.
key — массив тегов для фильтрации сообщений. */
this.sandbox.subscribe("GetColumnsValues", messageHandler, context, ["key"]);
```

При **адресном** режиме метод `messageHandler` должен возвращать объект, который обрабатывается как результат публикации сообщения. При **широковещательном** режиме метод `messageHandler` ничего не возвращает.

Метод messageHandler (адресный режим)

```
methods: {
    messageHandler: function(args) {
        /* Возвращение объекта, который обрабатывается как результат публикации сообщения. */
        return { };
    }
}
```

Метод messageHandler (широковещательный режим)

```
methods: {
    messageHandler: function(args) {
    }
}
```

Методы (methods)

Для реализации методов необходимо использовать свойство `methods`, которое содержит коллекцию методов, формирующих бизнес-логику схемы и влияющих на модель представления. По умолчанию контекст методов является контекстом модели представления.

Назначение свойства `methods`:

1. Создавать новые методы.
2. Расширять базовые методы родительских схем.

Бизнес-правила (rules и businessRules)

Бизнес-правила — это механизмы приложения, которые пользовательскими средствами позволяют настраивать поведение полей на странице или детали. Для реализации бизнес-правил необходимо использовать свойства `rules` и `businessRules`. Свойство `businessRules` используется для бизнес-правил,

© 2022 Terrasoft. Все права защищены.

которые созданы или изменены мастером разделов или мастером деталей.

Назначение бизнес-правил:

- Скрытие или отображение полей.
- Блокировка или доступность редактирования полей.
- Обязательность или необязательность заполнения полей.
- Фильтрация справочных полей в зависимости от значений в других полях.

Функциональность бизнес-правил реализована в клиентском модуле `BusinessRuleModule`. Для использования функциональности бизнес-правил необходимо в список зависимостей схемы добавить модуль `BusinessRuleModule`.

Пример добавления модуля `BusinessRuleModule` в список зависимостей

```
define("CustomPageModule", ["BusinessRuleModule"],
    function(BusinessRuleModule) {
        return {
            /* Реализация клиентского модуля. */
        };
    });
```

Типы бизнес-правил определены в перечислении `RuleType` модуля `BusinessRuleModule`.

Особенности бизнес-правил

Особенности объявления бизнес-правил:

- Все бизнес-правила описываются в свойстве `rules` схемы.
- Бизнес-правила применяются к колонкам модели представления, а не к элементам управления.
- Бизнес-правило имеет название.
- Параметры бизнес-правила задаются в конфигурационном объекте.

Особенности бизнес-правил, определенных в свойстве `businessRules`:

- Генерируются мастерами разделов или деталей.
- При создании нового бизнес-правила мастер генерирует его имя и добавляет его в клиентскую схему модели представления страницы записи.
- При описании сгенерированного бизнес-правила не используются перечисления модуля бизнес-правил `BusinessRuleModule`.
- Маркерные комментарии `/**SCHEMA_BUSINESS_RULES*/` обязательны к использованию для работы мастеров.
- При выполнении имеют более высокий приоритет.
- При отключении бизнес-правила свойству `enabled` конфигурационного объекта присваивается значение `false`.

© 2022 Terrasoft. Все права защищены.

- При удалении бизнес-правила конфигурационный объект остается в клиентской схеме модели представления страницы записи, но свойству `removed` присваивается значение `true`.

Важно. Не рекомендуется редактировать свойство `businessRules` клиентской схемы.

Редактирование существующего бизнес-правила

После редактирования мастером созданного вручную бизнес-правила, остается неизменным конфигурационный объект бизнес-правила в свойстве `rules` модели представления страницы записи. При этом будет создана новая версия конфигурационного объекта бизнес-правила с тем же именем в свойстве `businessRules`.

При обработке бизнес-правила во время выполнения приложения приоритет отдается бизнес-правилу, определенному в свойстве `businessRules`. Поэтому последующие изменения этого бизнес-правила в свойстве `rules` никак не повлияют на систему.

На заметку. При удалении или отключении бизнес-правила более высокий приоритет имеют изменения, выполненные в конфигурационном объекте свойства `businessRules`.

Модули (`modules`)

Для реализации модулей необходимо использовать свойство `modules`, конфигурационный объект которого отвечает за объявление и конфигурирование модулей и деталей, загружаемых на страницу. Маркерные комментарии `/**SCHEMA_MODULES*/` обязательны к использованию для работы мастеров.

На заметку. Для загрузки детали на страницу используется свойство `details`. Поскольку деталь по сути является модулем, то правильным будет использование свойства `modules`.

Пример использования свойства `modules`

```
modules: /**SCHEMA_MODULES*/{
    /* Загрузка модуля.
    Заголовок модуля. Должен быть таким же, как свойство name в массиве diff. */
    "TestModule": {
        /* Опционально. Идентификатор загружаемого модуля. Если не указан, будет сгенерирован си
        "moduleId": "myModuleId",.
        /* Если параметр не указан, будет использован BaseSchemaModuleV2 для загрузки. */
        "moduleName": "MyTestModule",
        /* Конфигурационный объект. При загрузке модуля передается как instanceConfig. В нем хра
        "config": {
            "isSchemaConfigInitialized": true,
            "schemaName": "MyTestSchema",
            "useHistoryState": false,
```

© 2022 Terrasoft. Все права защищены.

```
            /* Дополнительные параметры модуля. */
            "parameters": {
                /* Параметры, передаваемые в схему при ее инициализации. */
                "viewModelConfig": {
                    masterColumnName: "PrimaryContact"
                }
            }
        }
    },

    /* Загрузка детали.
    Имя детали. */
    "Project": {
            /* Название схемы детали. */
            "schemaName": "ProjectDetailV2",
            "filter": {
            /* Колонка схемы объекта раздела. */
            "masterColumn": "Id",
            /* Колонка схемы объекта детали. */
            "detailColumn": "Opportunity"
        }
    }
}/**SCHEMA_MODULES*/
```

Массив модификаций (diff)

Для реализации массива модификаций необходимо использовать свойство `diff`, которое содержит массив конфигурационных объектов. **Назначение** массива модификаций — построение представления модуля в интерфейсе системы. Каждый элемент массива представляет собой метаданные, на основании которых генерируются различные элементы управления интерфейса. Маркерные комментарии `/**SCHEMA_DIFF*/` обязательны к использованию для работы мастеров.

Механизм псевдонимов alias

При разработке новых версий периодически возникает необходимость переместить элементы страницы в новые зоны. В ситуациях, когда пользователи проводили кастомизацию страницы записи, подобные изменения могут привести к непредсказуемым последствиям. **Механизм псевдонимов** `alias` обеспечивает частичную обратную совместимость при изменении пользовательского интерфейса в новых версиях продукта путем взаимодействия с построителем `diff` — классом `json-applier`, осуществляющим слияние параметров базовой схемы и клиентских расширяющих схем.

Свойство `alias` содержит информацию о предыдущем названии элемента. Эта информация учитывается при построении массива модификаций `diff`, сообщая о необходимости учитывать элементы не только с новым именем, но и с именем, указанным в `alias`. Фактически, `alias` представляет собой конфигурационный объект, который связывает новый и старый элементы. При построении массива модификаций `diff` благодаря конфигурационному объекту `alias` можно исключить применение некоторых свойств и операций по отношению к элементу, в котором он объявлен. Объект `alias` может

© 2022 Terrasoft. Все права защищены.

быть добавлен в любой элемент массива модификаций `diff`.

Связь между представлением и моделью

Назначение свойства `bindTo` — указание связи между атрибутом модели представления и свойством объекта представления.

Объявляется в свойстве `values` конфигурационных объектов массива `diff`.

Ниже представлен пример использования свойства `bindTo`.

Пример использования свойства `bindTo`

```
diff: [
    {
        "operation": "insert",
        "parentName": "CombinedModeActionButtonsCardLeftContainer",
        "propertyName": "items",
        "name": "MainContactButton",
        /* Свойства, передаваемые в конструктор компонента. */
        "values": {
            /* Тип добавляемого элемента — кнопка. */
            "itemType": Terrasoft.ViewItemType.BUTTON,
            /* Привязка заголовка кнопки к локализуемой строке схемы. */
            "caption": {bindTo: "Resources.Strings.OpenPrimaryContactButtonCaption"},
            /* Привязка метода-обработчика нажатия кнопки. */
            "click": {bindTo: "onOpenPrimaryContactClick"},
            /* Стиль отображения кнопки. */
            "style": Terrasoft.controls.ButtonEnums.style.GREEN,
            /* Привязка свойства доступности кнопки. */
            "enabled": {bindTo: "ButtonEnabled"}
        }
    }
]
```

Вкладки — объекты, которые в свойстве `propertyName` содержат значение `tabs`.

Для **заголовков вкладок** реализован альтернативный способ использования свойства `bindTo`.

Альтернативный способ использования свойства `bindTo`

```
...
{
    "operation": "insert",
    "name": "GeneralInfoTab",
    "parentName": "Tabs",
    /* Указывает на то, что данный объект является вкладкой. */
    "propertyName": "tabs",
```

© 2022 Terrasoft. Все права защищены.

```
    "index": 0,
    "values": {
        /* $ заменяет использование bindTo: {...}. */
        "caption": "$Resources.Strings.GeneralInfoTabCaption",
        "items": []
    }
},
...
```

Правила объявления свойства `diff`

- **Корректное использование конвертеров**.

  Конвертер — это функция, которая выполняется в окружении `viewModel` и, получая значения свойства `viewModel`, должна вернуть результат соответствующего типа. Для корректной работы мастеров значение свойства `diff` должно быть представлено в формате JSON. Поэтому значением конвертера должно быть имя метода модели представления, а не inline-функция.

  Пример правильного использования конвертера

```
methods: {
    someFunction: function(val) {
        /* ... */
    }
},

diff: /**SCHEMA_DIFF*/[
  {
      /* ... */
      "bindConfig": {
          "converter": "someFunction"
      }
      /* ...*/
  }
]/**SCHEMA_DIFF*/
```

  Пример неправильного использования конвертера

```
diff: /**SCHEMA_DIFF*/[
    {
        /* ... */
        "bindConfig": {
            "converter": function(val) {
            /* ... */
            }
```

© 2022 Terrasoft. Все права защищены.

```
        }
    }
]/**SCHEMA_DIFF*/
```

Пример правильного использования генератора

```
methods: {
    someFunction: function(val) {
        /* ... */
    }
},

diff: /**SCHEMA_DIFF*/[
  {
      /* ... */
      "values": {
          "generator": "someFunction"
      }
      /* ... */
  }
]/**SCHEMA_DIFF*/
```

Пример неправильного использования генератора

```
diff: /**SCHEMA_DIFF*/[
    {
        /* ... */
        "values": {
            "generator": function(val) {
                /* ... */
            }
        }
    }
]/**SCHEMA_DIFF*/
```

- **Родительский элемент**.

  Родительский элемент (контейнер) — это DOM-элемент, в который модуль отрисует свое представление. Для корректной работы мастерам необходимо, чтобы родительский контейнер содержал только один дочерний элемент.

  Пример правильного размещения представления в родительском элементе

© 2022 Terrasoft. Все права защищены.

```
<div id="OpportunityPageV2Container" class="schema-wrap one-el" data-item-marker="Opportunity
    <div id="CardContentWrapper" class="card-content-container page-with-left-el" data-item-m
</div>
```

Пример неправильного размещения представления в родительском элементе

```
<div id="OpportunityPageV2Container" class="schema-wrap one-el" data-item-marker="Opportunity
    <div id="CardContentWrapper" class="card-content-container page-with-left-el" data-item-m
    <div id="DuplicateContainer" class="DuplicateContainer"></div>
</div>
```

При добавлении, изменении, перемещении элемента (операции `insert`, `merge`, `move`) в свойстве `diff` необходимо указать свойство `parentName` — имя родительского элемента.

Пример правильного определения элемента представления в свойстве diff

```
{
  "operation": "insert",
  "name": "SomeName",
  "propertyName": "items",
  "parentName": "SomeContainer",
  "values": {}
}
```

Пример неправильного определения элемента представления в свойстве diff

```
{
  "operation": "insert",
  "name": "SomeName",
  "propertyName": "items",
  "values": {}
}
```

Если отсутствует свойство `parentName`, то при открытии мастера отобразится ошибка о невозможности настройки страницы мастером.

Значение свойства `parentName` должно соответствовать имени родительского элемента в соответствующей базовой схеме страницы. Так, например, для страниц записи это `CardContentContainer`.

Если в качестве родительского элемента в свойстве `parentName` указать имя несуществующего элемента-контейнера, то возникнет ошибка `"Схема не может иметь более одного корневого объекта"` (`"Schema cannot have more than one root object"`), поскольку добавляемый элемент будет помещен в

© 2022 Terrasoft. Все права защищены.

корневой контейнер.

- **Уникальность имен.**

  Каждый элемент в массиве `diff` должен иметь уникальное имя.

  Пример правильного добавления элементов в массив diff

  ```
  {
    "operation": "insert",
    "name": "SomeName",
    "values": { }
  },
  {
    "operation": "insert",
    "name": "SomeSecondName",
    "values": { }
  }
  ```

  Пример неправильного добавления элементов в массив diff

  ```
  {
    "operation": "insert",
    "name": "SomeName",
    "values": { }
  },
  {
    "operation": "insert",
    "name": "SomeName",
    "values": { }
  }
  ```

- **Размещение элементов представления**.

  Чтобы иметь возможность настраивать и изменять элементы представления, они должны находиться на **сетке разметки**. В Creatio каждый ряд сетки разметки имеет 24 ячейки (столбца). Для размещения на сетке используется свойство `layout`.

  Свойства элемента сетки:

  - `column` — индекс левого столбца.
  - `row` — индекс верхней строки.
  - `colSpan` — количество занимаемых столбцов.
  - `rowSpan` — количество занимаемых строк.

  Пример размещения элементов

© 2022 Terrasoft. Все права защищены.

```
{
  "operation": "insert",
  "parentName": "ParentContainerName",
  "propertyName": "items",
  "name": "ItemName",
  "values": {
      /* Размещение элемента. */
     "layout": {
        /* Начать с нулевого столбца. */
        "column": 0,
        /* Разместиться в пятой строке сетки. */
        "row": 5,
        /* Занять 12 столбцов в ширину. */
        "colSpan": 12,
        /* Занять один ряд в высоту. */
        "rowSpan": 1
     },
     "contentType": Terrasoft.ContentType.ENUM
  }
}
```

- **Количество операций**.

  Если клиентская схема изменяется без помощи мастера, то для корректной работы мастера рекомендуется добавлять не более одной операции для одного элемента в редактируемой схеме.

Свойства (properties)

Для реализации свойств необходимо использовать свойство `properties`, которое содержит JavaScript-объект.

Пример использования свойства `properties` в схеме `SectionTabsSchema` пакета `NUI` представлен ниже.

Пример использования свойства `properties`

```
define("SectionTabsSchema", [],
    function() {
        return {
            ...
            /* Объявление свойства properties. */
            properties: {
                /* Свойство parameters. Массив. */
                parameters: [],
                /* Свойство modulesContainer. Объект. */
                modulesContainer: {}
            },
            methods: {
```

© 2022 Terrasoft. Все права защищены.

```
                ...
                /* Метод инициализации. Всегда выполняется первым. */
                init: function(callback, scope) {
                    ...
                    /* Вызов метода, в котором используются свойства модели представления. */
                    this.initContainers();
                    ...
                },
                ...
                /* Метод, в котором используются свойства. */
                initContainers: function() {
                    /* Использование свойства modulesContainer. */
                    this.modulesContainer.items = [];
                    ...
                    /* Использование свойства parameters. */
                    this.Terrasoft.each(this.parameters, function(config) {
                        config = this.applyConfigs(config);
                        var moduleConfig = this.getModuleContainerConfig(config);
                        var initConfig = this.getInitConfig();
                        var container = viewGenerator.generatePartial(moduleConfig, initConfig)[
                        this.modulesContainer.items.push(container);
                    }, this);
                },
                ...
            },
        ...
        }
    });
```

Переопределить метод миксина

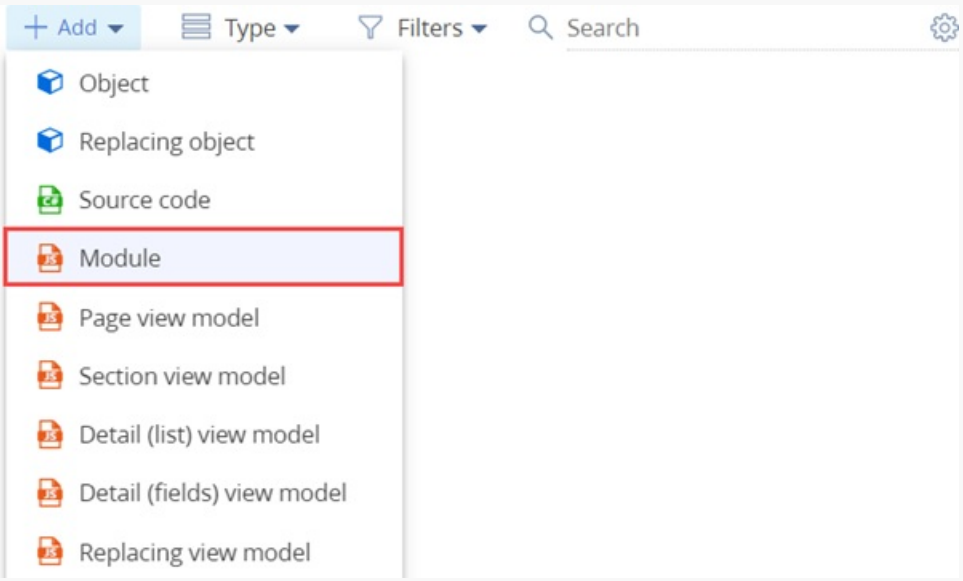
Легкий

Пример. Подключить миксин `ContentImageMixin` к пользовательской схеме, переопределить метод миксина.

1. Создать миксин

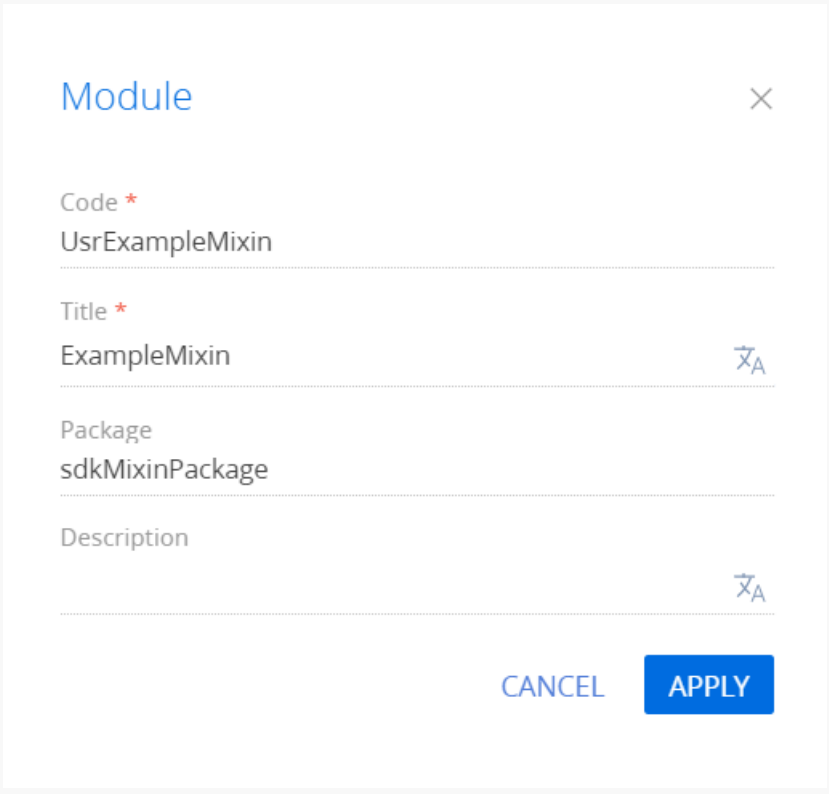
1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Модуль*] ([*Add*] —> [*Module*]).

© 2022 Terrasoft. Все права защищены.



3. В дизайнере схем заполните свойства схемы:

   - [*Код*] ([*Code*]) — "`UsrExampleMixin`".
   - [*Заголовок*] ([*Title*]) — "`ExampleMixin`".



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

4. В дизайнере схем добавьте исходный код.

Исходный код модуля

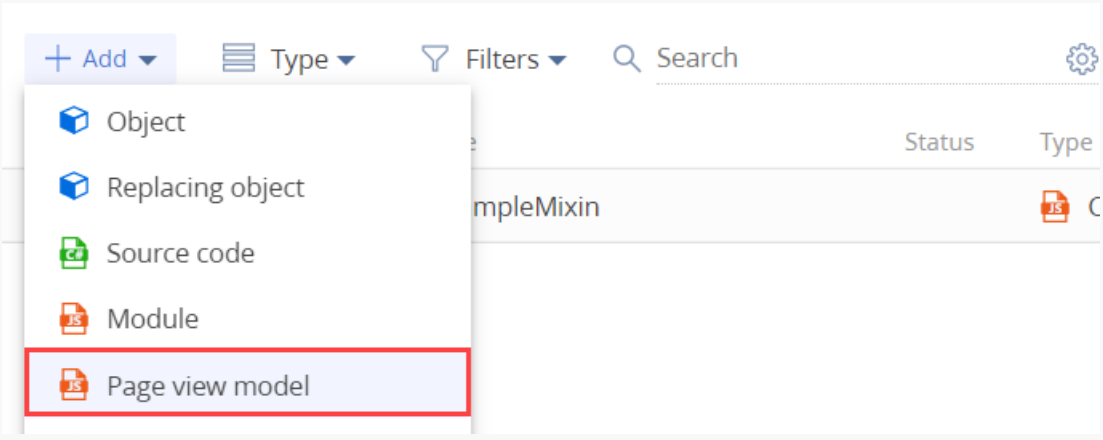
© 2022 Terrasoft. Все права защищены.

```
/* Определение модуля. */
define("ContentImageMixin", [ContentImageMixinV2Resources], function() {
    /* Определение класса ContentImageMixin. */
    Ext.define("Terrasoft.configuration.mixins.ContentImageMixin", {
        /* Псевдоним (сокращенное название класса). */
        alternateClassName: "Terrasoft.ContentImageMixin",
        /* Функциональность миксина. */
        getImageUrl: function() {
            var primaryImageColumnValue = this.get(this.primaryImageColumnName);
            if (primaryImageColumnValue) {
                return this.getSchemaImageUrl(primaryImageColumnValue);
            } else {
                var defImageResource = this.getDefaultImageResource();
                return this.Terrasoft.ImageUrlBuilder.getUrl(defImageResource);
            }
        }
    });
    return Ext.create(Terrasoft.ContentImageMixin);
});
```

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

2. Подключить миксин

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

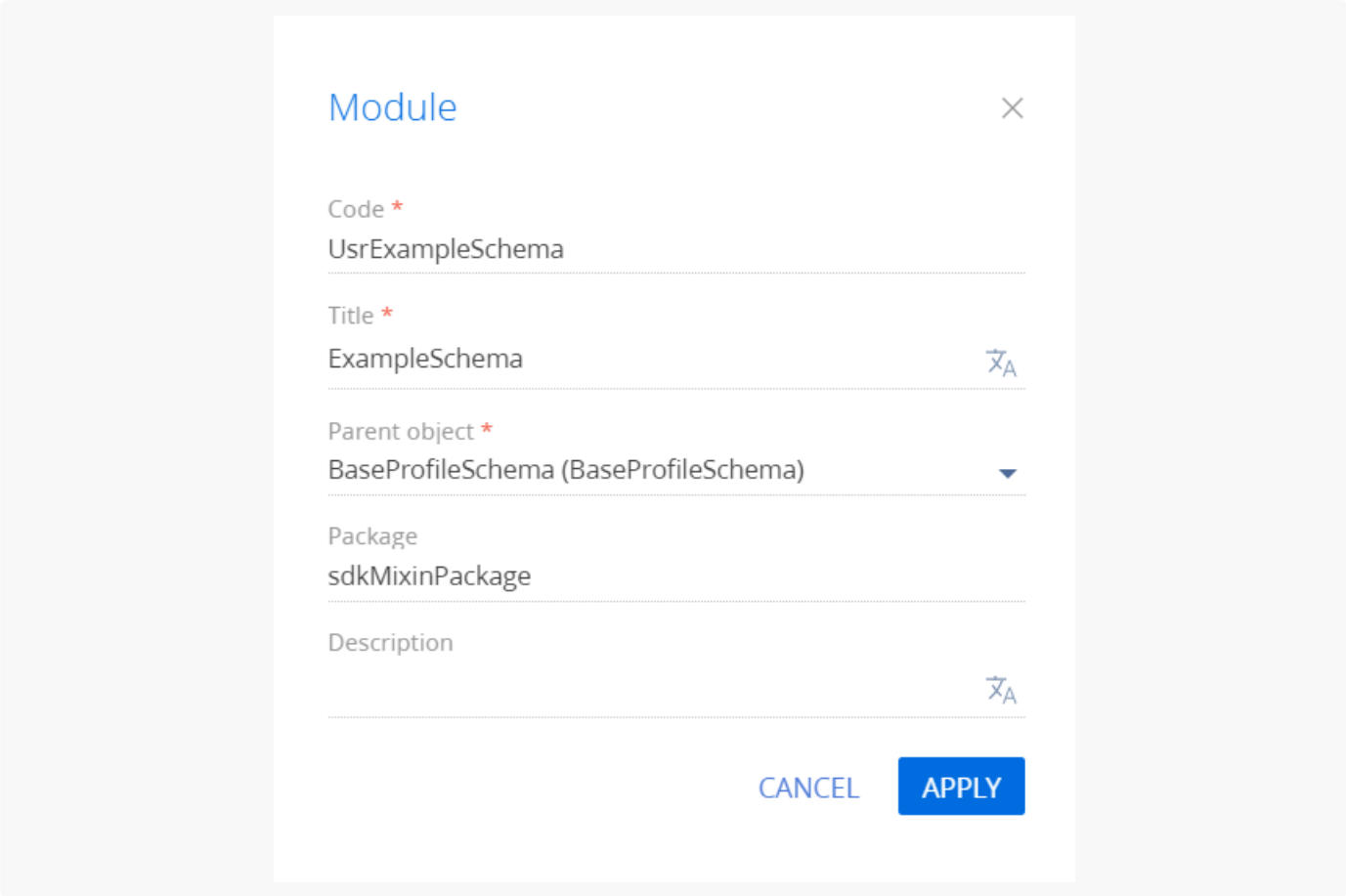


3. В дизайнере схем заполните свойства схемы:

- [*Код*] ([*Code*]) — "UsrExampleSchema".
- [*Заголовок*] ([*Title*]) — "ExampleSchema".

© 2022 Terrasoft. Все права защищены.

- [*Родительский объект*] ([*Parent object*]) — "BaseProfileSchema".



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

Чтобы использовать миксин, подключите его в блоке `mixins` пользовательской схемы `ExampleSchema`.

3. Переопределить метод миксина

В дизайнере схем добавьте исходный код. В блоке `method` переопределите метод миксина `getReadImageURL()`. Используйте переопределенную функцию в блоке `diff`.

Исходный код модуля

```
/* Объявление модуля. Обязательно укажите как зависимость модуль ContentImageMixin, в котором об
define("UsrExampleSchema", ["ContentImageMixin"], function() {
    return {
        entitySchemaName: "ExampleEntity",
        mixins: {
            /* Подключение миксина к схеме. */
            ContentImageMixin: "Terrasoft.ContentImageMixin"
        },
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
```

© 2022 Terrasoft. Все права защищены.

```
        diff: /**SCHEMA_DIFF*/[
            {
                "operation": "insert",
                "parentName": "AddRightsItemsHeaderImagesContainer",
                "propertyName": "items",
                "name": "AddRightsReadImage",
                "values": {
                    "classes": {
                        "wrapClass": ["rights-header-image"]
                    },
                    "getSrcMethod": "getReadImageUrl",
                    "imageTitle": resources.localizableStrings.ReadImageTitle,
                    "generator": "ImageCustomGeneratorV2.generateSimpleCustomImage"
                }
            }]/**SCHEMA_DIFF*/,
        methods: {
            getReadImageUrl: function() {
                /* Пользовательская реализация. */
                console.log("Contains custom logic");
                /* Вызов метода миксина. */
                this.mixins.ContentImageMixin.getImageUrl.apply(this, arguments);
            }
        },
        rules: {}
    };
});
```

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

Примеры объявления методов

Средний

Пример. К базовой логике метода `setValidationConfig` класса `Terrasoft.configuration.BaseSchemaViewModel` добавьте логику проверки заполнения колонки [*Email*].

Пример замещенного метода

```
methods: {
    /* Имя метода. */
    setValidationConfig: function() {
        /* Вызов логики метода setValidationConfig схемы родителя. */
        this.callParent(arguments);
        /* Установка валидации на колонку [Email]. */
```

© 2022 Terrasoft. Все права защищены.

```
        this.addColumnValidator("Email", EmailHelper.getEmailValidator);
    }
}
```

Пример нового метода

```
methods: {
    /* Имя метода. */
    getBlankSlateHeaderCaption: function() {
        /* Получение значения колонки MasterColumnInfo. */
        var masterColumnInfo = this.get("MasterColumnInfo");
        /* Возвращение результата работы метода. */
        return masterColumnInfo ? masterColumnInfo.caption : "";
    },
    /* Имя метода. */
    getBlankSlateIcon: function() {
        /* Возвращение результата работы метода. */
        return this.Terrasoft.ImageUrlBuilder.getUrl(this.get("Resources.Images.BlankSlateIcon")
    }
}
```

Пример использования массива модификаций

Средний

```
diff: /**SCHEMA_DIFF*/[
  {
      "operation": "insert",
      "name": "CardContentWrapper",
      "values": {
          "id": "CardContentWrapper",
          "itemType": Terrasoft.ViewItemType.CONTAINER,
          "wrapClass": ["card-content-container"],
          "items": []
      }
  },
  {
      "operation": "insert",
      "name": "CardContentContainer",
      "parentName": "CardContentWrapper",
      "propertyName": "items",
```

© 2022 Terrasoft. Все права защищены.

```
        "values": {
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            "items": []
        }
    },
    {
        "operation": "insert",
        "name": "HeaderContainer",
        "parentName": "CardContentContainer",
        "propertyName": "items",
        "values": {
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            "wrapClass": ["header-container-margin-bottom"],
            "items": []
        }
    },
    {
        "operation": "insert",
        "name": "Header",
        "parentName": "HeaderContainer",
        "propertyName": "items",
        "values": {
            "itemType": Terrasoft.ViewItemType.GRID_LAYOUT,
            "items": [],
            "collapseEmptyRow": true
        }
    }
]/**SCHEMA_DIFF*/
```

Пример использования механизма alias при многократном замещении схемы

Средний

Есть начальный элемент массива модификаций `diff` с именем "Name" и некоторым набором свойств. Элемент расположен в контейнере `Header`. Эта схема замещена несколько раз, при этом элемент с именем "Name" всячески модифицируется и перемещается.

Свойство diff базовой схемы

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Операция вставки. */
        "operation": "insert",
        /* Имя элемента-родителя, в который осуществляется вставка. */
```

© 2022 Terrasoft. Все права защищены.

```
        "parentName": "Header",
        /* Имя свойства элемента-родителя, с которым производится операция. */
        "propertyName": "items",
        /* Имя элемента. */
        "name": "Name",
        /* Объект значений свойств элемента. */
        "values": {
            /* Разметка. */
            "layout": {
                /* Номер колонки. */
                "column": 0,
                /* Номер строки. */
                "row": 1,
                /* Количество объединенных колонок. */
                "colSpan": 24
            }
        }
    }
] /**SCHEMA_DIFF*/
```

Свойство diff после первого замещения базовой схемы

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Операция объединения свойств двух элементов. */
        "operation": "merge",
        "name": "Name",
        "values": {
            "layout": {
                "column": 0,
                /* Номер строки. Элемент перемещен. */
                "row": 8,
                "colSpan": 24
            }
        }
    }
] /**SCHEMA_DIFF*/
```

Свойство diff после второго замещения базовой схемы

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Операция перемещения. */
        "operation": "move",
        "name": "Name",
```

© 2022 Terrasoft. Все права защищены.

```
        /* Имя элемента-родителя, в который осуществляется перемещение. */
        "parentName": "SomeContainer"
    }
] /**SCHEMA_DIFF*/
```

В новой версии элемент с именем "Name" был перемещен из элемента `SomeContainer` в элемент `ProfileContainer` и должен там остаться, несмотря на кастомизации клиента. Для этого элемент получает новое имя "NewName" и к нему добавляется конфигурационный объект `alias`.

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Операция вставки. */
        "operation": "insert",
        /* Имя элемента-родителя, в который осуществляется вставка. */
        "parentName": "ProfileContainer",
        /* Имя свойства элемента-родителя, с которым производится операция. */
        "propertyName": "items",
        /* Новое имя элемента. */
        "name": "NewName",
        /* Объект значений свойств элемента. */
        "values": {
            /* Привязка к значению свойства или функции. */
            "bindTo": "Name",
            /* Разметка. */
            "layout": {
                /* Номер колонки. */
                "column": 0,
                /* Номер строки. */
                "row": 0,
                /* Количество объединенных колонок. */
                "colSpan": 12
            }
        },

        /* Конфигурационный объект alias. */
        "alias": {
        /* Старое имя элемента. */
        "name": "Name",
        /* Массив игнорируемых свойств пользовательского замещения. */
        "excludeProperties": [ "layout" ],
        /* Массив игнорируемых операций пользовательского замещения. */
        "excludeOperations": [ "remove", "move" ]
        }
    }
] /**SCHEMA_DIFF*/
```

© 2022 Terrasoft. Все права защищены.

В новом элементе добавился `alias`, изменился родительский элемент и его расположение на странице записи. В свойстве `excludeProperties` хранится набор свойств, которые будут проигнорированы при применении разницы, а внутри `excludeOperations` хранится набор операций, которые не будут применяться к этому элементу из замещений.

В данном примере исключены свойства `layout` всех наследников для элемента с именем "Name", а также запрещены операции `remove` и `move`. Это говорит о том, что элемент с именем "NewName" будет содержать только корневое свойство `layout` и все свойства элемента "Name" из замещений, кроме `Layout`. Это же касается и операций.

Результат для построителя массива модификаций `diff`

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* Операция вставки. */
        "operation": "insert",
        /* Имя элемента-родителя, в который осуществится вставка. */
        "parentName": "ProfileContainer",
        /* Имя свойства элемента-родителя, с которым производится операция. */
        "propertyName": "items",
        /* Новое имя элемента. */
        "name": "NewName",
        /* Объект значений свойств элемента. */
        "values": {
            /* Привязка к значению свойства или функции. */
            "bindTo": "Name",
            /* Разметка. */
            "layout": {
                /* Номер колонки. */
                "column": 0,
                /* Номер строки. */
                "row": 0,
                /* Количество объединенных колонок. */
                "colSpan": 12
            },
        }
    },
] /**SCHEMA_DIFF*/
```

Свойство attributes JS

Легкий

Свойство `attributes` клиентской схемы содержит конфигурационный объект со своими свойствами.

Основные свойства

© 2022 Terrasoft. Все права защищены.

`dataValueType`

Тип данных атрибута. Будет использоваться при генерации представления. Доступные типы данных представлены перечислением `Terrasoft.DataValueType`.

Возможные значения (`DataValueType`)

GUID	0
TEXT	1
INTEGER	4
FLOAT	5
MONEY	6
DATE_TIME	7
DATE	8
TIME	9
LOOKUP	10
ENUM	11
BOOLEAN	12
BLOB	13
IMAGE	14
CUSTOM_OBJECT	15
IMAGELOOKUP	16
COLLECTION	17
COLOR	18
LOCALIZABLE_STRING	19
ENTITY	20
ENTITY_COLLECTION	21

© 2022 Terrasoft. Все права защищены.

ENTITY_COLUMN_MAPPING_COLLECTION	22
HASH_TEXT	23
SECURE_TEXT	24
FILE	25
MAPPING	26
SHORT_TEXT	27
MEDIUM_TEXT	28
MAXSIZE_TEXT	29
LONG_TEXT	30
FLOAT1	31
FLOAT2	32
FLOAT3	33
FLOAT4	34
LOCALIZABLE_PARAMETER_VALUES_LIST	35
METADATA_TEXT	36
STAGE_INDICATOR	37

`type`

Тип колонки. Необязательный параметр, который используется во внутренних механизмах `BaseViewModel`. Доступные типы колонок представлены перечислением `Terrasoft.ViewModelColumnType`.

Возможные значения (`ViewModelColumnType`)

© 2022 Terrasoft. Все права защищены.

ENTITY_COLUMN	0
CALCULATED_COLUMN	1
VIRTUAL_COLUMN	2
RESOURCE_COLUMN	3

`value`

Значение атрибута. Значение этого параметра будет установлено в модель представления при ее создании. В атрибуте `value` можно задавать числовые, строковые и логические значения. Если тип атрибута подразумевает использование значения ссылочного типа (массив, объект, коллекция и т. д.), то его начальное значение необходимо инициализировать с помощью методов.

Пример использования

Пример использования основных свойств атрибутов

```
attributes: {
    /* Имя атрибута. */
    "NameAttribute": {
        /* Тип данных. */
        "dataValueType": this.Terrasoft.DataValueType.TEXT,
        /* Тип колонки. */
        "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
        /* Значение по умолчанию. */
        "value": "NameValue"
    }
}
```

Дополнительные свойства

`caption`

Заголовок атрибута.

`isRequired`

Признак обязательности заполнения атрибута.

`dependencies`

© 2022 Terrasoft. Все права защищены.

Зависимость от другого атрибута модели. Например, установка атрибута в зависимости от значения другого атрибута. Используется для создания вычисляемых полей.

`lookupListConfig`

Свойство, отвечающее за свойства поля-справочника. Использование данного параметра описано в статье Настроить фильтрацию значений справочного поля. Это конфигурационный объект, который может содержать в себе опциональные свойства.

Опциональные свойства

`columns`	Массив имен колонок, которые будут добавлены к запросу дополнительно к колонке [*Id*] и первичной для отображения колонке.
`orders`	Массив конфигурационных объектов, которые определяют сортировку данных при отображении.
`filter`	Метод, возвращающий объект класса `Terrasoft.BaseFilter` или его наследника, который, в свою очередь, будет применен к запросу. Не может использоваться совместно со свойством `filters`.
`filters`	Массив фильтров (методов, возвращающих коллекции класса `Terrasoft.FilterGroup`). Не может использоваться совместно со свойством `filter`.

Пример использования

Пример использования дополнительных свойств атрибутов

```
attributes: {
    /* Имя атрибута. */
    "Client": {
        /* Заголовок атрибута. */
        "caption": { "bindTo": "Resources.Strings.Client" },
        /* Атрибут обязателен для заполнения. */
        "isRequired": true
    },

    /* Имя атрибута. */
    "ResponsibleDepartment": {
        lookupListConfig: {
            /* Дополнительные колонки. */
            columns: "SalesDirector",
            /* Колонка сортировки. */
            orders: [ { columnPath: "FromBaseCurrency" } ],
            /* Функция определения фильтра. */
```

© 2022 Terrasoft. Все права защищены.

```
            filter: function()
            {
                /* Возвращает фильтр по колонке [Type], которая равна константе Competitor
                return this.Terrasoft.createColumnFilterWithParameter(
                this.Terrasoft.ComparisonType.EQUAL,
                "Type",
                ConfigurationConstants.AccountType.Competitor);
            }
        }
    },
    /* Имя атрибута. */
    "Probability": {
        /* Определение зависимости колонки. */
        "dependencies": [
        {
            /* Зависит от колонки [Stage]. */
            "columns": [ "Stage" ],
            /* Имя метода-обработчика изменения колонки [Stage].
            Метод setProbabilityByStage() определен в свойстве methods объекта схемы. */
            "methodName": "setProbabilityByStage"
        }
        ]
    }
},
methods: {
    /* Метод-обработчик изменения колонки [Stage]. */
    setProbabilityByStage: function()
    {
        /* Получение значения колонки [Stage]. */
        var stage = this.get("Stage");
        /* Условие изменения колонки [Probability]. */
        if (stage.value && stage.value ===
        ConfigurationConstants.Opportunity.Stage.RejectedByUs)
        {
            /* Установка значения колонки [Probability]. */
            this.set("Probability", 0);
        }
    }
}
```

Свойство messages

Средний

Свойство `messages` клиентской схемы содержит конфигурационный объект со своими свойствами.

© 2022 Terrasoft. Все права защищены.

Свойства

`mode`

Режим сообщения. Доступные режимы представлены перечислением `Terrasoft.MessageMode`.

Возможные значения (`MessageMode`)

`PTP`	Адресное.
`BROADCAST`	Широковещательное.

`direction`

Направление сообщения. Доступные режимы представлены перечислением `Terrasoft.MessageDirectionType`.

Возможные значения (`MessageDirectionType`)

`PUBLISH`	Публикация.
`SUBSCRIBE`	Подписка.
`BIDIRECTIONAL`	Двунаправленное.

Свойства rules и businessRules JS

Легкий

Свойства `rules` и `businessRules` клиентской схемы содержат конфигурационный объект со своими свойствами.

Основные свойства

`ruleType`

Тип правила. Определяется значением перечисления `BusinessRuleModule.enums.RuleType`.

Возможные значения (`BusinessRuleModule.enums.RuleType`)

© 2022 Terrasoft. Все права защищены.

`BINDPARAMETER`	Тип бизнес-правила. Используется для настройки привязки свойств одной колонки к значениям различных параметров. Например, чтобы настроить видимость или доступность колонки в зависимости от значения другой колонки.
`FILTRATION`	Тип бизнес-правила. Используется для настройки фильтрации значений колонки модели представления. Например, для фильтрации определенной колонки с типом `LOOKUP` в зависимости от значения текущего состояния страницы.

`property`

Используется для типа бизнес-правила `BINDPARAMETER`. Свойство элемента управления. Задается значением перечисления `BusinessRuleModule.enums.Property`.

Возможные значения (`BusinessRuleModule.enums.Property`)

`VISIBLE`	Видимость колонки.
`ENABLED`	Доступность колонки.
`REQUIRED`	Обязательность для заполнения.
`READONLY`	Колонка доступна для чтения.

`conditions`

Используется для типа бизнес-правила `BINDPARAMETER`. Массив условий применения правила. Каждое условие представляет собой конфигурационный объект.

Свойства конфигурационного объекта

© 2022 Terrasoft. Все права защищены.

`leftExpression`	Выражение левой части условия. Представляет собой конфигурационный объект.
	Свойства конфигурационного объекта
	
	`type`
	
	Тип выражения. Задается значением из перечисления `BusinessRuleModule.enums.ValueType`.
	
	Возможные значения (`BusinessRuleModule.enums.ValueType`)
	
	`CONSTANT` — Константа.
	`ATTRIBUTE` — Значение колонки модели представления.
	`SYSSETTING` — Системная настройка.
	`SYSVALUE` — Системное значение. Элемент списка системных значений `Terrasoft.core.enums.SystemValueType`.
	
	`attribute`
	
	Название колонки модели.
	
	`attributePath`
	
	Мета-путь к колонке справочной схемы.
	
	`value`
	
	Значение для сравнения.

`comparisonType`	Тип сравнения. Задается значением из перечисления `Terrasoft.core.enums.ComparisonType`.
`rightExpression`	Выражение правой части условия. Аналогично `leftExpression`.

`logical`

Используется для типа бизнес-правила `BINDPARAMETER`. Логическая операция объединения условий из свойства conditions. Задается значением из перечисления `Terrasoft.LogicalOperatorType`.

© 2022 Terrasoft. Все права защищены.

`autocomplete`

Используется для типа бизнес-правила `FILTRATION`. Признак обратной фильтрации. Может принимать значения `true` или `false`.

`autoClean`

Используется для типа бизнес-правила `FILTRATION`. Признак автоматической очистки значения при изменении колонки, по которой осуществляется фильтрация. Может принимать значения `true` или `false`.

`baseAttributePatch`

Используется для типа бизнес-правила `FILTRATION`. Мета-путь к колонке справочной схемы, по которой будет выполняться фильтрация. При построении пути к колонке применяется принцип обратной связи так же, как в `EntitySchemaQuery`. Путь формируется относительно схемы, на которую ссылается колонка модели.

`comparisonType`

Используется для типа бизнес-правила `FILTRATION`. Тип операции сравнения. Задается значением из перечисления `Terrasoft.ComparisonType`.

`type`

Используется для типа бизнес-правила `FILTRATION`. Тип значения, с которым будет сравниваться `baseAttributePatch`. Задается значением из перечисления `BusinessRuleModule.enums.ValueType`.

`attribute`

Используется для типа бизнес-правила `FILTRATION`. Имя колонки модели представления. Это свойство описывается в том случае, если указан тип значения (`type`) `ATTRIBUTE`.

`attributePath`

Используется для типа бизнес-правила `FILTRATION`. Мета-путь к колонке схемы объекта. При построении пути к колонке применяется принцип обратной связи так же, как в `EntitySchemaQuery`. Путь формируется относительно схемы, на которую ссылается колонка модели.

`value`

Используется для типа бизнес-правила `FILTRATION`. Значение для фильтрации. Это свойство описывается в том случае, если указан тип значения (`type`) `ATTRIBUTE`.

© 2022 Terrasoft. Все права защищены.

Дополнительные свойства

Дополнительные свойства используются только для свойства `businessRules`.

`uId`

Уникальный идентификатор правила. Значение типа "`GUID`".

`enabled`

Признак активности. Может принимать значения `true` или `false`.

`removed`

Признак, указывающий, является ли правило удаленным. Может принимать значения `true` или `false`.

`invalid`

Признак, указывающий, является ли правило корректным. Может принимать значения `true` или `false`.

Примеры использования

Пример созданного мастером бизнес-правила BINDPARAMETER

```
define("SomePage", [], function() {
    return {
        /* ... */
        businessRules: /**SCHEMA_BUSINESS_RULES*/{
        /* Набор правил для колонки Type модели представления. */
        "Type": {
            /* Сгенерированный мастером код правила. */
            "ca246daa-6634-4416-ae8b-2c24ea61d1f0": {
                /* Уникальный идентификатор правила. */
                "uId": "ca246daa-6634-4416-ae8b-2c24ea61d1f0",
                /* Признак активности. */
                "enabled": true,
                /* Признак, указывающий, является ли правило удаленным. */
                "removed": false,
                /* Признак, указывающий, является ли правило корректным. */
                "invalid": false,
                /* Тип правила. */
                "ruleType": 0,
                /* Код свойства, которое регулирует правило. */
                "property": 0,
```

© 2022 Terrasoft. Все права защищены.

```
                /* Логическая взаимосвязь между несколькими условиями правила. */
                "logical": 0,
                /* Массив условий, при выполнении которых срабатывает правило.
                Сравнивает значение Account.PrimaryContact.Type со значением в колонке Type
                "conditions": [
                    {
                    /* Тип операции сравнения. */
                    "comparisonType": 3,
                    /* Выражение левой части условия. */
                    "leftExpression": {
                        /* Тип выражения — колонка (атрибут) модели представления. */
                        "type": 1,
                        /* Название колонки модели представления. */
                        "attribute": "Account",
                        /* Путь к колонке в справочной схеме Account, значение которой сра
                        "attributePath": "PrimaryContact.Type"
                    },
                    /* Выражение правой части условия. */
                    "rightExpression": {
                        /* Тип выражения — колонка (атрибут) модели представления. */
                        "type": 1,
                        /* Название колонки модели представления. */
                        "attribute": "Type"
                    }
                    }
                ]
            }
        }
        }/**SCHEMA_BUSINESS_RULES*/
        /* ... */
    };
});
```

Пример созданного мастером бизнес-правила FILTRATION

```
define("SomePage", [], function() {
    return {
        /* ... */
        businessRules: /**SCHEMA_BUSINESS_RULES*/{
        /* Набор правил для колонки Type модели представления. */
        "Account": {
            /* Сгенерированный мастером код правила. */
            "a78b898c-c999-437f-9102-34c85779340d": {
                /* Уникальный идентификатор правила. */
                "uId": "a78b898c-c999-437f-9102-34c85779340d",
                /* Признак активности. */
                "enabled": true,
```

© 2022 Terrasoft. Все права защищены.

```
                /* Признак, указывающий, является ли правило удаленным. */
                "removed": false,
                /* Признак, указывающий, является ли правило корректным. */
                "invalid": false,
                /* Тип правила. */
                "ruleType": 1,
                /* Путь к колонке для фильтрации в справочной схеме Account, на которую ссы
                "baseAttributePatch": "PrimaryContact.Type",
                /* Тип операции сравнения в фильтре. */
                "comparisonType": 3,
                /* Тип выражения — колонка (атрибут) модели представления. */
                "type": 1,
                /* Название колонки модели представления, по значению которой будет выполня
                "attribute": "Type"
            }
        }
        }/**SCHEMA_BUSINESS_RULES*/
        /* ... */
    };
});
```

Свойство diff JS

Средний

Свойство `diff` клиентской схемы содержит массив конфигурационных объектов со своими свойствами.

Свойства

`operation`

Операции с элементами.

Возможные значения

© 2022 Terrasoft. Все права защищены.

`set`	Значение элемента схемы устанавливается значением параметра `values`.
`merge`	Значения из родительских, замещаемых и замещающих схем сливаются вместе, при этом свойства из значения параметра `values` последнего наследника имеют приоритет.
`remove`	Элемент удаляется из схемы.
`move`	Элемент перемещается в другой родительский элемент.
`insert`	Элемент добавляется в схему.

`name`

Имя элемента схемы, с которым выполняется операция.

`parentName`

Имя родительского элемента схемы, в котором размещается элемент при операции `insert`, или в который перемещается элемент при операции `move`.

`propertyName`

Имя параметра родительского элемента при операции `insert`. Также используется при операции `remove`, если нужно удалить только определенные параметры элемента, а не весь элемент.

`index`

Индекс, в который перемещается или добавляется параметр. Параметр используется с операциями `insert` и `move`. Если параметр не указан, то вставка идет последним элементом массива.

`values`

Объект, свойства которого будут установлены либо объединены со свойствами элемента схемы. Используется при операциях `set`, `merge` и `insert`.

Набор основных элементов, которые можно отобразить на странице, представлены перечислением `Terrasoft.ViewItemType`.

Возможные значения (`ViewItemType`)

`GRID_LAYOUT`	0	Элемент-сетка, включающий в себя размещение других элементов управления.

© 2022 Terrasoft. Все права защищены.

TAB_PANEL	1	Набор вкладок.
DETAIL	2	Деталь.
MODEL_ITEM	3	Элемент модели представления.
MODULE	4	Модуль.
BUTTON	5	Кнопка.
LABEL	6	Надпись.
CONTAINER	7	Контейнер.
MENU	8	Выпадающий список.
MENU_ITEM	9	Элемент выпадающего списка.
MENU_SEPARATOR	10	Разделитель выпадающего списка.
SECTION_VIEWS	11	Представления раздела.
SECTION_VIEW	12	Представление раздела.
GRID	13	Реестр.
SCHEDULE_EDIT	14	Планировщик.
CONTROL_GROUP	15	Группа элементов.
RADIO_GROUP	16	Группа переключателей.
DESIGN_VIEW	17	Настраиваемое представление.
COLOR_BUTTON	18	Цвет.
IMAGE_TAB_PANEL	19	Набор вкладок с иконками.
HYPERLINK	20	Гиперссылка.
INFORMATION_BUTTON	21	Информационная кнопка из всплывающей подсказкой.
TIP	22	Всплывающая подсказка.
COMPONENT	23	Компонент.
PROGRESS_BAR	30	Индикатор.

© 2022 Terrasoft. Все права защищены.

`alias`

Конфигурационный объект.

Свойства объекта `alias`

`name`	Имя элемента, с которым связан новый элемент. По этому имени будут находиться элементы в замещенных схемах и связываться с новым элементом. Значение свойства `name` элемента массива модификаций `diff` не должно быть равным свойству `alias.name`.
`excludeProperties`	Массив свойств объекта `values` элемента массива модификаций `diff`, которые не будут применяться при построении `diff`.
`excludeOperations`	Массив операций, которые не должны применяться к данному элементу при построении массива модификаций `diff`.

Пример использования

Пример использования объекта `alias`

```
/* Массив diff. */
diff: /**SCHEMA_DIFF*/ [
  {
    /* Операция, совершаемая с элементом. */
    "operation": "insert",
    /* Новое имя элемента. */
    "name": "NewElementName",
    /* Значения элемента. */
    "values": {
        /* ... */
    },
    /* Конфигурационный объект alias. */
    "alias": {
        /* Предыдущее имя элемента. */
        "name": "OldElementName",
        /* Массив исключаемых свойств. */
        "excludeProperties": [ "layout", "visible", "bindTo" ],
        /* Массив игнорируемых операций. */
        "excludeOperations": [ "remove", "move", "merge" ]
    }
  },
  /* ... */
]
```

© 2022 Terrasoft. Все права защищены.

Виды модулей

Легкий

Базовые модули

В Creatio реализованы **базовые модули**:

- `ext-base` — реализует функциональность фреймворка `ExtJs`.
- `terrasoft` пространства имен и объектов `Terrasoft` — реализует доступ к системным операциям, переменным ядра и т. д.
- `sandbox` — реализует механизм обмена сообщениями между модулями.

Доступ к модулям `ext-base`, `terrasoft` и `sandbox`

```
// Определение модуля и получение ссылок на модули-зависимости.
define("ExampleModule", ["ext-base", "terrasoft", "sandbox"],
    // Ext – ссылка на объект, дающий доступ к возможностям фреймворка ExtJs.
    // Terrasoft – ссылка на объект, дающий доступ к системным переменным, переменным ядра и т.д
    // sandbox – используется для обмена сообщениями между модулями.
    function (Ext, Terrasoft, sandbox) {
});
```

Базовые модули используются в большинстве клиентских модулей. Указывать базовые модули в зависимостях `["ext-base", "terrasoft", "sandbox"]` не обязательно. После создания объекта класса модуля объекты `Ext`, `Terrasoft` и `sandbox` будут доступны как свойства объекта `this.Ext`, `this.Terrasoft`, `this.sandrenderbox`.

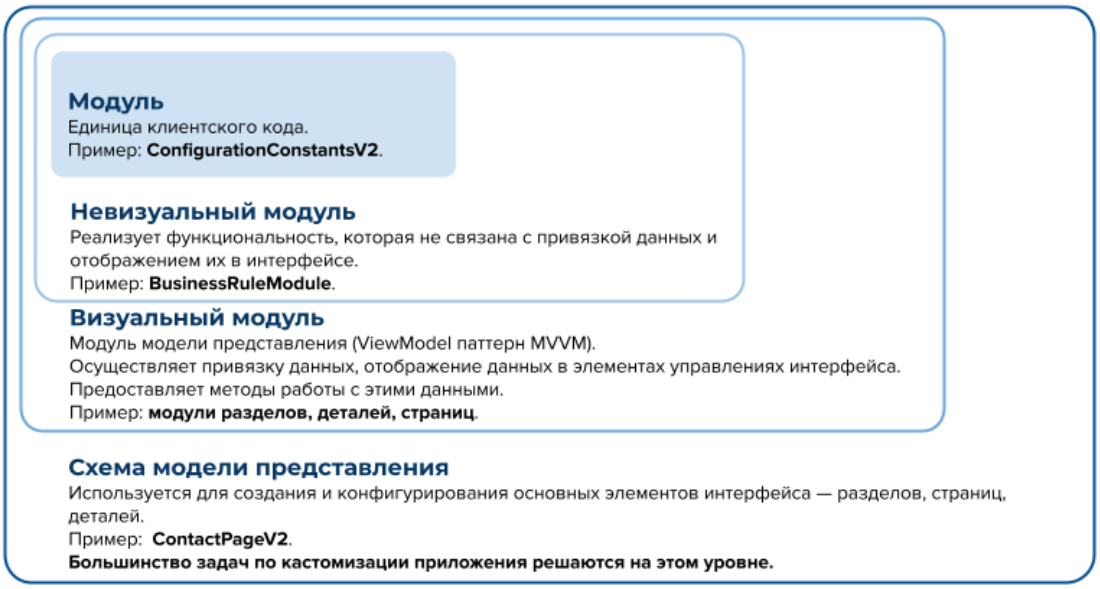
Клиентские модули

Виды клиентских модулей

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

Иерархия клиентских модулей представлена на рисунке ниже.

© 2022 Terrasoft. Все права защищены.



Разработка клиентских модулей описана в статье Разработка конфигурационных элементов.

Невизуальный модуль

Назначение невизуального модуля — реализация функциональности системы, которая, как правило, не сопряжена с привязкой данных и отображением их в интерфейсе. Примерами невизуальных модулей в системе являются модули бизнес-правил (`BusinessRuleModule`) и утилитные модули, которые реализуют служебные функции.

Разработка невизуального модуля описана в статье Схема модуля.

Визуальный модуль

К визуальным относятся модули, которые реализуют в системе модели представления (`ViewModel`) согласно шаблону MVVM.

Назначение визуального модуля — инкапсуляция данных, которые отображаются в элементах управления графического интерфейса, и методов работы с данными. Примерами визуальных модулей в системе являются модули разделов, деталей, страниц.

Разработка визуального модуля описана в статье Схема модели представления.

Модуль расширения и замещающий клиентский модуль

Назначение модулей расширения и замещающих клиентских модулей — расширение функциональности базовых модулей.

Разработка модуля расширения и замещающего клиентского модуля описана в статье Схема замещающей модели представления.

© 2022 Terrasoft. Все права защищены.

Особенности клиентских модулей

Методы клиентских модулей

- `init()` — метод реализует логику, выполняемую при загрузке модуля. При загрузке модуля клиентское ядро автоматически вызывает этот метод первым. Как правило, в методе `init()` выполняется подписка на события других модулей и инициализация значений.
- `render(renderTo)` — метод реализует логику визуализации модуля. При загрузке модуля и наличии метода клиентское ядро автоматически его вызывает. Для корректного отображения данных перед их визуализацией должен отработать механизм связывания представления (`View`) и модели представления (`ViewModel`). Поэтому, как правило, в методе `render()` выполняется запуск механизма — вызов у объекта представления метода `bind()`. Если модуль загружается в контейнер, то в качестве аргумента метода `render()` будет передана ссылка на этот контейнер. Метод `render()` в обязательном порядке должен реализовываться визуальными модулями.

Вызов одного модуля из другого. Утилитные модули

Несмотря на то что модуль является изолированной программной единицей, он может использовать функциональность других модулей. Для этого достаточно в качестве зависимости импортировать тот модуль, функциональность которого предполагается использовать. Доступ к экземпляру модуля–зависимости осуществляется через аргумент фабричной функции.

Так, в процессе разработки вспомогательные и служебные методы общего назначения можно группировать в отдельные **утилитные модули** и затем импортировать их в те модули, в которых необходима эта функциональность.

Работа с ресурсами

Ресурсы — дополнительные свойства схемы. Ресурсы добавляются в клиентскую схему на панели свойств дизайнера (кнопка +). Чаще всего используемые ресурсы — локализуемые строки (свойство [*Локализуемые строки*] ([*Localizable strings*])) и изображения (свойство [*Изображения*] ([*Images*])). Ресурсы содержатся в специальном модуле с именем `[ИмяКлиентскогоМодуля]Resources`, который автоматически генерирует ядро приложения для каждого клиентского модуля.

Чтобы получить доступ к модулю ресурсов из клиентского модуля, необходимо в качестве зависимости импортировать модуль ресурсов в клиентский модуль. В коде модуля необходимо использовать локализуемые ресурсы, а не строковые литералы или константы.

Использование модулей расширения

Модули расширения базовой функциональности не поддерживают наследование в традиционном его представлении.

Особенности создания расширяющего модуля:

1. Скопировать программный код расширяемого модуля.
2. Внести изменения в функциональность.

В замещающем модуле нельзя использовать ресурсы замещаемого модуля — все ресурсы должны

© 2022 Terrasoft. Все права защищены.

заново создаваться в замещающей схеме.

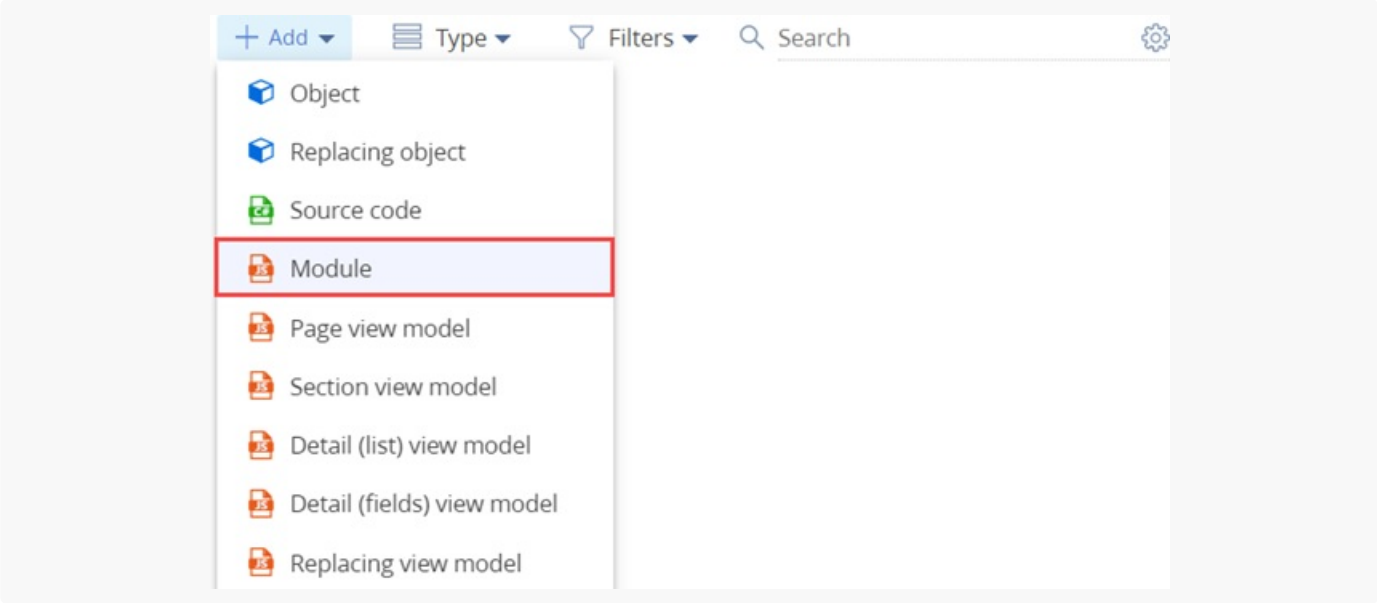
Создать стандартный модуль

Средний

Пример. Создать стандартный модуль, который содержит методы `init()` и `render()`. Каждый метод должен отображать информационное сообщение. При загрузке модуля на клиент ядро сначала вызовет метод `init()`, а затем — метод `render()`, о чем должны оповестить соответствующие информационные сообщения.

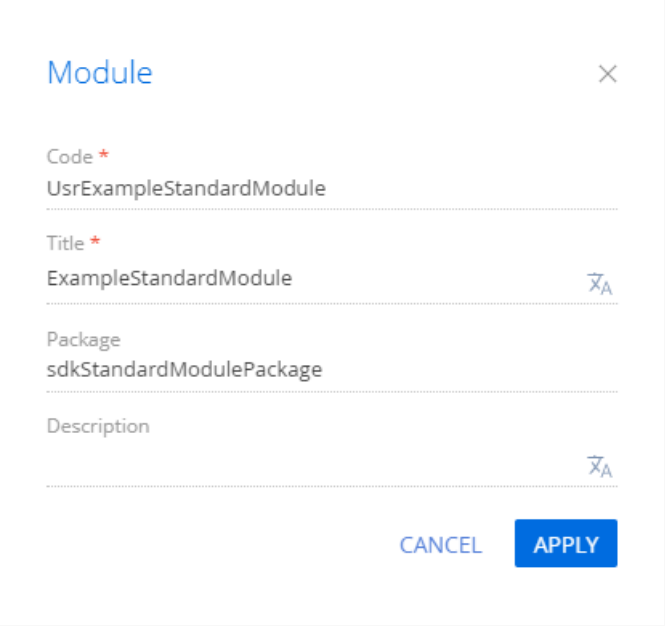
1. Создать визуальный модуль

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



3. В дизайнере схем заполните свойства схемы:
   - [*Код*] ([*Code*]) — "`UsrExampleStandardModule`".
   - [*Заголовок*] ([*Title*]) — "`ExampleStandardModule`".

© 2022 Terrasoft. Все права защищены.



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

4. В дизайнере схем добавьте исходный код.

Исходный код модуля

```
// Объявление модуля с именем UsrExampleStandartModule. Модуль не имеет никаких зависимостей,
// поэтому в качестве второго параметра передается пустой массив.
define("UsrExampleStandardModule", [],
    // Функция-фабрика возвращает объект модуля с двумя методами.
    function () {
        return {
            // Метод будет вызван ядром самым первым, сразу после загрузки на клиент.
            init: function () {
                alert("Calling the init() method of the UsrExampleStandardModule module");
            },
            // Метод будет вызван ядром при загрузке модуля в контейнер. Ссылка на контейнер
            // в качестве параметра renderTo. В информационном сообщении будет выведен id эле
            // в котором должны отображаться визуальные данные модуля. По умолчанию – centerP
            render: function (renderTo) {
                alert("Calling the render() method of the UsrExampleStandardModule module. Th
            }
        };
    });
```

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

2. Проверить визуальный модуль

Базовая версия Creatio позволяет проверить визуальный модуль, выполнить его загрузку на клиент и

© 2022 Terrasoft. Все права защищены.

визуализацию. Для этого сформируйте адресную строку запроса.

Адресная строка запроса

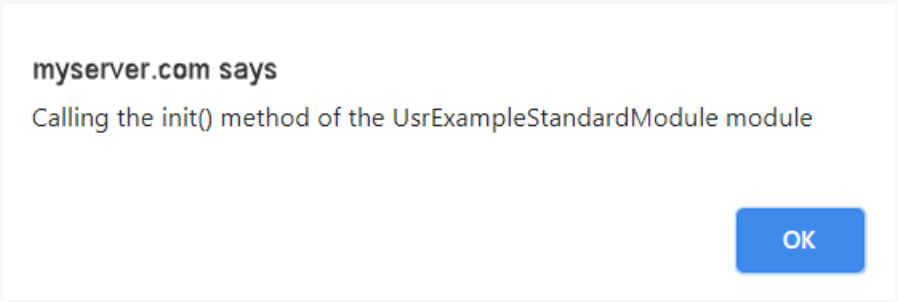
```
[АдресПриложения]/[НомерКонфигурации]/0/NUI/ViewModule.aspx#[ИмяМодуля]
```

Пример адресной строки запроса

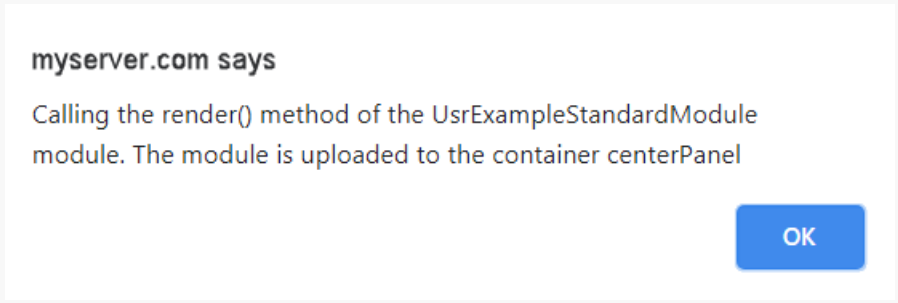
```
http://myserver.com/CreatioWebApp/0/NUI/ViewModule.aspx#UsrExampleStandardModule
```

На клиент будет возвращен модуль `UsrExampleStandardModule`.

Вызов метода `init()` модуля `UsrExampleStandardModule`



Вызов метода `render()` модуля `UsrExampleStandardModule`



Создать утилитный модуль

Средний

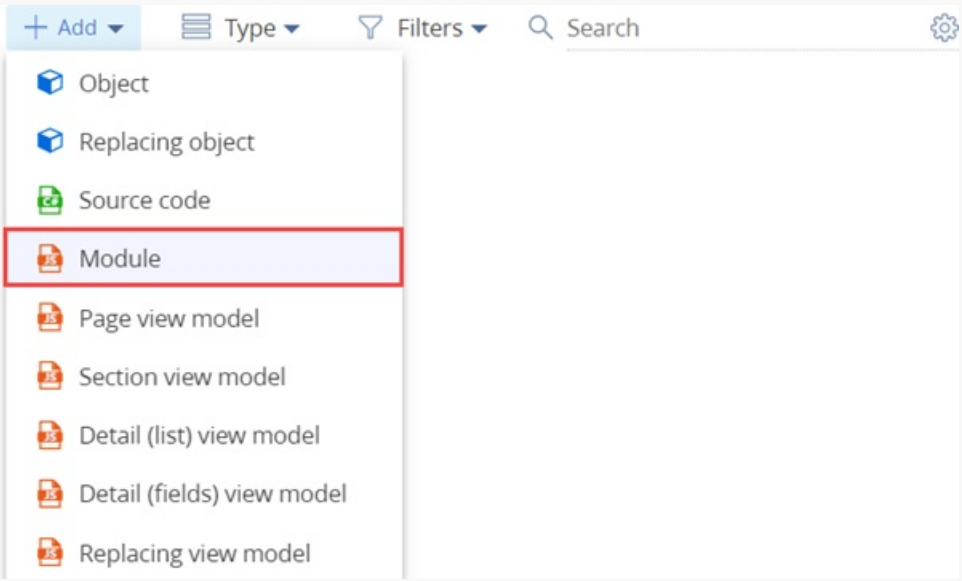
Пример. Создать стандартный модуль, который содержит методы `init()` и `render()`. Каждый метод должен отображать информационное сообщение. При загрузке модуля на клиент ядро сначала вызовет метод `init()`, а затем — метод `render()`, о чем должны оповестить соответствующие информационные сообщения. Метод отображения информационного окна

© 2022 Terrasoft. Все права защищены.

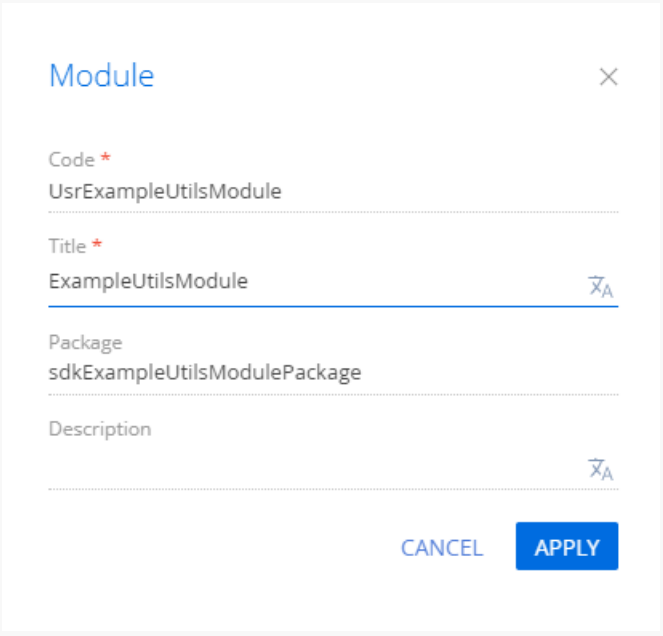
необходимо вынести в отдельный утилитный модуль.

1. Создать утилитный модуль

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



3. В дизайнере схем заполните свойства схемы:
   - [*Код*] ([*Code*]) — "`UsrExampleUtilsModule`".
   - [*Заголовок*] ([*Title*]) — "`ExampleUtilsModule`".



© 2022 Terrasoft. Все права защищены.

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

4. В дизайнере схем добавьте исходный код.

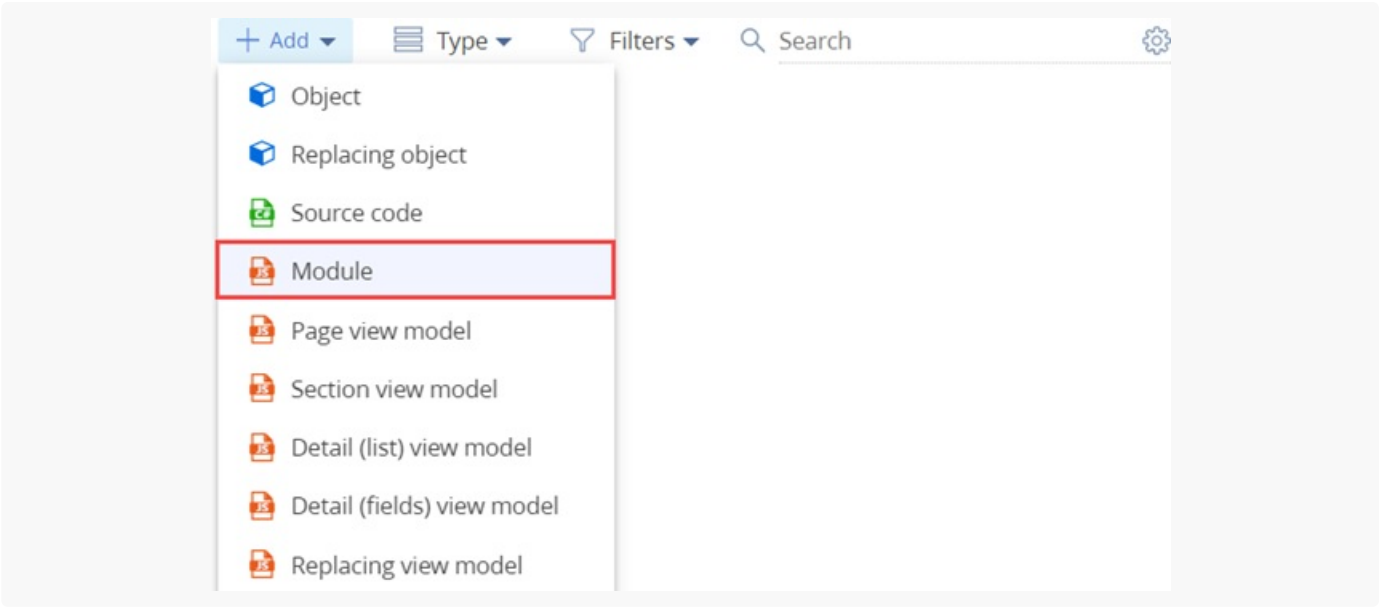
Исходный код утилитного модуля

```
// Объявление утилитного модуля. Модуль не имеет зависимостей и содержит один метод
// для отображения информационного сообщения.
define("UsrExampleUtilsModule", [],
    function () {
        return {
            // Метод, который отображает информационное окно с сообщением. Сообщение, выводим
            // передается в метод в качестве аргумента information.
            showInformation: function (information) {
                alert(information);
            }
        };
    });
```

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

2. Создать визуальный модуль

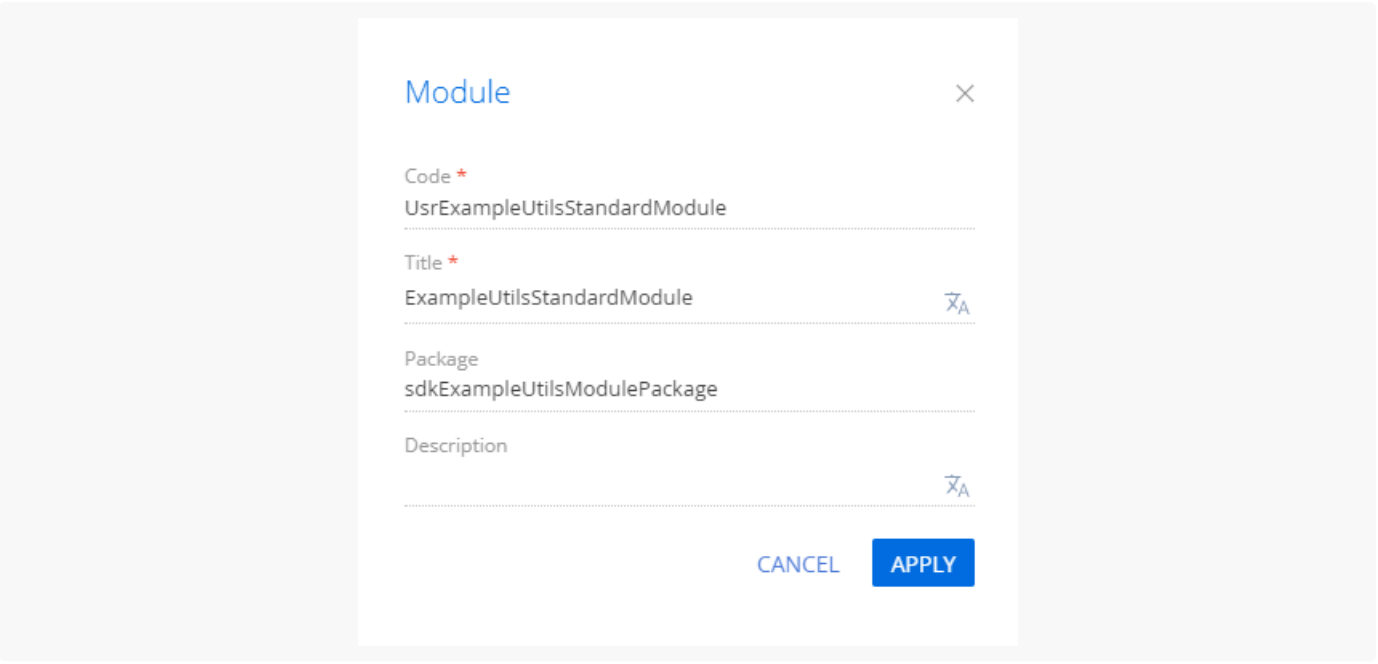
1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Модуль*] ([*Add*] —> [*Module*]).



3. В дизайнере схем заполните свойства схемы:

- [*Код*] ([*Code*]) — "`UsrExampleUtilsStandardModule`".
- [*Заголовок*] ([*Title*]) — "`ExampleUtilsStandardModule`".

© 2022 Terrasoft. Все права защищены.



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

4. В дизайнере схем добавьте исходный код.

Исходный код модуля

```
// В модуль импортируется модуль-зависимость UsrExampleUtilsModule для доступа к утилитному м
// Аргумент функции-фабрики – ссылка на загруженный утилитный модуль.
define("UsrExampleUtilsStandardModule", ["UsrExampleUtilsModule"],
    function (UsrExampleUtilsModule) {
        return {
            // В функциях init() и render() вызывается утилитный метод для отображения информ
            // с сообщением, которое передается в качестве аргумента утилитному методу.
            init: function () {
                UsrExampleUtilsModule.showInformation("Calling the init() method of the UsrEx
            },
            render: function (renderTo) {
                UsrExampleUtilsModule.showInformation("Calling the render() method of the Usr
            }
        };
    });
```

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

3. Проверить визуальный модуль

Базовая версия Creatio позволяет проверить визуальный модуль, выполнить его загрузку на клиент и визуализацию. Для этого сформируйте адресную строку запроса.

© 2022 Terrasoft. Все права защищены.

Адресная строка запроса

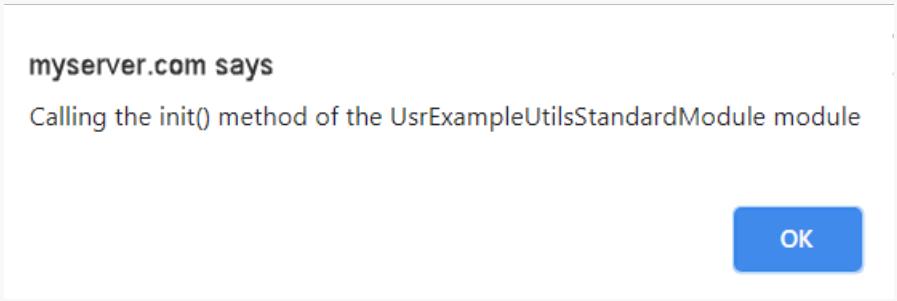
```
[АдресПриложения]/[НомерКонфигурации]/0/NUI/ViewModule.aspx#[ИмяМодуля]
```

Пример адресной строки запроса

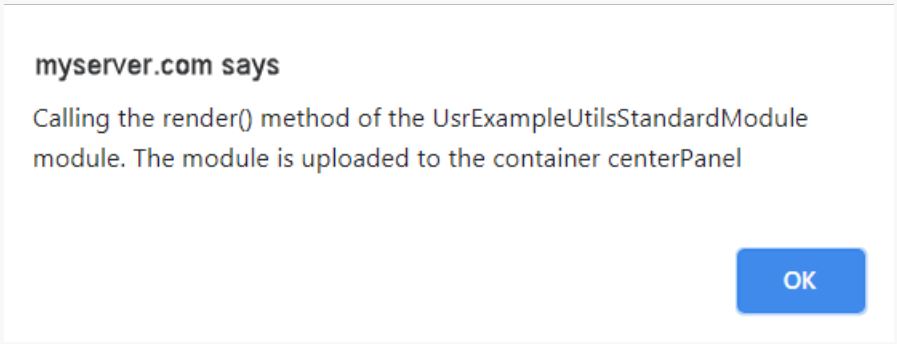
```
http://myserver.com/CreatioWebApp/0/NUI/ViewModule.aspx#UsrExampleUtilsStandardModule
```

На клиент будет возвращен модуль `UsrExampleUtilsStandardModule`.

Вызов метода `init()` модуля `UsrExampleUtilsStandardModule`



Вызов метода `render()` модуля `UsrExampleUtilsStandardModule`



Компоненты

Средний

Элементы управления — это объекты, используемые для организации интерфейса между пользователем и приложением. Как правило, они отображаются в навигационных панелях, диалоговых окнах и на панелях инструментов. К ним относятся кнопки, признаки (флажки), поля и т. д.

Все элементы управления наследуются от класса `Terrasoft.Component` (полное имя — `Terrasoft.controls.Component`), который, в свою очередь, наследуется от `Terrasoft.BaseObject`. Благодаря тому, что в родительском классе (`Component`) объявлен миксин `Bindable`, имеется возможность

© 2022 Terrasoft. Все права защищены.

связывания свойств элемента управления с требуемыми свойствами, методами или атрибутами модели представления.

В элементе управления объявляются события, необходимые для корректной работы. Также каждый элемент содержит в себе события, унаследованные от класса `Terrasoft.Component`:

- `added` — срабатывает после того, как компонент был добавлен в контейнер.
- `afterrender` — срабатывает после того, как компонент был отрендерен и его HTML-представление попало в DOM.
- `afterrerender` — срабатывает после того, как компонент отрабатывает рендеринг и его HTML представление обновилось в DOM.
- `beforererender` — срабатывает перед тем, как компонент был отрендерен и его HTML представление попало в DOM.
- `destroy` — срабатывает перед окончательным уничтожением элемента управления.
- `destroyed` — срабатывает после удаления компонента.
- `init` — срабатывает, когда инициализация компонента завершена.

Подробнее о событиях класса `Terrasoft.Component` можно узнать из [библиотеки классов клиентской части ядра платформы].

Также элемент управления может подписываться на события браузера и определять свои собственные события.

Элемент управления определяется в массиве модификаций `diff` модуля, в котором его необходимо расположить.

Массив модификаций `diff` модуля

```
// Массив модификаций diff модуля.
diff: [{
    // Операция вставки.
    "operation": "insert",
    // Имя элемента-родителя, в который осуществится вставка элемента управления.
    "parentName": "CardContentContainer",
    // Имя свойства элемента-родителя, с которым производится операция.
    "propertyName": "items",
    // Имя элемента управления.
    "name": "ExampleButton",
    // Значения элемента управления.
    "values": {
        // Тип элемента управления.
        "itemType": "Terrasoft.ViewItemType.BUTTON",
        // Заголовок элемента управления.
        "caption": "ExampleButton",
        // Связывание события элемента управления с функцией.
        "click": {"bindTo": "onExampleButtonClick"},
        // Стиль элемента управления.
        "style": Terrasoft.controls.ButtonEnums.style.GREEN
```

© 2022 Terrasoft. Все права защищены.

```
    }
}]
```

Внешний вид элемента управления определяется шаблоном (`template <tpl>`). Во время рендеринга элемента управления в представление страницы генерация представления элемента выполняется согласно заданному шаблону.

Элемент управления не должен иметь никакой бизнес логики — она определяется в модуле, в который добавляется элемент управления.

Элемент управления имеет атрибуты `styles`, `selectors`, которые определены в классе-родителе `Terrasoft.Component`. Эти атрибуты предоставляют возможность гибкой настройки стилей.

Класс модуля

 Средний

Объявление класса модуля

Объявление классов — функция JavaScript-фреймворка `ExtJs`. Для объявления классов используется стандартный механизм библиотеки — метод `define()` глобального объекта `Ext`.

Пример объявления класса с помощью метода `define()`

```
// Название класса с соблюдением пространства имен.
Ext.define("Terrasoft.configuration.ExampleClass", {
    // Сокращенное название класса.
    alternateClassName: "Terrasoft.ExampleClass",
    // Название класса, от которого происходит наследование.
    extend: "Terrasoft.BaseClass",
    // Блок для объявления статических свойств и методов.
    static: {
        // Пример статического свойства.
        myStaticProperty: true,

        // Пример статического метода.
        getMyStaticProperty: function () {
            // Пример доступа к статическому свойству.
            return Terrasoft.ExampleClass.myStaticProperty;
        }
    },
    // Пример динамического свойства.
    myProperty: 12,
    // Пример динамического метода класса.
    getMyProperty: function () {
        return this.myProperty;
    }
```

© 2022 Terrasoft. Все права защищены.

```
});
```

Примеры создания экземпляров класса представлены ниже.

Пример создания экземпляра класса по полному имени

```
// Создание экземпляра класса по полному имени.
var exampleObject = Ext.create("Terrasoft.configuration.ExampleClass");
```

Пример создания экземпляра класса по псевдониму

```
// Создание экземпляра класса по сокращенному названию – псевдониму.
var exampleObject = Ext.create("Terrasoft.ExampleClass");
```

Наследование класса модуля

В большинстве случаев класс модуля правильно наследовать от `Terrasoft.configuration.BaseModule` или `Terrasoft.configuration.BaseSchemaModule`, в которых реализованы **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. Возвращает представление модуля. Принимает единственный аргумент `renderTo`, в который будет добавлено представление объекта модуля.
- `destroy()` — метод, отвечающий за удаление представления модуля, удаление модели представления, отписку от ранее подписанных сообщений и уничтожение объекта класса модуля.

Ниже приведен пример класса модуля, наследуемого от `Terrasoft.BaseModule`. Данный модуль добавляет кнопку в DOM. При клике на кнопку выводится сообщение, после чего она удаляется из DOM.

Пример класса модуля, наследуемого от `Terrasoft.BaseModule`

```
define("ModuleExample", [], function () {
    Ext.define("Terrasoft.configuration.ModuleExample", {
        // Короткое название класса.
        alternateClassName: "Terrasoft.ModuleExample",
        // Класс, от которого происходит наследование.
        extend: "Terrasoft.BaseModule",
        /* Обязательное свойство. Если не определено, будет сгенерирована ошибка на уровне
           "Terrasoft.core.BaseObject", так как класс наследуется от "Terrasoft.BaseModule".*/
        Ext: null,
```

© 2022 Terrasoft. Все права защищены.

```
        /* Обязательное свойство. Если не определено, будет сгенерирована ошибка на уровне
           "Terrasoft.core.BaseObject", так как класс наследуется от "Terrasoft.BaseModule".*/
        sandbox: null,
        /* Обязательное свойство. Если не определено, будет сгенерирована ошибка на уровне
           "Terrasoft.core.BaseObject", так как класс наследуется от "Terrasoft.BaseModule".*/
        Terrasoft: null,
        // Модель представления.
        viewModel: null,
        // Представление. В качестве примера используется кнопка.
        view: null,
        /* Если не реализовать метод init() в текущем классе,
           то при создании экземпляра текущего класса будет вызван метод
           init() класса-родителя Terrasoft.BaseModule.*/
        init: function () {
            // Вызывает выполнение логики метода init() класса-родителя.
            this.callParent(arguments);
            this.initViewModel();
        },
        // Инициализирует модель представления.
        initViewModel: function () {
            /* Сохранение контекста класса модуля
               для доступа к нему из модели представления.*/
            var self = this;
            // Создание модели представления.
            this.viewModel = Ext.create("Terrasoft.BaseViewModel", {
                values: {
                    // Заголовок кнопки.
                    captionBtn: "Click Me"
                },
                methods: {
                    // Обработчик нажатия на кнопку.
                    onClickBtn: function () {
                        var captionBtn = this.get("captionBtn");
                        alert(captionBtn + " button was pressed");
                        /* Вызывает метод выгрузки представления и модели представления,
                           что приводит к удалению кнопки из DOM.*/
                        self.destroy();
                    }
                }
            });
        },
        /* Создает представление (кнопку),
           связывает ее с моделью представления и вставляет в DOM.*/
        render: function (renderTo) {
            // В качестве представления создается кнопка.
            this.view = this.Ext.create("Terrasoft.Button", {
                // Контейнер, в который будет помещена кнопка.
                renderTo: renderTo,
                // HTML-атрибут id.
```

© 2022 Terrasoft. Все права защищены.

```
                id: "example-btn",
                // Название класса.
                className: "Terrasoft.Button",
                // Заголовок.
                caption: {
                    // Связывает заголовок кнопки
                    // со свойством captionBtn модели представления.
                    bindTo: "captionBtn"
                },
                // Метод-обработчик события нажатия на кнопку.
                click: {
                    /* Связывает обработчик события нажатия на кнопку
                       с методом onClickBtn() модели представления.*/
                    bindTo: "onClickBtn"
                },
                /* Стиль кнопки. Возможные стили определены в перечислении
                   Terrasoft.controls.ButtonEnums.style.*/
                style: this.Terrasoft.controls.ButtonEnums.style.GREEN
            });
            // Связывает представление и модель представления.
            this.view.bind(this.viewModel);
            // Возвращает представление, которое будет вставлено в DOM.
            return this.view;
        },
        // Удаляет неиспользуемые объекты.
        destroy: function () {
            // Уничтожает представление, что приводит к удалению кнопки из DOM.
            this.view.destroy();
            // Удаляет неиспользуемую модель представления.
            this.viewModel.destroy();
        }
    });
    // Возвращает объект модуля.
    return Terrasoft.ModuleExample;
});
```

Мониторинг переопределения членов класса модуля

При наследовании класса модуля в классе-наследнике могут быть переопределены как публичные, так и приватные свойства и методы базового модуля.

В Creatio **приватными свойствами или методами класса** считаются те, названия которых начинаются с нижнего подчеркивания, например, `_privateMemberName`.

В Creatio присутствует функциональность **мониторинга переопределения приватных членов класса** — класс `Terrasoft.PrivateMemberWatcher`.

Назначение мониторинга — при определении пользовательского класса проверять выполнение переопределений приватных свойств или методов, которые объявлены в родительских классах. При

© 2022 Terrasoft. Все права защищены.

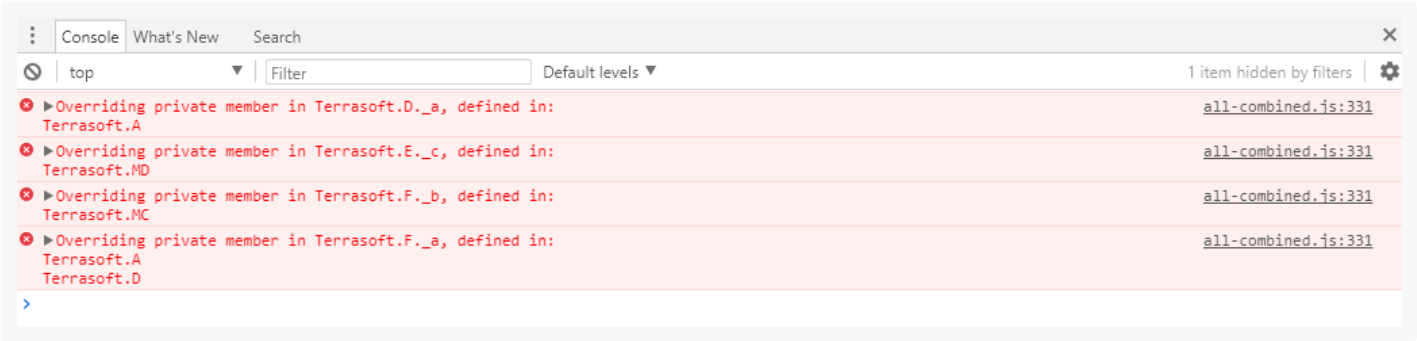
этом в режиме отладки отображается предупреждение в консоли браузера.

Например, в пользовательский пакет добавлена схема модуля, исходный код которой приведен ниже.

Пример переопределения приватных свойств класса модуля

```
define("UsrPrivateMemberWatcher", [], function() {
    Ext.define("Terrasoft.A", {_a: 1});
    Ext.define("Terrasoft.B", {extend: "Terrasoft.A"});
    Ext.define("Terrasoft.MC", {_b: 1});
    Ext.define("Terrasoft.C", {extend: "Terrasoft.B", mixins: {ma: "Terrasoft.MC"}});
    Ext.define("Terrasoft.MD", {_c: 1});
    // Переопределение свойства _a.
    Ext.define("Terrasoft.D", {extend: "Terrasoft.C", _a: 3, mixins: {mb: "Terrasoft.MD"}});
    // Переопределение свойства _c.
    Ext.define("Terrasoft.E", {extend: "Terrasoft.D", _c: 3});
    // Переопределение свойств _a и _b.
    Ext.define("Terrasoft.F", {extend: "Terrasoft.E", _b: 3, _a: 0});
});
```

После загрузки этого модуля в консоли отобразятся предупреждения о том, что приватные члены базовых классов были переопределены.



Инициализация экземпляра класса модуля

Синхронная инициализация

Модуль инициализируется синхронно, если при его загрузке явно не указано свойство `isAsync: true` конфигурационного объекта, передаваемого в качестве параметра метода `loadModule()`. Например, при выполнении

```
this.sandbox.loadModule([moduleName])
```

методы класса модуля будут загружены синхронно. Первым будет вызван метод `init()`, после которого

© 2022 Terrasoft. Все права защищены.

сразу же будет вызван метод `render()`.

Пример реализации синхронно инициализируемого модуля

```
define("ModuleExample", [], function () {
    Ext.define("Terrasoft.configuration.ModuleExample", {
        alternateClassName: "Terrasoft.ModuleExample",
        Ext: null,
        sandbox: null,
        Terrasoft: null,
        init: function () {
            // При инициализации выполнится первым.
        },
        render: function (renderTo) {
            // При инициализации модуля выполнится сразу же после метода init().
        }
    });
});
```

Асинхронная инициализация

Модуль инициализируется асинхронно, если при его загрузке явно указано свойство `isAsync: true` конфигурационного объекта, передаваемого в качестве параметра метода `loadModule()`. Например, при выполнении

```
this.sandbox.loadModule([moduleName], {
    isAsync: true
})
```

первым будет вызван метод `init()`, в который будет передан единственный параметр — callback-функция с контекстом текущего модуля. При вызове callback-функции будет вызван метод `render()` загружаемого модуля. Представление будет добавлено в DOM только после выполнения метода `render()`.

Пример реализации асинхронно инициализируемого модуля

```
define("ModuleExample", [], function () {
    Ext.define("Terrasoft.configuration.ModuleExample", {
        alternateClassName: "Terrasoft.ModuleExample",
        Ext: null,
        sandbox: null,
        Terrasoft: null,
        // При инициализации модуля выполнится первым.
        init: function (callback) {
```

© 2022 Terrasoft. Все права защищены.

```
            setTimeout(callback, 2000);
        },
        render: function (renderTo) {
            // Метод выполнится с задержкой в 2 секунды,
            // Задержка указана в аргументе функции setTimeout() в методе init().
        }
    });
});
```

Цепочки модулей

Цепочки модулей — механизм, который позволяет отобразить представление одной модели на месте представления другой модели. Например, для установки значения поля на текущей странице необходимо отобразить страницу `SelectData` для выбора значения из справочника. То есть, на месте контейнера модуля текущей страницы должно отобразиться представление модуля страницы выбора из справочника.

Для построения цепочки необходимо добавить свойство `keepAlive` в конфигурационный объект загружаемого модуля. Например, в модуле текущей страницы `CardModule` необходимо вызвать модуль выбора из справочника `selectDataModule`.

Пример вызова модуля выбора из справочника `selectDataModule` в модуле страницы `CardModu`

```
sandbox.loadModule("selectDataModule", {
    // Id представления загружаемого модуля.
    id: "selectDataModule_id",
    // Представление будет добавлено в контейнер текущей страницы.
    renderTo: "cardModuleContainer",
    // Указывает, чтобы текущий модуль не выгружался.
    keepAlive: true
});
```

После выполнения кода будет построена цепочка из модуля текущей страницы и модуля страницы выбора из справочника. Добавление еще одного элемента в цепочку позволяет из модуля текущей страницы `selectData` по нажатию на кнопку [*Добавить новую запись*] ([*Add new record*]) открыть новую страницу. Добавив в цепочку еще один элемент, из модуля текущей страницы `selectData` по нажатию на кнопку [*Добавить новую запись*] ([*Add new record*]) откроется новая страница. Таким образом, в цепочку модулей можно добавлять неограниченное количество экземпляров модулей.

Активный модуль (тот, который отображен на странице) — последний элемент цепочки. Если установить активным элемент из середины цепочки, то будут уничтожены все элементы, находящиеся в цепочке после него. Активация элемента цепочки выполняется путем вызова функции `loadModule()`, в параметр которой необходимо передать идентификатор модуля.

Пример вызова функции `loadModule()`

© 2022 Terrasoft. Все права защищены.

```
sandbox.loadModule("someModule", {
    id: "someModuleId"
});
```

Ядро уничтожит все элементы цепочки, после чего вызовет методы `init()` и `render()`. При этом в метод `render()` будет передан контейнер, который содержит предыдущий активный модуль. Модули цепочки могут работать, как и раньше — принимать и отправлять сообщения, сохранять данные и т. д.

Если при вызове метода `loadModule()` в конфигурационный объект не добавлять свойство `keepAlive` или добавить его со значением `keepAlive: false`, то цепочка модулей будет уничтожена.

Sandbox

Средний

Модуль в Creatio является изолированной программной единицей. Ему ничего не известно об остальных модулях системы, кроме имен модулей, от которых он зависит. Для организации взаимодействия модулей предназначен специальный объект — `sandbox`.

Sandbox предоставляет два ключевых **механизма взаимодействия модулей** в системе:

- Механизм обмена сообщениями между модулями.
- Загрузка и выгрузка модулей по требованию (для визуальных модулей).

Важно. Для того чтобы модуль мог взаимодействовать с другими модулями системы, он должен импортировать в качестве зависимости модуль `sandbox`.

Обмен сообщениями между модулями

Модули могут общаться друг с другом только посредством сообщений. Модуль, которому требуется сообщить об изменении своего состояния другим модулям в системе, публикует сообщение. Если модулю необходимо получать сообщения об изменении состояний других модулей, он должен быть подписанным на эти сообщения.

На заметку. Указывать базовые модули в зависимостях ["*ext-base*", "*terrasoft*", "*sandbox*"] не обязательно, если модуль экспортирует конструктор класса. Объекты `Ext`, `Terrasoft` и `sandbox` после создания объекта класса модуля будут доступны как свойства объекта: `this.Ext`, `this.Terrasoft`, `this.sandbox`.

Зарегистрировать сообщение

Чтобы модули могли обмениваться сообщениями, сообщения необходимо зарегистрировать.

Для регистрации сообщений модуля предназначен метод `sandbox.registerMessages(messageConfig)`, где `messageConfig` — конфигурационный объект сообщений модуля.

© 2022 Terrasoft. Все права защищены.

Конфигурационный объект является коллекцией "ключ-значение", в которой каждый элемент имеет вид, представленный ниже.

Конфигурационный объект сообщения

```
"MessageName": {
    mode: [Режим работы сообщения],
    direction: [Направление сообщения]
}
```

Здесь `MessageName` — ключ элемента коллекции, содержащий имя сообщения. Значением является конфигурационный объект, содержащий два свойства:

- `mode` — режим работы сообщения. Должно содержать значение перечисления `Terrasoft.MessageMode (Terrasoft.core.enums.MessageMode)`.
- `direction` — направление сообщения. Должно содержать значение перечисления `Terrasoft.MessageDirectionType (Terrasoft.core.enums.MessageDirectionType)`.

Режимы обмена сообщениями (свойство `mode`):

- **Широковещательный** — режим работы сообщения, при котором количество подписчиков заранее неизвестно. Соответствует значению перечисления `Terrasoft.MessageMode.BROADCAST`.
- **Адресный** — режим работы сообщения, при котором сообщение может быть обработано только одним подписчиком. Соответствует значению перечисления `Terrasoft.MessageMode.PTP.`

Важно. В адресном режиме подписчиков может быть несколько, но сообщение обработает только один, как правило, последний зарегистрированный подписчик.

Направления сообщения (свойство `direction`):

- **Публикация** — модуль может только опубликовать сообщение в `sandbox`. Соответствует значению перечисления `Terrasoft.MessageDirectionType.PUBLISH`.
- **Подписка** — модуль может только подписаться на сообщение, опубликованное из другого модуля. Соответствует значению перечисления `Terrasoft.MessageDirectionType.SUBSCRIBE`.
- **Двунаправленное** — позволяет публиковать и подписываться на одно и то же сообщение в разных экземплярах одного и того же класса или в рамках одной и той же иерархии наследования схем. Соответствует значению перечисления `Terrasoft.MessageDirectionType.BIDIRECTIONAL`.

В схемах модели представления регистрировать сообщения с помощью метода `sandbox.registerMessages()` не нужно. Достаточно объявить конфигурационный объект сообщений в свойстве `messages`.

Для отказа от регистрации сообщений в модуле можно воспользоваться методом `sandbox.unRegisterMessages(messages)`, где `messages` — имя или массив имен сообщений.

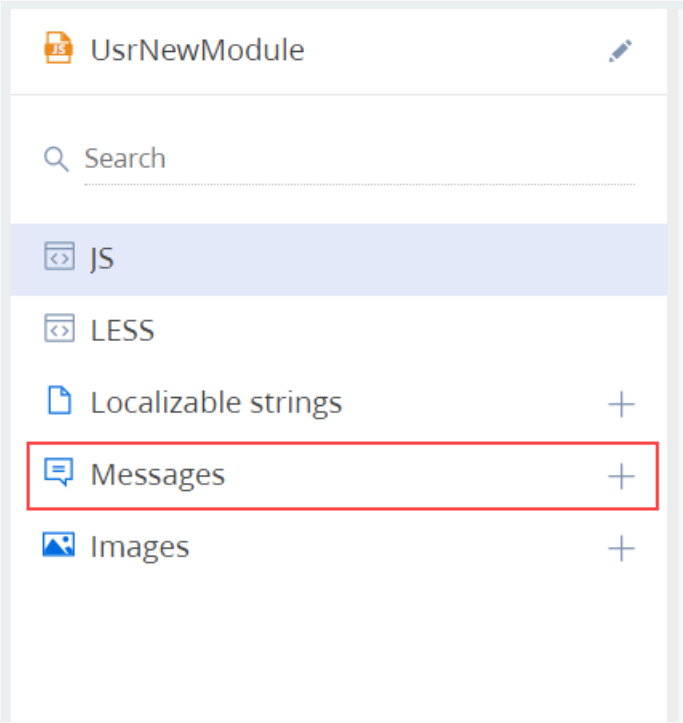
© 2022 Terrasoft. Все права защищены.

Добавить сообщение в схему модуля

Зарегистрировать сообщения можно также, добавив их в схему модуля с помощью дизайнера.

Для добавления сообщения в схему модуля:

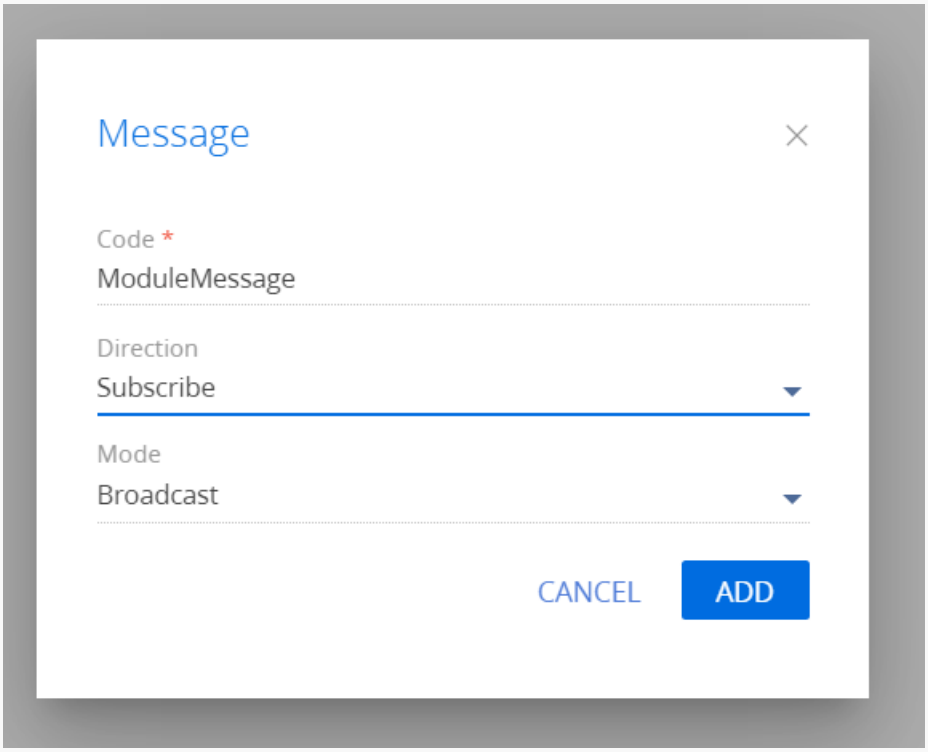
1. В области свойств модуля дизайнера схемы модуля добавьте сообщение в узел [*Сообщения*]([*Messages*]).



2. Для добавленного сообщения установите необходимые свойства:

   - [*Название*] ([*Name*]) — имя сообщения, совпадающее с ключом в конфигурационном объекте модуля.
   - [*Направление*] ([*Direction*]) — направление сообщения. Возможные значения "`Подписка`" ("`Follow`") и "`Публикация`" ("`Publish`").
   - [*Режим*] ([*Mode*]) — режим работы сообщения. Возможные значения "`Широковещательное`" ("`Broadcast`") и "`Адрес`" ("`Address`").

© 2022 Terrasoft. Все права защищены.



Важно. В схемах модели представления добавлять сообщения в структуру схемы не нужно.

Опубликовать сообщение

Для публикации сообщения предназначен метод `sandbox.publish(messageName , messageArgs, tags)`.

Для сообщения, опубликованного с массивом тегов, будут вызваны только те обработчики, для которых совпадает хотя бы один тег. Сообщения, опубликованные без тегов, смогут обработать только подписчики без тегов.

Подписаться на сообщение

Подписаться на сообщение можно, используя метод `sandbox.subscribe(messageName, messageHandler, scope, tags)`.

Загрузка и выгрузка модулей

При работе с пользовательским интерфейсом Creatio может возникнуть необходимость загрузки не объявленных как зависимости модулей во время выполнения приложения.

Загрузить модуль

Для загрузки не объявленных в качестве зависимостей модулей предназначен метод `sandbox.loadModule(moduleName, config)`. Параметры метода:

- `moduleName` — название модуля.

© 2022 Terrasoft. Все права защищены.

- `config` — конфигурационный объект, содержащий параметры модуля. Обязательный параметр для визуальных модулей.

Примеры вызова метода `sandbox.loadModule()`

```
// Загрузка модуля без дополнительных параметров.
this.sandbox.loadModule("ProcessListenerV2");
// Загрузка модуля с дополнительными параметрами.
this.sandbox.loadModule("CardModuleV2", {
    renderTo: "centerPanel",
    keepAlive: true,
    id: moduleId
});
```

Выгрузить модуль

Для выгрузки модуля необходимо использовать метод `sandbox.unloadModule(id, renderTo, keepAlive)`. Параметры метода:

- `id` — идентификатор модуля.
- `renderTo` — название контейнера, из которого необходимо удалить представление визуального модуля. Обязателен для визуальных модулей.
- `keepAlive` — признак сохранения модели модуля. При выгрузке модуля ядро может сохранить его модель для возможности использовать ее свойства, методы, сообщения. Не рекомендуется к использованию.

Примеры вызова метода `sandbox.unloadModule()`

```
...
// Метод получения идентификатора выгружаемого модуля.
getModuleId: function() {
    return this.sandbox.id + "_ModuleName";
},
...
// Выгрузка невизуального модуля.
this.sandbox.unloadModule(this.getModuleId());
...
// Выгрузка визуального модуля, ранее загруженного в контейнер "ModuleContainer".
this.sandbox.unloadModule(this.getModuleId(), "ModuleContainer");
```

Создать цепочку модулей

Иногда возникает необходимость показать представление некой модели на месте представления другой модели. Например, для установки значения определенного поля на текущей странице нужно отобразить

© 2022 Terrasoft. Все права защищены.

страницу выбора значения из справочника `SelectData`. В таких случаях нужно, чтобы модуль текущей страницы не выгружался, а на месте его контейнера отображалось представление модуля страницы выбора из справочника. Для этого можно использовать цепочки модулей.

Чтобы начать построение цепочки, достаточно добавить свойство `keepAlive` в конфигурационный объект загружаемого модуля.

Реализовать обмен сообщениями между модулями

 Средний

Пример. Создать модуль, и реализовать в нем публикацию и подписку на сообщения:

- адресное `MessageToSubscribe` с направлением [*Подписка*] ;
- широковещательное `MessageToPublish` с направлением [*Публикация*].

1. Создайте модуль

Создайте модуль `UsrSomeModule`, наследующего базовый класс `BaseModule`.

Объявление класса модуля

```
define("UsrSomeModule", [], function() {
    Ext.define("Terrasoft.configuration.UsrSomeModule", {
        alternateClassName: "Terrasoft.UsrSomeModule",
        extend: "Terrasoft.BaseModule",
        Ext: null,
        sandbox: null,
        Terrasoft: null,

        init: function() {
            this.callParent(arguments);
        },
        destroy: function() {
            this.callParent(arguments);
        }
    });
    return Terrasoft.UsrSomeModule;
});
```

2. Зарегистрируйте сообщения модуля.

© 2022 Terrasoft. Все права защищены.

1. Добавьте в код модуля свойство `messages`, в котором опишите конфигурационные объекты сообщений.
2. Измените метод `init()` добавив в нем вызов метода `sandbox.registerMessages()`, который выполнит регистрацию сообщений.

Регистрация сообщений модуля

```
...
// Коллекция конфигурационных объектов сообщений.
messages: {
    "MessageToSubscribe": {
        mode: Terrasoft.MessageMode.PTP,
        direction: Terrasoft.MessageDirectionType.SUBSCRIBE
    },
    "MessageToPublish": {
        mode: Terrasoft.MessageMode.BROADCAST,
        direction: Terrasoft.MessageDirectionType.PUBLISH
    }
};
...
init: function() {
    this.callParent(arguments);
    // Регистрация коллекции сообщений.
    this.sandbox.registerMessages(this.messages);
}
...
```

3. Опубликуйте сообщение.

1. Добавьте в код модуля метод `processMessages()`, в котором выполните вызов метода публикации `sandbox.publish()` сообщения `MessageToPublish`.
2. Измените метод `init()` добавив в нем вызов метода `processMessages()`.

Публикация сообщения

```
...
init: function() {
    this.callParent(arguments);
    this.sandbox.registerMessages(this.messages);
    this.processMessages();
},
...
processMessages: function() {
    this.sandbox.publish("MessageToPublish", null, [this.sandbox.id]);
}
```

© 2022 Terrasoft. Все права защищены.

```
...
```

При определенных бизнес-задачах необходимо опубликовать сообщение, которое будет принимать результат от модуля-подписчика. При публикации сообщения в адресном режиме результат вернет метод-обработчик сообщения в модуле-подписчике. При публикации сообщения в широковещательном режиме результат его обработки можно получить через объект, передаваемый в качестве аргумента для метода-обработчика.

Пример публикации адресного сообщения

```
...
// Объявление сообщения.
messages: {
    ...
    "MessageWithResult": {
        mode: Terrasoft.MessageMode.PTP,
        direction: Terrasoft.MessageDirectionType.PUBLISH
    }
}
...
processMessages: function() {
    ...
    // Публикация сообщения и получение результата его обработки подписчиком.
    var result = this.sandbox.publish("MessageWithResult", {arg1:5, arg2:"arg2"}, ["resultTag"])
    // Вывод результата в консоль браузера.
    console.log(result);
}
...
```

Пример публикации широковещательного сообщения

```
// Объявление сообщения.
messages: {
    ...
    "MessageWithResultBroadcast": {
        mode: Terrasoft.MessageMode.BROADCAST,
        direction: Terrasoft.MessageDirectionType.PUBLISH
    }
}
...
processMessages: function() {
    ...
    var arg = {};
    // Публикация сообщения и получение результата его обработки подписчиком.
    // В методе-обработчике подписчика в объект нужно добавить свойство result,
```

© 2022 Terrasoft. Все права защищены.

```
    // в которое следует записать результат обработки.
    this.sandbox.publish("MessageWithResultBroadcast", arg, ["resultTag"]);
    // Вывод результата в консоль браузера.
    console.log(arg.result);
}
...
```

4. Подпишитесь на сообщение.

1. Измените метод `processMessages()`, добавив в него вызов метода подписки `sandbox.subscribe()` на сообщение `MessageToSubscribe`.
2. В параметрах метода обязательно укажите метод-обработчик `onMessageSubscribe()` и добавьте его в исходный код модуля.
3. Метод `onMessageSubscribe()` вернет результат в модуль, который опубликовал сообщение.

Подписка на сообщение

```
...
processMessages: function() {
    this.sandbox.subscribe("MessageToSubscribe", this.onMessageSubscribe, this, ["resultTag"]);
    this.sandbox.publish("MessageToPublish", null, [this.sandbox.id]);
},
onMessageSubscribe: function(args) {
    console.log("'MessageToSubscribe' received");
    // Изменение параметра.
    args.arg1 = 15;
    args.arg2 = "new arg2";
    // Обязательный возврат результата.
    return args;
}
...
```

5. Реализуйте отмену регистрации сообщений.

Для этого измените код базового метода `destroy()`, добавив в него вызов метода отмены регистрации сообщений `sandbox.unRegisterMessages()`.

Отмена регистрации сообщений модуля

```
...
destroy: function() {
    if (this.messages) {
```

© 2022 Terrasoft. Все права защищены.

```
        var messages = this.Terrasoft.keys(this.messages);
        // Отмена регистрации массива сообщений.
        this.sandbox.unRegisterMessages(messages);
    }
    this.callParent(arguments);
}
...
```

Ниже приведен полный листинг кода модуля, в котором реализован механизм обмена сообщениями.

UsrSomeModule.js

```
define("UsrSomeModule", [], function() {
    Ext.define("Terrasoft.configuration.UsrSomeModule", {
        alternateClassName: "Terrasoft.UsrSomeModule",
        extend: "Terrasoft.BaseModule",
        Ext: null,
        sandbox: null,
        Terrasoft: null,
        messages: {
            "MessageToSubscribe": {
                mode: Terrasoft.MessageMode.PTP,
                direction: Terrasoft.MessageDirectionType.SUBSCRIBE
            },
            "MessageToPublish": {
                mode: Terrasoft.MessageMode.BROADCAST,
                direction: Terrasoft.MessageDirectionType.PUBLISH
            }
        },
        init: function() {
            this.callParent(arguments);
            this.sandbox.registerMessages(this.messages);
            this.processMessages();
        },
        processMessages: function() {
            this.sandbox.subscribe("MessageToSubscribe", this.onMessageSubscribe, this, ["result
            this.sandbox.publish("MessageToPublish", null, [this.sandbox.id]);
        },
        onMessageSubscribe: function(args) { console.log("'MessageToSubscribe' received");
            args.arg1 = 15;
            args.arg2 = "new arg2";
            return args;
        },
        destroy: function() {
            if (this.messages) {
                var messages = this.Terrasoft.keys(this.messages);
                this.sandbox.unRegisterMessages(messages);
```

© 2022 Terrasoft. Все права защищены.

```
            }
            this.callParent(arguments);
        }
    });
    return Terrasoft.UsrSomeModule;
});
```

Пример асинхронного обмена сообщениями

Средний

Пример. Организовать асинхронный обмен сообщениями между модулями.

В коде модуля, публикующего сообщение, необходимо при публикации в качестве аргумента передать конфигурационный объект, включающий функцию обратного вызова (callback-функцию).

В коде модуля-подписчика при подписке на сообщение в методе-обработчике вернуть результат асинхронно, через callback аргумента опубликованного сообщения.

Пример публикации сообщения и получения результата

```
this.sandbox.publish("AsyncMessageResult",
//Объект, передаваемый как аргумент функции-обработчика в подписчике.
{
    // Функция обратного вызова.
    callback: function(result) {
        this.Terrasoft.showInformation(result);
    },
    // Контекст выполнения функции обратного вызова.
    scope: this
});
```

Пример подписки на сообщение

```
this.sandbox.subscribe("AsyncMessageResult",
// Функция-обработчик сообщения
function(config) {
    // Обработка входящего параметра.
    var config = config || {};
    var callback = config.callback;
    var scope = config.scope || this;
    // Подготовка результирующего сообщения.
```

© 2022 Terrasoft. Все права защищены.

```
    var result = "Message from callback function";
    // Выполнение функции обратного вызова.
    if (callback) {
        callback.call(scope, result);
    }
},
// Контекст выполнения функции-обработчика сообщения.
 this);
```

Пример использования двунаправленных сообщений

Средний

В схеме `BaseEntityPage`, которая является базовой для всех схем модели представления страницы записи, зарегистрировано сообщение `CardModuleResponse`.

```
define("BaseEntityPage", [...], function(...) {
    return {
        messages: {
            ...
            "CardModuleResponse": {
                "mode": this.Terrasoft.MessageMode.PTP,
                "direction": this.Terrasoft.MessageDirectionType.BIDIRECTIONAL
            },
            ...
        },
        ...
    };
});
```

Сообщение публикуется, например, после сохранения измененной записи.

```
define("BasePageV2", [..., "LookupQuickAddMixin", ...],
    function(...) {
        return {
            ...
            methods: {
                ...
                onSaved: function(response, config) {
                    ...
                        this.sendSaveCardModuleResponse(response.success);
                    ...
```

© 2022 Terrasoft. Все права защищены.

```
                },
                ...
                sendSaveCardModuleResponse: function(success) {
                    var primaryColumnValue = this.getPrimaryColumnValue();
                    var infoObject = {
                        action: this.get("Operation"),
                        success: success,
                        primaryColumnValue: primaryColumnValue,
                        uId: primaryColumnValue,
                        primaryDisplayColumnValue: this.get(this.primaryDisplayColumnName),
                        primaryDisplayColumnName: this.primaryDisplayColumnName,
                        isInChain: this.get("IsInChain")
                    };
                    return this.sandbox.publish("CardModuleResponse", infoObject, [this.sandbox.
                },
                ...
            },
            ...
        };
    });
```

Эта функциональность реализована в дочерней схеме `BasePageV2` (т.е. схема `BaseEntityPage` является родительской для `BasePageV2`). Также в `BasePageV2` в качестве зависимости указан миксин `LookupQuickAddMixin`, в котором выполняется подписка на сообщение `CardModuleResponse`.

На заметку. Миксин — это класс-примесь, предназначенный для расширения функциональности других классов. Миксины расширяют функциональность схемы, при этом позволяя не дублировать часто употребляемую логику в методах схемы. Миксины отличаются от остальных модулей, подключаемых в список зависимостей, способом вызова методов из схемы модуля — к методам миксина можно обращаться напрямую, как к методам схемы.

```
define("LookupQuickAddMixin", [...],
        function(...) {
            Ext.define("Terrasoft.configuration.mixins.LookupQuickAddMixin", {
                alternateClassName: "Terrasoft.LookupQuickAddMixin",
                ...
                // Объявление сообщения.
                _defaultMessages: {
                    "CardModuleResponse": {
                        "mode": this.Terrasoft.MessageMode.PTP,
                        "direction": this.Terrasoft.MessageDirectionType.BIDIRECTIONAL
                    }
                },
                ...
                // Метод регистрации сообщения.
```

© 2022 Terrasoft. Все права защищены.

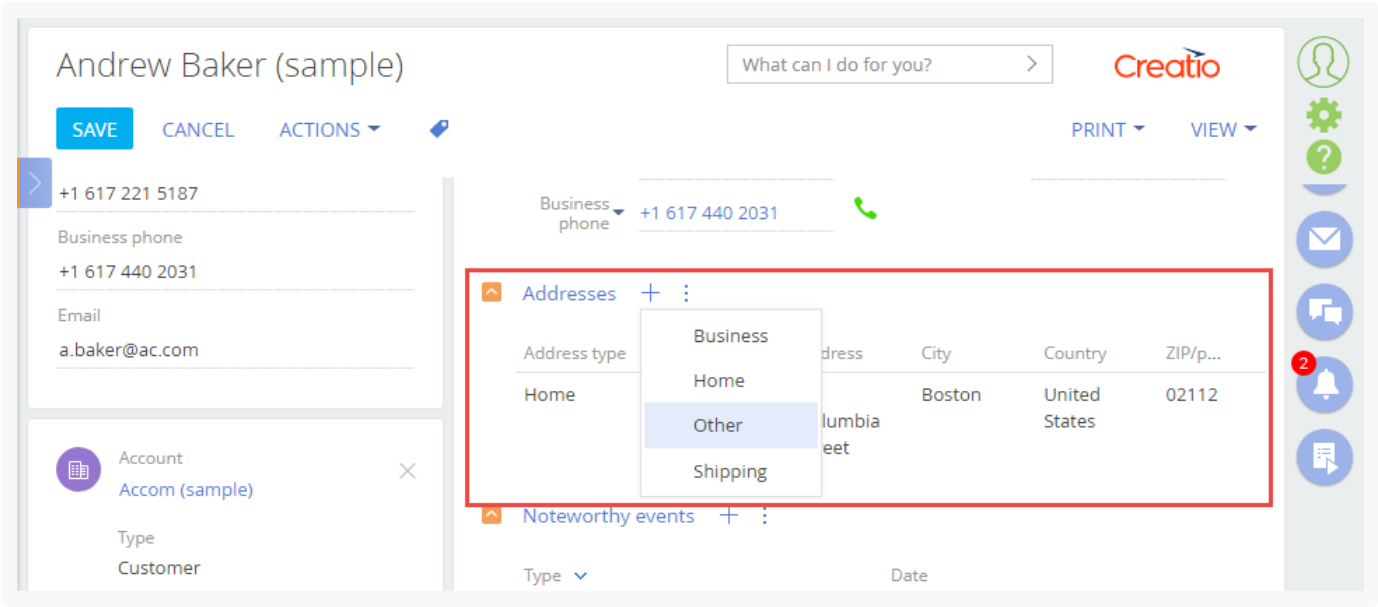
```
            _registerMessages: function() {
                this.sandbox.registerMessages(this._defaultMessages);
            },
            ...
            // Инициализация экземпляра класса.
            init: function(callback, scope) {
                ...
                this._registerMessages();
                ...
            },
            ...
            // Выполняется после добавления новой записи в справочник.
            onLookupChange: function(newValue, columnName) {
                ...
                // Здесь выполняется цепочка вызовов методов.
                // В результате будет вызван метод _subscribeNewEntityCardModuleResponse().
                ...
            },
            ...
            // Метод, в котором выполняется подписка на сообщение "CardModuleResponse".
            // В callback-функции выполняется установка в справочное поле значения,
            // отправленного при публикации сообщения.
            _subscribeNewEntityCardModuleResponse: function(columnName, config) {
                this.sandbox.subscribe("CardModuleResponse", function(createdObj) {
                    var rows = this._getResponseRowsConfig(createdObj);
                    this.onLookupResult({
                        columnName: columnName,
                        selectedRows: rows
                    });
                }, this, [config.moduleId]);
            },
            ...
        });
        return Terrasoft.LookupQuickAddMixin;
    });
```

Последовательность работы с двунаправленным сообщением можно рассмотреть на примере добавления нового адреса на странице контакта.

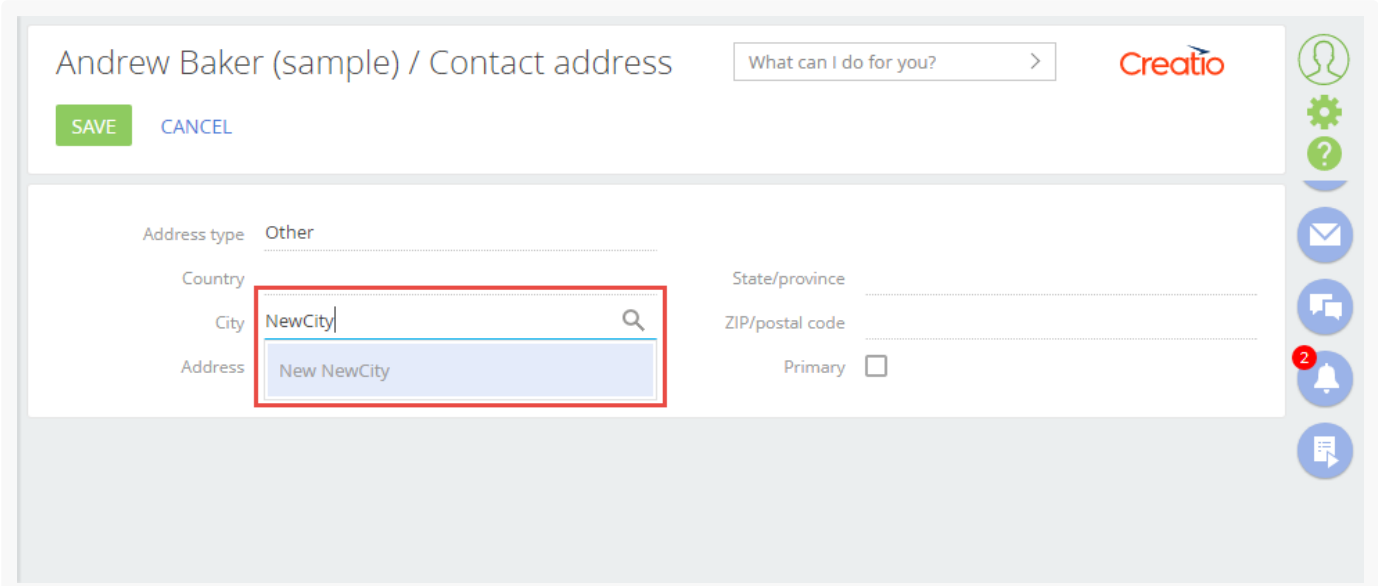
1. После выполнения команды добавления новой записи на детали [*Адреса*] ([*Addresses*]) выполняется загрузка модуля `ContactAddressPageV2` в цепочку модулей и открывается страница адреса контакта.

Добавление новой записи на детали [*Адреса*] ([*Addresses*])

© 2022 Terrasoft. Все права защищены.



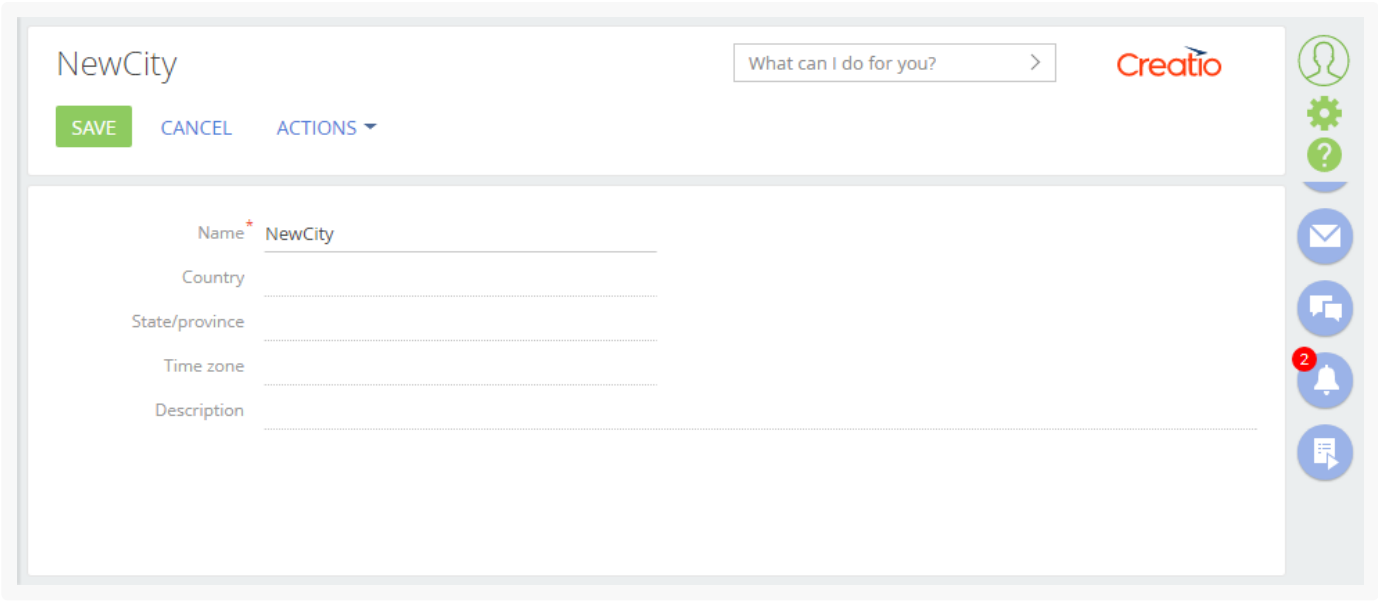
Страница адреса контакта



Поскольку схема `ContactAddressPageV2` имеет в своей иерархии наследования как `BaseEntityPage`, так и `BasePageV2`, то в ней уже зарегистрировано сообщение `CardModuleResponse`. Это сообщение также регистрируется в методе `_registerMessages()` миксина `LookupQuickAddMixin` при его инициализации как модуля-зависимости `BasePageV2`.

2. При быстром добавлении нового значения в справочные поля страницы `ContactAddressPageV2` (например, нового города вызывается метод `onLookupChange()` миксина `LookupQuickAddMixin`. В этом методе, кроме выполнения загрузки модуля `CityPageV2` в цепочку модулей, также вызывается метод `_subscribeNewEntityCardModuleResponse()`, в котором осуществляется подписка на сообщение `CardModuleResponse`. После этого откроется страница города (схема `CityPageV2`).

© 2022 Terrasoft. Все права защищены.



3. Поскольку схема `CityPageV2` также имеет в иерархии наследования схему `BasePageV2`, то после сохранения записи (кнопка [*Сохранить*] ([*Save*]) выполняется метод `onSaved()`, реализованный в базовой схеме. Этот метод в свою очередь вызывает метод `sendSaveCardModuleResponse()`, в котором осуществляется публикация сообщения `CardModuleResponse`. При этом передается объект с необходимыми результатами сохранения.
4. После публикации сообщения выполняется callback-функция подписчика (см. метод `_subscribeNewEntityCardModuleResponse()` миксина `LookupQuickAddMixin`), в которой обрабатываются результаты сохранения нового города в справочник.

Таким образом, публикация и подписка на двунаправленное сообщение были выполнены в рамках одной иерархии наследования схем, в которой базовой является схема `BasePageV2`, содержащая всю необходимую функциональность.

Настроить загрузку и выгрузку модулей

Средний

Пример. Создать визуальный модуль `UsrCardModule` и выполнить его загрузку в модуль `UsrModule`.

1. Создать класс визуального модуля

Создайте класс модуля `UsrCardModule`, наследующего базовый класс `BaseSchemaModule`. Класс должен быть инстанцируемым, то есть возвращать функцию-конструктор. В таком случае при загрузке модуля извне можно будет передать в конструктор необходимые параметры.

Объявление класса модуля UsrCardModule

```
// Модуль, возвращающий экземпляр класса.
define("UsrCardModule", [...], function(...) {
```

© 2022 Terrasoft. Все права защищены.

```
    Ext.define("Terrasoft.configuration.UsrCardModule", {
        // Псевдоним класса.
        alternateClassName: "Terrasoft.UsrCardModule",
        // Родительский класс.
        extend: "Terrasoft.BaseSchemaModule",
        // Признак, что параметры схемы установлены извне.
        isSchemaConfigInitialized: false,
        // Признак, что при загрузке модуля используется состояние истории.
        useHistoryState: true,
        // Название схемы отображаемой сущности.
        schemaName: "",
        // Признак использования режима совместного отображения с реестром раздела.
        // Если значение false, то на странице присутствует SectionModule.
        isSeparateMode: true,
        // Название схемы объекта.
        entitySchemaName: "",
        // Значение первичной колонки.
        primaryColumnValue: Terrasoft.GUID_EMPTY,
        // Режим работы страницы записи. Возможные значения
        // ConfigurationEnums.CardStateV2.ADD|EDIT|COPY
        operation: ""
    });
    // Возврат экземпляра класса.
    return Terrasoft.UsrCardModule;
}
```

2. Создать класс модуля, в который будет выполнена загрузка визуального модуля

Создайте класс модуля `UsrModule`, наследующего класс `BaseModel`.

Объявление класса модуля UsrModule

```
define("UsrModule", [...], function(...) {
    Ext.define("Terrasoft.configuration.UsrModule", {
        alternateClassName: "Terrasoft.UsrModule",
        extend: "Terrasoft.BaseModel",
        Ext: null,
        sandbox: null,
        Terrasoft: null,
    });
}
```

3. Загрузить модуль

© 2022 Terrasoft. Все права защищены.

Существует возможность при загрузке модуля передавать аргументы в конструктор класса инстанцируемого модуля. Для этого в конфигурационный объект `config` (параметр метода `sandbox.loadModule()`) нужно добавить свойство `instanceConfig`, которому необходимо присвоить объект с нужными значениями.

Создайте в модуле `UsrModule` конфигурационный объект `configObj`, свойства которого будут переданы в свойства конструктора. Затем загрузите визуальный модуль `UsrCardModule` с помощью метода `sandbox.loadModule()`.

Загрузка модуля с передачей параметров

```
...
init: function() {
    this.callParent(arguments);
    // Объект, свойства которого содержат значения,
    // передаваемые как параметры конструктора
    var configObj = {
        isSchemaConfigInitialized: true,
        useHistoryState: false,
        isSeparateMode: true,
        schemaName: "QueueItemEditPage",
        entitySchemaName: "QueueItem",
        operation: ConfigurationEnums.CardStateV2.EDIT,
        primaryColumnValue: "{3B58C589-28C1-4937-B681-2D40B312FBB6}"
    };

    // Загрузка модуля.
    this.sandbox.loadModule("UsrCardModule", {
        renderTo: "DelayExecutionModuleContainer",
        id: this.getQueueItemEditModuleId(),
        keepAlive: true,
        // Указание значений, передаваемых в конструктор модуля.
        instanceConfig: configObj
        }
    });
}
...
```

Для передачи дополнительных параметров в модуль при его загрузке в конфигурационном объекте `config` предусмотрено свойство `parameters`. Такое же свойство должно быть определено и в классе модуля (или в одном из его родительских классов).

На заметку. Свойство `parameters` уже определено в классе `Terrasoft.BaseModule`.

Таким образом, при создании экземпляра модуля свойство `parameters` модуля будет

© 2022 Terrasoft. Все права защищены.

проинициализировано значениями, переданными в свойстве `parameters` объекта `config`.

Объект Sandbox JS

Средний

Sandbox — компонент ядра, который служит диспетчером при взаимодействии модулей системы. Sandbox предоставляет механизм обмена сообщениями между модулями и загрузки модулей по требованию в интерфейс приложения.

Методы

`registerMessages(messageConfig)`

Регистрирует сообщения модуля.

Параметры

`{Object} messageConfig`

Конфигурационный объект сообщений модуля.

Конфигурационный объект является коллекцией "ключ-значение", в которой каждый элемент имеет вид, представленный ниже.

Элемент конфигурационного объекта

```
// Ключ элемента коллекции, содержащий имя сообщения.
"MessageName": {
    // Значение элемента коллекции.
    mode: [Режим работы сообщения],
    direction: [Направление сообщения]
}
```

Свойства значения элемента коллекции

`{Terrasoft.MessageMode} mode`	Режим работы сообщения, содержит значение перечисления. `Terrasoft.MessageMode (Terrasoft.core.enums.MessageMode)` Возможные значения(`Terrasoft.MessageMode`)

© 2022 Terrasoft. Все права защищены.

	`BROADCAST`
	Широковещательный режим работы сообщения, при котором количество подписчиков заранее неизвестно.
	`PTP`
	Адресный режим работы сообщения, при котором сообщение может быть обработано только одним подписчиком.

`{Terrasoft.MessageDirectionType} direction`	Направление сообщения, содержит значение перечисления.
	`Terrasoft.MessageDirectionType (Terrasoft.core.enums.MessageDirectionType)`
	Возможные значения (`Terrasoft.MessageDirectionType`)
	`PUBLISH`
	Направление сообщения — публикация; модуль может только опубликовать сообщение в `sandbox`.
	`SUBSCRIBE`
	Направление сообщения — подписка; модуль может только подписаться на сообщение, опубликованное из другого модуля.
	`BIDIRECTIONAL`
	Направление сообщения — двунаправленное; позволяет публиковать и подписываться на одно и то же сообщение в разных экземплярах одного и того же класса или в рамках одной и той же иерархии наследования схем.

`unRegisterMessages(messages)`

Отменяет регистрацию сообщений.

Параметры

© 2022 Terrasoft. Все права защищены.

`{String|Array} messages`	Имя или массив имен сообщений.

`publish(messageName , messageArgs, tags)`

Публикует сообщение в `sandbox`.

Параметры

`{String} messageName`	Строка, содержащая имя сообщения, например, "MessageToSubscribe".
`{Object} messageArgs`	Объект, передаваемый в качестве аргумента в метод-обработчик сообщения в модуле-подписчике. Если в методе-обработчике нет входящих параметров, то параметру `messageArgs` необходимо присвоить значение `null`.
`{Array} tags`	Массив тегов, позволяющий однозначно определить модуль, отправляющий сообщение. Как правило, используется значение `[this.sandbox.id]`. По массиву тегов `sandbox` определяет подписчиков и публикаторов сообщения.

Примеры использования

Примеры использования метода `publish()`

```
// Публикация сообщения без аргумента и тегов.
this.sandbox.publish("MessageWithoutArgsAndTags");
// Публикация сообщения без аргументов для метода-обработчика.
this.sandbox.publish("MessageWithoutArgs", null, [this.sandbox.id]);
// Публикация сообщения с аргументом для метода-обработчика.
this.sandbox.publish("MessageWithArgs", {arg1: 5, arg2: "arg2"}, ["moduleName"]);
// Публикация сообщения с произвольным массивом тегов.
this.sandbox.publish("MessageWithCustomIds", null, ["moduleName","otherTag"]);
```

`subscribe(messageName, messageHandler, scope, tags)`

Выполняет подписку на сообщение.

Параметры

© 2022 Terrasoft. Все права защищены.

`{String} messageName`	Строка, содержащая имя сообщения, например, "MessageToSubscribe".
`{Function} messageHandler`	Метод-обработчик, вызываемый при получении сообщения. Это может быть анонимная функция или метод модуля. В определении метода может быть указан параметр, значение которого должно быть передано при публикации сообщения с помощью метода `sandbox.publish()`.
`{Object} scope`	Контекст выполнения метода-обработчика `messageHandler`.
`{Array} tags`	Массив тегов, позволяющий однозначно определить модуль, отправляющий сообщение. По массиву тегов `sandbox` определяет подписчиков и публикаторов сообщения.

Примеры использования

Примеры использования метода `subscribe()`

```
// Подписка на сообщение без аргументов для метода-обработчика.
// Метод-обработчик — анонимная функция. Контекст выполнения — текущий модуль.
// Метод getsandboxid() должен вернуть тег, совпадающий с тегом опубликованного сообщения.
this.sandbox.subscribe("MessageWithoutArgs", function(){console.log("Message without argume
// Подписка на сообщение с аргументом для метода-обработчика.
this.sandbox.subscribe("MessageWithArgs", function(args){console.log(args)}, this, ["module
// Подписка на сообщение с произвольным тегом.
// Тег может быть любым из массива тегов опубликованного сообщения.
// Метод-обработчик myMsgHandler должен быть реализован отдельно.
this.sandbox.subscribe("MessageWithCustomIds", this.myMsgHandler, this, ["otherTag"]);
```

`loadModule(moduleName, config)`

Загружает модуль.

Параметры

`{String} moduleName`	Название модуля.
`{Object} config`	Конфигурационный объект, содержащий параметры модуля. Обязательный параметр для визуальных модулей. Свойства конфигурационного объекта --- `{String} id`

© 2022 Terrasoft. Все права защищены.

Идентификатор модуля. Если не указан, то сформируется автоматически.

`{String} renderTo`

Название контейнера, в котором будет отображено представление визуального модуля. Передается в качестве аргумента в метод `render()` загружаемого модуля. Обязателен для визуальных модулей.

`{Boolean} keepAlive`

Признак добавление модуля в цепочку модулей. Используется для навигации между представлениями модулей в браузере.

`{Boolean} isAsync`

Признак асинхронной инициализации модуля.

`{Object} instanceConfig`

Предоставляет возможность при загрузке модуля передавать аргументы в конструктор класса инстанцируемого модуля. Для этого свойству `instanceConfig` необходимо присвоить объект с нужными значениями.

> **На заметку.** Инстанцируемым является модуль, возвращающий функцию-конструктор.

Передавать в экземпляр модуля можно свойства следующих типов:

- `string`;
- `boolean`;
- `number`;
- `date` (значение будет скопировано);
- `object` (только объекты-литералы. Нельзя передавать экземпляры классов, наследники HTMLElement и т. п.).

При передаче параметров в конструктор модуля наследника `Terrasoft.BaseObject` действует следующее ограничение: нельзя передать параметр, не описанный в классе модуля или в одном из родительских классов.

© 2022 Terrasoft. Все права защищены.

`{Object} parameters`

Используется для передачи дополнительных параметров в модуль при его загрузке.

Такое же свойство должно быть определено и в классе модуля (или в одном из его родительских классов).

> **На заметку.** Свойство `parameters` уже определено в классе `Terrasoft.BaseModule`.

Таким образом, при создании экземпляра модуля свойство `parameters` модуля будет проинициализировано значениями, переданными в свойстве `parameters` объекта `config`.

Примеры использования

Примеры использования метода `loadModule()`

```
// Загрузка модуля без дополнительных параметров.
this.sandbox.loadModule("ProcessListenerV2");
// Загрузка модуля с дополнительными параметрами.
this.sandbox.loadModule("CardModuleV2", {
    renderTo: "centerPanel",
    keepAlive: true,
    id: moduleId
});
```

`unloadModule(id, renderTo, keepAlive)`

Выгружает модуль.

Параметры

`{String} id`	Идентификатор модуля.
`{String} renderTo`	Название контейнера, из которого необходимо удалить представление визуального модуля. Обязателен для визуальных модулей.
`{Boolean} keepAlive`	Признак сохранения модели модуля. При выгрузке модуля ядро может сохранить его модель для возможности использовать ее свойства, методы, сообщения.

© 2022 Terrasoft. Все права защищены.

Примеры использования

Примеры использования метода `unloadModule()`

```
...
// Метод получения идентификатора выгружаемого модуля.
getModuleId: function() {
    return this.sandbox.id + "_ModuleName";
},
...
// Выгрузка невизуального модуля.
this.sandbox.unloadModule(this.getModuleId());
...
// Выгрузка визуального модуля, ранее загруженного в контейнер "ModuleContainer".
this.sandbox.unloadModule(this.getModuleId(), "ModuleContainer");
```

Передача сообщений по WebSocket

Сложный

Для транслирования сообщений, полученных по WebSocket, подписчикам внутри системы в Creatio используется схема `ClientMessageBridge`.

Механизм передачи сообщений

Если для сообщения, полученного по WebSocket, в расширяющих схемах `ClientMessageBridge` не определена дополнительная логика, то используется широковещательная отправка сообщения внутри системы через `sandbox` с именем "SocketMessageReceived". Подписавшись на это сообщение, можно обработать данные, полученные по WebSocket.

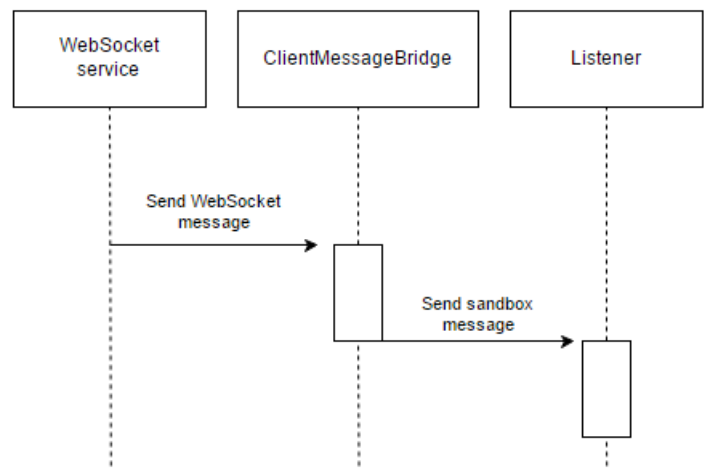
Для дополнительных настроек публикуемого сообщения, то необходимо выполнить следующе действия:

1. Создать замещающую клиентскую схему, в которой родительской схемой будет выступать "ClientMessageBridge".
2. В свойстве `messages` привязать к схеме сообщение с необходимыми настройками.
3. Выполнить перегрузку родительского метода `init()` для добавления сообщения, полученного по WebSocket, в конфигурационный объект сообщений схемы.
4. Выполнить перегрузку базового метода `afterPublishMessage()` для отслеживания момента рассылки сообщения.

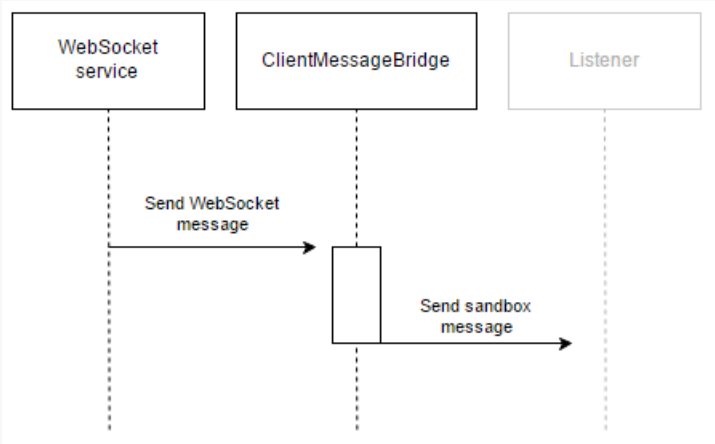
Механизм сохранения истории сообщений

В идеальном случае на момент публикации сообщения внутри системы присутствует обработчик "Listener".

© 2022 Terrasoft. Все права защищены.



Но возможна ситуация, когда обработчик еще не загружен.



В этом случае, чтобы не потерять необработанные сообщения, Creatio выполняет следующие действия:

1. Сообщения сохраняются в истории.
2. Перед публикацией каждого сообщения проверяется наличие обработчика.
3. Когда обработчик загружен, ему публикуются все сохраненные сообщения в порядке их получения.
4. После публикации сообщений из истории вся история очищается.

Для реализации описанной возможности необходимо при добавлении нового конфигурационного объекта, указать в нем признак `isSaveHistory` равным `true`.

Пример конфигурирования обработки сообщения с сохранением в историю

```
init: function() {
   // Вызов родительского метода init().
   this.callParent(arguments);
   // Добавление нового конфигурационного объекта в коллекцию конфигурационных объектов.
   this.addMessageConfig({
       // sender – имя сообщения, получаемого по WebSocket.
```

© 2022 Terrasoft. Все права защищены.

```
        sender: "OrderStepCalculated",
        // messageName — имя сообщения, с которым оно будет разослано внутри системы.
        messageName: "OrderStepCalculated",
        // isSaveHistory — признак, указывающий на необходимость сохранения истории.
        isSaveHistory: true
    });
},
```

В классе `ClientMessageBridge` реализованы абстрактные методы родительского класса `BaseMessageBridge`, которые обеспечивают сохранение истории сообщений и работу с ними с использованием `localStorage` браузера:

- `saveMessageToHistory()` — обеспечивает сохранение нового сообщения в коллекции сообщений;
- `getMessagesFromHistory()` — обеспечивает получение массива сообщений по передаваемому имени;
- `deleteSavedMessages()` — удаляет все сохраненные сообщения по передаваемому имени.

Для реализации работы с другим типом хранилища требуется создать наследник класса `BaseMessageBridge` и добавить в него необходимую реализацию абстрактных методов `saveMessageToHistory()`, `getMessagesFromHistory()` и `deleteSavedMessages()`.

Настроить нового подписчика

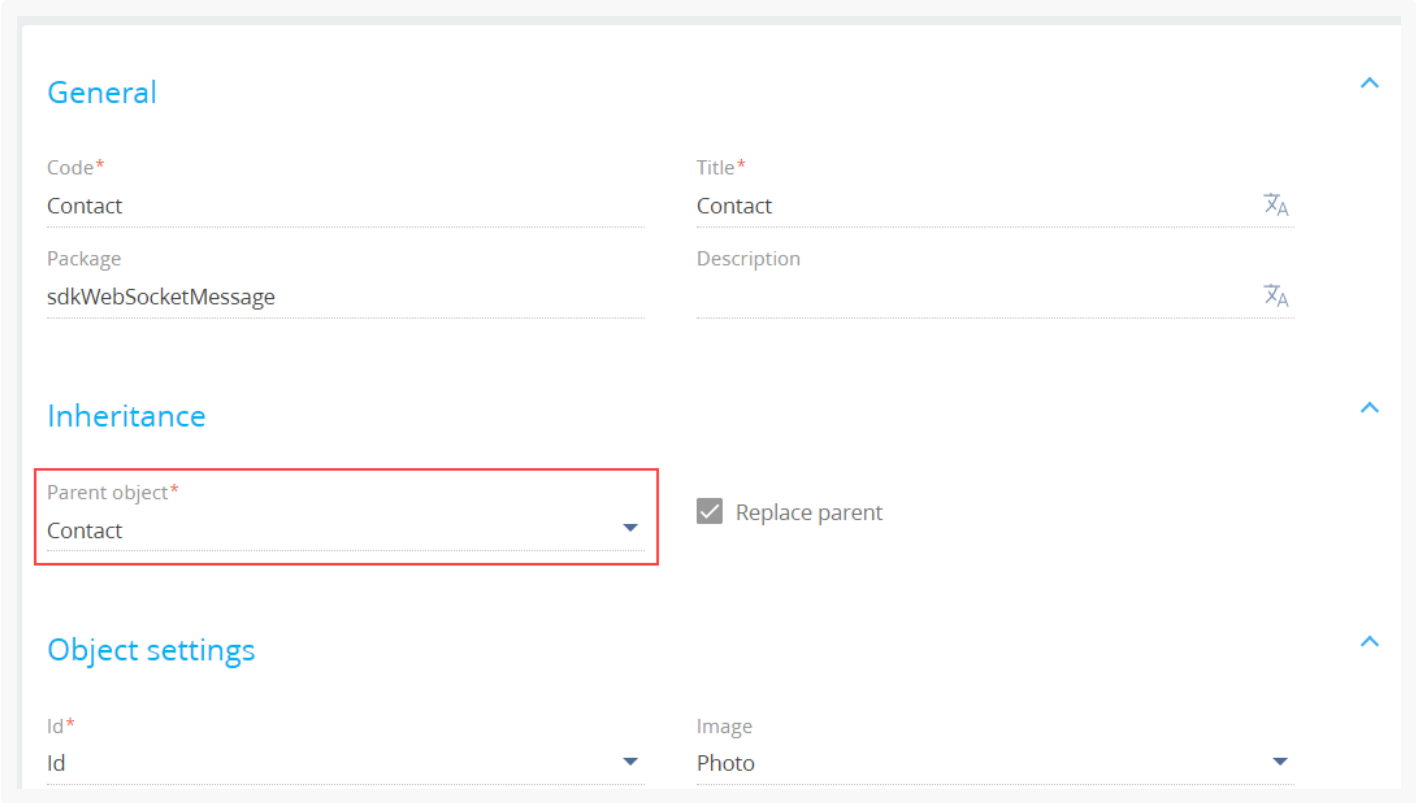
 Сложный

Пример. При сохранении контакта на стороне сервера необходимо по WebSocket опубликовать сообщение с именем `NewUserSet`, содержащее информацию о дне рождения и ФИО контакта. На стороне клиента необходимо реализовать рассылку сообщений `NewUserSet` внутри системы. Дополнительно перед рассылкой необходимо обработать свойство `birthday` сообщения, полученного по WebSocket, а после рассылки вызвать метод `afterPublishUserBirthday` утилитного класса. В завершение необходимо реализовать подписку на сообщение, разосланное на стороне клиента, например, в схеме страницы контакта.

1. Создать замещающий объект [*Контакт*]

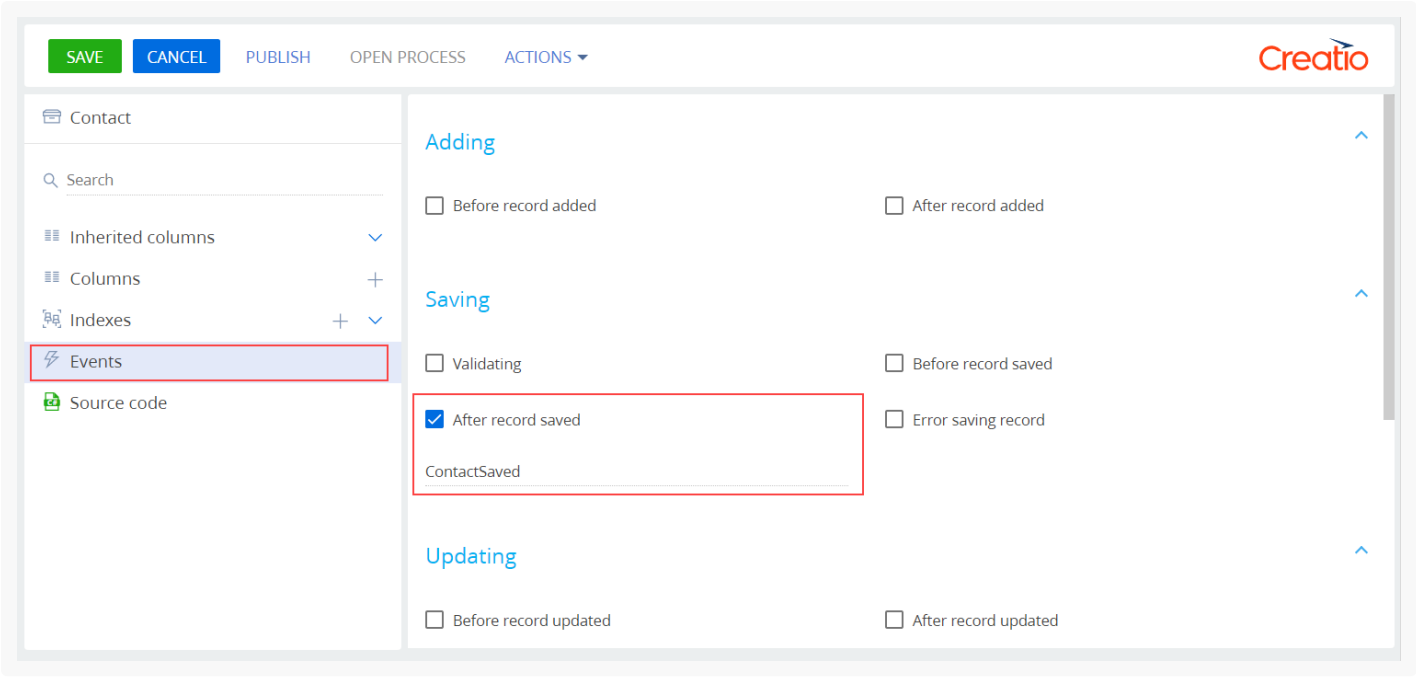
Прежде чем добавлять функциональность публикации сообщения по WebSocket, создайте замещающий объект, указав родителем [*Контакт*] ([*Contact*]).

© 2022 Terrasoft. Все права защищены.



2. Создать событие "После сохранения записи"

Добавьте функциональность публикации сообщения по WebSocket после сохранения записи контакта. Для этого в списке событий объекта выберите событие "`После сохранения записи`" ("`After record saved`").



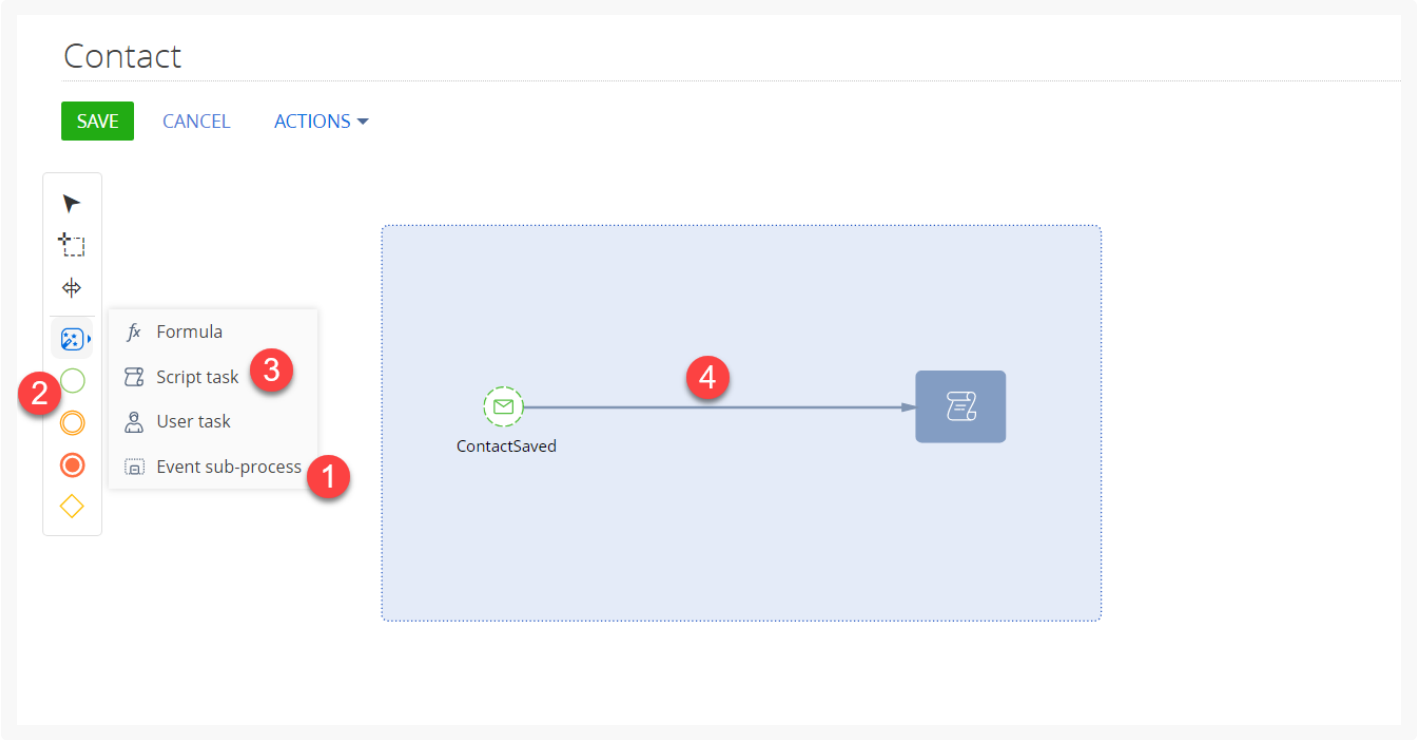
3. Реализовать событийный подпроцесс

© 2022 Terrasoft. Все права защищены.

В обработчике события "`После сохранения записи`" реализуйте событийный подпроцесс, который запускается сообщением `ContactSaved`.

1. Добавьте элемент событийного подпроцесса (1).
2. Добавьте элемент сообщения (2), указав имя сообщения `ContactSaved`.
3. Добавьте элемент сценария (3).
4. Соедините объект сообщения и сценария связью (4).

Создание подпроцесса для обработки сообщения



4. Добавить логику публикации сообщения по WebSocket

Для добавления логики публикации сообщения по WebSocket откройте элемент [*Задание-сценарий*] событийного подпроцесса и добавьте следующий исходный код.

Пример добавления логики публикации сообщения по WebSocket

```
// Получение имени контакта.
string userName = Entity.GetTypedColumnValue<string>("Name");
// Получение даты дня рождения контакта.
DateTime birthDate = Entity.GetTypedColumnValue<DateTime>("BirthDate");
// Формирование текста сообщения.
string messageText = "{\"birthday\": \"" + birthDate.ToString("s") + "\", \"name\": \"" + userNa
// Присвоение сообщению имени.
string sender = "NewUserSet";
// Публикация сообщения по WebSocket.
MsgChannelUtilities.PostMessageToAll(sender, messageText);
```

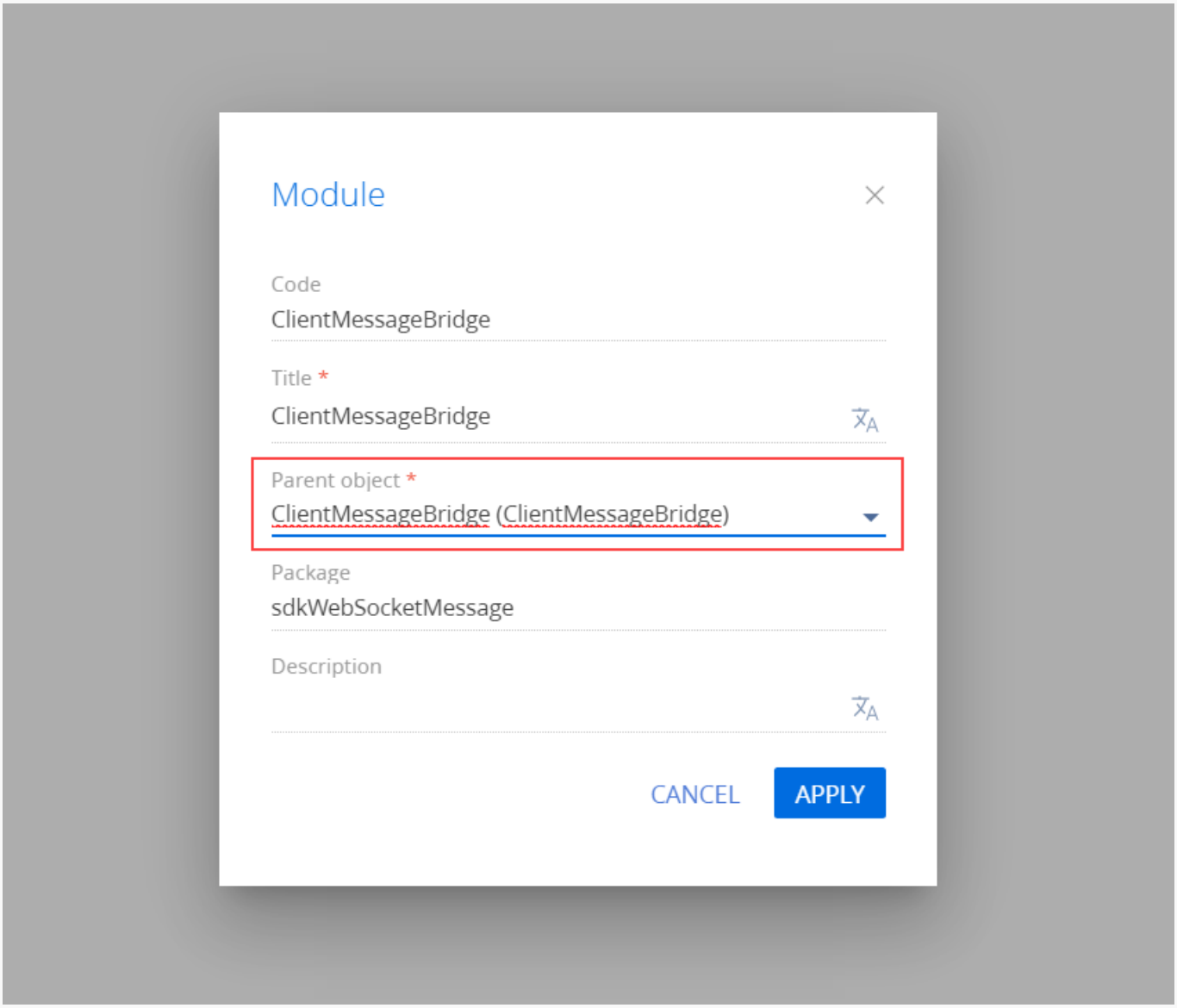
© 2022 Terrasoft. Все права защищены.

```
return true;
```

После этого сохраните и опубликуйте событийный подпроцесс.

5. Реализовать рассылку сообщения внутри приложения

Для этого в пользовательском пакете создайте замещающую модель представления, указав в качестве родительского объекта схему `ClientMessageBridge` пакета `NUI`.



Для реализации рассылки широковещательного сообщения `NewUserSet` добавьте в схему следующий исходный код.

ClientMessageBridge.js

© 2022 Terrasoft. Все права защищены.

```
define("ClientMessageBridge", ["ConfigurationConstants"],
    function(ConfigurationConstants) {
        return {
            // Сообщения.
            messages: {
                // Имя сообщения.
                "NewUserSet": {
                    // Тип сообщения — широковещательное, без указания конкретного подписчика.
                    "mode": Terrasoft.MessageMode.BROADCAST,
                    // Направление сообщения — публикация.
                    "direction": Terrasoft.MessageDirectionType.PUBLISH
                }
            },
            methods: {
                // Инициализация схемы.
                init: function() {
                    // Вызов родительского метода.
                    this.callParent(arguments);
                    // Добавление нового конфигурационного объекта в коллекцию конфигурационных
                    this.addMessageConfig({
                        // Имя сообщения, получаемого по WebSocket.
                        sender: "NewUserSet",
                        // Имя сообщения, с которым оно будет разослано.
                        messageName: "NewUserSet"
                    });
                },
                // Метод, выполняемый после публикации сообщения.
                afterPublishMessage: function(
                    // Имя сообщения, с которым оно было разослано.
                    sandboxMessageName,
                    // Содержимое сообщения.
                    webSocketBody,
                    // Результат отправки сообщения.
                    result,
                    // Конфигурационный объект рассылки сообщения.
                    publishConfig) {
                    // Проверка, что сообщение соответствует добавленному в конфигурационный объ
                    if (sandboxMessageName === "NewUserSet") {
                        // Сохранение содержимого в локальные переменные.
                        var birthday = webSocketBody.birthday;
                        var name = webSocketBody.name;
                        // Вывод содержимого в консоль браузера.
                        window.console.info("Опубликовано сообщение: " + sandboxMessageName +
                            ". Данные: name: " + name +
                            "; birthday: " + birthday);
                    }
                }
```

© 2022 Terrasoft. Все права защищены.

```
            }
        };
    });
```

Здесь в свойстве `messages` к схеме привязывается широковещательное сообщение `NewUserSet`, которое может публиковаться внутри системы. В свойстве `methods` выполняется перегрузка родительского метода `init()` для добавления сообщения, полученного по `WebSocket` в конфигурационый объект сообщений схемы. Для того чтобы отслеживать момент рассылки сообщения, выполняется перегрузка родительского метода `afterPublishMessage`.

После сохранения схемы и обновления страницы в браузере, сообщения `NewUserSet`, полученные по WebSocket, будут рассылаться внутри системы, о чем будет сигнализировать сообщение в консоли браузера в режиме отладки.

6. Реализовать подписку на сообщение

Чтобы получить объектпо WebSocket, подпишитесь на получение сообщения `NewUserSet` в любой схеме, например, в схеме страницы контакта. Для этого создайте замещающую модель представления (см. п. 5), указав в качестве родительского объекта "ContactPageV2".

Затем в схему добавьте следующий исходный код:

ContactPageV2.js

```
define("ContactPageV2", [],
    function(BusinessRuleModule, ConfigurationConstants) {
        return {
            // Имя схемы объекта.
            entitySchemaName: "Contact",
            messages: {
                // Имя сообщения.
                "NewUserSet": {
                    // Тип сообщения – широковещательное, без указания конкретного подписчика.
                    "mode": Terrasoft.MessageMode.BROADCAST,
                    // Направление сообщения – подписка.
                    "direction": Terrasoft.MessageDirectionType.SUBSCRIBE
                }
            },
            methods: {
                // Инициализация схемы.
                init: function() {
                    // Вызов родительского метода init().
                    this.callParent(arguments);
                    // Подписка на прием сообщения NewUserSet.
                    this.sandbox.subscribe("NewUserSet", this.onNewUserSet, this);
                },
                // Обработчик события получения сообщения NewUserSet.
                onNewUserSet: function(args) {
```

© 2022 Terrasoft. Все права защищены.

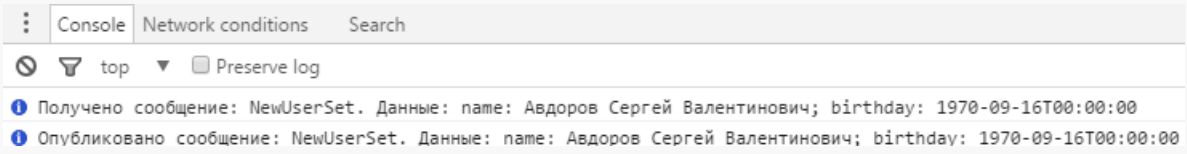
```
                    // Сохранение содержимого сообщения в локальные переменные.
                    var birthday = args.birthday;
                    var name = args.name;
                    // Вывод содержимого в консоль браузера.
                    window.console.info("Получено сообщение: NewUserSet. Данные: name: " +
                        name + "; birthday: " + birthday);
                }

            }
        };
    });
```

Здесь в свойстве `messages` к схеме привязывается широковещательное сообщение `NewUserSet`, на которое можно подписаться. В свойстве `methods` выполняется перегрузка родительского метода `init()` для подписки на сообщение `NewUserSet`, с указанием метода-обработчика `onNewUserSet()`, в котором выполняется обработка полученного в сообщении объекта и вывод результата в консоль браузера.

После добавления исходного кода сохраните схему и обновите страницу приложения в браузере.

Результат выполнения примера

Результатом выполнения примера будет получение двух информационных сообщений в консоли браузера после сохранения контакта.



Класс ClientMessageBridge JS

Сложный

Класс `ClientMessageBridge` используется для транслирования сообщений, полученных по WebSocket, подписчикам внутри системы.

Для дополнительной обработки сообщений перед публикацией и изменением имени, с которым сообщение будет рассылаться внутри системы, необходимо реализовать расширяющую схему. В расширяющей схеме, используя доступный API, можно сконфигурировать отправку для конкретного типа сообщений.

Свойства

`WebSocketMessageConfigs` Array

© 2022 Terrasoft. Все права защищены.

Коллекция конфигурационных объектов.

`LocalStoreName` `String`

Название хранилища, в котором сохраняется история сообщений.

`LocalStore` `Terrasoft.LocalStore`

Экземпляр класса, реализующий доступ к локальному хранилищу.

Методы

`init()`

Инициализирует значения по умолчанию.

`getSandboxMessageListenerExists(sandboxMessageName)`

Проверяет наличие слушателей сообщения с переданным именем. Возвращаемое значение: `Boolean` — результат проверки наличия слушателя сообщения.

Параметры

`sandboxMessageName: String`	Имя сообщения, с которым оно будет разослано внутри системы.

`publishMessageResult(sandboxMessageName, webSocketMessage)`

Публикует сообщение внутри системы. Возвращаемое значение: * — результат, полученный от обработчика сообщения.

Параметры

`sandboxMessageName: String`	Имя сообщения, с которым оно будет разослано внутри системы.
`webSocketMessage: Object`	Сообщение полученное по WebSocket.

`beforePublishMessage(sandboxMessageName, webSocketBody, publishConfig)`

Обработчик, вызываемый перед публикацией сообщения внутри системы.

Параметры

© 2022 Terrasoft. Все права защищены.

`sandboxMessageName: String`	Имя сообщения, с которым оно будет разослано внутри системы.
`webSocketBody: Object`	Сообщение, полученное по WebSocket.
`publishConfig: Object`	Конфигурационный объект рассылки сообщения.

`afterPublishMessage(sandboxMessageName, webSocketBody, result, publishConfig)`

Обработчик, вызываемый после публикации сообщения внутри системы.

Параметры

`sandboxMessageName: String`	Имя сообщения, с которым оно будет разослано внутри системы.
`webSocketBody: Object`	Сообщение, полученное по WebSocket. result: * — результат выполнения публикации сообщения внутри системы.
`publishConfig: Object`	Конфигурационный объект рассылки сообщения.

`addMessageConfig(config)`

Добавляет новый конфигурационный объект в коллекцию конфигурационных объектов.

Параметры

`config: Object`	Конфигурационный объект.

Пример конфигурационного объекта

```
{
    "sender": "webSocket sender key 1",
    "messageName": "sandbox message name 1",
    "isSaveHistory": true
}
```

Здесь:

- `sender: String` — имя сообщения, которое ожидается получить по WebSocket.
- `messageName: String` — имя сообщения, с которым оно будет разослано внутри системы.

© 2022 Terrasoft. Все права защищены.

- `isSaveHistory: Boolean` — признак, отвечающий за необходимость сохранения истории сообщений.

`saveMessageToHistory(sandboxMessageName, webSocketBody)`

Сохраняет сообщение в хранилище, если подписчик отсутствует, а в конфигурационном объекте указан признак необходимости сохранения.

Параметры

`sandboxMessageName: String`	Имя сообщения, с которым оно будет разослано внутри системы.
`webSocketBody: Object`	Сообщение, полученное по WebSocket.

`getMessagesFromHistory(sandboxMessageName)`

Возвращает массив сохраненных сообщений из хранилища.

Параметры

`sandboxMessageName: String`	Имя сообщения, с которым оно будет разослано внутри системы.

`deleteSavedMessages(sandboxMessageName)`

Удаляет из хранилища сохраненные сообщения.

Параметры

`sandboxMessageName: String`	Имя сообщения, с которым оно будет разослано внутри системы.

Операции с данными (front-end)

Средний

Работа с данными во front-end модулях Creatio реализована с помощью высокоуровневого класса `EntitySchemaQuery`, который предназначен для построения запросов на выборку из базы данных.

Особенности класса `EntitySchemaQuery`:

- Запрос на выборку данных `EntitySchemaQuery` строится с учетом прав доступа текущего пользователя.
- Механизм кэширования позволяет оптимизировать выполнение операций за счет обращения к кэшированным результатам запроса без дополнительного обращения к базе данных.

© 2022 Terrasoft. Все права защищены.

Алгоритм работы с данными во front-end модулях Creatio:

1. Создайте экземпляр класса `EntitySchemaQuery`.
2. Укажите корневую схему.
3. Сформируйте путь к колонке корневой схемы для добавления колонки в запрос.
4. Создайте экземпляры фильтров.
5. Добавьте фильтры в запрос.
6. Отфильтруйте результаты запроса.

Сформировать пути к колонкам относительно корневой схемы

Основой механизма построения запроса `EntitySchemaQuery` является корневая схема. **Корневая схема** — таблица базы данных, относительно которой строятся пути к колонкам в запросе, в том числе к колонкам присоединяемых таблиц. Для использования в запросе колонки из произвольной таблицы необходимо корректно задать путь к этой колонке.

Сформировать путь к колонке по прямым связям

Шаблон формирования пути к колонке по **прямым связям**
`ИмяСправочнойКолонки.ИмяКолонкиСхемыИзСправочнойСхемы`.

Например, есть корневая схема `[City]` из справочной колонкой `[Country]`, которая через колонку `Id` связана со справочной схемой `[Country]`.

Путь к колонке с наименованием страны, которой принадлежит город по прямым связям `Country.Name`. Здесь:

- `Country` — имя справочной колонки корневой схемы `[City]` (ссылается на схему `[Country]`).
- `Name` — имя колонки из справочной схемы `[Country]`.

Сформировать путь к колонке по обратным связям

Шаблон формирования пути к колонке по **обратным связям**
`[ИмяПрисоединяемойСхемы:ИмяКолонкиДляСвязиПрисоединяемойСхемы:ИмяКолонкиДляСвязиТекущейСхемы].ИмяКолонкиПрисоединяемойСхемы`
.

Путь к колонке с именем контакта, который добавил город по обратным связям `[Contact:Id:CreatedBy].Name`. Здесь:

- `Contact` — имя присоединяемой схемы.

© 2022 Terrasoft. Все права защищены.

- `Id` — имя колонки схемы `[Contact]` для установки связи присоединяемой схемы.
- `CreatedBy` — имя справочной колонки схемы `[City]` для установки связи текущей схемы.
- `Name` — значение справочной колонки схемы `[City]`.

Если в качестве колонки для связи у текущей схемы выступает колонка `[Id]`, то ее можно не указывать: `[ИмяПрисоединяемойСхемы:ИмяКолонкиДляСвязиПрисоединяемойСхемы].ИмяКолонкиКорневойСхемы`. Например, `[Contact:City].Name`.

Добавить колонки в запрос

Колонка запроса `EntitySchemaQuery` представляет собой экземпляр класса `Terrasoft.EntityQueryColumn`. В свойствах экземпляра колонки необходимо указать основные **характеристики**:

- Заголовок.
- Значение для отображения.
- Признаки использования.
- Порядок и позицию сортировки.

Для добавления колонок в запрос предназначен метод `addColumn()`, который возвращает экземпляр добавленной в запрос колонки. Имя колонки относительно корневой схемы в методах `addColumn()` формируется согласно правилам, описанным в статье Сформировать пути к колонкам корневой схемы. Разновидности метода `addColumn()` позволяют добавлять в запрос колонки с различными параметрами и представлены в таблице ниже.

Разновидности метода `addColumn()`

Тип колонки	Метод
Колонка по заданному пути относительно корневой схемы	`addColumn(column, columnAlias)`
Экземпляр класса колонки запроса	
Колонка — параметр	`addParameterColumn(paramValue, paramDataType, columnAlias)`
Колонка — функция	`addFunctionColumn(columnPath, functionType, columnAlias)`
Колонка — агрегирующая функция	`addAggregationSchemaColumnFunctionColumn(columnPath, aggregationType, columnAlias)`

Получить результат запроса

Результат выполнения запроса `EntitySchemaQuery` — коллекция сущностей Creatio. Каждый экземпляр коллекции — строка набора данных, возвращаемого запросом.

Способы получения результатов запроса:

© 2022 Terrasoft. Все права защищены.

- Вызов метода `getEntity()` для получения конкретной строки набора данных по заданному первичному ключу.
- Вызов метода `getEntityCollection()` для получения всего результирующего набора данных.

Управлять фильтрами в запросе

Фильтр — набор условий, применяемых при отображении данных запроса. В терминах SQL фильтр представляет собой отдельный предикат (условие) оператора `WHERE`.

Для создания простого фильтра в `EntitySchemaQuery` используется метод `createFilter()`, который возвращает созданный объект фильтра `Terrasoft.CompareFilter`. В `EntitySchemaQuery` реализованы методы создания фильтров специального вида.

Экземпляр `EntitySchemaQuery` имеет свойство `filters`, которое представляет собой коллекцию фильтров данного запроса (экземпляр класса `Terrasoft.FilterGroup`). Экземпляр класса `Terrasoft.FilterGroup` представляет собой коллекцию элементов `Terrasoft.BaseFilter`.

Алгоритм добавления фильтра в запрос:

- Создайте экземпляр фильтра для данного запроса (метод `createFilter()`, методы создания фильтров специального вида).
- Добавьте созданный экземпляр фильтра в коллекцию фильтров запроса (метод `add()` коллекции).

По умолчанию фильтры, добавляемые в коллекцию `filters`, объединяются между собой логической операцией `AND`. Свойство `logicalOperation` коллекции `filters` позволяет указать логическую операцию, которой необходимо объединять фильтры. Свойство `logicalOperation` принимает значения перечисления `Terrasoft.core.enums.LogicalOperatorType` (`AND` — И, `OR` — ИЛИ).

В запросах `EntitySchemaQuery` реализована возможность управления фильтрами, участвующими в построении результирующего набора данных. Каждый элемент коллекции `filters` имеет свойство `isEnabled`, которое определяет, участвует ли данный элемент в построении результирующего запроса (`true` — участвует, `false` — не участвует). Аналогичное свойство `isEnabled` также определено для коллекции `filters`. Установив это свойство в `false`, можно отключить фильтрацию для запроса, при этом коллекция фильтров запроса останется неизменной. Таким образом, изначально сформировав коллекцию фильтров запроса, в дальнейшем можно использовать различные комбинации, не внося изменений в саму коллекцию.

Формирование путей колонок в фильтрах `EntitySchemaQuery` осуществляется в соответствии с общими правилами формирования путей к колонкам относительно корневой схемы.

Примеры формирования путей к колонкам

Средний

Путь к колонке корневой схемы

© 2022 Terrasoft. Все права защищены.

- Корневая схема: `[Contact]`.
- Колонка с адресом контакта: `Address`.

Пример создания запроса `EntitySchemaQuery`, возвращающего значения этой колонки

```
/* Создаем экземпляр класса EntitySchemaQuery с корневой схемой Contact. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем колонку Address, задаем для нее псевдоним Address. */
esq.addColumn("Address", "Address");
```

Путь к колонке схемы по прямым связям

- Корневая схема: `[Contact]`.
- Колонка с названием контрагента: `Account.Name`.
- Колонка с именем основного контакта у контрагента: `Account.PrimaryContact.Name`.

Пример создания запроса `EntitySchemaQuery`, возвращающего значения этих колонок

```
/* Создаем экземпляр класса EntitySchemaQuery с корневой схемой Contact. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем справочную колонку Account. Затем добавляем колонку Name из схемы Account, на котор
esq.addColumn("Account.Name", "AccountName");
/* Добавляем справочную колонку Account. Затем добавляем справочную колонку PrimaryContact из сх
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
```

Путь к колонке схемы по обратным связям

- Корневая схема: `[Contact]`.
- Колонка с именем контакта, который добавил город: `[Contact:Id:CreatedBy].Name`.

Пример создания запроса `EntitySchemaQuery`, возвращающего значения этой колонки

```
/* Создаем экземпляр класса EntitySchemaQuery с корневой схемой Contact. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Присоединяем к корневой схеме еще одну схему Contact по колонке Owner и выбираем из нее колон
```

© 2022 Terrasoft. Все права защищены.

```
esq.addColumn("[Contact:Id:Owner].Name", "OwnerName");
/* К справочной колонке Account присоединяем схему Contact по колонке PrimaryContact и выбираем
esq.addColumn("Account.[Contact:Id:PrimaryContact].Name", "PrimaryContactName");
```

Примеры добавления колонок в запрос

Средний

Колонка из корневой схемы

Пример. Добавить в коллекцию колонок запроса колонку из корневой схемы.

Пример добавления в коллекцию колонок запроса колонки из корневой схемы

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addColumn("DurationInMinutes", "ActivityDuration");
```

Аггрегирующая колонка

Пример 1. Добавить в коллекцию колонок запроса аггрегирующую колонку с типом аггрегации `SUM`, применяющимся ко всем записям таблицы.

Пример добавления в коллекцию колонок запроса аггрегирующей колонки

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addAggregationSchemaColumn("DurationInMinutes", Terrasoft.AggregationType.SUM, "ActivitiesDu
```

Пример 2. Добавить в коллекцию колонок запроса аггрегирующую колонку с типом аггрегации `COUNT`, применяющимся к уникальным записям таблицы.

Пример добавления в коллекцию колонок запроса аггрегирующей колонки

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
```

© 2022 Terrasoft. Все права защищены.

```
    rootSchemaName: "Activity"
});
esq.addAggregationSchemaColumn("DurationInMinutes", Terrasoft.AggregationType.COUNT, "UniqueActi
```

Колонка-параметр

Пример. Добавить в коллекцию колонок запроса колонку-параметр с типом данных `TEXT`.

Пример добавления в коллекцию колонок запроса колонки-параметра

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addParameterColumn("DurationInMinutes", Terrasoft.DataValueType.TEXT, "DurationColumnName");
```

Колонка-функция

Пример 1. Добавить в коллекцию колонок запроса колонку-функцию с типом функции `LENGTH` (размер значения в байтах).

Пример добавления в коллекцию колонок запроса колонки-функции

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addFunctionColumn("Photo.Data", Terrasoft.FunctionType.LENGTH, "PhotoLength");
```

Пример 2. Добавить в коллекцию колонок запроса колонку-функцию с типом `DATE_PART` (часть даты). В качестве значения используется день недели.

Пример добавления в коллекцию колонок запроса колонки-функции

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addDatePartFunctionColumn("StartDate", Terrasoft.DatePartType.WEEK_DAY, "StartDay");
```

© 2022 Terrasoft. Все права защищены.

Пример 3. Добавить в коллекцию колонок запроса колонку-функцию с типом `MACROS`, который не требует параметризации — `PRIMARY_DISPLAY_COLUMN` (первичная колонка для отображения).

Пример добавления в коллекцию колонок запроса колонки-функции

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addMacrosColumn(Terrasoft.QueryMacrosType.PRIMARY_DISPLAY_COLUMN, "PrimaryDisplayColumnValue
```

Примеры получения результатов запроса

Средний

Строка набора данных по заданному первичному ключу

Пример. Получить конкретную строку набора данных по заданному первичному ключу.

Пример получения конкретной строки набора данных

```
/* Получаем Id объекта карточки. */
var recordId = this.get("Id");
/* Создаем экземпляр класса Terrasoft.EntitySchemaQuery с корневой схемой Contact. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем колонку с именем основного контакта контрагента. */
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
/* Получаем одну запись из выборки по Id объекта карточки и отображаем ее в информационном окне.
esq.getEntity(recordId, function(result) {
    if (!result.success) {
        /* Например, обработка/логирование ошибки. */
        this.showInformationDialog("Ошибка запроса данных");
        return;
    }
    this.showInformationDialog(result.entity.get("PrimaryContactName"));
}, this);
```

© 2022 Terrasoft. Все права защищены.

На заметку. При получении справочных колонок метод `this.get()` возвращает объект, а не идентификатор записи в базе данных. Чтобы получить идентификатор, необходимо использовать свойство `value`, например, `this.get('Account').value`.

Результирующий набор данных

Пример. Получить весь результирующий набор данных.

Пример получения всего набора данных

```
var message = "";
/* Создаем экземпляр класса Terrasoft.EntitySchemaQuery с корневой схемой Contact. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем колонку с названием контрагента, который относится к данному контакту. */
esq.addColumn("Account.Name", "AccountName");
/* Добавляем колонку с именем основного контакта контрагента. */
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
/* Получаем всю коллекцию записей и отображаем ее в информационном окне. */
esq.getEntityCollection(function (result) {
    if (!result.success) {
        /* Например, обработка/логирование ошибки. */
        this.showInformationDialog("Ошибка запроса данных");
        return;
    }
    result.collection.each(function (item) {
        message += "Account name: " + item.get("AccountName") +
        " - primary contact name: " + item.get("PrimaryContactName") + "\n";
    });
    this.showInformationDialog(message);
}, this);
```

Примеры управления фильтрами в запросе

Средний

Пример управления фильтрами в запросе

© 2022 Terrasoft. Все права защищены.

```
/* Создание экземпляра запроса с корневой схемой Contact. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
esq.addColumn("Name");
esq.addColumn("Country.Name", "CountryName");

/* Создание экземпляра первого фильтра. */
var esqFirstFilter = esq.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL, "Countr

/* Создание экземпляра второго фильтра. */
var esqSecondFilter = esq.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL, "Count

/* Фильтры в коллекции фильтров запроса будут объединяться логическим оператором OR. */
esq.filters.logicalOperation = Terrasoft.LogicalOperatorType.OR;

/* Добавление созданных фильтров в коллекцию запроса. */
esq.filters.add("esqFirstFilter", esqFirstFilter);
esq.filters.add("esqSecondFilter", esqSecondFilter);

/* В данную коллекцию попадут объекты - результаты запроса, отфильтрованные по двум фильтрам. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            // Обработка элементов коллекции.
        });
    }
}, this);

/* Для второго фильтра указывается, что он не будет участвовать в построении результирующего зап
esqSecondFilter.isEnabled = false;

/* В данную коллекцию попадут объекты - результаты запроса, отфильтрованные только по первому фи
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            /* Обработка элементов коллекции. */
        });
    }
}, this);
```

Пример использования других методов создания фильтров

```
/* Создание экземпляра запроса с корневой схемой Contact. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
```

© 2022 Terrasoft. Все права защищены.

```
});
esq.addColumn("Name");
esq.addColumn("Country.Name", "CountryName");

/* Выбираем все контакты, в которых не указана страна. */
var esqFirstFilter = esq.createColumnIsNullFilter("Country");

/* Выбираем все контакты, даты рождения которых находятся в промежутке между 1.01.1970 и 1.01.19
var dateFrom = new Date(1970, 0, 1, 0, 0, 0, 0);
var dateTo = new Date(1980, 0, 1, 0, 0, 0, 0);
var esqSecondFilter = esq.createColumnBetweenFilterWithParameters("BirthDate", dateFrom, dateTo)

/* Добавление созданных фильтров в коллекцию запроса. */
esq.filters.add("esqFirstFilter", esqFirstFilter);
esq.filters.add("esqSecondFilter", esqSecondFilter);

/* В данную коллекцию попадут объекты - результаты запроса, отфильтрованные по двум фильтрам. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            /* Обработка элементов коллекции. */
        });
    }
}, this);
```

Класс EntitySchemaQuery JS

Средний

Класс `EntitySchemaQuery` предназначен для построения запросов на выборку из базы данных.

Методы

`abortQuery()`

Прерывает выполнение запроса.

`addAggregationSchemaColumn(columnPath, aggregationType, [columnAlias], aggregationEvalType)`

Создает и добавляет в коллекцию колонок запроса экземпляр функциональной колонки `Terrasoft.FunctionQueryColumn` с заданным типом `AGGREGATION`.

Параметры

`{String} columnPath`	Путь колонки для добавления (указывается относительно

© 2022 Terrasoft. Все права защищены.

	`rootSchema`).

`{Terrasoft.AggregationType} aggregationType`	Тип используемой агрегирующей функции.
	
	Возможные значения (`Terrasoft.AggregationType`)
	
	`AVG`
	
	Среднее значение всех элементов.
	
	`COUNT`
	
	Количество всех элементов.
	
	`MAX`
	
	Максимальное значение среди всех элементов.
	
	`MIN`
	
	Минимальное значение среди всех элементов.
	
	`NONE`
	
	Тип агрегирующей функции не определен.
	
	`SUM`
	
	Сумма значений всех элементов.

`{String} columnAlias`	Псевдоним колонки (необязательный параметр).

`{Terrasoft.AggregationEvalType} aggregationEvalType`	Область применения агрегирующей функции.
	
	Возможные значения (`Terrasoft.AggregationEvalType`)
	
	`NONE`
	
	Область применения агрегирующей функции не определена.
	
	`ALL`
	
	Применяется для всех элементов.

© 2022 Terrasoft. Все права защищены.

	`DISTINCT` Применяется к уникальным значениям.

`addColumn(column, [columnAlias], [config])`

Создает и добавляет в коллекцию колонок запроса экземпляр колонки `Terrasoft.EntityQueryColumn`.

Параметры

`{String/Terrasoft.BaseQueryColumn} column`	Путь колонки для добавления (указывается относительно `rootSchema`) или экземпляр колонки запроса `Terrasoft.BaseQueryColumn`.
`{String} columnAlias`	Псевдоним колонки (необязательный параметр).
`{Object} config`	Конфигурационный объект колонки запроса.

`addDatePartFunctionColumn(columnPath, datePartType, [columnAlias])`

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции `Terrasoft.FunctionQueryColumn` с типом `DATE_PART`.

Параметры

© 2022 Terrasoft. Все права защищены.

`{String} columnPath`	Путь колонки для добавления (указывается относительно `rootSchema`).

`{Terrasoft.DatePartType} datePartType`	Часть даты, используемая в качестве значения
	
	Возможные значения (`Terrasoft.DatePartType`)
	
	`NONE`
	Пустое значение.
	
	`DAY`
	День.
	
	`WEEK`
	Неделя.
	
	`MONTH`
	Месяц.
	
	`YEAR`
	Год.
	
	`WEEK_DAY`
	День недели.
	
	`HOUR`
	Час.
	
	`HOUR_MINUTE`
	Минута.

`{String} columnAlias`	Псевдоним колонки (необязательный параметр).

`addDatePeriodMacrosColumn(macrosType, [macrosValue], [columnAlias])`

© 2022 Terrasoft. Все права защищены.

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции `Terrasoft.FunctionQueryColumn` с типом `MACROS`, который требует параметризации. Например, следующие N дней, 3-й квартал года, и т. д.

Параметры

`{Terrasoft.QueryMacrosType} macrosType`	Тип макроса колонки.
`{Number/Date} macrosValue`	Вспомогательная переменная для макроса (необязательный параметр).
`{String} columnAlias`	Псевдоним колонки (необязательный параметр).

`addFunctionColumn(columnPath, functionType, [columnAlias])`

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции `Terrasoft.FunctionQueryColumn`.

Параметры

© 2022 Terrasoft. Все права защищены.

`{String} columnPath`	Путь колонки для добавления (указывается относительно `rootSchema`).

`{Terrasoft.FunctionType} functionType`	Тип функции.
	
	Возможные значения (`Terrasoft.FunctionType`)
	
	`NONE`
	Тип функционального выражения не определен.
	
	`MACROS`
	Подстановка по макросу.
	
	`AGGREGATION`
	Агрегирующая функция.
	
	`DATE_PART`
	Часть даты.
	
	`LENGTH`
	Размер значения в байтах. Используется для бинарных данных.

`{String} columnAlias`	Псевдоним колонки (необязательный параметр).

`addMacrosColumn(macrosType, [columnAlias])`

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции `Terrasoft.FunctionQueryColumn` с типом `MACROS`, который не требует параметризации (например, текущий месяц, текущий пользователь, первичная колонка, и т. д.).

Параметры

`{Terrasoft.QueryMacrosType} macrosType`	Тип макроса колонки. Возможные значения (`Terrasoft.QueryMacrosType`) `NONE`

© 2022 Terrasoft. Все права защищены.

Тип макроса не определен.

`CURRENT_USER`

Текущий пользователь.

`CURRENT_USER_CONTACT`

Контакт текущего пользователя.

`YESTERDAY`

Вчера.

`TODAY`

Сегодня.

`TOMORROW`

Завтра.

`PREVIOUS_WEEK`

Предыдущая неделя.

`CURRENT_WEEK`

Текущая неделя.

`NEXT_WEEK`

Следующая неделя.

`PREVIOUS_MONTH`

Предыдущий месяц.

`CURRENT_MONTH`

Текущий месяц.

© 2022 Terrasoft. Все права защищены.

`NEXT_MONTH`

Следующий месяц.

`PREVIOUS_QUARTER`

Предыдущий квартал.

`CURRENT_QUARTER`

Текущий квартал.

`NEXT_QUARTER`

Следующий квартал.

`PREVIOUS_HALF_YEAR`

Предыдущее полугодие.

`CURRENT_HALF_YEAR`

Текущее полугодие.

`NEXT_HALF_YEAR`

Следующее полугодие.

`PREVIOUS_YEAR`

Предыдущий год.

`CURRENT_YEAR`

Текущий год.

`PREVIOUS_HOUR`

Предыдущий час.

`CURRENT_HOUR`

Текущий час.

© 2022 Terrasoft. Все права защищены.

	`NEXT_HOUR`
	Следующий час.
	
	`NEXT_YEAR`
	Следующий год.
	
	`NEXT_N_DAYS`
	Следующие N дней. Требует параметризации.
	
	`PREVIOUS_N_DAYS`
	Предыдущие N дней. Требует параметризации.
	
	`NEXT_N_HOURS`
	Следующие N часов. Требует параметризации.
	
	`PREVIOUS_N_HOURS`
	Предыдущие N часов. Требует параметризации.
	
	`PRIMARY_COLUMN`
	Первичная колонка.
	
	`PRIMARY_DISPLAY_COLUMN`
	Первичная колонка для отображения.
	
	`PRIMARY_IMAGE_COLUMN`
	Первичная колонка для отображения изображения.

`{String} columnAlias`	Псевдоним колонки (необязательный параметр).

`addParameterColumn(paramValue, paramDataType, [columnAlias])`

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-параметра `Terrasoft.ParameterQueryColumn`.

© 2022 Terrasoft. Все права защищены.

Параметры

{Mixed} paramValue	Значение параметра. Значение должно соответствовать типу данных.
{Terrasoft.DataValueType} paramDataType	Тип данных параметра.
{String} columnAlias	Псевдоним колонки (необязательный параметр).

---

createBetweenFilter(leftExpression, rightLessExpression, rightGreaterExpression)

Создает экземпляр `Between`-фильтра.

Параметры

{Terrasoft.BaseExpression} leftExpression	Выражение, проверяемое в фильтре.
{Terrasoft.BaseExpression} rightLessExpression	Начальное выражение диапазона фильтрации.
{Terrasoft.BaseExpression} rightGreaterExpression	Конечное выражение диапазона фильтрации.

---

createColumnBetweenFilterWithParameters(columnPath, lessParamValue, greaterParamValue, paramData

Создает экземпляр `Between`-фильтра для проверки попадания колонки в заданный диапазон.

Параметры

{String} columnPath	Путь к проверяемой колонке относительно корневой схемы `rootSchema`.
{Mixed} lessParamValue	Начальное значение фильтра.
{Mixed} greaterParamValue	Конечное значение фильтра.
{Terrasoft.DataValueType} paramDataType	Тип данных параметра.

© 2022 Terrasoft. Все права защищены.

`createColumnFilterWithParameter(comparisonType, columnPath, paramValue, paramDataType)`

Создает экземпляр `Compare`-фильтра для сравнения колонки с заданным значением.

Параметры

`{Terrasoft.ComparisonType} comparisonType`	Тип операции сравнения.
`{String} columnPath`	Путь к проверяемой колонке относительно корневой схемы `rootSchema`.
`{Mixed} paramValue`	Значение параметра.
`{Terrasoft.DataValueType} paramDataType`	Тип данных параметра.

`createColumnInFilterWithParameters(columnPath, paramValues, paramDataType)`

Создает экземпляр `In`-фильтра для проверки совпадения значения заданной колонки со значением одного из параметров.

Параметры

`{String} columnPath`	Путь к проверяемой колонке относительно корневой схемы `rootSchema`.
`{Array} paramValues`	Массив значений параметров.
`{Terrasoft.DataValueType} paramDataType`	Тип данных параметра.

`createColumnIsNotNullFilter(columnPath)`

Создает экземпляр `IsNull`-фильтра для проверки заданной колонки.

Параметры

`{String} columnPath`	Путь к проверяемой колонке относительно корневой схемы `rootSchema`.

`createColumnIsNullFilter(columnPath)`

© 2022 Terrasoft. Все права защищены.

Создает экземпляр `IsNull`-фильтра для проверки заданной колонки.

Параметры

`{String} columnPath`	Путь к проверяемой колонке относительно корневой схемы `rootSchema`.

`createCompareFilter(comparisonType, leftExpression, rightExpression)`

Создает экземпляр `Compare`-фильтра.

Параметры

`{Terrasoft.ComparisonType} comparisonType`	Тип операции сравнения.
`{Terrasoft.BaseExpression} leftExpression`	Выражение, проверяемое в фильтре.
`{Terrasoft.BaseExpression} rightExpression`	Выражение фильтрации.

`createExistsFilter(columnPath)`

Создает экземпляр `Exists`-фильтра для сравнения типа `[Существует по заданному условию]` и устанавливает в качестве проверяемого значения выражение колонки, расположенной по заданному пути.

Параметры

`{String} columnPath`	Путь к колонке, для выражения которой строится фильтр.

`createFilter(comparisonType, leftColumnPath, rightColumnPath)`

Создает экземпляр фильтра класса `Terrasoft.CompareFilter` для сравнения значений двух колонок.

Параметры

© 2022 Terrasoft. Все права защищены.

{Terrasoft.ComparisonType} comparisonType	Тип операции сравнения.
{String} leftColumnPath	Путь к проверяемой колонке относительно корневой схемы `rootSchema`.
{String} rightColumnPath	Путь к колонке-фильтру относительно корневой схемы `rootSchema`.

`createFilterGroup()`

Создает экземпляр группы фильтров.

`createInFilter(leftExpression, rightExpressions)`

Создает экземпляр `In`-фильтра.

Параметры

{Terrasoft.BaseExpression} leftExpression	Выражение, проверяемое в фильтре.
{Terrasoft.BaseExpression[]} rightExpressions	Массив выражений, которые будут сравниваться с `leftExpression`.

`createIsNotNullFilter(leftExpression)`

Создает экземпляр `IsNull`-фильтра.

Параметры

{Terrasoft.BaseExpression} leftExpression	Выражение, которое проверяется по условию `IS NOT NULL`.

`createIsNullFilter(leftExpression)`

Создает экземпляр `IsNull`-фильтра.

Параметры

© 2022 Terrasoft. Все права защищены.

{Terrasoft.BaseExpression} leftExpression	Выражение, которое проверяется по условию `IS NULL`.

`createNotExistsFilter(columnPath)`

Создает экземпляр `Exists`-фильтра для сравнения типа `[Не существует по заданному условию]` и устанавливает в качестве проверяемого значения выражение колонки, расположенной по заданному пути.

Параметры

{String} columnPath	Путь к колонке, для выражения которой строится фильтр.

`createPrimaryDisplayColumnFilterWithParameter(comparisonType, paramValue, paramDataType)`

Создает объект фильтра для сравнения первичной колонки для отображения со значением параметра.

Параметры

{Terrasoft.ComparisonType} comparisonType	Тип сравнения.
{Mixed} paramValue	Значение параметра.
{Terrasoft.DataValueType} paramDataType	Тип данных параметра.

`destroy()`

Удаляет экземпляр объекта. Если экземпляр уже удален, запишет в консоль сообщение об ошибке. Вызывает виртуальный метод `onDestroy` для переопределения в подклассах.

`enablePrimaryColumnFilter(primaryColumnValue)`

Включает фильтрацию по первичному ключу.

Параметры

{String/Number} primaryColumnValue	Значение первичного ключа.

© 2022 Terrasoft. Все права защищены.

`error(message)`

Записывает сообщение об ошибке в журнал сообщений.

Параметры

`{String} message`	Сообщение об ошибке, которое будет записано в журнал сообщений.

`execute(callback, scope)`

Запрос на выполнение запроса на сервере.

Параметры

`{Function} callback`	Функция, которая будет вызвана при получении ответа от сервера.
`{Object} scope`	Контекст, в котором будет вызвана функция `callback`.

`{Object} getDefSerializationInfo()`

Возвращает объект с дополнительной информацией для сериализации.

`getEntity(primaryColumnValue, callback, scope)`

Возвращает экземпляр сущности по заданному первичному ключу `primaryColumnValue`. После получения данных вызывает функцию `callback` в контексте `scope`.

Параметры

`{String/Number} primaryColumnValue`	Значение первичного ключа.
`{Function} callback`	Функция, которая будет вызвана при получении ответа от сервера.
`{Object} scope`	Контекст, в котором будет вызвана функция `callback`.

`getEntityCollection(callback, scope)`

Возвращает коллекцию экземпляров сущности, представляющих результаты выполнения текущего запроса. После получения данных вызывает функцию `callback` в контексте `scope`.

© 2022 Terrasoft. Все права защищены.

Параметры

`{Function} callback`	Функция, которая будет вызвана при получении ответа от сервера.
`{Object} scope`	Контекст, в котором будет вызвана функция `callback`.

`{Object} getTypeInfo()`

Возвращает информацию о типе элемента.

`log(message, [type])`

Записывает сообщение в журнал сообщений.

Параметры

`{String|Object} message`	Сообщение, будет записано в журнал сообщений.
`{Terrasoft.LogMessageType} type`	Тип журнала сообщений `callback` (необязательный параметр). По умолчанию `console.log`. Возможные значения заданы перечислением `Terrasoft.core.enums.LogMessageType`.

`onDestroy()`

Удаляет все подписки на события и уничтожает объект.

`serialize(serializationInfo)`

Сериализует объект в json.

Параметры

`{String} serializationInfo`	Результат в json.

`setSerializableProperty(serializableObject, propertyName)`

Задает имя свойства объекту, если он не пустой или не является функцией.

Параметры

© 2022 Terrasoft. Все права защищены.

`{Object} serializableObject`	Сериализуемый объект.
`{String} propertyName`	Имя свойства.

`warning(message)`

Записывает предупреждающее сообщение в журнал сообщений.

Параметры

`{String} message`	Сообщение, будет записано в журнал сообщений.

Класс DataManager JS

Средний

Класс DataManager JS

Класс `DataManager` — синглтон, который доступен через глобальный объект `Terrasoft`. Данный класс предоставляет хранилище `dataStore`. В хранилище может быть загружено содержимое одной или нескольких таблиц базы данных.

```
dataStore: {
    /* Коллекция данных типа DataManagerItem схемы SysModule. */
    SysModule: sysModuleCollection,
 /* Коллекция данных типа DataManagerItem схемы SysModuleEntity. */
    SysModuleEntity: sysModuleEntityCollection
}
```

Каждая запись коллекции представляет запись из соответствующей таблицы в базе данных.

Свойства

`{Object} dataStore`

Хранилище коллекций данных.

`{String} itemClassName`

Имя класса записи. Содержит значение `Terrasoft.DataManagerItem`.

© 2022 Terrasoft. Все права защищены.

Методы

`{Terrasoft.Collection} select(config, callback, scope)`

Если `dataStore` не содержит коллекцию данных с именем `config.entitySchemaName`, то метод формирует и выполняет запрос в базу данных, затем возвращает полученные данные, иначе возвращает коллекцию данных из `dataStore`.

Параметры

`{Object} config`	Конфигурационный объект.
	
	Свойства конфигурационного объекта
	
	`{String} entitySchemaName`
	Имя схемы.
	
	`{Terrasoft.FilterGroup} filters`
	Условия.

`{Function} callback`	Функция обратного вызова.
`{Object} scope`	Контекст выполнения функции обратного вызова.

`{Terrasoft.DataManagerItem} createItem(config, callback, scope)`

Создает новую запись типа `config.entitySchemaName` со значениями колонок `config.columnValues`.

Параметры

© 2022 Terrasoft. Все права защищены.

`{Object} config`	Конфигурационный объект.
	
	Свойства конфигурационного объекта
	
	`{String} entitySchemaName`
	
	Имя схемы.
	
	`{Object} columnValues`
	
	Значения колонок записи.

`{Function} callback`	Функция обратного вызова.
`{Object} scope`	Контекст выполнения функции обратного вызова.

`{Terrasoft.DataManagerItem} addItem(item)`

Добавляет запись `item` в коллекцию данных схемы.

Параметры

`{Terrasoft.DataManagerItem} item`	Добавляемая запись.

`{Terrasoft.DataManagerItem} findItem(entitySchemaName, id)`

Возвращает запись из коллекции данных схемы с именем `entitySchemaName` и с идентификатором `id`.

Параметры

`{String} entitySchemaName`	Название коллекции данных.
`{String} id`	Идентификатор записи.

`{Terrasoft.DataManagerItem} remove(item)`

Устанавливает признак `isDeleted` для записи `item`, при фиксации изменений запись будет удалена из базы данных.

Параметры

© 2022 Terrasoft. Все права защищены.

{Terrasoft.DataManagerItem} item	Удаляемая запись.

`removeItem(item)`

Удаляет запись из коллекции данных схемы.

Параметры

{Terrasoft.DataManagerItem} item	Удаляемая запись.

`{Terrasoft.DataManagerItem} update(config, callback, scope)`

Обновляет запись со значением первичной колонки `config.primaryColumnValue` значениями из `config.columnValues`.

Параметры

{Object} config	Конфигурационный объект. Свойства конфигурационного объекта {String} entitySchemaName Имя схемы. {String} primaryColumnValue Значение первичной колонки. {Mixed} columnValues Значения колонок.
{Function} callback	Функция обратного вызова.
{Object} scope	Контекст выполнения функции обратного вызова.

`{Terrasoft.DataManagerItem} discardItem(item)`

Отменяет изменения для записи `item`, сделанные в текущей сессии работы с `DataManager`.

© 2022 Terrasoft. Все права защищены.

Параметры

`{Terrasoft.DataManagerItem} item`	Запись, по которой будут отменены изменения.

`{Object} save(config, callback, scope)`

Выполняет сохранение в базу данных коллекций данных схем, указанных в `config.entitySchemaNames`.

Параметры

`{Object} config`	Конфигурационный объект. Свойства конфигурационного объекта `{String[]} entitySchemaName` Имя схемы, сохранение которой необходимо выполнить. Если свойство не заполнено, сохранение выполняется для всех схем.
`{Function} callback`	Функция обратного вызова.
`{Object} scope`	Контекст выполнения функции обратного вызова.

Класс DataManagerItem JS

Свойства

`{Terrasoft.BaseViewMode} viewModel`

Объектная проекция записи в базе данных.

Методы

`setColumnValue(columnName, columnValue)`

Устанавливает новое значение `columnValue` для колонки с именем `columnName`.

Параметры

© 2022 Terrasoft. Все права защищены.

{String} columnName	Название колонки.
{String} columnValue	Значение колонки.

`{Mixed} getColumnValue(columnName)`

Возвращает значение колонки с именем `columnName`.

Параметры

{String} columnName	Название колонки.

`{Object} getValues()`

Возвращает значения всех колонок записи.

`remove()`

Устанавливает признак `isDeleted` для записи.

`discard()`

Отменяет изменения для записи, сделанные в текущей сессии работы с `DataManager`.

`{Object} save(callback, scope)`

Фиксирует изменения в базе данных.

Параметры

{Function} callback	Функция обратного вызова.
{Object} scope	Контекст выполнения функции обратного вызова.

`{Boolean} getIsNew()`

Возвращает признак того, что запись новая.

`{Boolean} getIsChanged()`

Возвращает признак того, что запись была изменена.

© 2022 Terrasoft. Все права защищены.

Примеры использования

Получение записей из таблицы [*Contact*]

```
/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaName: "Contact",
    /* Убирать дубли в результирующем наборе данных. */
    isDistinct: true
};
/* Получение данных. */
Terrasoft.DataManager.select(config, function (collection) {
    /* Сохранение полученных записей в локальное хранилище. */
    collection.each(function (item) {
        Terrasoft.DataManager.addItem(item);
    });
}, this);
```

Добавление новой записи в объект `DataManager`

```
/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaName: "Contact",
    /* Значения колонок. */
    columnValues: {
        Id: "00000000-0000-0000-0000-000000000001",
        Name: "Name1"
    }
};
/* Создание новой записи. */
Terrasoft.DataManager.createItem(config, function (item) {
    Terrasoft.DataManager.addItem(item);
}, this);
```

Получение записи и изменение значения колонки

```
/* Получение записи. */
var item = Terrasoft.DataManager.findItem("Contact",
     "00000000-0000-0000-0000-000000000001");
/* Установка нового значения "Name2" колонке Name. */
item.setColumnValue("Name", "Name2");
```

© 2022 Terrasoft. Все права защищены.

Удаление записи из `DataManager`

```
/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaName: "Contact",
    /* Значение первичной колонки. */
    primaryColumnValue: "00000000-0000-0000-0000-000000000001"
};
/* Устанавливает признак isDeleted для записи item. */
Terrasoft.DataManager.remove(config, function () {
}, this);
```

Отмена изменений, сделанных в текущей сессии работы с `DataManager`

```
/* Получение записи. */
var item = Terrasoft.DataManager.findItem("Contact",
     "00000000-0000-0000-0000-000000000001");
/* Отмена изменений для записи. */
Terrasoft.DataManager.discardItem(item);
```

Фиксация изменений в базе данных

```
/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaNames: ["Contact"]
};
/* Сохранение изменений в базу данных. */
Terrasoft.DataManager.save(config, function () {
}, this);
```

Понятие элемента управления

Средний

Элементы управления — это объекты, используемые для организации взаимодействия между пользователем и приложением. Это, например, кнопки, поля ввода, элементы выбора и т. д.

Все элементы управления Creatio наследуются от класса `Terrasoft.controls.Component`. Описание классов, которые реализуют компоненты, содержится в документации Библиотека JS классов.

В соответствии с принципом открытости-закрытости, невозможно добавить пользовательскую логику

© 2022 Terrasoft. Все права защищены.

непосредственно в существующий элемент управления.

Алгоритм реализации пользовательской логики элемента управления:

1. Создайте клиентский модуль.
2. В модуле объявите класс, наследующий функциональность существующего элемента управления.
3. В классе-наследнике реализуйте требуемую функциональность.
4. Добавьте новый элемент в интерфейс приложения.

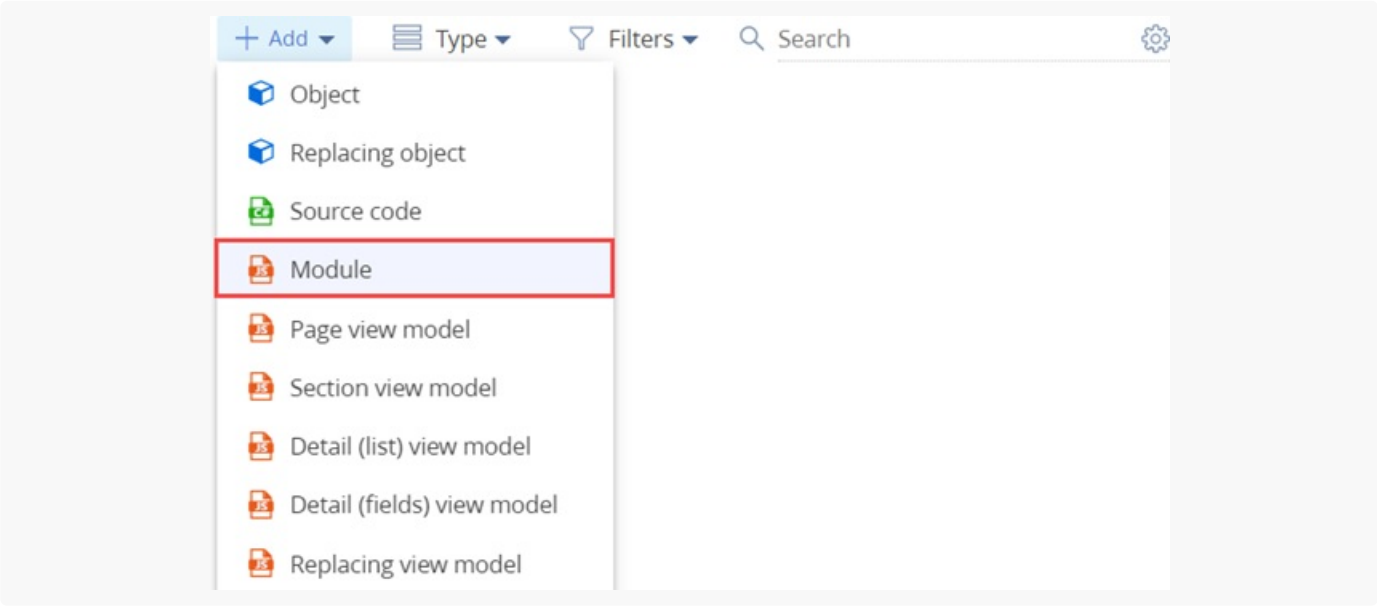
Создать элемент управления

Сложный

Пример. Создать элемент управления, который позволяет вводить только целые числа в заданном диапазоне. При нажатии кнопки Enter проверять введенное значение и выводить соответствующее сообщение, если число выходит за диапазон заданных значений. В качестве родительского использовать элемент управления `Terrasoft.controls.IntegerEdit`.

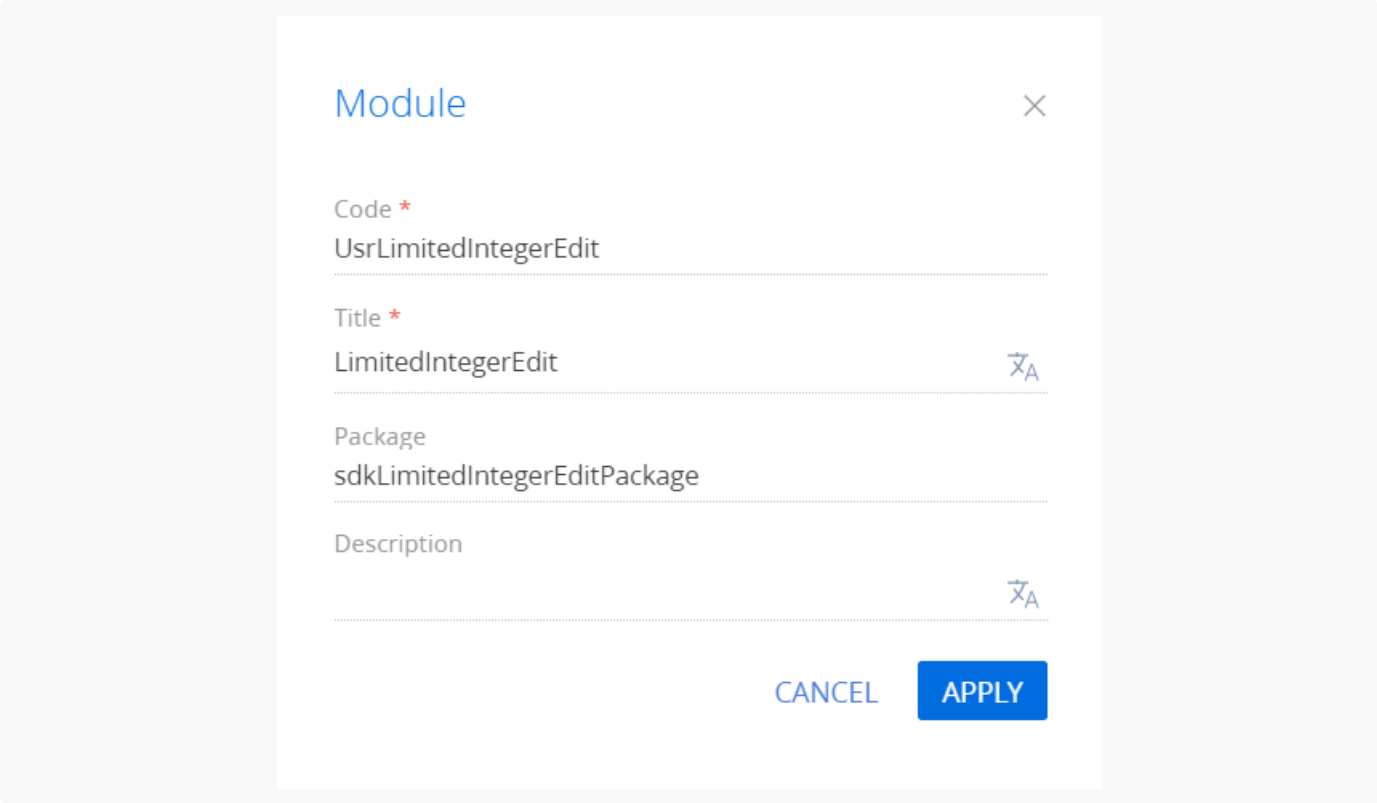
1. Создать модуль

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



3. В дизайнере схем заполните свойства схемы:
   - [*Код*] ([*Code*]) — "`UsrLimitedIntegerEdit`".
   - [*Заголовок*] ([*Title*]) — "`LimitedIntegerEdit`".

© 2022 Terrasoft. Все права защищены.



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

2. Создать класс элемента управления

В дизайнере схем добавьте исходный код.

Исходный код модуля

```
define("UsrLimitedIntegerEdit", [], function () {
    // Объявление класса элемента управления.
    Ext.define("Terrasoft.controls.UsrLimitedIntegerEdit", {
        // Базовый класс.
        extend: "Terrasoft.controls.IntegerEdit",
        // Псевдоним (сокращенное название класса).
        alternateClassName: "Terrasoft.UsrLimitedIntegerEdit",
        // Наименьшее допустимое значение.
        minLimit: -1000,
        // Наибольшее допустимое значение.
        maxLimit: 1000,
        // Метод проверки на вхождение в диапазон допустимых значений.
        isOutOfLimits: function (numericValue) {
            if (numericValue < this.minLimit || numericValue > this.maxLimit) {
                return true;
            }
            return false;
        },
```

© 2022 Terrasoft. Все права защищены.

```
        // Переопределение метода-обработчика события нажатия клавиши Enter.
        onEnterKeyPressed: function () {
            // Вызов базовой функциональности.
            this.callParent(arguments);
            // Получение введенного значения.
            var value = this.getTypedValue();
            // Приведение к числовому типу.
            var numericValue = this.parseNumber(value);
            // Проверка на вхождение в диапазон допустимых значений.
            var outOfLimits = this.isOutOfLimits(numericValue);
            if (outOfLimits) {
                // Формирование предупреждающего сообщения.
                var msg = "Value " + numericValue + " is out of limits [" + this.minLimit + ", "
                // Изменение конфигурационного объекта для показа предупреждающего сообщения.
                this.validationInfo.isValid = false;
                this.validationInfo.invalidMessage = msg;
            }
            else{
                // Изменение конфигурационного объекта для скрытия предупреждающего сообщения.
                this.validationInfo.isValid = true;
                this.validationInfo.invalidMessage ="";
            }
            // Вызов логики отображения предупреждающего сообщения.
            this.setMarkOut();
        },
    });
});
```

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

На заметку. Логику, аналогичную логике метода `onEnterKeyPressed()`, можно использовать в обработчике события потери фокуса `onBlur()`.

Кроме стандартных свойств `extend` и `alternateClassName`, в класс добавлены свойства `minLimit` и `maxLimit`, которые задают диапазон допустимых значений. Для этих свойств указаны значения по умолчанию.

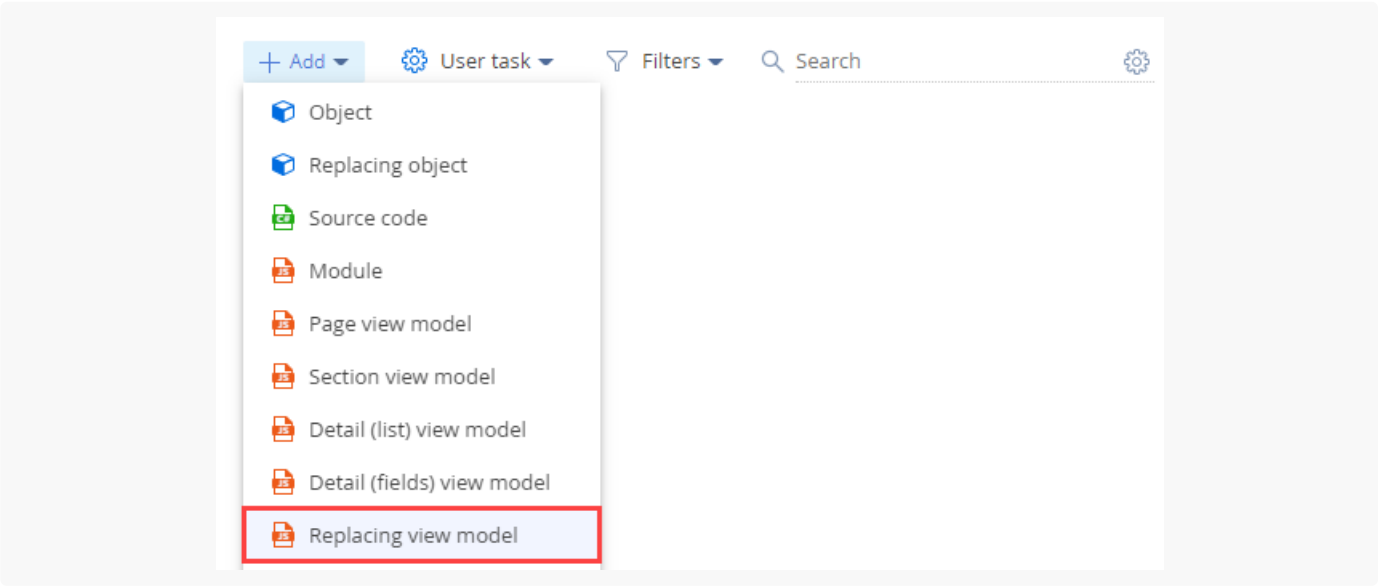
Пользовательская логика элемента управления реализована в переопределенном методе `onEnterKeyPressed`. После вызова базовой логики, в которой выполняется генерация событий изменения значения, введенное значение проверяется на допустимость. Если число недопустимо, то в поле ввода отображается соответствующее предупреждающее сообщение. Для проверки вхождения введенного значения в диапазон допустимых значений предусмотрен метод `isOutOfLimits`.

Несмотря на вывод соответствующего предупреждения, при текущей реализации введенное значение все равно сохраняется и передается в модель представления схемы, в которой будет использован компонент.

© 2022 Terrasoft. Все права защищены.

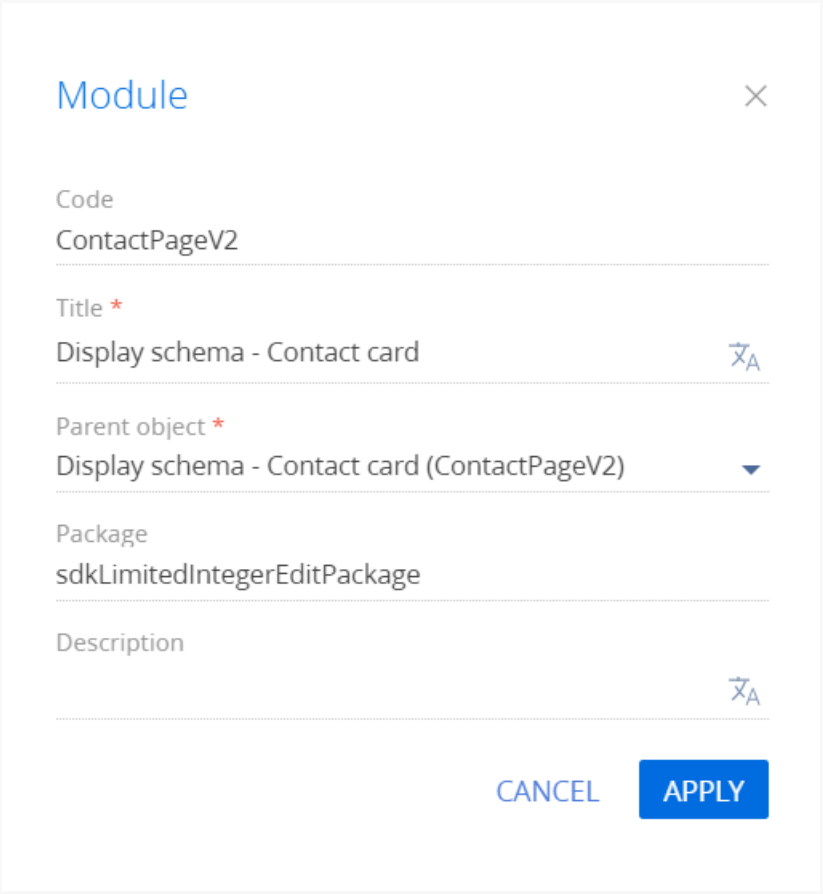
3. Добавить элемент управления в интерфейс приложения

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



3. В дизайнере схем свойству [*Родительский объект*] ([*Parent object*]) задайте значение "`Display schema - Contact card`" пакета [*ContactPageV2*]. После подтверждения выбранного родительского объекта остальные свойства будут заполнены автоматически.

© 2022 Terrasoft. Все права защищены.



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

4. В дизайнере схем добавьте исходный код.

Исходный код модуля

```
// Объявление модуля. Обязательно следует указать как зависимость
// модуль, в котором объявлен класс элемента управления.
define("ContactPageV2", ["UsrLimitedIntegerEdit"],
    function () {
        return {
            attributes: {
                // Атрибут, связываемый со значением в элементе управления.
                "ScoresAttribute": {
                    // Тип данных атрибута – целочисленный.
                    "dataValueType": this.Terrasoft.DataValueType.INTEGER,
                    // Тип атрибута – виртуальная колонка.
                    "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                    // Значение по умолчанию.
                    "value": 0
                }
            },
            diff: /**SCHEMA_DIFF*/[
                {
```

© 2022 Terrasoft. Все права защищены.

```
 "operation": "insert",
 "parentName": "ProfileContainer",
 "propertyName": "items",
 "name": "Scores",
 "values": {
  "contentType": Terrasoft.ContentType.LABEL,
                 "caption": {"bindTo": "Resources.Strings.ScoresCaption"},
  "layout": {
                     "column": 0,
                     "row": 6,
                     "colSpan": 24
                 }
 }
 },
 {

  // Операция добавления компонента на страницу.
  "operation": "insert",
  // Мета-имя родительского контейнера, в который добавляется поле.
  "parentName": "ProfileContainer",
  // Поле добавляется в коллекцию компонентов
  // родительского элемента.
  "propertyName": "items",
  // Имя колонки схемы, к которой привязан компонент.
  "name": "ScoresValue",
  "values": {
   // Тип элемента управления — компонент.
   "itemType": Terrasoft.ViewItemType.COMPONENT,
   // Название класса.
   "className": "Terrasoft.UsrLimitedIntegerEdit",
   // Свойство value компонента связано с атрибутом ScoresAttribute.
   "value": { "bindTo": "ScoresAttribute" },
   // Значения для свойства minLimit.
   "minLimit": -300,
   // Значения для свойства maxLimit.
   "maxLimit": 300,
   // Свойства расположения компонента в контейнере.
   "layout": {
    "column": 0,
    "row": 6,
    "colSpan": 24
   }
  }
 }
]/**SCHEMA_DIFF*/
     };
 });
```

© 2022 Terrasoft. Все права защищены.

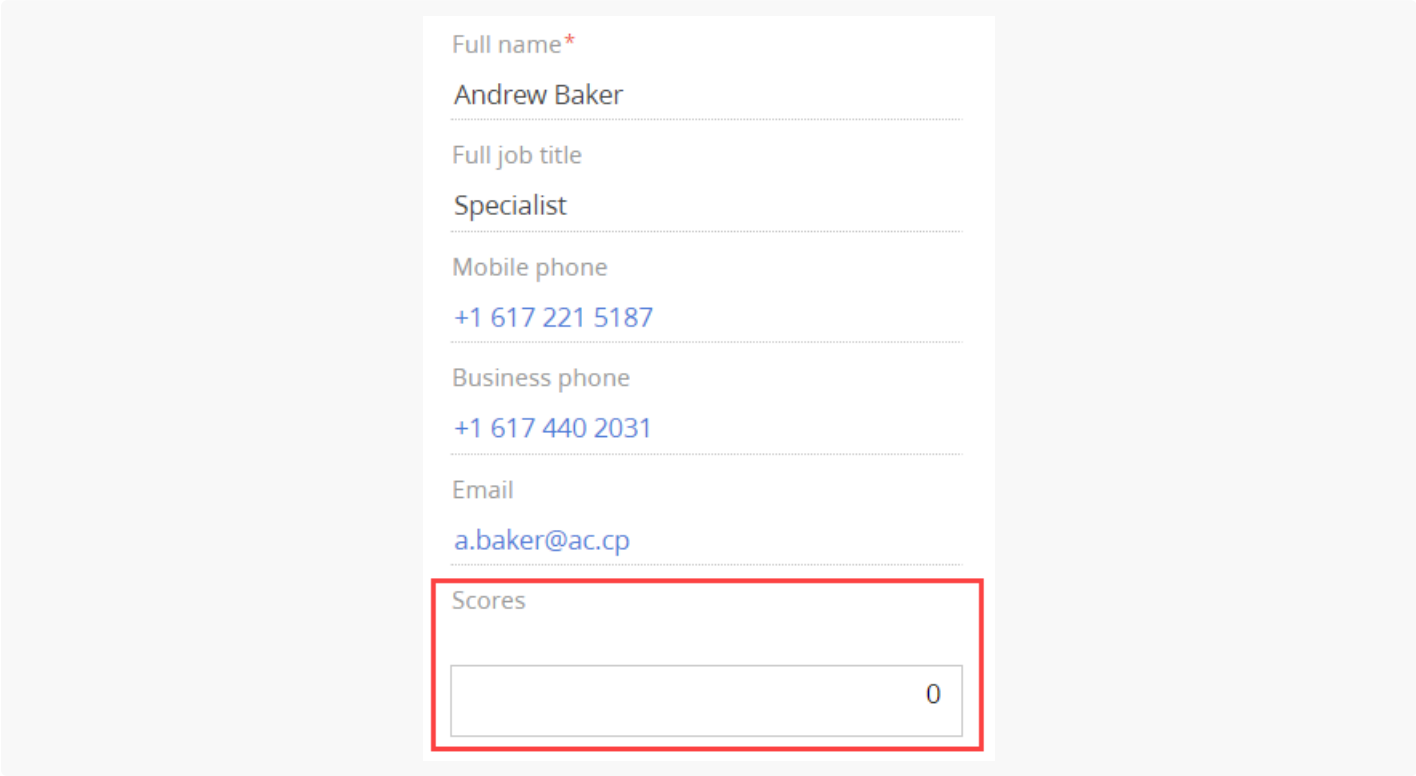
5. На панели инструментов дизайнера нажмите [*Сохранить*] ([*Save*]).

Значение атрибута `ScoresAttribute` связано со значением, введенным в поле ввода элемента управления. Вместо атрибута можно использовать целочисленную колонку объекта, связанного со схемой страницы редактирования записи.

В свойство `diff` добавлен конфигурационный объект, определяющий значения свойств экземпляра элемента управления. Значение свойства `value` связано с атрибутом `ScoresAttribute`. Свойствам `minLimit` и `maxLimit` присвоены значения, указывающие допустимый диапазон ввода. Если в конфигурационном объекте явно не указать свойства `minLimit` и `maxLimit`, то по умолчанию будет использоваться диапазон допустимых значений `[-1000, 1000]`.

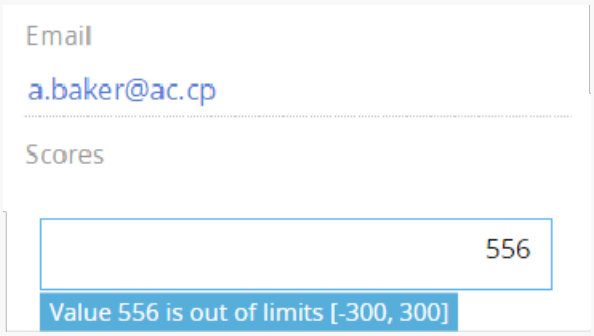
4. Проверить результат выполнения примера

В результате выполнения примера, на странице редактирования записи контакта будет добавлено поле ввода числовых значений.



При вводе в поле недопустимого значения отобразится предупреждающее сообщение.

© 2022 Terrasoft. Все права защищены.



Создать элемент управления для редактирования исходного кода

Средний

1. Подключить миксин

Для использования миксина в элементе управления добавьте его в свойстве `mixins`:

Подключение миксина

```
mixins: {
    SourceCodeEditMixin: "Terrasoft.SourceCodeEditMixin"
},
```

2. Реализовать абстрактные методы миксина

Функциональность миксина получает значение, вызывая абстрактный **getter**-метод `getSourceCodeValue()`, задача которого — вернуть строку для редактирования. В каждом конкретном случае "подмешивания" функциональности должен быть реализован свой **getter**-метод:

Реализация метода получения строкового значения

```
getSourceCodeValue: function () {
    // Метод getValue() реализован в базовом классе Terrasoft.BaseEdit.
    return this.getValue();
},
```

После завершения редактирования миксин вызовет абстрактный **setter**-метод `setSourceCodeValue()` для сохранения результата. В каждом конкретном случае "подмешивания" функциональности должен быть реализован свой **setter**-метод.

Реализация метода установки результирующей строки

© 2022 Terrasoft. Все права защищены.

```
setSourceCodeValue: function (value) {
    // Метод setValue() реализован в базовом классе Terrasoft.BaseEdit.
    this.setValue(value);
},
```

3. Вызвать метод openSourceCodeBox()

Для открытия окна редактирования исходного кода вызовите метод миксина `openSourceCodeBox()`. Следует обратить внимание на то, что метод вызван в контексте экземпляра основного класса. Например, при вызове метода `onSourceButtonClick()` компонента.

Реализация вызова метода открытия окна редактора исходного кода

```
onSourceButtonClick: function () {
    this.mixins.SourceCodeEditMixin
      .openSourceCodeBox.call(this);
},
```

4. Реализовать метод удаления миксина

После завершения работы с экземпляром основного класса происходит его удаление из памяти. Так как `SourceCodeEditMixin` требует определенных ресурсов, их также необходимо освободить. Для этого вызывается метод миксина `destroySourceCode()` в контексте экземпляра основного класса.

```
onDestroy: function () {
    this.mixins.SourceCodeEditMixin
      .destroySourceCode.apply(this, arguments);
    this.callParent(arguments);
}
```

Полный исходный код примера

SomeControl.js

```
// Добавление модуля миксина в зависимости.
define("SomeControl", ["SomeControlResources", "SourceCodeEditMixin"],
  function (resources) {
      Ext.define("Terrasoft.controls.SomeControl", {
          extend: "Terrasoft.BaseEdit",
          alternateClassName: "Terrasoft.SomeControl",
```

© 2022 Terrasoft. Все права защищены.

```
        // Подключение миксина.
        mixins: {
            SourceCodeEditMixin: "Terrasoft.SourceCodeEditMixin"
        },

        // Реализация метода получения строкового значения.
        getSourceCodeValue: function () {
            // Метод getValue() реализован в базовом классе Terrasoft.BaseEdit.
            return this.getValue();
        },

        // Реализация метода установки результирующей строки.
        setSourceCodeValue: function (value) {
            // Метод setValue() реализован в базовом классе Terrasoft.BaseEdit.
            this.setValue(value);
        },

        // Реализация вызова метода открытия окна редактора исходного кода.
        onSourceButtonClick: function () {
            this.mixins.SourceCodeEditMixin
              .openSourceCodeBox.call(this);
        },

        // Реализация вызова очистки ресурсов миксина.
        onDestroy: function () {
            this.mixins.SourceCodeEditMixin
              .destroySourceCode.apply(this, arguments);
            this.callParent(arguments);
        }
    });
});
```

Класс SourceCodeEditMixin JS

Сложный

Класс `SourceCodeEditMixin` предназначен для реализации элемента управления, предоставляющего возможность редактирования строкового значения, содержащего HTML, JavaScript или LESS код.

`SourceCodeEditMixin` — это миксин, который предназначен не для самостоятельного использования, а для обогащения других классов возможностью редактирования строки, используя удобный интерфейс. Концепция миксина напоминает концепцию множественного наследования.

Свойства

© 2022 Terrasoft. Все права защищены.

`sourceCodeEdit` `Terrasoft.SourceCodeEdit`

Экземпляр элемента управления редактора исходного кода.

`sourceCodeEditContainer` `Terrasoft.Container`

Экземпляр контейнера, в котором размещен редактор исходного кода.

Методы

`openSourceCodeEditModalBox()`

Метод, реализующий открытие модального окна редактирования исходного кода.

`loadSourceCodeValue()`

Абстрактный метод. Должен быть реализован в основном классе. Реализует логику получения значения для редактирования.

`saveSourceCodeValue()`

Абстрактный метод. Должен быть реализован в основном классе. Реализует логику сохранения результата редактирования в объект основного класса.

Параметры

`{String} value`	Результат редактирования.

`destroySourceCodeEdit()`

Метод, реализующий очистку ресурсов, используемых миксином.

`getSourceCodeEditModalBoxStyleConfig()`

Возвращает объект типа "ключ-значение", описывающий стили, которые будут установлены на модальное окно редактора исходного кода.

`getSourceCodeEditStyleConfig()`

Возвращает объект типа ключ-значение, описывающий стили, которые будут применены к элементам управления редактора исходного кода.

© 2022 Terrasoft. Все права защищены.

`getSourceCodeEditConfig()`

Возвращает объект типа "ключ-значение", описывающий свойства, с которыми будет создан экземпляр редактора исходного кода.

Свойства созданного объекта

`{Boolean} showWhitespaces`	Отображение невидимых строк. По умолчанию: `false`.

`{SourceCodeEditEnums.Language} language`	Синтаксис языка. Выбирается из перечисления `SourceCodeEditEnums.Language`. По умолчанию: `SourceCodeEditEnums.Language.JAVASCRIPT`.
	
	Возможные значения (`SourceCodeEditEnums.Language`)
	
	`JAVASCRIPT`
	JavaScript
	
	`CSHARP`
	C#
	
	`LESS`
	LESS
	
	`CSS`
	CSS
	
	`SQL`
	SQL
	
	`HTML`
	HTML

`{SourceCodeEditEnums.Theme} theme`	Тема редактора. Выбирается из перечисления `SourceCodeEditEnums.Theme`. По умолчанию: `SourceCodeEditEnums.Theme.CRIMSON_EDITOR`. Возможные значения (`SourceCodeEditEnums.Theme`)

© 2022 Terrasoft. Все права защищены.

	`SQLSERVER`
	Тема редактора SQL.
	
	`CRIMSON_EDITOR`
	Тема редактора Crimson.

`{Boolean} showLineNumbers`	Отображение номеров строк. По умолчанию: `true`.
`{Boolean} showGutter`	Установка зазора между столбцами. По умолчанию: `true`.
`{Boolean} highlightActiveLine`	Подсветка активной линии. По умолчанию: `true`.
`{Boolean} highlightGutterLine`	Подсветка линии в межстолбцовом промежутке. По умолчанию: `true`.

Библиотека JS классов

 Сложный

Документация по программному интерфейсу (JavaScript API) front-end части ядра платформы (JavaScript-ядра) доступна на отдельном web-ресурсе.

© 2022 Terrasoft. Все права защищены.