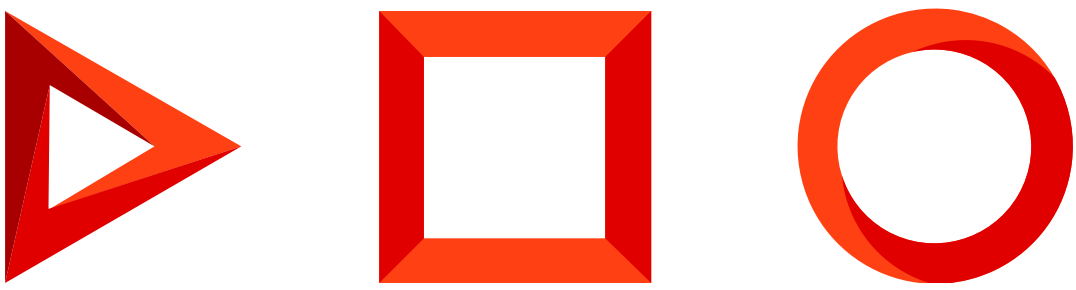
Creatio Academy

Front-end разработка

Sandbox

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

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.

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`.

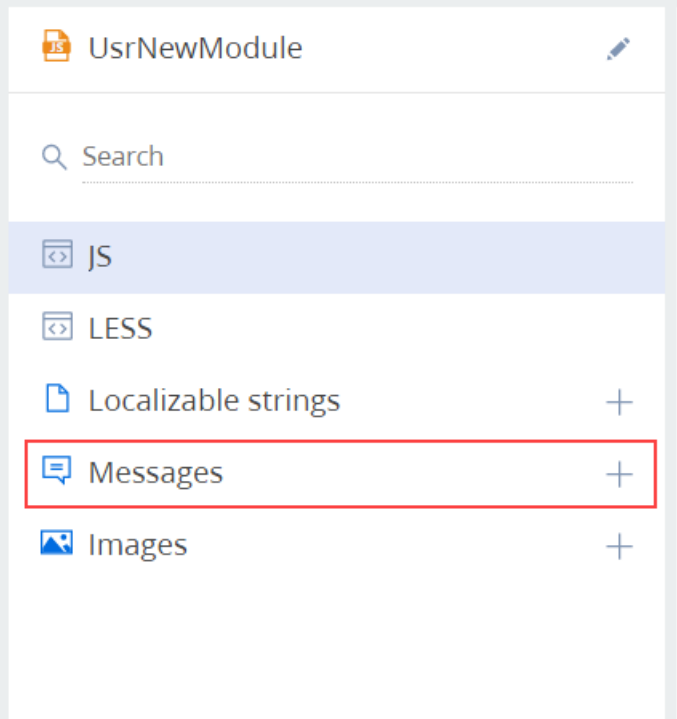
Для отказа от регистрации сообщений в модуле можно воспользоваться методом `sandbox.unRegisterMessages(messages)`, где `messages` — имя или массив имен сообщений.

Добавить сообщение в схему модуля

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

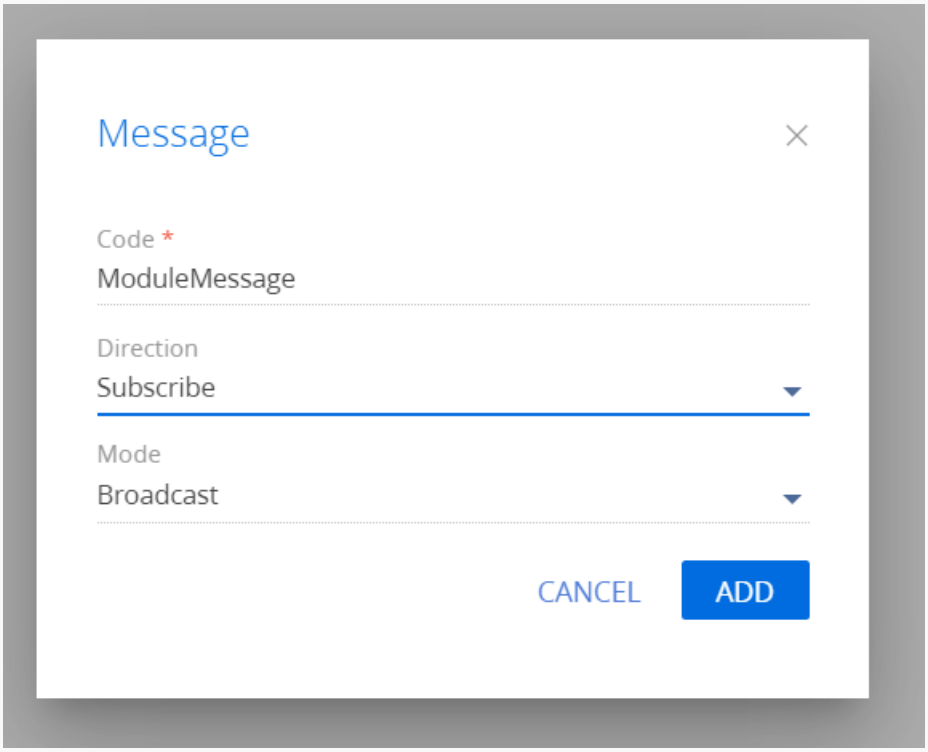
Для добавления сообщения в схему модуля:

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`").



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.

- `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

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.

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]);
}
```

```
...
```

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,
```

```
    // 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) {
```

```
        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);
```

```
            }
            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.
```

```
    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);
                    ...
```

```
                },
                ...
                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.

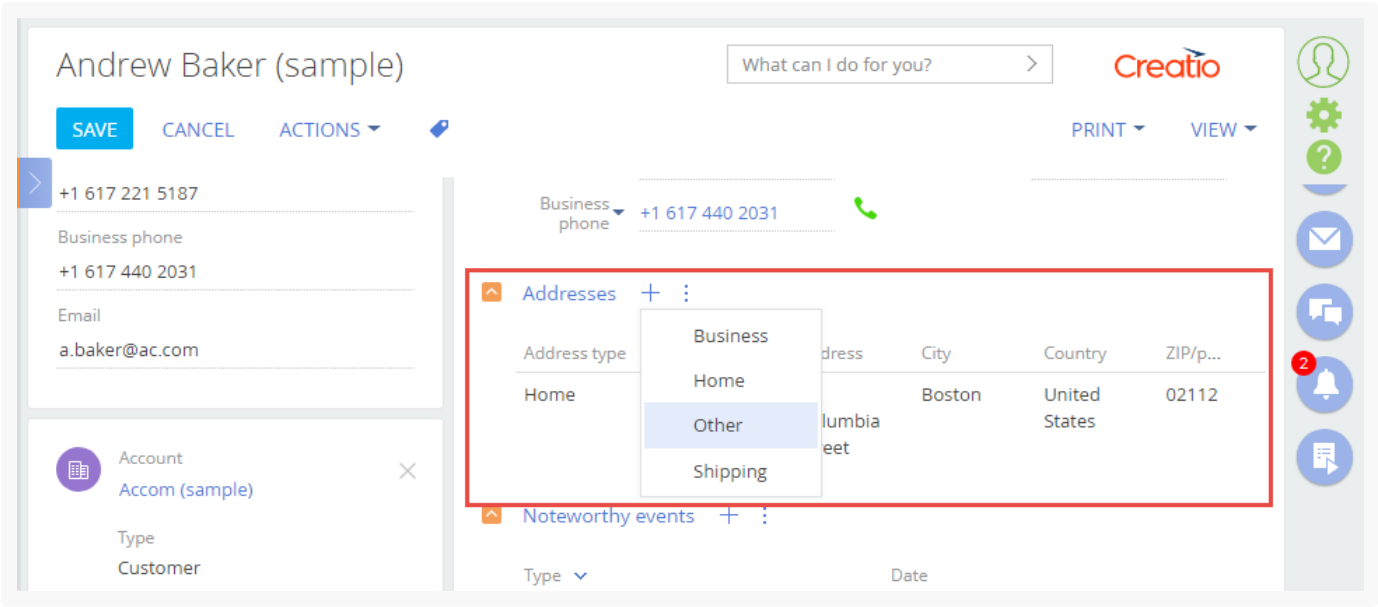
```
define("LookupQuickAddMixin", [...],
        function(...) {
            Ext.define("Terrasoft.configuration.mixins.LookupQuickAddMixin", {
                alternateClassName: "Terrasoft.LookupQuickAddMixin",
                ...
                // Объявление сообщения.
                _defaultMessages: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

```
                _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;
        });
```

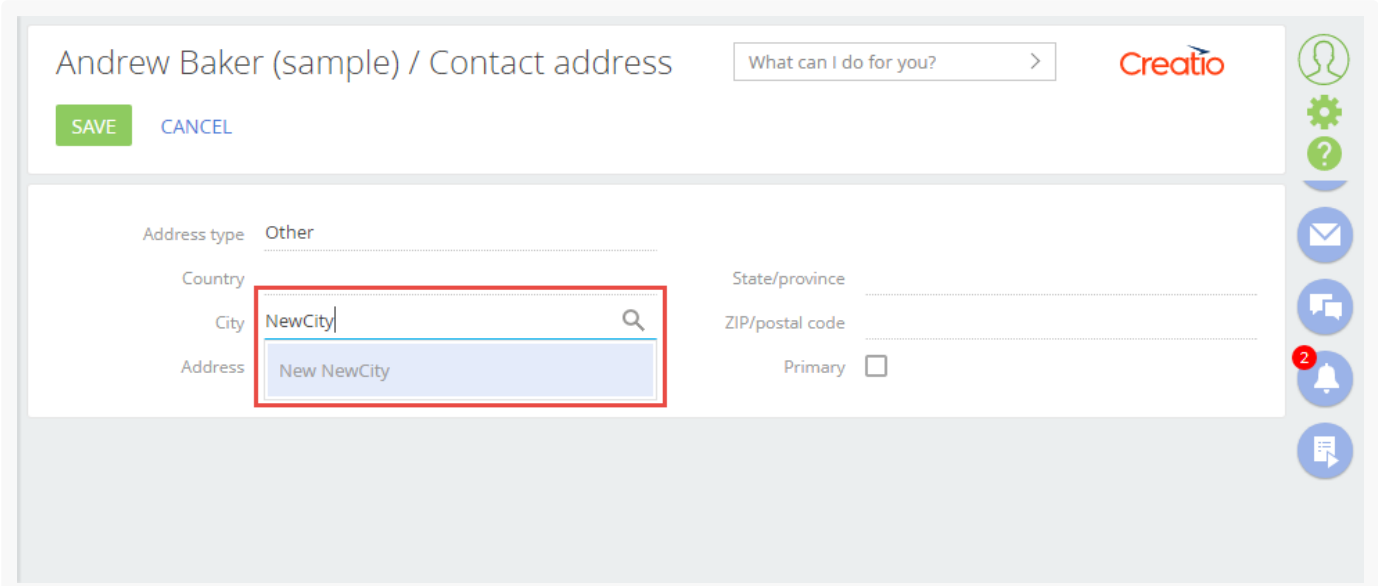
Последовательность работы с двунаправленным сообщением можно рассмотреть на примере добавления нового адреса на странице контакта.

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*])

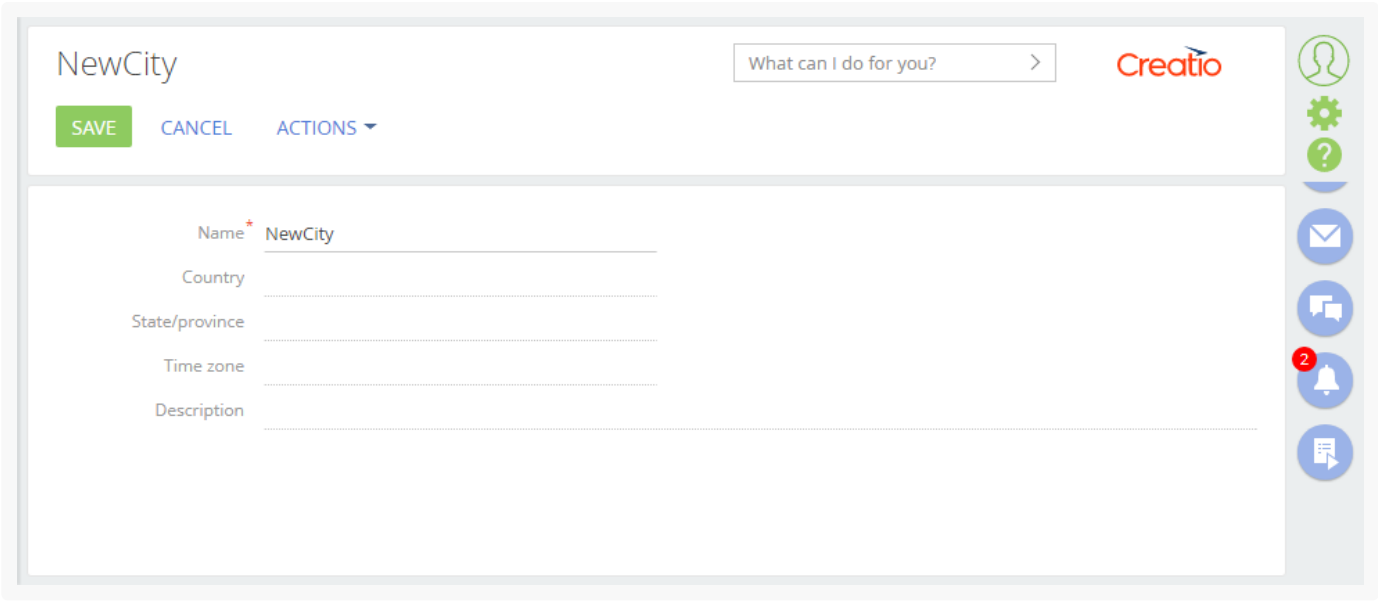


Страница адреса контакта



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`).



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(...) {
```

```
    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

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

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`)

	`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

`{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

`{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`

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.

`{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.

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");
```