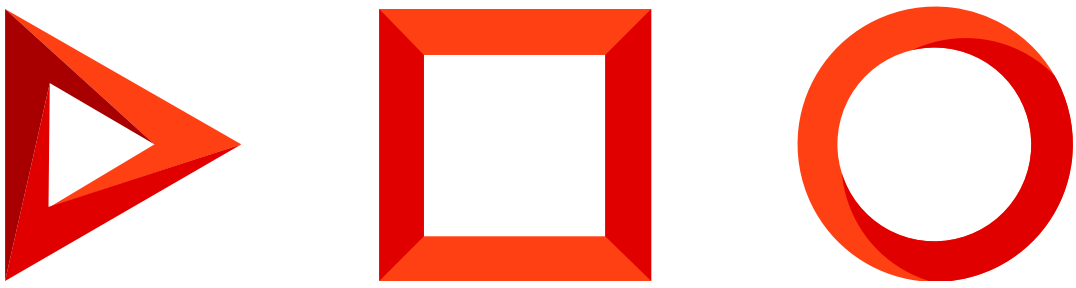
Creatio Academy

# Разработка мобильного приложения

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

**Общие принципы работы мобильного приложения** 6
Архитектура мобильного приложения 6
Экспорт данных в пакетном режиме 10
Жизненный цикл страниц в мобильном приложении 11
Фоновое обновление конфигурации в мобильном приложении 12
Автоматическое разрешение конфликтов при синхронизации 15
**Класс** BaseConfigurationPage 16
Методы 16
**Класс** PageNavigator 16
Методы 17
**Класс** Router 18
Методы 18
**Манифест мобильного приложения** 19
Свойства конфигурационного объекта для настроек синхронизации 19
Свойства конфигурационного объекта для настройки синхронизации модели 20
Свойства конфигурационного объекта фильтра модели 20
Свойство SyncOptions.ModelDataImportConfig.QueryFilter 24
**Настроить меню мобильного приложения** 26
Реализация примера 26
**Настроить стартовую страницу и разделы меню мобильного приложения** 26
Реализация примера 27
**Настроить конфигурацию моделей** 28
Реализация примера 28
**Использовать функцию поиска подстроки для поиска данных** 29
Реализация примера 29
**Загрузить данные моделей при синхронизации** 29
Реализация примера 30
**Отобразить страницу на планшете во весь экран** 31
Реализация примера 32
**Добавить стандартную деталь с колонками** 33
Алгоритм реализации примера 33
**Добавить пользовательский дашборд в мобильное приложение** 37
Алгоритм реализации примера 38
**Добавить кнопку для отображения имени контакта** 40
Алгоритм реализации примера 41
**Свойство** ModuleGroups 45

Свойство конфигурационного объекта 45
**Свойство** Modules **46**
Свойства конфигурационного объекта 46
**Свойство** Icons **47**
Свойства конфигурационного объекта 47
**Свойства** DefaultModuleImageId **и** DefaultModuleImageIdV2 **47**
**Свойство** Models **48**
Свойства конфигурационного объекта 48
**Свойство** PreferedFilterFuncType **49**
Функции фильтрации (Terrasoft.FilterFunctions) 49
**Свойство** CustomSchemas **50**
**Свойство** SyncOptions **50**
Свойства конфигурационного объекта для настроек синхронизации 51
Свойства конфигурационного объекта для настройки синхронизации модели 51
Свойства конфигурационного объекта фильтра модели 52
Свойство SyncOptions.ModelDataImportConfig.QueryFilter 56
**Реестр мобильного приложения** **57**
**Настроить реестр раздела** **57**
Реализация примера 57
**Класс** GridPage **58**
Методы 58
**Отладка мобильного приложения** **61**
**Бизнес-правила мобильного приложения** **61**
**Выполнить фильтрацию** **61**
Пример 1 62
Пример 2 62
Пример 3 63
**Выделить поле по условию** **64**
Реализация примера 64
**Сбросить отрицательные значения в** 0 **65**
Реализация примера 65
**Сгенерировать заголовок активности** **65**
Реализация примера 66
**Свойства объекта** config **66**
Базовые бизнес-правила 66
Бизнес-правило Обязательность заполнения (Terrasoft.RuleTypes.Requirement) 67
Бизнес-правило Видимость (Terrasoft.RuleTypes.Visibility) 68
Бизнес-правило Доступность (Terrasoft.RuleTypes.Activation) 69
Бизнес-правило Фильтрация (Terrasoft.RuleTypes.Filtration) 70

Бизнес-правило Взаимная фильтрация (Terrasoft.RuleTypes.MutualFiltration) 71
Бизнес-правило Регулярное выражение (Terrasoft.RuleTypes.RegExp) 72
Пользовательские бизнес-правила 73
**Получение данных и настроек по дашбордам** **76**
**Примеры запросов к сервису** AnalyticsService **77**
Методы 77
**Класс** AnalyticsService **79**
Методы 79

# Общие принципы работы мобильного приложения

Легкий

## Архитектура мобильного приложения

### Реализация приложений для мобильных устройств

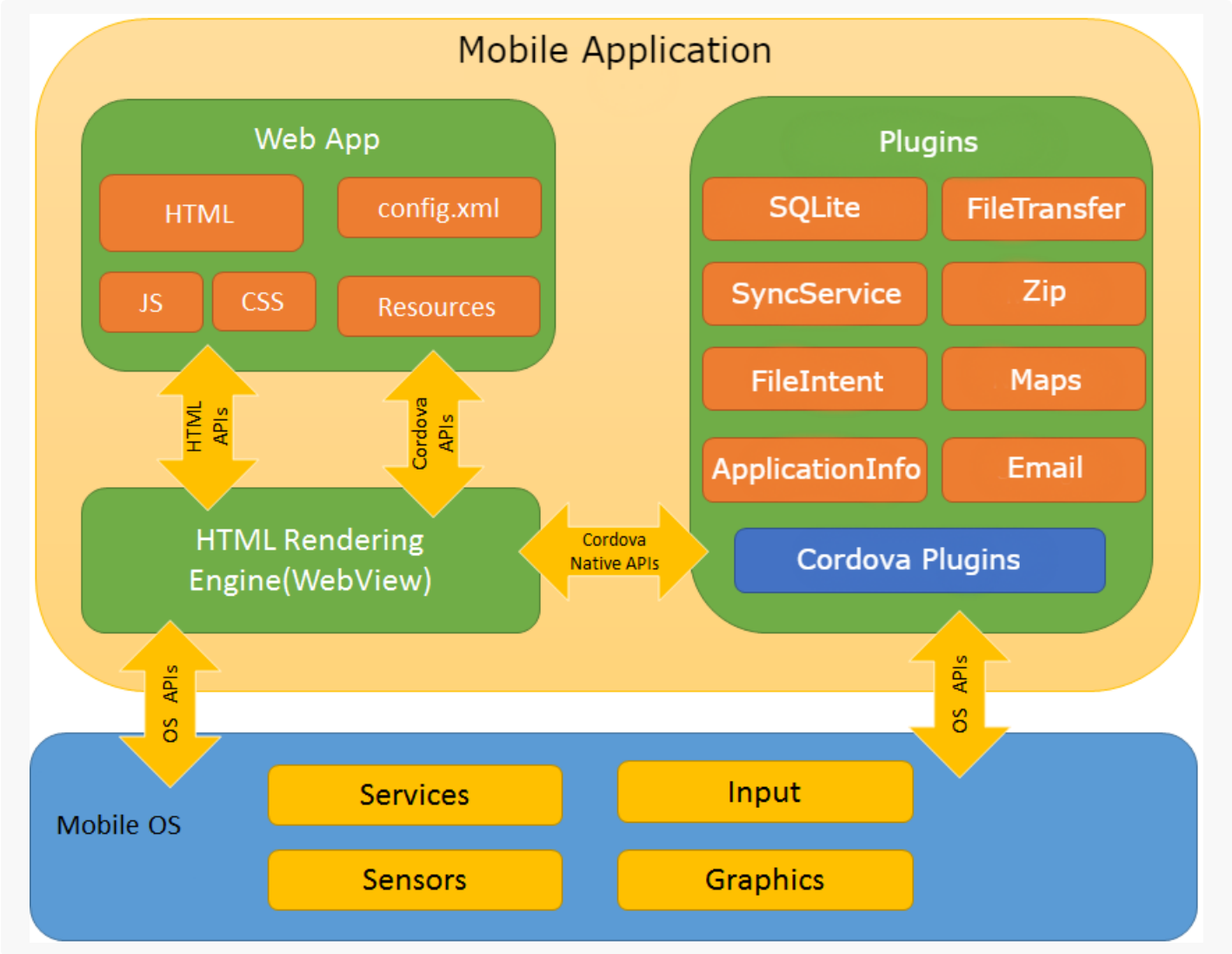
Существует три подхода технической реализации приложений для мобильных устройств:

**Мобильное native-приложение** — это специально разработанное приложение под конкретную мобильную платформу (iOS, Android, Windows Phone). Такое приложение разрабатывается на языке высокого уровня и компилируется в т. н. native-код ОС, обеспечивающий максимальную производительность. Главным недостатком мобильных приложений этого типа является низкая переносимость между мобильными платформами.

**Мобильное web-приложение** — специализированный web-сайт, адаптированный для просмотра и функционирования на мобильном устройстве. Такое приложение хоть и не зависит от платформы, однако требует постоянного подключения к сети, т. к. физически размещено не на мобильном устройстве, а на отдельном сервере.

**Гибридное приложение** — мобильное приложение, "упакованное" в native-оболочку. Такое приложение, как и native, устанавливается из онлайн-магазина и имеет доступ к тем же возможностям мобильного устройства, но разрабатываетсяс помощью web-языков HTML5, CSS и JavaScript. В отличие от native-приложения является легкопереносимым между различными платформами, однако несколько уступает в производительности. Мобильное приложение Creatio относится к этому типу приложений.

Схема архитектуры мобильного приложения:




Для создания гибридных приложений мобильное приложение Creatio использует возможности фреймворка Apache Cordova. Фреймворк Cordova обладает следующими преимуществами:

- Предоставляет доступ к программному интерфейсу мобильного устройства (API) для взаимодействия с базой данных или оборудованием (например, камерой или картой памяти).
- Предоставляет native-плагины для работы с API разных мобильных платформ (iOS, Android, Windows Phone и др.). Кроме того, разработка пользовательских плагинов позволяет добавлять функциональность и расширять API. Перечень доступных платформ и функциональность базовых native-плагинов Cordova содержится в документации Cordova.

Ядро мобильного приложения Creatio предоставляет унифицированный интерфейс для взаимодействия всех остальных клиентских частей приложения. Используемые ядром JavaScript-файлы условно можно разделить на базовые и конфигурационные категории.

Базовые скрипты содержатся в сборке приложения, публикуемой в магазине приложений, и включают в себя следующие элементы:

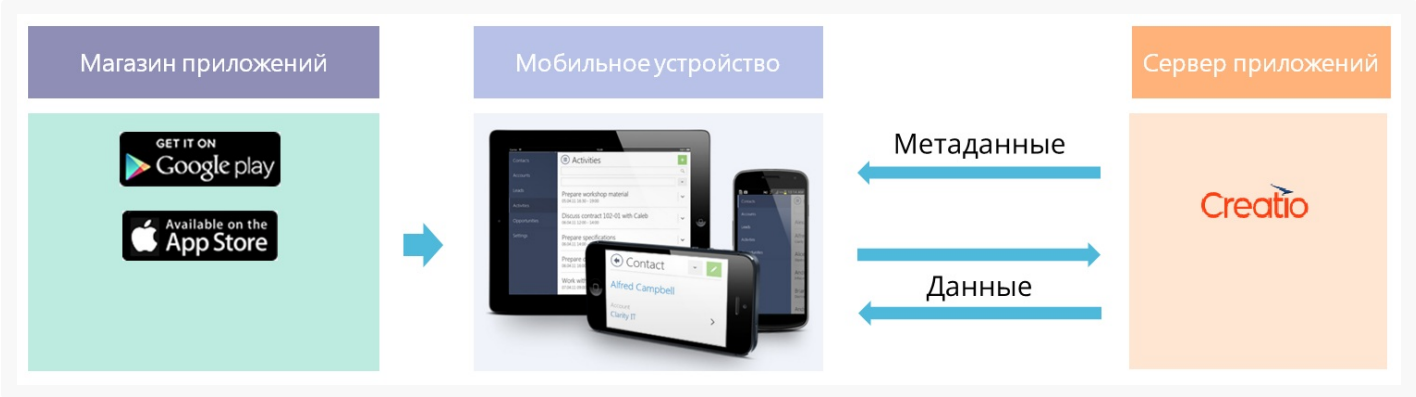
- MVC-компоненты (представления страниц, контроллеры, модели).
- Модули синхронизации (импорт и экспорт данных, импорт метаданных, импорт файлов и т. д.).

- Клиентские классы веб-сервисов.
- Классы, предоставляющие доступ к native-плагинам.

Приложение получает конфигурационные файлы в ходе синхронизации с сервером Creatio и сохраняет локально в файловой системе устройства. Конфигурационные файлы включают в себя манифест мобильного приложения Creatio, а также схемы и настройки разделов.

## Схема работы мобильного приложения Creatio

Опубликованное в магазине мобильное приложение Creatio представляет собой набор модулей, необходимых для синхронизации с сервером — основным приложением. Именно в основном приложении хранятся все необходимые настройки мобильного приложения и данные. Условно функционирование мобильного приложения можно представить в виде следующей схемы:




После установки приложения на мобильное устройство пользователь, указав параметры соединения с сервером Creatio, получает метаданные (структура приложения, системные данные) и данные от сервера.

Такая схема работы дает очевидное преимущество — мобильное приложение совместимо со всеми существующими продуктами Creatio. Каждый продукт, каждый отдельно взятый сайт клиента может содержать собственный набор настроек мобильного приложения, свою логику работы, даже свой визуальный интерфейс. Все что нужно сделать мобильному пользователю — установить мобильное приложение и выполнить синхронизацию с сайтом Creatio.

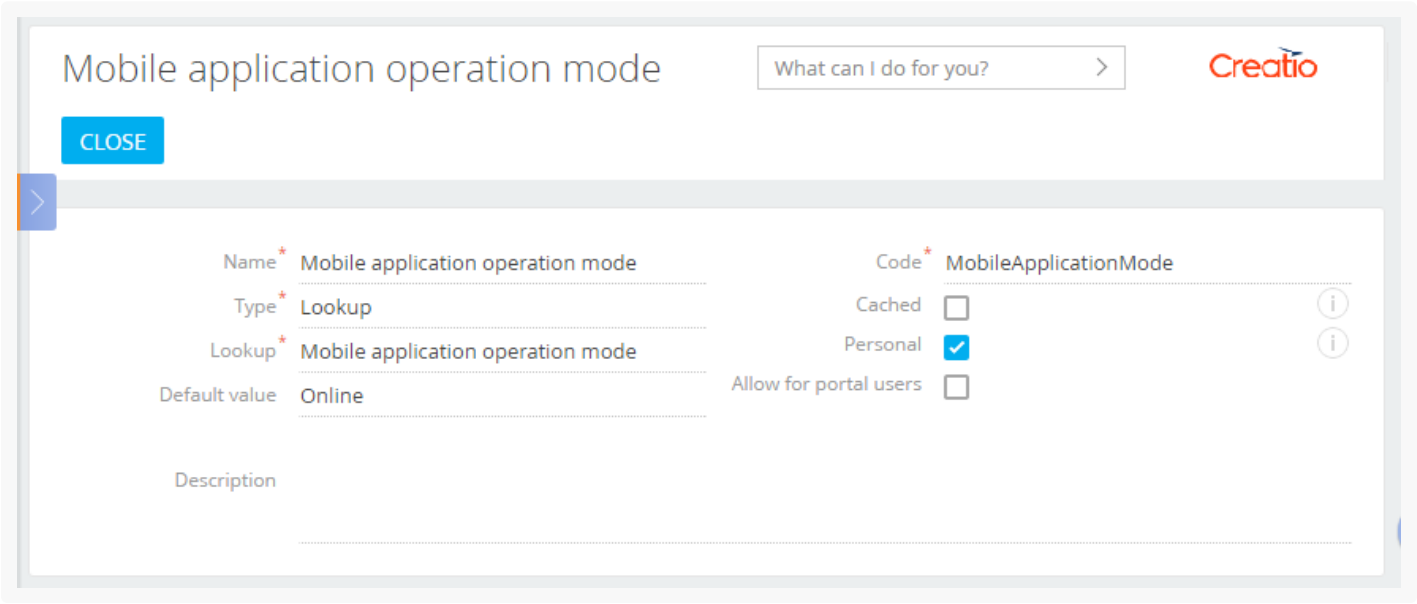
## Режимы работы мобильного приложения Creatio

Мобильное приложение Creatio может работать в следующих режимах:

- **Гибридный режим**. Гибридный режим предназначен для работы с данными и автоматически включается при отсутствии стабильного соединения с сервером Creatio. Этот режим позволяет создавать новые записи и работать с расписанием. Также реализована возможность работы с недавними записями раздела (10 записей), с которыми взаимодействовал пользователь.
- **Online**. Для online-режима необходимо наличие интернет-соединения. При использовании этого режима пользователь работает напрямую с сервером Creatio, в качестве которого выступает основное приложение. Синхронизация конфигурационных изменений выполняется автоматически в режиме реального времени.
- **Offline**. Для offline-режима наличие интернет-соединения требуется только для первичного импорта и синхронизации. При использовании этого режима данные сохраняются локально на мобильном

устройстве. Для получения конфигурационных изменений и актуализации данных необходимо вручную выполнять синхронизацию с сервером Creatio.

За режим работы мобильного приложения отвечает системная настройка [ *Режим работы мобильного приложения* ] ([ *Mobile application operation mode* ]) в Creatio. Если нужно изменить режим работы одновременно для всех пользователей мобильного приложения, необходимо установить нужное значение этой настройки без установленного свойства [ *Персональная* ]. Если же необходимо для разных пользователей указать разные режимы, то нужно эти изменения делать непосредственно пользователям, устанавливая свойство [ *Персональная* ]. У этих пользователей должны быть права на изменение системных настроек.




## Синхронизация мобильного приложения с Creatio

В зависимости от режима работы приложения, синхронизация с сервером Creatio выполняет разные задачи. В случае online-режима синхронизация нужна только для получения изменений в конфигурации. А в случае offline-режима синхронизация необходима как для получений обновлений, так и для передачи или получения измененных или новых данных. Общая схема синхронизации для offline-режима:


Сначала приложение выполняет аутентификацию. При этом, выполняя logout, на сервере уничтожается текущая активная сессия. Далее у сервера запрашиваются данные для формирования пакета разницы. Приложение анализирует эти данные и запрашивает измененные или новые конфигурационные схемы. После загрузки схем приложение получает системные данные, к которым относятся кешируемые справочники (так называемые “простые” справочники), системные настройки и т. д. Затем идет обмен данными с сервером.

Отличие синхронизации в online-режиме заключается в том, что у нее нет последних двух этапов — экспорта и импорта.

**На заметку.** В версии мобильного приложения 7.8.6 реализован еще один этап синхронизации — “Актуализация данных”. Если эта функциональность включена, то данный этап выполняется последним, после экспорта и импорта данных. Суть этапа в следующем: приложение сравнивает доступные на сервере данные с локальными и, в случае наличия разницы, загружает недостающие данные или удаляет неактуальные. Этот механизм предусматривает ситуацию, которая возможна в случае перераспределения прав доступа или удаления данных на сервере. Для его включения в манифесте в секции `SyncOptions,` в свойстве `ModelDataImportConfig` для нужного объекта-модели установить значение `true` для свойства `IsAdministratedByRights`.

## Экспорт данных в пакетном режиме

По умолчанию мобильное приложение отправляет каждое произведенное пользователем изменение данных поочередно. Таким образом одно изменение производит как минимум один запрос на сервер. При достаточно большом количестве изменений выполнение таких запросов может занять существенное время.

Начиная с версии 7.9, стало возможным отправлять данные в пакетном режиме (batch mode), что позволяет значительно ускорить отправку данных на сервер.

Для включения пакетного режима отправки данных необходимо в манифесте мобильного приложения в секции `SyncOptions` установить для свойства `UseBatchExport` значение `true`. В результате все пользовательские изменения будут сгруппированы в несколько пакетных запросов согласно типу операции, выполняемой пользователем. Возможные типы операций — вставка, обновление и удаление.

# Жизненный цикл страниц в мобильном приложении

Во время навигации по мобильному приложению для каждой страницы выполняется ряд этапов — открытие, закрытие, выгрузка, возврат к странице и т. п. Время, прошедшее от загрузки страницы в память мобильного устройства и до ее окончательной выгрузки из памяти, принято называть жизненным циклом страницы.

Для каждого этапа жизненного цикла предусмотрены события страницы. Использование событий дает возможность расширять функциональность. К основным событиям относятся:

- инициализация представления;
- завершение инициализации класса;
- загрузка страницы;
- загрузка данных;
- закрытие страницы.

Понимание этапов выполнения жизненного цикла страницы позволяет качественно и максимально эффективно расширять логику страниц.

## Этапы жизненного цикла

**Важно.** На экране телефона может отображаться только одна страница. На экране планшета — одна страница в портретной ориентации и две в ландшафтной. В связи с этим жизненный цикл страниц имеет отличия для телефона и планшета.

### Открытие страницы

При первом открытии страницы выполняется загрузка скриптов, требуемых для ее работы. Далее инициализируется контроллер и создается представление.

События открытия страницы генерируются в следующей последовательности:

1. `initializeView` — инициализация представления.
2. `pageLoadComplete` — событие завершения загрузки страницы.

3. `launch` — инициирует загрузку данных.

### Закрытие страницы

Во время закрытия страницы ее представление удаляется из объектной модели документа (Document object model, DOM), а контроллер удаляется из памяти устройства.

Закрытие страницы происходит в следующих случаях:

- Нажата кнопка [ *Назад* ]. В таком случае удаляется последняя страница.
- Выполнен переход в другой раздел. В таком случае удаляются все страницы, которые были открыты ранее.

Событие завершения закрытия страницы — `pageUnloadComplete`.

### Выгрузка страницы

Выгрузка страницы происходит в случае, когда выполняется переход к другой странице в том же разделе. При этом текущая страница становится неактивной. Она может оставаться видимой на экране устройства. Например, если на планшете открыть страницу просмотра из реестра, то страница реестра останется видимой. В такой же ситуации на телефоне страница реестра не будет отображаться, но будет оставаться в памяти. Это и отличает выгрузку от закрытия страницы.

Событие выгрузки страницы — `pageUnloadComplete` (совпадает с событием закрытия страницы).

### Возврат к странице

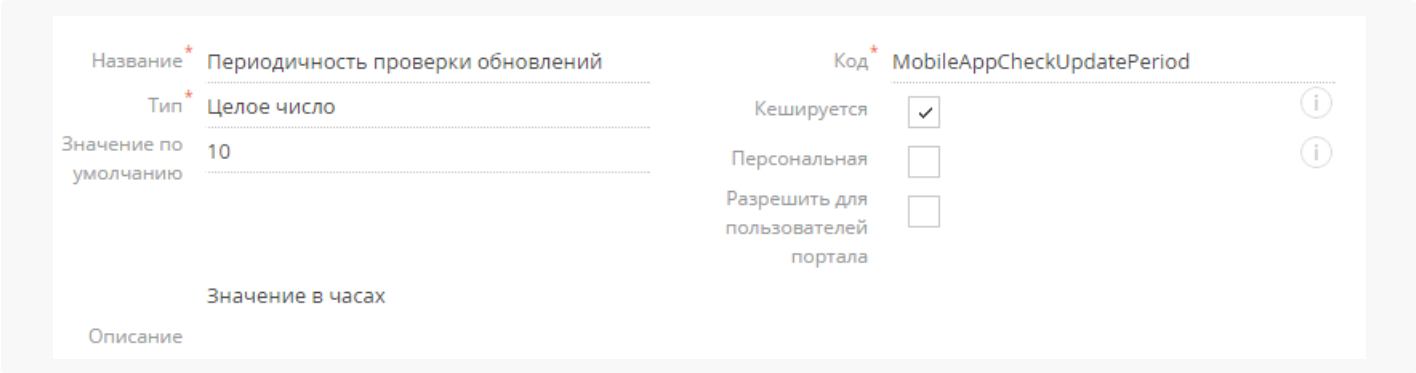
Возврат к выгруженной ранее странице происходит при нажатии на кнопку [ *Назад* ].

Событие возврата к странице — `pageLoadComplete`.

**Важно.** В приложении может использоваться только один экземпляр страницы. Поэтому, если последовательно открыть две одинаковые страницы, то при возврате к первой из них повторно выполняется обработчик события `launch`. Это следует учитывать при разработке.

## Фоновое обновление конфигурации в мобильном приложении

В мобильном приложении Creatio реализован механизм синхронизации структуры приложения, который может работать в автоматическом фоновом режиме. Для управления этим процессом необходимо использовать системную настройку [ *Периодичность проверки обновлений* ] ([ *Update checks frequency* ]).




Эта настройка указывает по истечении какого времени (в часах) мобильное приложение может запросить изменения конфигурации у Creatio. Если настройке установить значение 0, то приложение будет всегда загружать обновления конфигурации.

## Условия работы

Приложение запускает синхронизацию структуры в фоновом режиме только при соблюдении следующих условий:

- на мобильном устройстве используется платформа iOS или Android;
- синхронизация ранее не была запущена;
- с момента последней синхронизации структуры прошло больше времени, чем указано в системной настройке [ *Периодичность проверки обновлений* ] ([ *Update checks frequency* ]);
- осуществляется запуск приложения, или приложение активируется (т.е. если оно было ранее свернуто или в него переходят из другого приложения).

Если в ходе обновления структуры изменения были получены, то для применения полученных изменений приложение автоматически перезапустится когда пользователь свернет его или перейдет в другое приложение.

## Особенности работы на разных платформах

1. На платформе Android фоновый режим реализован через параллельно запущенный сервис. Такой подход гарантирует, что запущенная синхронизация гарантировано завершится, даже если вручную выгрузить приложение из памяти устройства.
2. На платформе iOS для запуска синхронизации в фоновом режиме используется второй `webView`, в то время как само приложение работает в основном `webView`. Это гарантирует нормальную работу пользователя в приложении при одновременно запущенной синхронизации структуры.

   В отличие от реализации на платформе Android это не гарантирует завершения синхронизации на 100%, поскольку синхронизация может быть прервана при выгрузке приложения вручную либо если это сделает платформа iOS.

В приложении под iOS (начиная с версии 7.17.2) вместо `UIWebView` используется `WKWebView`, что привело к использованию фреймворка Cordova версии 6.1.1, а также минимально поддерживаемой версия iOS 11.

`WKWebView` имеет следующие особенности:

- Не допускается использовать абсолютные пути (для ресурсов, скриптов, iframe и т.п.).
- Не допускается использовать кросс-доменные ссылки (для ресурсов, скриптов, iframe и т.п.).
- Данные `localStorage` не сохранятся при переходе на `WKWebView`.
- Не рекомендуется использование iframe.

Изменения в iOS WebView в Apache Cordova подробнее описаны в документации Cordova.

Если необходимо вставить на страницу ссылку на локальный файл, то необходимо конвертировать путь к нему через метод `Terrasoft.util.toUrlScheme`.

**Вставка на страницу ссылки на локальный файл**

```
this.element.setStyle('background-image', 'url("' + Terrasoft.util.toUrlScheme(value) +'")');
```

Пример ссылок представлен ниже.

Недопустимая ссылка

```
file:///var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C271D9871/Documents/BP
```

Ссылка после конвертации

```
app://localhost/_app_file_/var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C27
```

Использование плагина `inappbrowser` имеет следующие особенности:

- Все пути должны быть относительными и находиться внутри корневой папки сайта.

```
<img src="images/2.jpg">
```

- Не допускается использовать кросс-доменные ссылки (для ресурсов, скриптов, iframe и т.п.).

```
<img src="https://www.worldometers.info/img/worldometers-logo.gif">
<img src="file:///var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C271D9871
```

- При открытии сайта необходимо указывать абсолютный путь.

```
cordova.InAppBrowser.open("file:///var/mobile/Containers/Data/Application/DE..9871/Documents/
```

Открытие сайта через плагин `inappbrowser` имеет особенности, аналогичные особенностям `WKWebView`.

# Автоматическое разрешение конфликтов при синхронизации

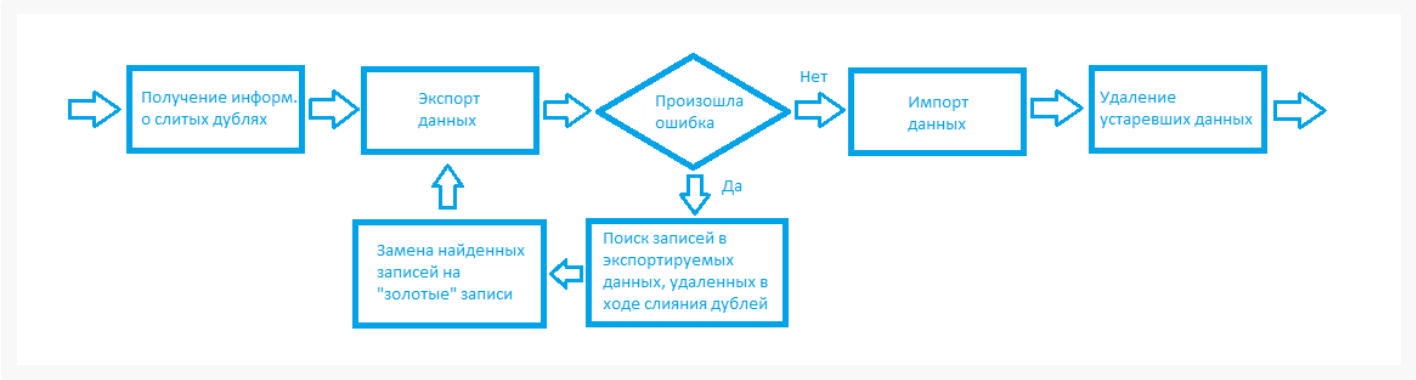
В ходе синхронизации мобильного приложения, работающего в offline-режиме, могут возникать ситуации, когда переданные в Creatio данные не могут быть сохранены по ряду причин. К таким причинам могут относиться следующие:

- Запись была слита в Creatio с другой дублирующейся записью, поэтому ее не существует.
- Запись была удалена из Creatio.

Каждая из приведенных выше ситуаций обрабатывается мобильным приложением автоматически.

## Слияние дублей

Алгоритм разрешения конфликта, возникающего по причине использования данных, которые в Creatio были удалены в ходе процедуры слияния дублей:




Как видно на схеме, в ходе синхронизации приложение сначала забирает на сервере информацию о том, по каким записям с момента последней синхронизации производилось слияние дублей. А именно какие записи были удалены и какие записи их заменили. Если в ходе экспорта не было никаких ошибок, то далее выполняется импорт. Если же произошла ошибка, связанная с исключением внешнего ключа ( `Foreign Key Exception` ), или ошибка, связанная с тем, что на сервере не была найдена какая-то из записей ( `Item Not Found Exception` ), то выполняется процедура разрешения этого конфликта со следующими этапами:

- В экспортируемых данных ищутся колонки, содержащие “старую” запись.
- В найденных колонках “старая” запись заменяется новой, в которой данные объединялись.

После этого запись повторно отправляется в Creatio. Как только заканчивается импорт и появляется информация о слитых дублях, локально производится удаление “старых” записей.

## Запись не найдена

В случае, когда сервер возвращает ошибку, свидетельствующую о том, что измененная пользователем запись в Creatio не найдена, приложение выполняет следующие действия:

1. Проверяет наличие записи в списке записей, удаленных в ходе слияния дублей.

2. Если в списке удаленных записи нет, то приложение удаляет ее локально.
3. Удаляет информацию по этой записи из лога синхронизации.

Таким образом, приложение считает этот конфликт разрешенным и продолжает экспорт данных.

# Класс BaseConfigurationPage JS

Сложный

Классы контроллеров страниц наследуются от класса `Terrasoft.controller.BaseConfigurationPage`, который предоставляет методы обработки событий жизненного цикла.

## Методы

---

`initializeView(view)`

Вызывается после того как было создано (но еще не было отрисовано) представление страницы в DOM. На этом этапе можно подписываться на события классов представления, выполнять дополнительные манипуляции с DOM.

---

`pageLoadComplete(isLaunch)`

Предоставляет возможность расширения логики, которая выполняется как при загрузке страницы, так и при возврате. Значение параметра `isLaunch` равное `true` указывает на то, что страница загружается первый раз.

---

`launch()`

Вызывается только при открытии страницы. Метод инициирует начало загрузки данных. Если требуется загрузка дополнительных данных, то правильно будет делать это в методе `launch()`.

---

`pageUnloadComplete()`

Предоставляет возможность расширения логики, которая выполняется как при закрытии страницы, так и при ее выгрузке.

# Класс PageNavigator JS

Сложный

Управлением жизненным циклом страниц занимается класс `Terrasoft.PageNavigator`. Класс предоставляет возможности открытия и закрытия страниц, обновления неактуальных данных, а также хранения истории открытых страниц.

# Методы

`forward(openingPageConfig)`

Метод открывает страницу с учетом свойств конфигурационного объекта-параметра `openingPageConfig`. Основные свойства этого объекта представлены в таблице.

Свойства объекта `openingPageConfig`

| | |
|---|---|
| `controllerName` | Имя класса контроллера. |
| `viewXType` | Тип представления по xtype. |
| `type` | Тип страницы из перечисления `Terrasoft.core.enums.PageType`. |
| `modelName` | Имя модели страницы. |
| `pageSchemaName` | Название схемы страницы в конфигурации. |
| `isStartPage` | Признак, указывающий на то, что страница должна быть первой. Если до этого уже были открыты страницы, то они будут закрыты. |
| `isStartRecord` | Признак, указывающий на то, что страница карточки просмотра/редактирования должна быть первой после реестра. Если есть другие открытые страницы после реестра, они закрываются. |
| `recordId` | Идентификатор записи открываемой страницы. |
| `detailConfig` | Настройки стандартной детали. |

`backward()`

Метод закрывает страницу.

`markPreviousPagesAsDirty(operationConfig)`

Метод отмечает все предыдущие страницы как неактуальные. После возврата к предыдущим страницам для каждой из них вызовется метод `refreshDirtyData()`, который выполняет повторную загрузку данных или актуализирует данные на основании объекта `operationConfig`.

`refreshPreviousPages(operationConfig, currentPageHistoryItem)`

Метод выполняет для всех предыдущих страниц повторную загрузку данных или актуализирует данные на основании `operationConfig`. Если установлено значение для параметра

`currentPageHistoryItem`, метод выполняет те же действия для предшествующих страниц.

---

`refreshAllPages(operationConfig, excludedPageHistoryItems)`

Метод выполняет для всех страниц повторную загрузку данных или актуализирует данные на основании `operationConfig`. Если установлен параметр `excludedPageHistoryItems`, метод исключает из актуализации указанные страницы.

# Класс Router JS

Сложный

Маршрутизация используется для управления визуальными компонентами: страницами, пикерами и др. Маршрут имеет три состояния:

1. `Load` — выполняет открытие текущего маршрута.
2. `Unload` — выполняет закрытие текущего маршрута при возврате.
3. `Reload` — выполнят восстановление предыдущего маршрута при возврате.

Для маршрутизации используется класс `Terrasoft.Router` и его основные методы `add()`, `route()`, `back()`.

## Методы

---

`add(name, config)`

Добавляет маршрут.

### Параметры

| | |
|---|---|
| `name` | Уникальное имя маршрута. В случае повторного добавления, последний переопределит предыдущий. |
| `config` | Описывает имена функций обработчиков состояний маршрута. В свойстве `handlers` устанавливаются обработчики состояний маршрута. |

**Пример использования**

```
Terrasoft.Router.add("record", {
  handlers: {
     load: "loadPage",
     reload: "reloadPage",
     unload: "unloadLastPage"
  }
```

```
});
```

`route(name, scope, args, config)`

Выполняет открытие маршрута.

Параметры

| | |
|---|---|
| `name` | Имя маршрута. |
| `scope` | Контекст функции обработчиков состояний. |
| `args` | Параметры функций обработчиков состояний. |
| `config` | Дополнительные параметры маршрута. |

**Пример использования**

```
var mainPageController = Terrasoft.util.getMainController();
Terrasoft.Router.route("record", mainPageController, [{pageSchemaName: "MobileActivityGridPag
```

`back()`

Выполняет закрытие текущего маршрута и восстановление предыдущего.

# Манифест мобильного приложения

Сложный

Описывает параметры для настройки синхронизации данных. Содержит конфигурационный объект со свойствами, представленными в таблице.

## Свойства конфигурационного объекта для настроек синхронизации

`ImportPageSize`

Количество страниц, импортируемых в одном потоке.

`PagesInImportTransaction`

Количество потоков импорта.

---

`SysSettingsImportConfig`

Массив импортируемых системных настроек.

---

`SysLookupsImportConfig`

Массив импортируемых системных справочников.

---

`ModelDataImportConfig`

Массив моделей, для которых будут загружаться данные при синхронизации.

В массиве моделей `ModelDataImportConfig` для каждой модели можно указать дополнительные параметры синхронизации, список загружаемых колонок, а также условия фильтрации загружаемых данных модели. Если при синхронизации должна загружаться полная модель, в массиве просто указывается объект с именем модели. Если к модели должны применяться дополнительные условия при синхронизации, в массив `ModelDataImportConfig` добавляется конфигурационный объект со свойствами.

## Свойства конфигурационного объекта для настройки синхронизации модели

---

`Name`

Название модели (см. свойство `Models` конфигурационного объекта манифеста).

---

`SyncColumns`

Массив колонок модели, для которых импортируются данные. Помимо явно перечисленных колонок, при синхронизации в обязательном порядке будут импортироваться системные колонки ( `CreatedOn` , `CreatedBy` , `ModifiedOn` , `ModifiedBy` ) и колонка, первичная для отображения.

---

`SyncFilter`

Фильтр, накладываемый на модель при импорте модели.

Фильтр `SyncFilter` , накладываемый на модель при импорте модели представляет собой конфигурационный объект со свойствами.

## Свойства конфигурационного объекта фильтра модели

---

`type`

Тип фильтра. Задается значением перечисления `Terrasoft.FilterTypes`. Необязательное свойство. По умолчанию `Terrasoft.FilterTypes.Simple`.

Возможные значения ( `Terrasoft.FilterTypes` )

| | |
|---|---|
| `Simple` | фильтр с одним условием |
| `Group` | групповой фильтр с несколькими условиями |

---

`logicalOperation`

Логическая операция объединения коллекции фильтров (для фильтров с типом `Terrasoft.FilterTypes.Group`). Задается значением перечисления `Terrasoft.FilterLogicalOperations`. Значение по умолчанию - `Terrasoft.FilterLogicalOperations.And`.

Возможные значения ( `Terrasoft.FilterLogicalOperations` )

| | |
|---|---|
| `Or` | логическая операция ИЛИ |
| `And` | логическая операция И |

---

`subfilters`

Коллекция фильтров, применяемых к модели. Обязательное свойство для типа фильтра `Terrasoft.FilterTypes.Group`. Фильтры между собой объединяются логической операцией, указанной в свойстве `logicalOperation`. Каждый фильтр представляет собой конфигурационный объект фильтра.

---

`property`

Название колонки модели, по которой выполняется фильтрация. Обязательное свойство для типа фильтра `Terrasoft.FilterTypes.Simple`.

---

`valueIsMacrosType`

Признак, определяющий, является ли значение для фильтрации макросом. Необязательное свойство. Может принимать значения: `true`, если для фильтрации используется макрос, иначе — `false`.

---

`value`

Значение для фильтрации колонки, указанной в свойстве `property`. Обязательное свойство для типа фильтра `Terrasoft.FilterTypes.Simple`. Может задаваться непосредственно значением для фильтрации (в том числе, может быть `null`) либо макросом (для этого свойство `valueIsMacrosType` должно иметь значение `true`). Макросы, которые можно использовать в качестве значения свойства, содержатся в

перечислении `Terrasoft.ValueMacros`.

Возможные значения (`Terrasoft.ValueMacros`)

| | |
|---|---|
| `CurrentUserContactId` | идентификатор текущего пользователя |
| `CurrentDate` | текущая дата |
| `CurrentDateTime` | текущие дата и время |
| `CurrentDateEnd` | полная дата окончания текущей даты |
| `CurrentUserContactName` | имя текущего контакта |
| `CurrentUserContact` | идентификатор и имя текущего контакта |
| `SysSettings` | значение системной настройки. Имя системной настройки передается в свойстве `macrosParams` |
| `CurrentTime` | текущее время |
| `CurrentUserAccount` | идентификатор и имя контрагента текущего пользователя |
| `GenerateUId` | сгенерированный идентификатор |

---

`macrosParams`

Значения, которые передаются в макрос в качестве параметра. Необязательное свойство. В настоящее время используется только для макроса `Terrasoft.ValueMacros.SysSettings`.

---

`isNot`

Определяет, применяется к фильтру оператор отрицания. Необязательное свойство. Принимает значение `true`, если к фильтру применяется оператор отрицания, иначе — `false`.

---

`funcType`

Тип функции, которая применяется к колонке модели, заданой в свойстве `property`. Необязательное свойство. Может принимать значения перечисления `Terrasoft.FilterFunctions`. Значения аргументов для функций фильтрации задаются в свойстве `funcArgs`. Значение, с которым сравнивается результат функции, задается свойством `value`.

Возможные значения (`Terrasoft.FilterFunctions`)

| | |
|---|---|
| `SubStringOf` | определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки `property` |
| `ToUpper` | приводит значения колонки, заданной в `property,` к верхнему регистру |
| `EndsWith` | проверяет, оканчивается ли значение колонки `property` значением, переданным в качестве аргумента |
| `StartsWith` | проверяет, начинается ли значение колонки `property` значением, переданным в качестве аргумента |
| `Year` | возвращает год по значению колонки `property` |
| `Month` | возвращает месяц по значению колонки `property` |
| `Day` | возвращает день по значению колонки `property` |
| `In` | проверяет вхождение значения колонки `property` в диапазон значений, переданных в качестве аргумента функции |
| `NotIn` | проверяет невхождение значения колонки `property` в диапазон значений, переданных в качестве аргумента функции |
| `Like` | определяет, совпадает ли значение колонки `property` с заданным шаблоном |

---

`funcArgs`

Массив значений аргументов для функции фильтрации, заданной в свойстве `funcType`. Порядок значений в массиве `funcArgs` должен соответствовать порядку параметров функции `funcType`.

---

`name`

Имя фильтра или группы фильтров. Необязательное свойство.

---

`modelName`

Название модели, для которой выполняется фильтрация. Необязательное свойство. Указывается, если фильтрация выполняется по колонкам связанной модели.

---

`assocProperty`

Колонка связанной модели, по которой осуществляется связь с основной моделью. В качестве колонки для связи у основной модели выступает первичная колонка.

---

`operation`

Тип операции фильтрации. Необязательный параметр. Может принимать значения из перечисления `Terrasoft.FilterOperation`. По умолчанию имеет значение `Terrasoft.FilterOperation.General`.

Возможные значения (`Terrasoft.FilterOperation`)

| | |
|---|---|
| `General` | стандартная фильтрация |
| `Any` | фильтрация с применением фильтра `exists` |

---

`compareType`

Тип операции сравнения в фильтре. Необязательный параметр. Принимает значения из перечисления `Terrasoft.ComparisonType`. По умолчанию — `Terrasoft.ComparisonType.Equal`.

Возможные значения (`Terrasoft.ComparisonType`)

| | |
|---|---|
| `Equal` | равно |
| `LessOrEqual` | меньше или равно |
| `NotEqual` | не равно |
| `Greater` | больше |
| `GreaterOrEqual` | больше или равно |
| `Less` | меньше |

## Свойство SyncOptions.ModelDataImportConfig.QueryFilter

Доступно в приложении, начиная с версии 7.12.1, и в мобильном приложении, начиная с версии 7.12.3.

Свойство синхронизации `QueryFilter` позволяет настроить фильтрацию данных указанной модели при импорте с помощью сервиса работы с данными DataService. Ранее для фильтрации данных использовалось свойство `SyncFilter`, а импорт выполнялся с помощью сервиса работы с данными OData.

**Важно.** Импорт данных с помощью сервиса работы с данными DataService доступен только для платформ Android и iOS. Для платформы Windows используется OData.

Формат фильтра `QueryFilter` представляет собой набор параметров в виде JSON-объекта, передаваемых в запросе к сервису работы с данными DataService.

**Пример exists-фильтра**

```
{
  "SyncOptions": {
     "ModelDataImportConfig": [
        {
           "Name": "ActivityParticipant",
           "QueryFilter": {
              "logicalOperation": 0,
              "filterType": 6,
              "rootSchemaName": "ActivityParticipant",
              "items": {
                 "ActivityFilter": {
                    "filterType": 5,
                    "leftExpression": {
                       "expressionType": 0,
                       "columnPath": "Activity.[ActivityParticipant:Activity].Id"
                    },
                    "subFilters": {
                       "logicalOperation": 0,
                       "filterType": 6,
                       "rootSchemaName": "ActivityParticipant",
                       "items": {
                          "ParticipantFilter":{
                             "filterType": 1,
                             "comparisonType": 3,
                             "leftExpression": {
                                "expressionType": 0,
                                "columnPath": "Participant"
                             },
                             "rightExpression": {
                                "expressionType": 1,
                                "functionType": 1,
                                "macrosType": 2
                             }
                          }
                       }
                    }
                 }
              }
           }
        }
     ]
  }
}
```

# Настроить меню мобильного

# Настроить меню мобильного приложения

 Сложный

**Пример.** Настроить меню мобильного приложения, состоящего из двух групп — основной группы и группы [ *Продажи* ].

## Реализация примера

**Свойство** `ModuleGroups`

```
// Групы модулей мобильного приложения.
"ModuleGroups": {
    // Настройка группы основного меню.
    "main": {
        // Позиция группы в главном меню.
        "Position": 0
    },
    // Настройка группы меню [Продажи].
    "sales": {
        // Позиция группы в главном меню.
        "Position": 1
    }
}
```

# Настроить стартовую страницу и разделы меню мобильного приложения

 Сложный

**Пример.** Настроить разделы приложения следующим образом:

1. Разделы основного меню: [ *Контакты* ], [ *Контрагенты* ].
2. Стартовая страница приложения: раздел [ *Контакты* ].

В блоке [ *LocalizableStrings* ] схемы манифеста должны быть созданы строки содержащие заголовки разделов:

- `ContactSectionTitle` со значением "Контакты".
- `AccountSectionTitle` со значением "Контрагенты".

# Реализация примера

**Свойство** `Modules`

```
// Модули мобильного приложения.
"Modules": {
    // Раздел "Контакт".
    "Contact": {
        // Группа меню приложения, в которой размещается раздел.
        "Group": "main",
        // Название модели, которая предоставляет данные раздела.
        "Model": "Contact",
        // Позиция раздела в группе меню.
        "Position": 0,
        // Заголовок раздела.
        "Title": "ContactSectionTitle",
        // Подключение пользовательского изображения к разделу.
        "Icon": {
            // Уникальный идентификатор изображения.
            "ImageId": "4c1944db-e686-4a45-8262-df0c7d080658"
        },
        // Подключение пользовательского изображения к разделу.
        "IconV2": {
            // Уникальный идентификатор изображения.
            "ImageId": "9672301c-e937-4f01-9b0a-0d17e7a2855c"
        },
        // Признак отображения в меню.
        "Hidden": false
    },
    // Раздел "Контрагент".
    "Account": {
        // Группа меню приложения, в которой размещается раздел.
        "Group": "main",
        // Название модели, которая предоставляет данные раздела.
        "Model": "Account",
        // Позиция раздела в группе меню.
        "Position": 1,
        // Заголовок раздела.
        "Title": "AccountSectionTitle",
        // Подключение пользовательского изображения к разделу.
        "Icon": {
            // Уникальный идентификатор изображения.
            "ImageId": "c046aa1a-d618-4a65-a226-d53968d9cb3d"
        },
        // Подключение пользовательского изображения к разделу.
        "IconV2": {
```

```
            // Уникальный идентификатор изображения.
            "ImageId": "876320ef-c6ac-44ff-9415-953de17225e0"
        },
        // Признак отображения в меню.
        "Hidden": false
    }
}
```

# Настроить конфигурацию моделей

Сложный

**Пример.** Добавить в манифест конфигурацию следующих моделей:

1. `Контакт` — указать названия схем страниц реестра, просмотра и редактирования, требуемые модели, модули расширения модели и страниц модели.
2. `Адрес контакта` — указать только модуль расширения модели.

## Реализация примера

**Свойство** `Models`

```
// Импортируемые модели.
"Models": {
    // Модель "Контакт"
    "Contact": {
        // Схема страницы реестра.
        "Grid": "MobileContactGridPage",
        // Схема страницы просмотра.
        "Preview": "MobileContactPreviewPage",
        // Схема страницы записи.
        "Edit": "MobileContactEditPage",
        // Названия моделей, от которых зависит модель "Контакт".
        "RequiredModels": [
            "Account", "Contact", "ContactCommunication", "CommunicationType", "Department",
            "ContactAddress", "AddressType", "Country", "Region", "City", "ContactAnniversary",
            "AnniversaryType", "Activity", "SysImage", "FileType", "ActivityPriority",
            "ActivityType", "ActivityCategory", "ActivityStatus"
        ],
        // Расширения модели.
        "ModelExtensions": [
            "MobileContactModelConfig"
        ],
```

```
        // Расширения страниц модели.
        "PagesExtensions": [
            "MobileContactRecordPageSettingsDefaultWorkplace",
            "MobileContactGridPageSettingsDefaultWorkplace",
            "MobileContactActionsSettingsDefaultWorkplace",
            "MobileContactModuleConfig"
        ]
    },
    // Модель "Адреса контактов".
    "ContactAddress": {
        // Страницы реестра, просмотра и редактирования сгенерированы автоматически.
        // Расширения модели.
        "ModelExtensions": [
            "MobileContactAddressModelConfig"
        ]
    }
}
```

# Использовать функцию поиска подстроки для поиска данных

Сложный

**Пример.** Для поиска данных использовать функцию поиска подстроки.

## Реализация примера

**Свойство** `PreferedFilterFuncType`

```
// Для поиска данных используется функция поиска подстроки.
"PreferedFilterFuncType": "Terrasoft.FilterFunctions.SubStringOf"
```

**Важно.** Если в секции `PreferedFilterFuncType` в качестве функции фильтрации данных задается функция, отличная от `Terrasoft.FilterFunctions.StartWith`, то при поиске в таблицах БД индексы использоваться не будут.

# Загрузить данные моделей при синхронизации

Сложный

**Пример.** При синхронизации в мобильное приложение должны загружаться данные для таких моделей:

1. `Активность` — загружаются все колонки. Выполняется фильтрация модели - загружаются только те активности, у которых участником является текущий пользователь.
2. `Тип активности` — загружается полная модель.

# Реализация примера

**Свойство** `SyncOptions`

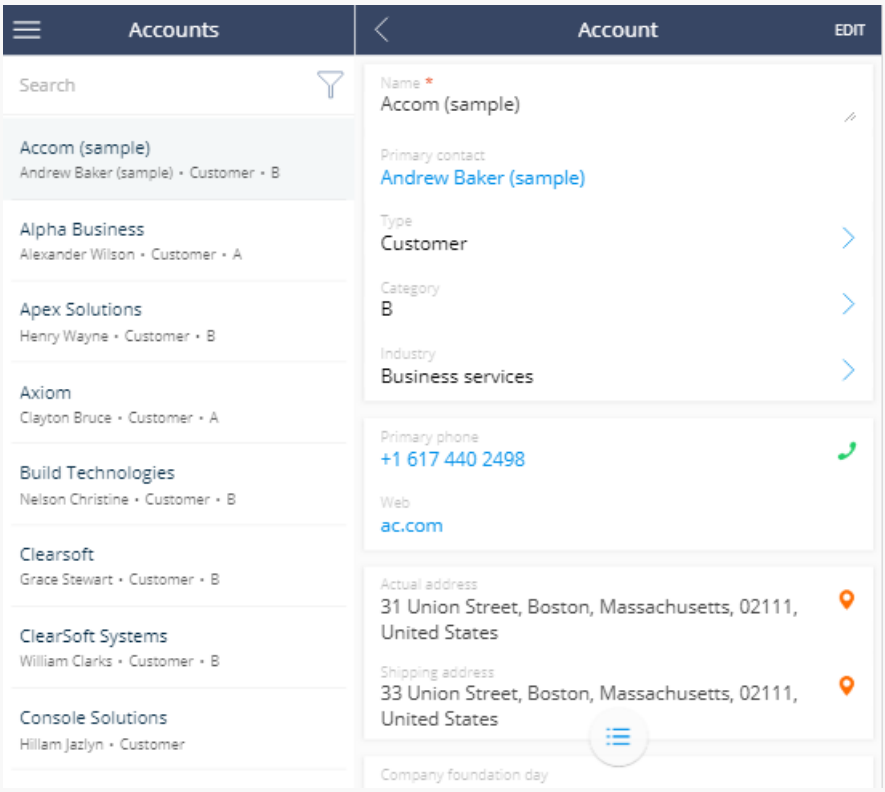
```
// Настройки синхронизации.
"SyncOptions": {
    // Количество страниц, импортируемых в одном потоке.
    "ImportPageSize": 100,
    // Количество потоков импорта.
    "PagesInImportTransaction": 5,
    // Массив импортируемых системных настроек.
    "SysSettingsImportConfig": [
        "SchedulerDisplayTimingStart", "PrimaryCulture", "PrimaryCurrency", "MobileApplicationMo
    ],
    // Массив импортируемых системных справочников.
    "SysLookupsImportConfig": [
        "ActivityCategory", "ActivityPriority", "ActivityResult", "ActivityResultCategory", "Act
    // Массив моделей, для которых будут загружаться данные при синхронизации.
    "ModelDataImportConfig": [
        // Конфигурирование модели Activity.
        {
            "Name": "Activity",
            // Фильтр, накладываемый на модель при импорте.
            "SyncFilter": {
                // Название колонки модели, по которой выполняется фильтрация.
                "property": "Participant",
                // Название модели, для которой выполняется фильтрация.
                "modelName": "ActivityParticipant",
                // Колонка связанной модели, по которой осуществляется связь с основной моделью.
                "assocProperty": "Activity",
                // Тип операции фильтрации.
                "operation": "Terrasoft.FilterOperations.Any",
                // Для фильтрации используется макрос.
                "valueIsMacros": true,
                // Значение для фильтрации колонки – идентификатор и имя текущего контакта.
                "value": "Terrasoft.ValueMacros.CurrentUserContact"
```

```
            },
            // Массив колонок модели, для которых импортируются данные.
            "SyncColumns": [
                "Title", "StartDate", "DueDate", "Status", "Result", "DetailedResult", "Activity
            ]
        },
        // Модель ActivityType загружается полностью.
        {
            "Name": "ActivityType",
            "SyncColumns": []
        }
    ]
}
```

# Отобразить страницу на планшете во весь экран

Легкий

При просмотре страницы раздела мобильного приложения Creatio на планшете по умолчанию срабатывает режим отображения реестра раздела в левой области экрана.




**Пример.** Отобразить страницу на планшете во весь экран.

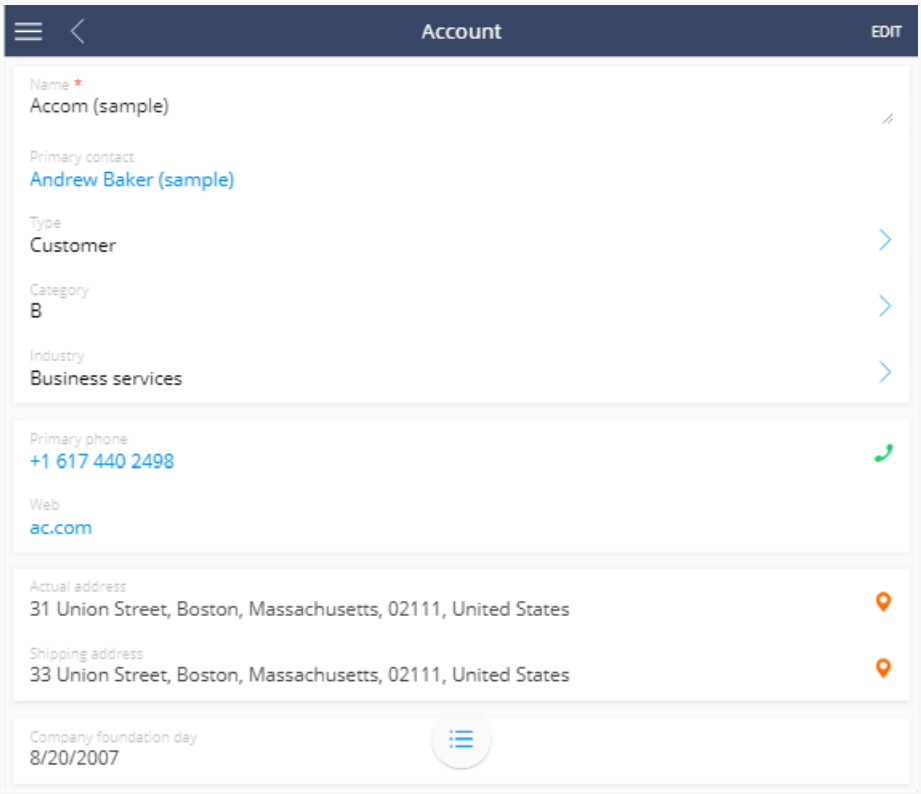
# Реализация примера

Для отображения страницы раздела во весь экран необходимо внести изменения в манифест мобильного приложения, добавив в него свойство `TabletViewMode` со значением "`SinglePage`".

**Свойство** `TabletViewMode`

```
{
    "TabletViewMode": "SinglePage",
    "CustomSchemas": [],
    "SyncOptions": {
        "SysSettingsImportConfig": [],
        "ModelDataImportConfig": []
    },
    "Modules": {},
    "Models": {}
}
```

После сохранения схемы и перезагрузки мобильного приложения страница раздела на планшете будет отображаться во весь экран.



# Добавить стандартную деталь с колонками

Сложный

Для добавления детали в раздел мобильного приложения Creatio необходимо использовать мастер мобильного приложения.

Однако, если объект детали не является объектом какого-либо раздела мобильного приложения Creatio, то на странице детали вместо значений будет отображаться идентификатор связанной записи раздела. Для отображения значений необходимо выполнить конфигурирование схемы страницы детали.

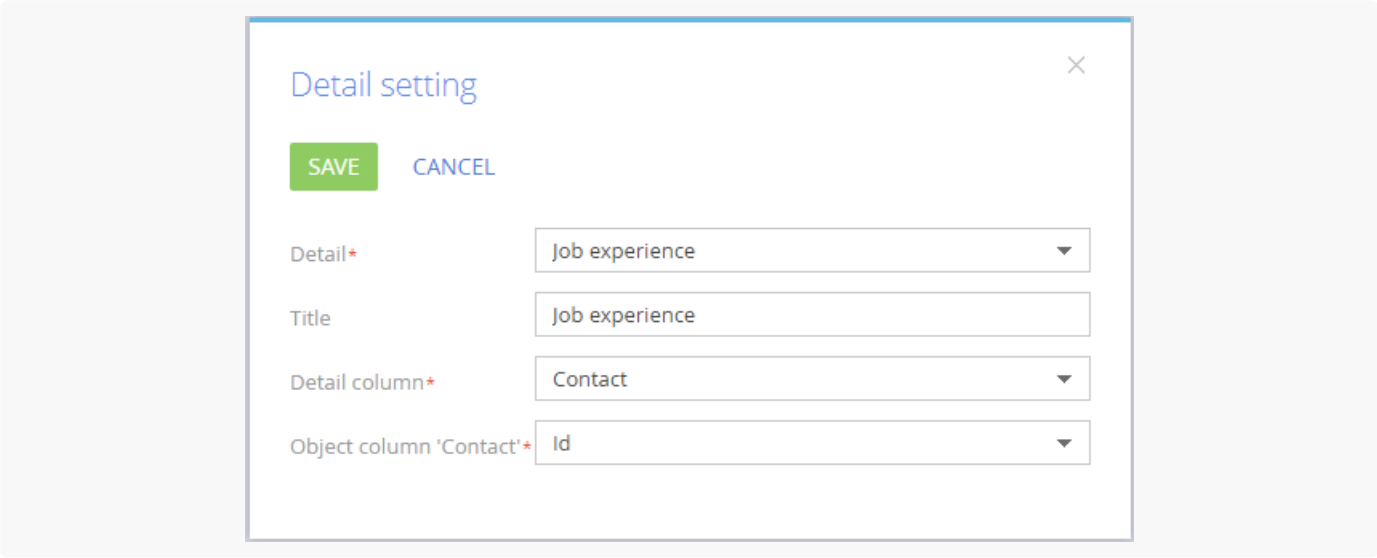
**Пример.** На страницу записи раздела [ *Контакты* ] мобильного приложения добавить деталь [ *Карьера* ]. В качестве основной отображать колонку [ *Должность* ].

## Алгоритм реализации примера

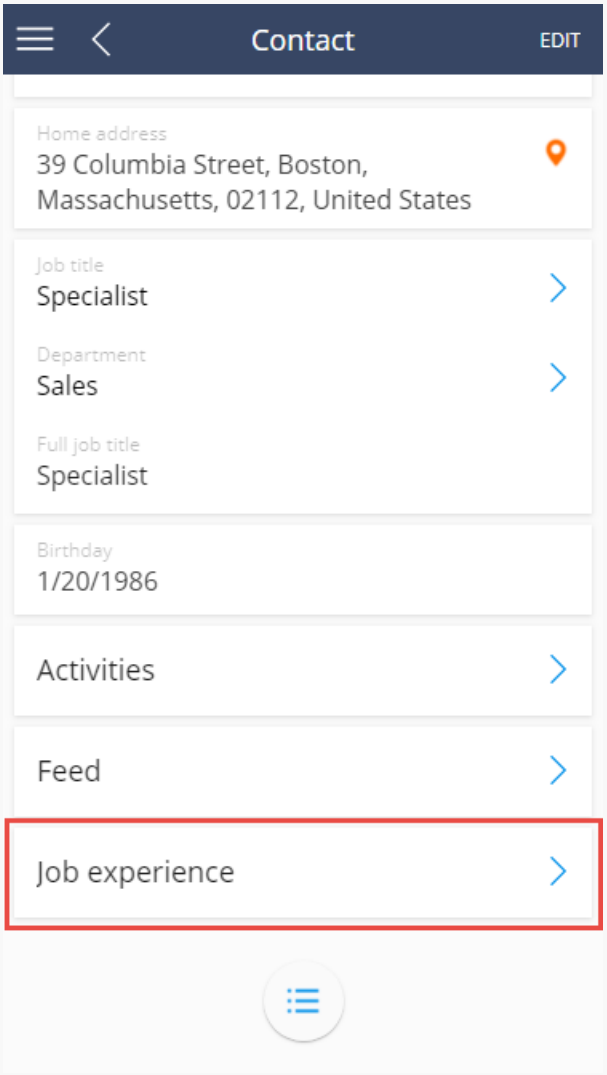
### 1. Добавить деталь [ *Карьера* ] с помощью мастера мобильного приложения

Чтобы добавить деталь на страницу записи, используйте мастер мобильного приложения. Для этого:

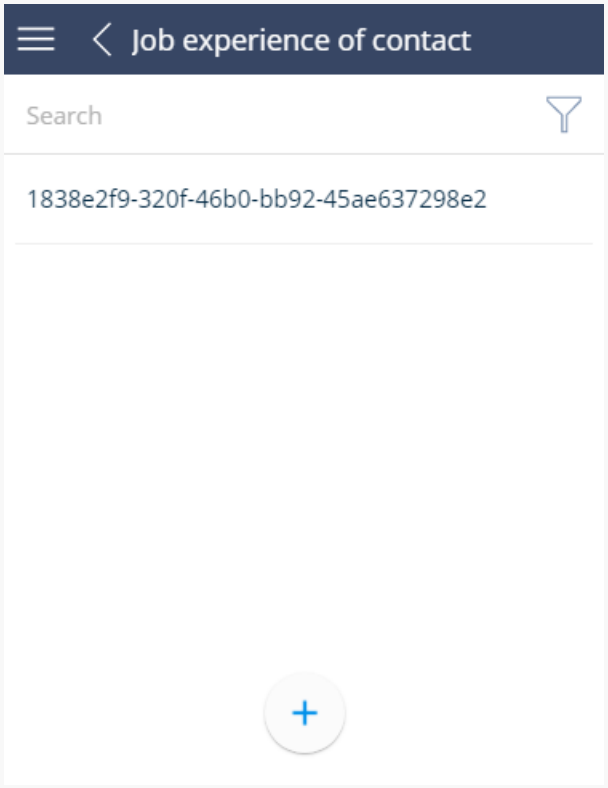
1. Откройте нужное рабочее место, например, [ *Основное рабочее место* ] ([ *Main workplace* ]) и нажмите кнопку [ *Настроить разделы* ] ([ *Set up sections* ]).
2. Выберите раздел [ *Контакты* ] и нажмите кнопку [ *Настроить детали* ] ([ *Details setup* ]).
3. Настройте деталь [ *Карьера* ] ([ *Job experience* ]).




После сохранения результатов настройки детали, раздела и рабочего места в мобильном приложении отобразится деталь [ *Карьера* ] ([ *Job experience* ]).



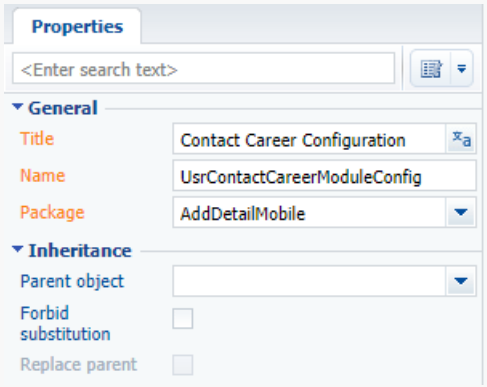

Однако, поскольку объект детали [ *Карьера* ] не является объектом раздела мобильного приложения Creatio, то на странице детали отображается значение основной колонки [ *Контакт* ] (идентификатор связанной записи контакта).




## 2. Создать схему модуля, в которой выполнить конфигурирование реестра детали

В разделе [ *Конфигурация* ] приложения Creatio в пользовательском пакете создайте клиентский модуль со следующими свойствами:

- [ *Заголовок* ] ([ *Title* ]) — "Настройки карьеры контакта" ("Contact Career Configuration").
- [ *Название* ] ([ *Name* ]) — "UsrContactCareerModuleConfig".




Добавьте в схему модуля исходный код.

```
UsrContactCareerModuleConfig
```

```
// Установка колонки [Должность] в качестве первичной.
Terrasoft.sdk.GridPage.setPrimaryColumn("ContactCareer", "JobTitle");
// Добавление колонки [Должность] в коллекцию первичных колонок.
Terrasoft.sdk.RecordPage.addColumn("ContactCareer", {
        name: "JobTitle",
        position: 1
    }, "primaryColumnSet");
// Удаление предыдущей первичной колонки [Контакт] из коллекции первичных колонок.
Terrasoft.sdk.RecordPage.removeColumn("ContactCareer", "Contact", "primaryColumnSet");
```

Здесь:

- `ContactCareer` — название таблицы, которая соответствует детали (как правило оно совпадает с названием объекта детали).
- `Job Title` — название колонки, которую требуется отобразить на странице.

## 3. Подключить схему модуля в манифесте мобильного приложения

Для применения настроек реестра детали, выполненный в модуле `UsrContactCareerModuleConfig`, выполните следующие шаги:

1. Откройте в дизайнере клиентского модуля схему манифеста мобильного приложения `MobileApplicationManifestDefaultWorkplace`. Эта схема создается в пользовательском пакете мастером мобильного приложения.
2. Добавьте модуль `UsrContactCareerModuleConfig` в секцию `PagesExtensions` модели `ContactCareer`.

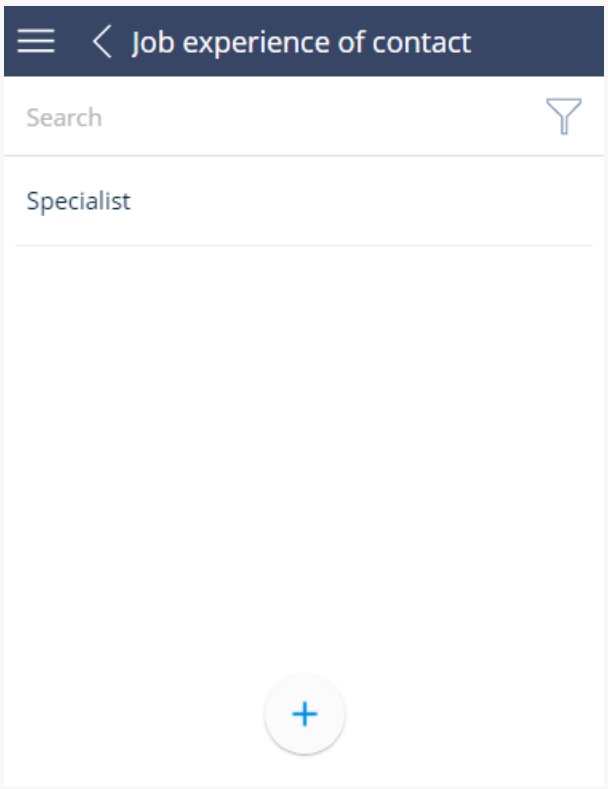
**ContactCareer**

```
{
    "SyncOptions": {
        ...
    },
    "Modules": {},
    "Models": {
        "ContactCareer": {
            "RequiredModels": [
                ...
            ],
            "ModelExtensions": [],
            "PagesExtensions": [
                ...
                "UsrContactCareerModuleConfig"
            ]
        },
        ...
```

```
        }
    }
```

3. Сохраните схему манифеста мобильного приложения.

В результате выполнения примера на странице детали [ *Карьера* ] ([ *Job experience* ]) будут отображаться записи по колонке [ *Должность* ].




**Важно.** Чтобы в мобильном приложении отображались сконфигурированные колонки, необходимо выполнить очистку кэша мобильного приложения. В некоторых случаях предварительно следует выполнить компиляцию приложения Creatio.

# Добавить пользовательский дашборд в мобильное приложение

Сложный

В приложении версии 7.10.3 (версия 7.10.5 мобильного приложения) добавлена поддержка итогов в мобильном приложении. Для получения настроек и данных итогов используется сервис `AnalyticsService`. Поддерживаются следующие дашборды — график, показатель, список и шкала.

Для добавления пользовательского типа дашборда в мобильное приложение необходимо:

1. Реализовать интерфейс настройки дашборда в приложении Creatio.

2. Добавить экземпляр реализованного пользовательского дашборда в приложение.
3. Реализовать отображение дашборда в мобильном приложении.

**Важно.** В данной статье описывается только реализация отображения дашборда в мобильном приложении.

Чтобы пользовательский тип дашборда отображался в мобильном приложении необходимо:

1. Реализовать получение данных пользовательского типа дашборда.
2. Добавить реализацию отображения дашборда в мобильном приложении.

**Пример.** На страницу итогов мобильного приложения добавить пользовательский дашборд, отображающий текущие дату и время.

# Алгоритм реализации примера

## 1. Реализация получения данных пользовательского типа дашборда

Чтобы осуществить получение данных каждого пользовательского типа дашборда необходимо создать класс, который должен реализовать интерфейс `IDashboardItemData` или наследоваться от базового класса `BaseDashboardItemData`. Также для этого класс должен быть декорирован атрибутом `DashboardItemData`. Для реализации класса необходимо в пользовательский пакет добавить схему [ *Исходный код* ].

Реализовать класс получения данных для пользовательского типа дашборда `CustomDashboardItem`.

**CustomDashboardItem**

```
namespace Terrasoft.Configuration
{
    using System;
    using Newtonsoft.Json.Linq;
    using Terrasoft.Core;

    // Атрибут, указывающий пользовательский тип дашборда.
    [DashboardItemData("CustomDashboardItem")]
    public class CustomDashboardItemData : BaseDashboardItemData
    {
        // Конструктор класса.
        public CustomDashboardItemData(string name, JObject config, UserConnection userConnectio
            : base(name, config, userConnection, timeZoneOffset)
        {

        }
```

```
        // Метод, возвращающий необходимые данные.
        public override JObject GetJson()
        {
            JObject itemObject = base.GetJson();
            itemObject["customValue"] = DateTime.Now.ToString();
            return itemObject;
        }
    }
}
```

## 2. Реализация отображения пользовательского типа дашборда

### 2.1. Добавить класс отображения данных

Для этого необходимо в пользовательском пакете создать клиентский модуль (например, `UsrMobileCustomDashboardItem`). В созданном модуле необходимо реализовать класс, расширяющий базовый класс `Terrasoft.configuration.controls.BaseDashboardItem`.

**UsrMobileCustomDashboardItem**

```
Ext.define("Terrasoft.configuration.controls.CustomDashboardItem", {
    extend: "Terrasoft.configuration.controls.BaseDashboardItem",
    // Отображает значение, переданное через свойство customValue
    updateRawConfig: function(config) {
        this.innerHtmlElement.setHtml(config.customValue);
    }

});
```

### 2.2. Добавить новый тип и реализующий его класс в перечисление `Terrasoft.DashboardItemClassName`

Для этого в модуле, созданном на предыдущем шаге, необходимо добавить исходный код.

**CustomDashboardItem**

```
Terrasoft.DashboardItemClassName.CustomDashboardItem = "Terrasoft.configuration.controls.CustomD
```

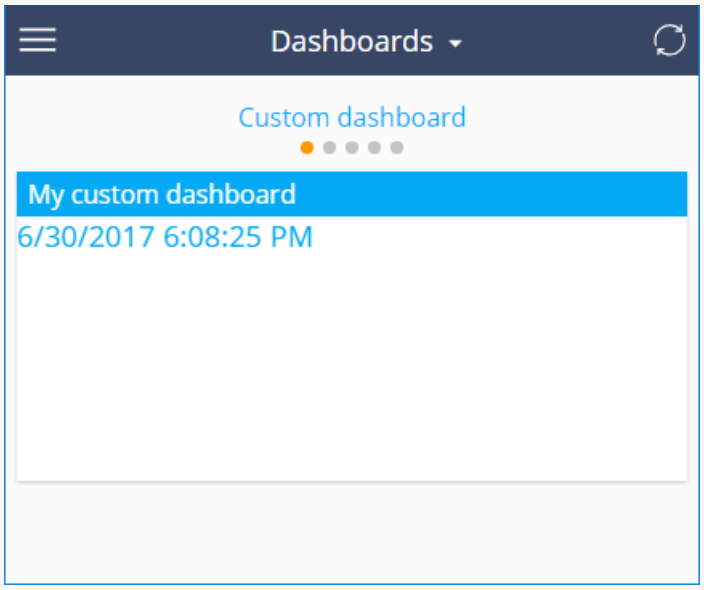
### 2.3. Добавить название созданной клиентской схемы в манифест мобильного приложения

В файле манифеста мобильного приложения необходимо в массив `CustomSchemas` добавить название созданной схемы модуля.

`CustomSchemas`

```
{
    "SyncOptions": {
        ...
    },
    "CustomSchemas": ["UsrMobileCustomDashboardItem"],
    "Modules": {...},
    "Models": {...}
}
```

После сохранения всех изменений дашборд будет отображаться в разделе [ *Итоги* ] мобильного приложения.




**Важно.** Чтобы дашборд отображался в мобильном приложении, он обязательно должен быть добавлен в основное приложение Creatio, с которым синхронизирована мобильная версия приложения.

# Добавить кнопку для отображения имени контакта

Сложный

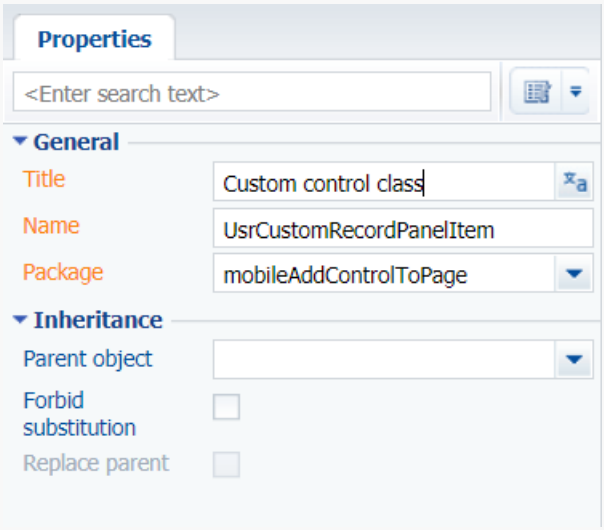
**Пример.** На страницу записи раздела [ *Контакты* ] мобильного приложения добавить кнопку, нажатие на которую будет показывать сообщение с полным именем контакта.

# Алгоритм реализации примера

## 1. Создать пользовательский класс-наследник `Terrasoft.RecordPanelItem`

В разделе [ *Конфигурация* ] приложения Creatio в пользовательском пакете создайте клиентский модуль со следующими свойствами:

- [ *Заголовок* ] ([ *Title* ]) — "`Класс пользовательского элемента управления`" ("`Custom control class`").
- [ *Название* ] ([ *Name* ]) — "`UsrCustomRecordPanelItem`".




В модуль добавьте исходный код.

**`CustomRecordPanelItem`**

```
Ext.define("Terrasoft.controls.CustomRecordPanelItem", {
    extend: "Terrasoft.RecordPanelItem",
    xtype: "cftestrecordpanelitem",
    // Конфигурационный объект создаваемого элемента.
    config: {
        items: [
            {
                xtype: "container",
                layout: "hbox",
                items: [
                    {
                        xtype: "button",
                        id: "clickMeButton",
                        text: "Full name",
                        flex: 1
                    }
```

```
                ]
            }
        ]
    },
    // Метод инициализирует созданный элемент и добавляет метод-обработчик нажатия кнопки.
    initialize: function() {
        var clickMeButton = Ext.getCmp("clickMeButton");
        clickMeButton.element.on("tap", this.onClickMeButtonClick, this);
    },
    // Метод-обработчик нажатия кнопки.
    onClickMeButtonClick: function() {
        var record = this.getRecord();
        Terrasoft.MessageBox.showMessage(record.getPrimaryDisplayColumnValue());
    }
});
```

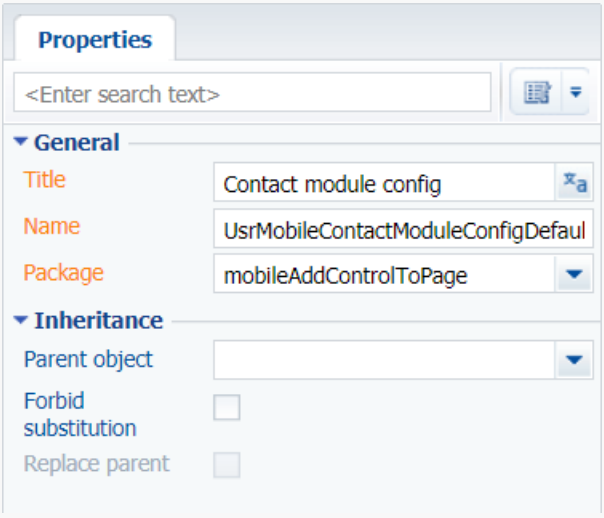
В классе описан конфигурационный объект созданного элемента управления и два метода:

- `initialize()` - метод-обработчик события нажатия кнопки;
- `onClickMeButtonClick()` - метод, который инициализует созданный элемент и присваивает событию нажатия на кнопку ссылку на метод-обработчик.

## 2. Создать схему модуля, в которой выполнить конфигурирование страницы раздела

В разделе [ *Конфигурация* ] приложения Creatio в пользовательском пакете создайте клиентский модуль со следующими свойствами:

- [ *Заголовок* ] ([ *Title* ]) — "Конфигурация раздела контактов" ("Contact module config").
- [ *Название* ] ([ *Name* ]) — "UsrMobileContactModuleConfigDefaultWorkplace".




Добавьте в схему модуля исходный код.

**UsrMobileContactModuleConfigDefaultWorkplace**

```
Terrasoft.sdk.RecordPage.addPanelItem("Contact", {
    xtype: "cftestrecordpanelitem",
    position: 1,
    componentConfig: {
    }
});
```

Здесь вызывается метода `addPanelItem()` класса `Terrasoft.sdk.RecordPage`, с помощью которого созданный элемент добавляется на страницу раздела.

## 3. Подключить схемы модулей в манифесте мобильного приложения

Для применения настроек страницы раздела, выполненных в модуле `UsrMobileContactModuleConfigDefaultWorkplace`, выполните следующие шаги:

1. Откройте в дизайнере клиентского модуля схему манифеста мобильного приложения `MobileApplicationManifestDefaultWorkplace`. Эта схема создается в пользовательском пакете мастером мобильного приложения.
2. Добавьте модуль `UsrCustomRecordPanelItem` в секцию `CustomSchemas`, а модуль `UsrContactCareerModuleConfig` в секцию `PagesExtensions` модели `Contact`.

   **Добавление модулей**

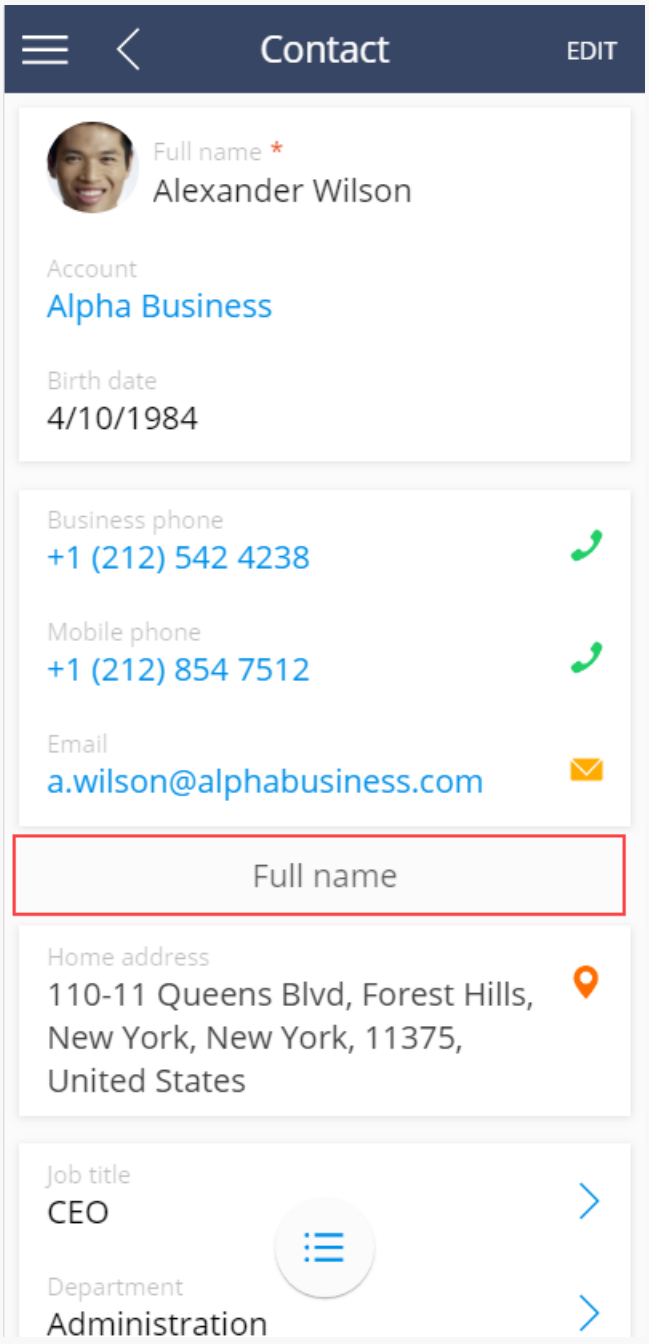
   ```
   {
       "CustomSchemas": [
           "UsrCustomRecordPanelItem.js"
       ],
       "SyncOptions": {},
       "Modules": {},
       "Models": {
           "Contact": {
               "RequiredModels": [],
               "ModelExtensions": [],
               "PagesExtensions": [
                   "UsrMobileContactModuleConfigDefaultWorkplace.js"
               ]
           }
       }
   }
   ```

3. Сохраните схему манифеста мобильного приложения.

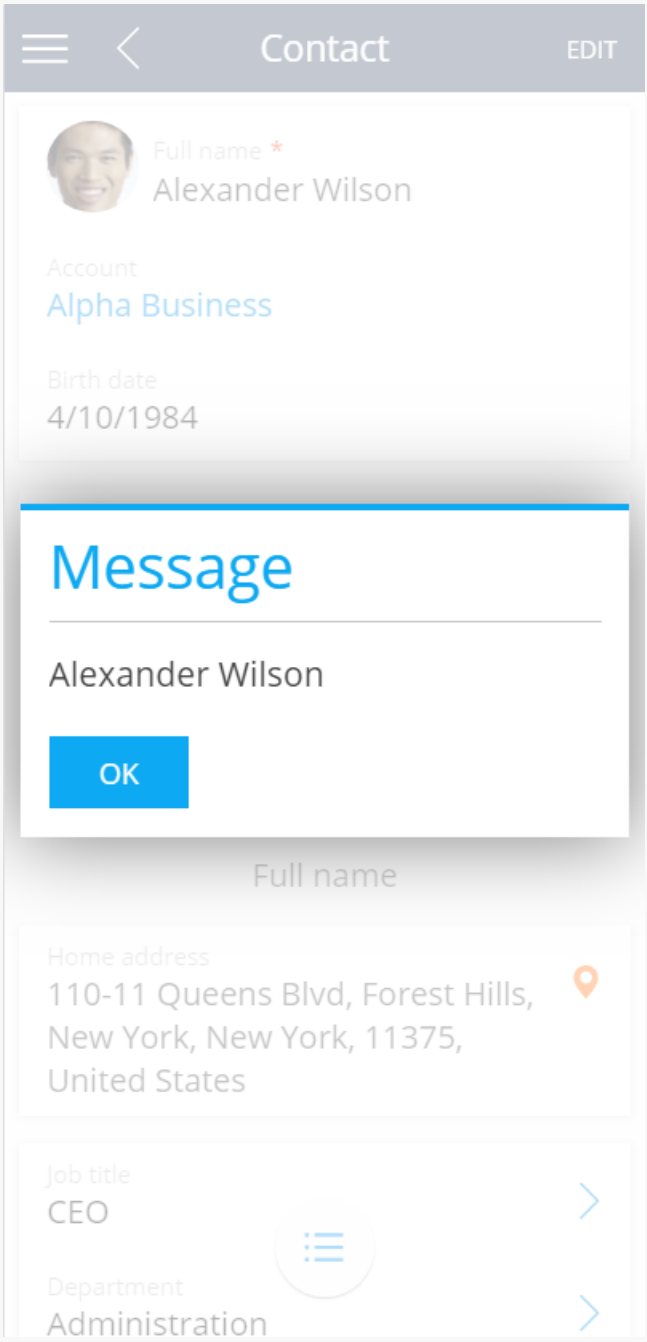
В результате выполнения примера на странице контакта появится элемент управления, при нажатии на

который отобразится сообщение с полным именем контакта.

Результат выполнения кейса. Добавление кнопки




Результат выполнения кейса. Нажатие на кнопку




# Свойство ModuleGroups JS

Сложный

Группы модулей приложения. Используется для настройки группы меню. Описывает верхнеуровневую настройку групп `Position`.

## Свойство конфигурационного объекта

`Position`

Позиция группы в главном меню. Начинается с 0.

# Свойство Modules JS

Сложный

Модули мобильного приложения. Модуль представляет собой раздел приложения. Каждый модуль в свойстве `Modules` конфигурационного объекта манифеста описывается конфигурационным объектом со свойствами, приведенными в таблице. Имя конфигурационного объекта раздела должно совпадать с названием модели, которая предоставляет данные раздела.

## Свойства конфигурационного объекта

---

`Group`

Группа меню приложения, в которой размещается раздел. Задается строкой с названием соответствующего раздела меню из свойства `ModuleGroups` конфигурационного объекта манифеста.

---

`Model`

Название модели, которая предоставляет данные раздела. Задается строкой с названием одной из моделей, объявленных в свойстве `Models` конфигурационного объекта манифеста.

---

`Position`

Позиция раздела в группе главного меню. Задается числовым значением, начиная с 0.

---

`Title`

Заголовок раздела. Строка с названием локализованного значения заголовка раздела. Локализованное значение заголовка раздела должно быть добавлено в блок [ *LocalizableStrings* ] схемы манифеста.

---

`Icon`

Свойство, предназначенное для подключения пользовательского изображения к разделу в меню пользовательского интерфейса версии 1.

---

`IconV2`

Свойство, предназначенное для подключения пользовательского изображения к разделу в меню пользовательского интерфейса версии 2.

---

`Hidden`

Признак, отображается ли данный раздел в меню (`true` — скрыт, `false` — отображается). Необязательное свойство. По умолчанию — `false`.

# Свойство Icons JS

Сложный

Свойство предназначено для подключения к мобильному приложению пользовательских изображений.

Задается массивом конфигурационных объектов, каждый из которых имеет свойства, приведенные в таблице.

## Свойства конфигурационного объекта

---

`ImageListId`

Идентификатор списка изображений.

---

`ImageId`

Идентификатор подключаемого изображения из списка `ImageListId`.

**Подключение пользовательских изображений**

```
// Подключение пользовательских изображений.
"Icons": [
    {
        // Идентификатор списка изображений.
        "ImageListId": "69c7829d-37c2-449b-a24b-bcd7bf38a8be",
        // Идентификатор подключаемого изображения.
        "ImageId": "4c1944db-e686-4a45-8262-df0c7d080658"
    }
]
```

# Свойства DefaultModuleImageId и DefaultModuleImageIdV2 JS

Сложный

Свойства предназначены для установки уникальных идентификаторов изображений по умолчанию для вновь создаваемых разделов или для разделов, у которых не указаны идентификаторы изображений в свойствах `Icon` или `IconV2` свойства `Modules` конфигурационного объекта манифеста.

**Установка уникальных идентификаторов изображений**

```
//Идентификатор изображения по умолчанию для пользовательского интерфейса V1.
"DefaultModuleImageId": "423d3be8-de6b-4f15-a81b-ed454b6d03e3",
//Идентификатор изображения по умолчанию для пользовательского интерфейса V2.
"DefaultModuleImageIdV2": "1c92d522-965f-43e0-97ab-2a7b101c03d4"
```

# Свойство Models JS

Сложный

Содержит импортируемые модели приложения. Каждая модель в свойстве описывается конфигурационным объектом с соответствующим именем. Свойства конфигурационного объекта модели представлены в таблице.

## Свойства конфигурационного объекта

---

`Grid`

Название схемы страницы реестра модели. Страница будет сгенерирована автоматически с именем `Mobile[Название_модели][Тип_страницы]Page`. Не обязателен для заполнения.

---

`Preview`

Название схемы страницы просмотра элемента модели. Страница будет сгенерирована автоматически с именем `Mobile[Название_модели][Тип_страницы]Page`. Не обязателен для заполнения.

---

`Edit`

Название схемы страницы элемента модели. Страница будет сгенерирована автоматически с именем `Mobile[Название_модели][Тип_страницы]Page`. Не обязателен для заполнения.

---

`RequiredModels`

Названия моделей, от которых зависит данная модель. Необязательное свойство. Здесь перечисляются все модели, колонки которых добавляются в текущую модель, а также колонки, на которые у текущей модели есть внешние ключи.

---

`ModelExtensions`

Расширения модели. Необязательное свойство. Представляет собой массив названий схем, в которых реализуются дополнительные настройки для модели (например, добавление в модель бизнес-правил, событий, значений по умолчанию для полей и т.д.).

`PagesExtensions`

Расширения страниц модели. Необязательное свойство. Представляет собой массив названий схем, в которых реализуются дополнительные настройки для различных типов страниц модели (добавление деталей, установка заголовков и т.д.).

# Свойство PreferedFilterFuncType JS

Сложный

Свойство предназначено для явного определения операции, которая будет использоваться при поиске и фильтрации данных в реестре (в разделах, деталях, справочниках). Значение для свойства задается перечислением `Terrasoft.FilterFunctions`.

## Функции фильтрации (Terrasoft.FilterFunctions)

`SubStringOf`

Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки `property`.

`ToUpper`

Приводит значения колонки, заданной в `property`, к верхнему регистру.

`EndsWith`

Проверяет, оканчивается ли значение колонки `property` значением, переданным в качестве аргумента.

`StartsWith`

Проверяет, начинается ли значение колонки `property` значением, переданным в качестве аргумента.

`Year`

Возвращает год по значению колонки `property`.

`Month`

Возвращает месяц по значению колонки `property`.

`Day`

Возвращает день по значению колонки `property`.

---

`In`

Проверяет вхождение значения колонки `property` в диапазон значений, переданных в качестве аргумента функции.

---

`NotIn`

Проверяет невхождение значения колонки `property` в диапазон значений, переданных в качестве аргумента функции.

---

`Like`

Определяет, совпадает ли значение колонки `property` с заданным шаблоном.

Если данное свойство явно не инициализировано в манифесте, то по умолчанию для поиска и фильтрации данных используется функция `Terrasoft.FilterFunctions.StartWith`, так как это обеспечивает использование соответствующих индексов в таблицах базы данных SQLite.

# Свойство CustomSchemas JS

Сложный

Свойство предназначено для подключения к мобильному приложению дополнительных схем (пользовательских схем с исходным кодом, написанным на JavaScript), расширяющих возможности приложения. Это могут быть, например, дополнительные классы, реализованные разработчиком в рамках проекта, либо утилитные классы, реализующие служебную функциональность, упрощающую работу разработчика, и т.д.

Значение свойства задается массивом с именами подключаемых пользовательских схем.

**Подключение дополнительных пользовательских схем регистрации действий и утилит**

```
// Подключение дополнительных пользовательских схем.
"CustomSchemas": [
    // Пользовательская схема регистрации действий.
    "MobileActionCheckIn",
    // Пользовательская схема утилит.
    "CustomMobileUtilities"
]
```

# Свойство SyncOptions JS

Сложный

Описывает параметры для настройки синхронизации данных. Содержит конфигурационный объект со свойствами, представленными в таблице.

## Свойства конфигурационного объекта для настроек синхронизации

---

`ImportPageSize`

Количество страниц, импортируемых в одном потоке.

---

`PagesInImportTransaction`

Количество потоков импорта.

---

`SysSettingsImportConfig`

Массив импортируемых системных настроек.

---

`SysLookupsImportConfig`

Массив импортируемых системных справочников.

---

`ModelDataImportConfig`

Массив моделей, для которых будут загружаться данные при синхронизации.

В массиве моделей `ModelDataImportConfig` для каждой модели можно указать дополнительные параметры синхронизации, список загружаемых колонок, а также условия фильтрации загружаемых данных модели. Если при синхронизации должна загружаться полная модель, в массиве просто указывается объект с именем модели. Если к модели должны применяться дополнительные условия при синхронизации, в массив `ModelDataImportConfig` добавляется конфигурационный объект со свойствами.

## Свойства конфигурационного объекта для настройки синхронизации модели

---

`Name`

Название модели (см. свойство `Models` конфигурационного объекта манифеста).

---

`SyncColumns`

Массив колонок модели, для которых импортируются данные. Помимо явно перечисленных колонок,

при синхронизации в обязательном порядке будут импортироваться системные колонки ( `CreatedOn` , `CreatedBy` , `ModifiedOn` , `ModifiedBy` ) и колонка, первичная для отображения.

---

`SyncFilter`

Фильтр, накладываемый на модель при импорте модели.

Фильтр `SyncFilter` , накладываемый на модель при импорте модели представляет собой конфигурационный объект со свойствами.

## Свойства конфигурационного объекта фильтра модели

---

`type`

Тип фильтра. Задается значением перечисления `Terrasoft.FilterTypes` . Необязательное свойство. По умолчанию `Terrasoft.FilterTypes.Simple` .

Возможные значения ( `Terrasoft.FilterTypes` )

| | |
|---|---|
| `Simple` | Фильтр с одним условием. |
| `Group` | Групповой фильтр с несколькими условиями. |

---

`logicalOperation`

Логическая операция объединения коллекции фильтров (для фильтров с типом `Terrasoft.FilterTypes.Group` ). Задается значением перечисления `Terrasoft.FilterLogicalOperations` . Значение по умолчанию - `Terrasoft.FilterLogicalOperations.And` .

Возможные значения ( `Terrasoft.FilterLogicalOperations` )

| | |
|---|---|
| `Or` | Логическая операция ИЛИ. |
| `And` | Логическая операция И. |

---

`subfilters`

Коллекция фильтров, применяемых к модели. Обязательное свойство для типа фильтра `Terrasoft.FilterTypes.Group` . Фильтры между собой объединяются логической операцией, указанной в свойстве `logicalOperation` . Каждый фильтр представляет собой конфигурационный объект фильтра.

---

`property`

Название колонки модели, по которой выполняется фильтрация. Обязательное свойство для типа фильтра `Terrasoft.FilterTypes.Simple`.

---

`valueIsMacrosType`

Признак, определяющий, является ли значение для фильтрации макросом. Необязательное свойство. Может принимать значения: `true`, если для фильтрации используется макрос, иначе — `false`.

---

`value`

Значение для фильтрации колонки, указанной в свойстве `property`. Обязательное свойство для типа фильтра `Terrasoft.FilterTypes.Simple`. Может задаваться непосредственно значением для фильтрации (в том числе, может быть `null`) либо макросом (для этого свойство `valueIsMacrosType` должно иметь значение `true`). Макросы, которые можно использовать в качестве значения свойства, содержатся в перечислении `Terrasoft.ValueMacros`.

Возможные значения (`Terrasoft.ValueMacros`)

| | |
|---|---|
| `CurrentUserContactId` | Идентификатор текущего пользователя. |
| `CurrentDate` | Текущая дата. |
| `CurrentDateTime` | Текущие дата и время. |
| `CurrentDateEnd` | Полная дата окончания текущей даты. |
| `CurrentUserContactName` | Имя текущего контакта. |
| `CurrentUserContact` | Идентификатор и имя текущего контакта. |
| `SysSettings` | Значение системной настройки. Имя системной настройки передается в свойстве `macrosParams`. |
| `CurrentTime` | Текущее время. |
| `CurrentUserAccount` | Идентификатор и имя контрагента текущего пользователя. |
| `GenerateUId` | Сгенерированный идентификатор. |

---

`macrosParams`

Значения, которые передаются в макрос в качестве параметра. Необязательное свойство. В настоящее время используется только для макроса `Terrasoft.ValueMacros.SysSettings`.

---

`isNot`

Определяет, применяется к фильтру оператор отрицания. Необязательное свойство. Принимает значение `true`, если к фильтру применяется оператор отрицания, иначе — `false`.

---

`funcType`

Тип функции, которая применяется к колонке модели, заданой в свойстве `property`. Необязательное свойство. Может принимать значения перечисления `Terrasoft.FilterFunctions`. Значения аргументов для функций фильтрации задаются в свойстве `funcArgs`. Значение, с которым сравнивается результат функции, задается свойством `value`.

Возможные значения (`Terrasoft.FilterFunctions`)

| | |
|---|---|
| `SubStringOf` | Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки `property`. |
| `ToUpper` | Приводит значения колонки, заданной в `property`, к верхнему регистру. |
| `EndsWith` | Проверяет, оканчивается ли значение колонки `property` значением, переданным в качестве аргумента. |
| `StartsWith` | Проверяет, начинается ли значение колонки `property` значением, переданным в качестве аргумента. |
| `Year` | Возвращает год по значению колонки `property`. |
| `Month` | Возвращает месяц по значению колонки `property`. |
| `Day` | Возвращает день по значению колонки `property`. |
| `In` | Проверяет вхождение значения колонки `property` в диапазон значений, переданных в качестве аргумента функции. |
| `NotIn` | Проверяет невхождение значения колонки `property` в диапазон значений, переданных в качестве аргумента функции. |
| `Like` | Определяет, совпадает ли значение колонки `property` с заданным шаблоном. |

---

`funcArgs`

Массив значений аргументов для функции фильтрации, заданной в свойстве `funcType`. Порядок значений в массиве `funcArgs` должен соответствовать порядку параметров функции `funcType`.

---

`name`

Имя фильтра или группы фильтров. Необязательное свойство.

---

`modelName`

Название модели, для которой выполняется фильтрация. Необязательное свойство. Указывается, если фильтрация выполняется по колонкам связанной модели.

---

`assocProperty`

Колонка связанной модели, по которой осуществляется связь с основной моделью. В качестве колонки для связи у основной модели выступает первичная колонка.

---

`operation`

Тип операции фильтрации. Необязательный параметр. Может принимать значения из перечисления `Terrasoft.FilterOperation`. По умолчанию имеет значение `Terrasoft.FilterOperation.General`.

Возможные значения (`Terrasoft.FilterOperation`)

| | |
|---|---|
| `General` | Стандартная фильтрация. |
| `Any` | Фильтрация с применением фильтра `exists`. |

---

`compareType`

Тип операции сравнения в фильтре. Необязательный параметр. Принимает значения из перечисления `Terrasoft.ComparisonType`. По умолчанию — `Terrasoft.ComparisonType.Equal`.

Возможные значения (`Terrasoft.ComparisonType`)

| | |
|---|---|
| `Equal` | Равно. |
| `LessOrEqual` | Меньше или равно. |
| `NotEqual` | Не равно. |
| `Greater` | Больше. |
| `GreaterOrEqual` | Больше или равно. |
| `Less` | Меньше. |

# Свойство SyncOptions.ModelDataImportConfig.QueryFilter

Доступно в приложении, начиная с версии 7.12.1, и в мобильном приложении, начиная с версии 7.12.3.

Свойство синхронизации `QueryFilter` позволяет настроить фильтрацию данных указанной модели при импорте с помощью сервиса работы с данными DataService. Ранее для фильтрации данных использовалось свойство `SyncFilter`, а импорт выполнялся с помощью сервиса работы с данными OData.

**Важно.** Импорт данных с помощью сервиса работы с данными DataService доступен только для платформ Android и iOS. Для платформы Windows используется OData.

Формат фильтра `QueryFilter` представляет собой набор параметров в виде JSON-объекта, передаваемых в запросе к сервису работы с данными DataService.

**Пример exists-фильтра**

```
{
  "SyncOptions": {
     "ModelDataImportConfig": [
        {
           "Name": "ActivityParticipant",
           "QueryFilter": {
              "logicalOperation": 0,
              "filterType": 6,
              "rootSchemaName": "ActivityParticipant",
              "items": {
                 "ActivityFilter": {
                    "filterType": 5,
                    "leftExpression": {
                       "expressionType": 0,
                       "columnPath": "Activity.[ActivityParticipant:Activity].Id"
                    },
                    "subFilters": {
                       "logicalOperation": 0,
                       "filterType": 6,
                       "rootSchemaName": "ActivityParticipant",
                       "items": {
                          "ParticipantFilter":{
                             "filterType": 1,
                             "comparisonType": 3,
                             "leftExpression": {
                                "expressionType": 0,
                                "columnPath": "Participant"
                             },
                             "rightExpression": {
                                "expressionType": 1,
                                "functionType": 1,
```

```
                                "macrosType": 2
                            }
                        }
                    }
                }
            }
        }
    }
  ]
 }
}
```

# Реестр мобильного приложения

Сложный

**Важно.** Актуально для мобильного приложения версии 7.11.1 и выше.

**SDK реестра** — это инструмент, позволяющий настраивать внешний вид реестра, сортировку, логику поиска и т. д. Он реализован в классе `Terrasoft.sdk.GridPage`.

# Настроить реестр раздела

Сложный

**Пример.** Настроить реестр раздела [ *Обращения* ] ([ *Cases* ]) таким образом, чтобы отображался заголовок с темой обращения, подзаголовок с датой регистрации и номером, а также описание обращения в виде многострочного поля.

## Реализация примера

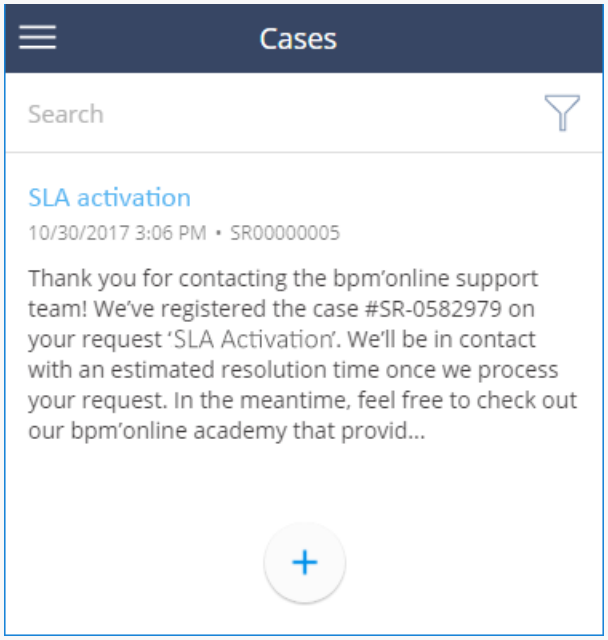
Для настройки реестра необходимо использовать приведенный ниже исходный код.

**Настройка реестра раздела [ *Обращения* ] ([ *Cases* ])**

```
// Настройка первичной колонки с темой обращения.
Terrasoft.sdk.GridPage.setPrimaryColumn("Case", "Subject");
// Установка подзаголовка с датой регистрации и номером обращения.
Terrasoft.sdk.GridPage.setSubtitleColumns("Case", ["RegisteredOn","Number"]);
// Добавление многострочного поля с описанием.
Terrasoft.sdk.GridPage.setGroupColumns("Case", [
{
```

```
    name: "Symptoms",
    isMultiline: true
}]);
```

В результате реестр будет отображаться в следующем виде.




# Класс GridPage JS

Сложный

Класс позволяет настраивать внешний вид реестра, сортировку, логику поиска и т. д.

## Методы

`setPrimaryColumn(modelName, column)`

Устанавливает первичную колонку для отображения. Настраивает отображение заголовка записи реестра.

### Параметры

| | |
|---|---|
| `modelName` | Название модели. |
| `column` | Название колонки. |

**Пример вызова**

```
Terrasoft.sdk.GridPage.setPrimaryColumn("Case", "Subject");
```

---

`setSubtitleColumns(modelName, columns)`

Устанавливает колонки, которые отображаются под заголовком. Настраивает отображение подзаголовка в виде списка колонок с разделителем.

Параметры

| | |
|---|---|
| `modelName` | Название модели. |
| `columns` | Массив колонок или конфигурационных объектов колонок. |

Пример вызова (вариант 1)

```
Terrasoft.sdk.GridPage.setSubtitleColumns("Case", ["RegisteredOn","Number"]);
```

Пример вызова (вариант 2)

```
Terrasoft.sdk.GridPage.setSubtitleColumns("Case", ["RegisteredOn", {
    name: "Number",
    convertFunction: function(values){
        return values.Number;

}]);
```

---

`setGroupColumns(modelName, columns)`

Устанавливает группу с колонками, которые отображаются вертикально. Настраивает отображение группы колонок.

Параметры

| | |
|---|---|
| `modelName` | Название модели. |
| `columns` | Массив колонок или конфигурационных объектов колонок. |

Пример вызова (вариант 1)

```
Terrasoft.sdk.GridPage.setGroupColumns("Case", ["Symptoms"]);
```

Пример вызова (вариант 2)

```
Terrasoft.sdk.GridPage.setGroupColumns("Case", [{
    name: "Symptoms",
    // Отображать как многострочное поле.
    isMultiline: true,
    // Имя локализованной строки.
    label: "CaseGridSymptomsColumnLabel",
    convertFunction: function(values) {
        return values.Symptoms;
    }
}]);
```

---

`setImageColumn()`

Устанавливает колонку изображения.

---

`setOrderByColumns()`

Устанавливает сортировку реестра.

---

`setSearchColumn()`

Устанавливает колонку поиска.

---

`setSearchColumns()`

Устанавливает колонки поиска.

---

`setSearchPlaceholder()`

Устанавливает текст подсказки в поле поиска.

---

`setTitle()`

Устанавливает заголовок страницы реестра.

# Отладка мобильного приложения

Средний

**Мобильное приложение Creatio** — приложение гибридного типа (т. е. мобильное приложение, "упакованное" в native-оболочку). Типы мобильных приложений описаны в статье Общие принципы работы мобильного приложения.

**Отладка мобильного приложения** — проверка правильности работы пользовательской функциональности мобильного приложения Creatio. Для отладки мобильного приложение можно использовать интегрированные инструменты разработчика браузера Google Chrome в режиме мобильного устройства. Подробнее об отладке клиентского кода приложения средствами инструментов разработчиков читайте в статье Front-end отладка.

Начиная с версии 7.17.4, прекращена поддержка эмулятора мобильного приложения Creatio и его использование больше невозможно. Эмулятор Mobile App был разработан для анализа обращений разработчиками и аналитиками компании Creatio. Для отладки рекомендуется использовать внешние инструменты разработки (например, Android Studio).

Работа с мобильным приложением Creatio описана в блоке статей Мобильное приложение.

# Бизнес-правила мобильного приложения

Сложный

**Бизнес-правила** — это один из механизмов Creatio, позволяющий настраивать поведение полей на странице записи. С их помощью можно, например, настроить обязательность и видимость полей, их доступность и т. п.

> **Важно.** Бизнес-правила работают только на страницах записи и просмотра записей.

Добавление бизнес-правила на страницу выполняется с помощью метода `Terrasoft.sdk.Model.addBusinessRule(name, config)`, где

- `name` — название модели, связанной со страницей записи, например, "`Contact`".
- `config` — объект, определяющий свойства бизнес-правила. Перечень свойств зависит от конкретного типа бизнес-правила.

В мобильном приложении существует возможность добавлять бизнес-правило, реализующее пользовательскую логику — пользовательское бизнес-правило. Для такого бизнес-правила предусмотрен тип `Terrasoft.RuleTypes.Custom`.

# Выполнить фильтрацию

Сложный

# Пример 1

**Пример.** Отфильтровать значения в колонке по условию.

На детали [ *Продукты в счете* ] при выборе значения из справочной колонки [ *Продукт* ] доступны только те продукты, у которых колонка [ *Active* ] содержит значение `true`.

## Реализация примера

**Пример фильтрации**

```
Terrasoft.sdk.Model.addBusinessRule("InvoiceProduct", {
    ruleType: Terrasoft.RuleTypes.Filtration,
    events: [Terrasoft.BusinessRuleEvents.Load],
    triggeredByColumns: ["Product"],
    filters: Ext.create("Terrasoft.Filter", {
        modelName: "Product",
        property: "Active",
        value: true
    })
});
```

# Пример 2

**Пример.** Отфильтровать значения в колонке по значению другой колонки.

На странице записи раздела [ *Счета* ], поле [ *Контакт* ] должно фильтроваться на основании значения в поле [ *Контрагент* ].

## Реализация примера

**Пример фильтрации**

```
Terrasoft.sdk.Model.addBusinessRule("Invoice", {
    ruleType: Terrasoft.RuleTypes.Filtration,
    events: [Terrasoft.BusinessRuleEvents.Load, Terrasoft.BusinessRuleEvents.ValueChanged],
    triggeredByColumns: ["Account"],
    filteredColumn: "Contact",
    filters: Ext.create("Terrasoft.Filter", {
        property: "Account"
    })
});
```

# Пример 3

**Пример.** Добавить и удалить фильтрацию по пользовательской логике.

## Реализация примера

**Пример фильтрации**

```
Terrasoft.sdk.Model.addBusinessRule("Activity", {
    name: "ActivityResultByAllowedResultFilterRule",
    position: 1,
    ruleType: Terrasoft.RuleTypes.Custom,
    triggeredByColumns: ["Result"],
    events: [Terrasoft.BusinessRuleEvents.ValueChanged, Terrasoft.BusinessRuleEvents.Load],
    executeFn: function(record, rule, column, customData, callbackConfig) {
        var allowedResult = record.get("AllowedResult");
        var filterName = "ActivityResultByAllowedResultFilter";
        if (!Ext.isEmpty(allowedResult)) {
            var allowedResultIds =  Ext.JSON.decode(allowedResult, true);
            var resultIdsAreCorrect = true;
            for (var i = 0, ln = allowedResultIds.length; i < ln; i++) {
                var item = allowedResultIds[i];
                if (!Terrasoft.util.isGuid(item)) {
                    resultIdsAreCorrect = false;
                    break;
                }
            }
            if (resultIdsAreCorrect) {
                var filter = Ext.create("Terrasoft.Filter", {
                    name: filterName,
                    property: "Id",
                    funcType: Terrasoft.FilterFunctions.In,
                    funcArgs: allowedResultIds
                });
                record.changeProperty("Result", {
                    addFilter: filter
                });
            } else {
                record.changeProperty("Result", {
                    removeFilter: filterName
                });
            }
        } else {
```

```
            record.changeProperty("Result", {
                removeFilter: filterName
            });
        }
        Ext.callback(callbackConfig.success, callbackConfig.scope, [true]);
    }
});
```

# Выделить поле по условию

 Сложный

**Пример.** Требуется выделить поле с результатом активности, если ее статус "Завершена", само поле [ *Результат* ] не заполнено и есть значение в колонке [ *ProcessElementId* ].

## Реализация примера

**Выделение поля по условию**

```
// Правило для страницы активности.
Terrasoft.sdk.Model.addBusinessRule("Activity", {
    // Название бизнес-правила.
    name: "ActivityResultRequiredByStatusFinishedAndProcessElementId",
    // Тип бизнес-правила: пользовательское.
    ruleType: Terrasoft.RuleTypes.Custom,
    // Правило инициируется колонками Status и Result.
    triggeredByColumns: ["Status", "Result"],
    // Правило отработает перед сохранением данных и после изменения данных.
    events: [Terrasoft.BusinessRuleEvents.ValueChanged, Terrasoft.BusinessRuleEvents.Save],
    // Функция-обработчик.
    executeFn: function(record, rule, column, customData, callbackConfig) {
        // Признак корректности свойства и правила.
        var isValid = true;
        // Значение колонки ProcessElementId.
        var processElementId = record.get("ProcessElementId");
        // Если значение не пустое.
        if (processElementId && processElementId !== Terrasoft.GUID_EMPTY) {
            // Установка признака корректности.
            isValid = !(record.get("Status.Id") === Terrasoft.Configuration.ActivityStatus.Finis
                Ext.isEmpty(record.get("Result")));
        }
        // Изменение свойств колонки Result.
        record.changeProperty("Result", {
            // Установка признака корректности колонки.
```

```
            isValid: {
                value: isValid,
                message: Terrasoft.LS["Sys.RequirementRule.message"]
            }
        });
        // Асинхронный возврат значений.
        Ext.callback(callbackConfig.success, callbackConfig.scope, [isValid]);
    }
});
```

# Сбросить отрицательные значения в 0

Сложный

**Пример.** Реализовать логику сбрасывания отрицательных значений в 0.

## Реализация примера

**Сброс отрицательных значений в 0**

```
Terrasoft.sdk.Model.addBusinessRule("Opportunity", {
    name: "OpportunityAmountValidatorRule",
    ruleType: Terrasoft.RuleTypes.Custom,
    triggeredByColumns: ["Amount"],
    events: [Terrasoft.BusinessRuleEvents.ValueChanged, Terrasoft.BusinessRuleEvents.Save],
    executeFn: function(model, rule, column, customData, callbackConfig) {
        var revenue = model.get("Amount");
        if ((revenue < 0) || Ext.isEmpty(revenue)) {
            model.set("Amount", 0, true);
        }
        Ext.callback(callbackConfig.success, callbackConfig.scope);
    }
});
```

# Сгенерировать заголовок активности

Сложный

**Пример.** Реализовать генерацию заголовка активности для решения FieldForce.

## Реализация примера

**Генерация заголовка активности**

```
Terrasoft.sdk.Model.addBusinessRule("Activity", {
    name: "FieldForceActivityTitleRule",
    ruleType: Terrasoft.RuleTypes.Custom,
    triggeredByColumns: ["Account", "Type"],
    events: [Terrasoft.BusinessRuleEvents.ValueChanged, Terrasoft.BusinessRuleEvents.Load],
    executeFn: function(record, rule, column, customData, callbackConfig, event) {
        if (event === Terrasoft.BusinessRuleEvents.ValueChanged || record.phantom) {
            var type = record.get("Type");
            var typeId = type ? type.get("Id") : null;
            if (typeId !== Terrasoft.Configuration.ActivityTypes.Visit) {
                Ext.callback(callbackConfig.success, callbackConfig.scope, [true]);
                return;
            }
            var account = record.get("Account");
            var accountName = (account) ? account.getPrimaryDisplayColumnValue() : "";
            var title = Ext.String.format("{0}: {1}", Terrasoft.LocalizableStrings.FieldForceTit
            record.set("Title", title, true);
        }
        Ext.callback(callbackConfig.success, callbackConfig.scope, [true]);
    }
});
```

# Свойства объекта config C#

Сложный

## Базовые бизнес-правила

Базовое бизнес-правило является абстрактным классом, т.е. все бизнес-правила должны быть его наследниками.

Свойства конфигурационного объекта `config` могут быть использованы наследниками базового бизнес-правила.

### Свойства конфигурационного объекта `config`

---

`ruleType`

Тип правила. Значение должно входить в перечисление `Terrasoft.RuleTypes`.

---

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила.

---

`message`

Текстовое сообщение, которое выводится под элементом управления, который связан с колонкой, в случае невыполнения бизнес-правила. Необходимо для правил, сообщающих пользователю предупреждающую информацию.

---

`name`

Уникальное имя бизнес-правила. Необходимо, если нужно удалить правило методами `Terrasoft.sdk`.

---

`position`

Позиция бизнес-правила, определяющая приоритет вызова правила в очереди текущих запускаемых правил.

---

`events`

Массив событий, определяющий время запуска бизнес-правил. Должен содержать значения, входящие в перечисление `Terrasoft.BusinessRuleEvents`.

Возможные значения (`Terrasoft.BusinessRuleEvents`)

| | |
|---|---|
| `Save` | Правило отработает перед сохранением данных. |
| `ValueChanged` | Правило отработает после изменения данных (при редактировании). |
| `Load` | Правило отработает при открытии страницы записи. |

# Бизнес-правило [ *Обязательность заполнения* ] (`Terrasoft.RuleTypes.Requirement`) C#

Определяет обязательность заполнения поля на странице записи.

## Свойства конфигурационного объекта `config`

---

`ruleType`

Для этого правила должно содержать значение `Terrasoft.RuleTypes.Requirement`.

---

`requireType`

Тип проверки. Значение должно входить в перечисление `Terrasoft.RequirementTypes`. Правило может проверять одну или все колонки из `triggeredByColumns`.

---

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила. Если тип проверки равен `Terrasoft.RequirementTypes.Simple`, то должна быть указана одна колонка в массиве.

Возможные значения ( `Terrasoft.RequirementTypes` )

| | |
|---|---|
| `Simple` | Проверка значения в одной колонке. |
| `OneOf` | Одна из колонок, указанных в `triggeredByColumns` должна быть обязательно заполнена. |

**Пример использования**

```
Terrasoft.sdk.Model.addBusinessRule("Contact", {
    ruleType: Terrasoft.RuleTypes.Requirement,
    requireType : Terrasoft.RequirementTypes.OneOf,
    events: [Terrasoft.BusinessRuleEvents.Save],
    triggeredByColumns: ["HomeNumber", "BusinessNumber"],
    columnNames: ["HomeNumber", "BusinessNumber"]
});
```

# Бизнес-правило [ *Видимость* ] (Terrasoft.RuleTypes.Visibility) C#

С помощью этого бизнес-правила можно скрывать и отображать поля по определенному условию.

## Свойства конфигурационного объекта config

---

`ruleType`

Для этого правила должно содержать значение `Terrasoft.RuleTypes.Visibility`.

---

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила.

---

`events`

Массив событий, определяющий время запуска бизнес-правил. Должен содержать значения, входящие в перечисление `Terrasoft.BusinessRuleEvents`.

---

`conditionalColumns`

Массив условий срабатывания бизнес-правила. Обычно это определенные значения колонок.

---

`dependentColumnNames`

Массив названий колонок, к которым применяется данное бизнес-правило.

**Пример использования**

```
Terrasoft.sdk.Model.addBusinessRule("Account", {
    ruleType: Terrasoft.RuleTypes.Visibility,
    conditionalColumns: [
        {name: "Type", value: Terrasoft.Configuration.Consts.AccountTypePharmacy}
    ],
    triggeredByColumns: ["Type"],
    dependentColumnNames: ["IsRx", "IsOTC"]
});
```

Поля, связанные с колонками `IsRx` и `IsOTC` будут отображены, если колонка `Type` будет содержать значение, определенное константой `Terrasoft.Configuration.Consts.AccountTypePharmacy`.

```
Terrasoft.Configuration.Consts = {
    AccountTypePharmacy: "d12dc11d-8c74-46b7-9198-5a4385428f9a"
};
```

Вместо константы можно использовать значение "`d12dc11d-8c74-46b7-9198-5a4385428f9a`".

# Бизнес-правило [ *Доступность* ] (`Terrasoft.RuleTypes.Activation`) C#

С помощью этого бизнес-правила можно делать поля недоступными (или наоборот — доступными) для ввода значений по определенному условию.

## Свойства конфигурационного объекта `config`

---

`ruleType`

Для этого правила должно содержать значение `Terrasoft.RuleTypes.Activation`.

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила.

`events`

Массив событий, определяющий время запуска бизнес-правил. Должен содержать значения, входящие в перечисление `Terrasoft.BusinessRuleEvents`.

`conditionalColumns`

Массив условий срабатывания бизнес-правила. Обычно это определенные значения колонок.

`dependentColumnNames`

Массив названий колонок, к которым применяется данное бизнес-правило.

Доступность поля, связанного с колонкой `Stock`, зависит от значения в колонке `IsPresence`.

**Пример использования**

```
Terrasoft.sdk.Model.addBusinessRule("ActivitySKU", {
    ruleType: Terrasoft.RuleTypes.Activation,
    events: [Terrasoft.BusinessRuleEvents.Load, Terrasoft.BusinessRuleEvents.ValueChanged],
    triggeredByColumns: ["IsPresence"],
    conditionalColumns: [
        {name: "IsPresence", value: true}
    ],
    dependentColumnNames: ["Stock"]
});
```

# Бизнес-правило [ *Фильтрация* ] (`Terrasoft.RuleTypes.Filtration`)

Это бизнес-правило можно использовать для фильтрации значений справочных колонок по условию, либо по значению другой колонки.

## Свойства конфигурационного объекта `config`

`ruleType`

Для этого правила должно содержать значение `Terrasoft.RuleTypes.Filtration`.

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила.

---

`events`

Массив событий, определяющий время запуска бизнес-правил. Должен содержать значения, входящие в перечисление `Terrasoft.BusinessRuleEvents`.

---

`filters`

Фильтр. Свойство должно содержать экземпляр класса `Terrasoft.Filter`.

---

`filteredColumn`

Колонка, на основании которой выполняется фильтрация значений.

# Бизнес-правило [ *Взаимная фильтрация* ] (`Terrasoft.RuleTypes.MutualFiltration`) C#

Это бизнес-правило позволяет выполнять взаимную фильтрацию двух справочных полей. Работает только с колонками, связанными отношением "один-ко-многим", например, [ *Страна* ] — [ *Город* ]. Для каждой связки полей необходимо создавать свое бизнес-правило. Например, для связок [ *Страна* ] — [ *Область* ] — [ *Город* ] и [ *Страна* ] — [ *Город* ] необходимо создать три бизнес-правила:

- [ *Страна* ] — [ *Область* ];
- [ *Область* ] — [ *Город* ];
- [ *Страна* ] — [ *Город* ].

## Свойства конфигурационного объекта `config`

---

`ruleType`

Для этого правила должно содержать значение `Terrasoft.RuleTypes.MutualFiltration`.

---

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила.

---

`connections`

Массив объектов, конфигурирующих отношение связок.

Взаимная фильтрация полей [Страна], [Область] и [Город]

```
Terrasoft.sdk.Model.addBusinessRule("ContactAddress", {
    ruleType: Terrasoft.RuleTypes.MutualFiltration,
    triggeredByColumns: ["City", "Region", "Country"],
    connections: [
        {
            parent: "Country",
            child: "City"
        },
        {
            parent: "Country",
            child: "Region"
        },
        {
            parent: "Region",
            child: "City"
        }
    ]
});
```

Взаимная фильтрация полей [Контакт], [Контрагент]

```
Terrasoft.sdk.Model.addBusinessRule("Activity", {
    ruleType: Terrasoft.RuleTypes.MutualFiltration,
    triggeredByColumns: ["Contact", "Account"],
    connections: [
        {
            parent: "Contact",
            child: "Account",
            connectedBy: "PrimaryContact"
        }
    ]
});
```

## Бизнес-правило [ *Регулярное выражение* ] (`Terrasoft.RuleTypes.RegExp`) C#

Выполняет проверку соответствия значения колонки регулярному выражению.

### Свойства конфигурационного объекта `config`

`ruleType`

Для этого правила должно содержать значение `Terrasoft.RuleTypes.RegExp`.

---

`RegExp`

Регулярное выражение, на соответствие которому проверяются все колонки из массива `triggeredByColumns`.

---

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила.

**Пример использования**

```
Terrasoft.sdk.Model.addBusinessRule("Contact", {
    ruleType: Terrasoft.RuleTypes.RegExp,
    regExp : /^([0-9\(\)\/\+ \-]*)$/
    triggeredByColumns: ["HomeNumber", "BusinessNumber"]
});
```

# Пользовательские бизнес-правила

При добавлении пользовательского бизнес-правила с помощью метода `Terrasoft.sdk.Model.addBusinessRule(name, config)` можно использовать свойства конфигурационного объекта `config` базового бизнес-правила. Также дополнительно предусмотрено свойство `executeFn`.

## Свойства конфигурационного объекта `config`

---

`ruleType`

Тип правила. Для пользовательских правил должно содержать значение `Terrasoft.RuleTypes.Custom`.

---

`triggeredByColumns`

Массив колонок, инициирующих срабатывание бизнес-правила.

---

`events`

Массив событий, определяющий время запуска бизнес-правил. Должен содержать значения из перечисления `Terrasoft.BusinessRuleEvents`. Значение по умолчанию: `Terrasoft.BusinessRuleEvents.ValueChanged`.

### Возможные значения ( `Terrasoft.BusinessRuleEvents` )

| | |
|---|---|
| `Save` | Правило сработает перед сохранением данных. |
| `ValueChanged` | Правило сработает после изменения данных (при редактировании). |
| `Load` | Правило сработает при открытии страницы записи. |

---

`executeFn`

Функция-обработчик, содержащая пользовательскую логику выполнения бизнес-правила.

## Свойства функции-обработчика executeFn

Функция-обработчик объявляется в свойстве `executeFn` .

**Сигнатура функции-обработчика**

```
executeFn: function(record, rule, checkColumnName, customData, callbackConfig, event) {    }
```

### Параметры

| | |
|---|---|
| `record` | Запись, для которой выполняется бизнес-правило. |
| `rule` | Экземпляр текущего бизнес-правила. |
| `checkColumnName` | Название колонки, которая вызвала срабатывание бизнес-правила. |
| `customData` | Объект, разделяемый между всем правилами. Не используется. Оставлен для совместимости с предыдущими версиями. |
| `callbackConfig` | Конфигурационный объект асинхронного возврата `Ext.callback` . |
| `event` | Событие, по которому было запущено бизнес-правило. |

При завершении работы функции необходимо обязательно вызвать или `callbackConfig.success` , или `callbackConfig.failure` .

**Варианты вызова**

```
Ext.callback(callbackConfig.success, callbackConfig.scope, [result]);
Ext.callback(callbackConfig.failure, callbackConfig.scope, [exception]);
```

Где:

- `result` — возвращаемое логическое значение, полученное при выполнении функции (`true` / `false`).
- `exception` — исключение типа `Terrasoft.Exception`, возникшее в функции-обработчике.

## Методы

В исходном коде функции-обработчика удобно использовать следующие методы модели, передаваемой в параметре `record`:

---

`get(columnName)`

Для получения значения колонки записи. Аргумент `columnName` должен содержать название колонки.

---

`set(columnName, value, fireEventConfig)`

Для установки значения колонки записи.

### Параметры

| | |
|---|---|
| `columnName` | Название колонки. |
| `value` | Значение, присваиваемое колонке. |
| `fireEventConfig` | Конфигурационный объект для установки свойств, передаваемых в событие изменения колонки. |

---

`changeProperty(columnName, propertyConfig)`

Для изменения свойств колонки, кроме ее значения. Аргумент `columnName` должен содержать название колонки, а `propertyConfig` — объект, устанавливающий свойства колонки.

### Свойства объекта `propertyConfig`

| | |
|---|---|
| `disabled` | Активность колонки. Если `true`, то элемент управления, связанный с колонкой, будет неактивным и закрытым для работы. |
| `readOnly` | Признак "только для чтения". Если `true`, то элемент управления, связанный с колонкой, будет доступен только для чтения. Если `false` — для чтения и записи. |
| `hidden` | Видимость колонки. Если `true`, то элемент управления, связанный с колонкой, будет скрыт. Если `false` — отображен. |
| `addFilter` | Добавить фильтр. Если свойство указано, то в нем должен быть указан фильтр типа `Terrasoft.Filter`, который будет добавлен к фильтрации колонки. Свойство используется только для справочных полей. |
| `removeFilter` | Удалить фильтр. Если свойство указано, то в нем должно быть указано имя фильтра, который будет удален из фильтрации колонки. Свойство используется только для справочных полей. |
| `isValid` | Признак корректности колонки. Если свойство указано, то оно изменит признак корректности элемента управления, связанного с колонкой. Если колонка некорректна, то это может означать отмену событий по сохранению записи, а также может привести к признанию записи некорректной. |

**Пример изменения свойств (но не значения) колонки `Owner`**

```
record.changeProperty("Owner", {
    disabled: false,
    readOnly: false,
    hidden: false,
    addFilter: {
        property: "IsChief",
        value: true
    },
    isValid: {
        value: false,
        message: LocalizableStrings["Owner_should_be_a_chief_only"]
    }
 });
```

# Получение данных и настроек по дашбордам

Сложный

Функциональность получения настроек и данных по дашбордам реализована в сервисе `AnalyticsService` и в утилитном классе `AnalyticsServiceUtils`, пакет `Platform`.

# Примеры запросов к сервису AnalyticsService

Сложный

**Заголовки запроса**

```
Accept:application/json
```

## Методы

`GetDashboardViewConfig()`

Запрос

```
POST /0/rest/AnalyticsService/GetDashboardViewConfig

{
    "id": "a71d5c04-dff7-4892-90e5-9e7cc2246915"
}
```

Ответ на запрос

```
{
  "items": [
    {
      "layout": {
        "column": 0,
        "row": 0,
        "colSpan": 12,
        "rowSpan": 5
      },
      "name": "Chart4",
      "itemType": 4,
      "widgetType": "Chart"
    }
```

```
  ]
}
```

---

GetDashboardData()

Запрос

```
POST /0/rest/AnalyticsService/GetDashboardData

{
    "id": "a71d5c04-dff7-4892-90e5-9e7cc2246915",
    "timeZoneOffset": 120
}
```

Ответ на запрос

```
{
    "items": [
        {
            "name": "Indicator1",
            "caption": "Среднее время выполнения активности",
            "widgetType": "Indicator",
            "style": "widget-green",
            "data": 2
        }
    ]
}
```

---

GetDashboardItemData()

Запрос

```
POST /0/rest/AnalyticsService/GetDashboardItemData

{
    "dashboardId": "a71d5c04-dff7-4892-90e5-9e7cc2246915",
    "itemName": "Chart4",
    "timeZoneOffset": 120
}
```

Ответ на запрос


```
{
  "name": "Chart4",
  "caption": "Invoice payment dynamics",
  "widgetType": "Chart",
  "chartConfig": {
    "xAxisDefaultCaption": null,
    "yAxisDefaultCaption": null,
    "seriesConfig": [
      {
        "type": "column",
        "style": "widget-green",
        "xAxis": {
          "caption": null,
          "dateTimeFormat": "Month;Year"
        },
        "yAxis": {
          "caption": "Actually paid",
          "dataValueType": 6
        },
        "schemaName": "Invoice",
        "schemaCaption": "Invoice",
        "useEmptyValue": null
      }
    ],
    "orderDirection": "asc"
  },
  "style": "widget-green",
  "data": []
}
```


# Класс AnalyticsService C#

Сложный

Класс реализует функциональность получения настроек и данных по дашбордам.

## Методы

```
Stream GetDashboardViewConfig(Guid id)
```

Возвращает настройки представления и виджетов на вкладке итогов по идентификатору страницы итогов.

---

```
Stream GetDashboardData(Guid id, int timeZoneOffset)
```

Возвращает данные по всем виджетам на вкладке итогов по идентификатору страницы итогов.

---

```
Stream GetDashboardItemData(Guid dashboardId, string itemName, int timeZoneOffset)
```

Возвращает данные по определенному виджету по идентификатору страницы итогов и имени виджета.

Здесь `timeZoneOffset` — смещение (в минутах) часового пояса относительно UTC. Данные по итогам будут получены с использованием этого часового пояса.