



Creatio
Mobile

Руководство
по разработке



ACCELERATE 

Содержание

1.	Документация по разработке Mobile Creatio	2
1.1.	Знакомство с платформой Mobile Creatio	2
1.1.1.	Архитектура мобильного приложения	2-6
1.2.	С чего начать разработку	6-7
1.2.1.	Отладка мобильного приложения	7-11
1.3.	Описание платформы	11-12
1.3.1.	Манифест мобильного приложения	12-13
1.3.1.1.	Манифест. Свойства интерфейса приложения	13-16
1.3.1.2.	Манифест. Свойства данных и бизнес-логики	16-19
1.3.1.3.	Манифест. Свойства синхронизации приложений	19-24
1.3.1.4.	Экспорт данных в пакетном режиме	24-25
1.3.2.	Жизненный цикл страниц в мобильном приложении	25-28
1.3.3.	Фоновое обновление конфигурации в мобильном приложении	28-29
1.3.4.	Получение настроек и данных раздела [Итоги]	29-32
1.3.5.	Автоматическое разрешение конфликтов при синхронизации	32-33
1.4.	Mobile SDK	33
1.4.1.	SDK реестра	33-36
1.4.2.	Бизнес-правила мобильного приложения	36-40
1.4.3.	Пользовательские бизнес-правила мобильного приложения	40-44
1.5.	Разработка Mobile Creatio на примерах	44-45
1.5.1.	Как добавить стандартную деталь с колонками	45-50
1.5.2.	Добавление пользовательского типа виджета в мобильное приложение	50-52
1.5.3.	Модификаторы доступа страницы в мобильном приложении	52-53
1.5.4.	Добавление элементов управления на страницу раздела	53-58
1.5.5.	Отображение страницы на планшете во весь экран	58-60

1 Документация по разработке Mobile Creatio



Знакомство с платформой Mobile Creatio (Section 1.1)

Архитектура, общая схема и режимы работы мобильного приложения Creatio.



С чего начать разработку (Section 1.2)

Как начать разработку мобильного приложения Creatio, используя инструменты разработки современных браузеров.



Описание платформы (Section 1.3)

Подробное описание компонентов и процессов мобильного приложения Creatio.



Mobile SDK (Section 1.4)

Модули, классы, методы и свойства.



Разработка Mobile Creatio на примерах (Section 1.5)

Как кастомизировать существующую и добавлять новую функциональность в мобильное приложение Creatio.

1.1 Знакомство с платформой Mobile Creatio

Содержание



Архитектура мобильного приложения (Section 1.1.1)

Архитектура, общая схема и режимы работы мобильного приложения Creatio.

1.1.1 Архитектура мобильного приложения

Уровень сложности



Общие положения

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.

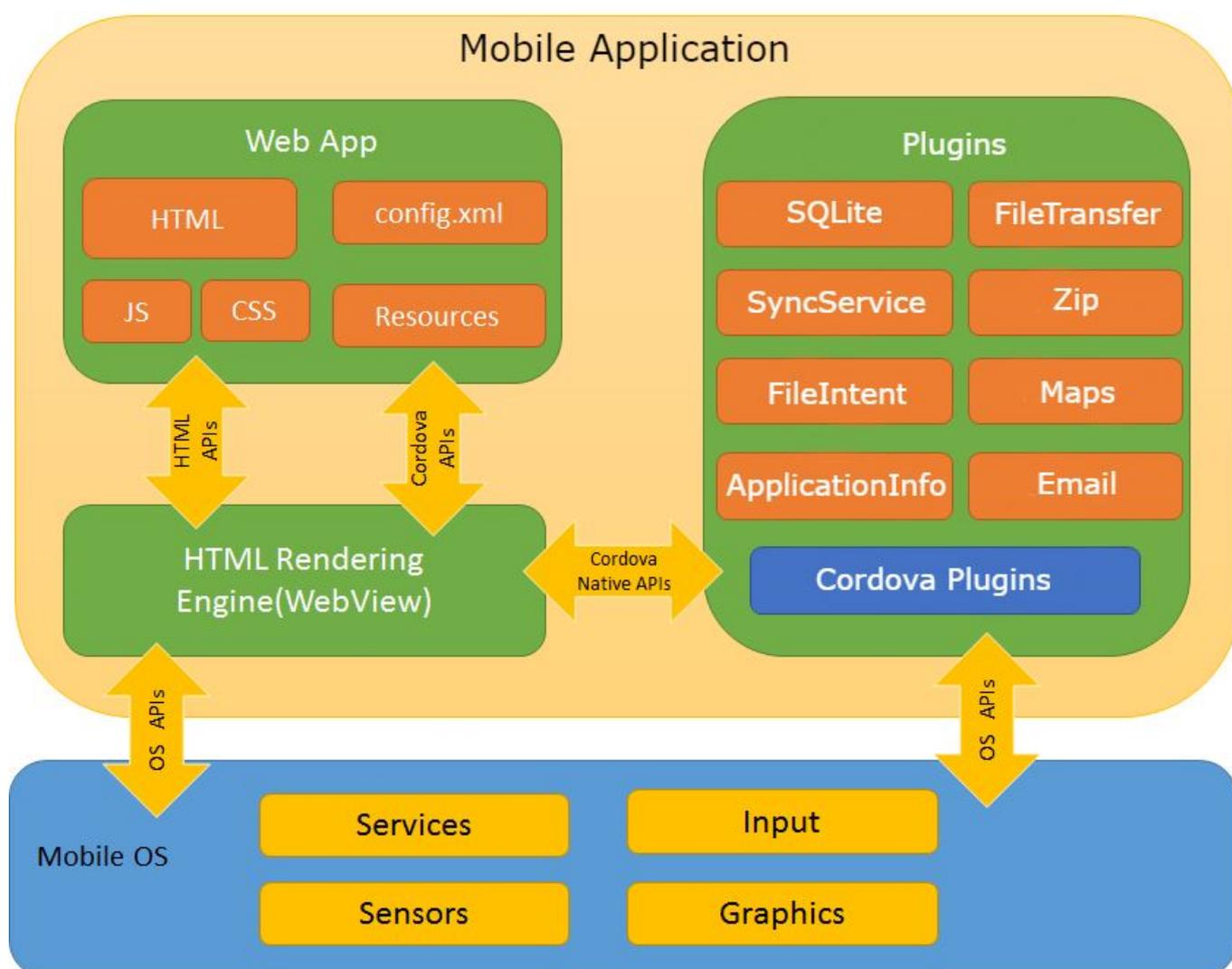
Гибридное приложение — мобильное приложение, "упакованное" в native-оболочку. Такое приложение, как и native, устанавливается из онлайн-магазина и имеет доступ к тем же возможностям мобильного устройства, но разрабатывается с помощью web-языков HTML5, CSS и JavaScript. В отличие от native-приложения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

Общие сведения по [установке и синхронизации мобильного приложения](#), а также специфика работы в онлайн и офлайн режимах приведены в документации Creatio.

Архитектура мобильного приложения Creatio

В общем виде архитектура мобильного приложения представлена на рис. 1.

Рис. 1. — Архитектура мобильного приложения



Для создания гибридных приложений, воспринимаемых мобильным устройством как native, мобильное приложение использует возможности фреймворка [Cordova](#). Фреймворк Cordova предоставляет доступ к программному интерфейсу мобильного устройства (API) для взаимодействия с базой данных или оборудованием, например, камерой или картой памяти. Также Cordova предоставляет т. н. 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 можно найти [здесь](#).

Ядро мобильного приложения предоставляет унифицированный интерфейс для взаимодействия всех остальных клиентских частей приложения. Используемые ядром Javascript-файлы условно можно разделить на две следующие категории:

1. Базовые:

- MVC-компоненты (представления страниц, контроллеры, модели);
- модули синхронизации (импорт\экспорт данных, импорт метаданных, импорт файлов и т. д.);
- клиентские классы веб-сервисов;
- классы, предоставляющие доступ к native-плагинам.

Базовые скрипты содержатся в сборке приложения, публикуемой в магазине приложений.

2. Конфигурационные:

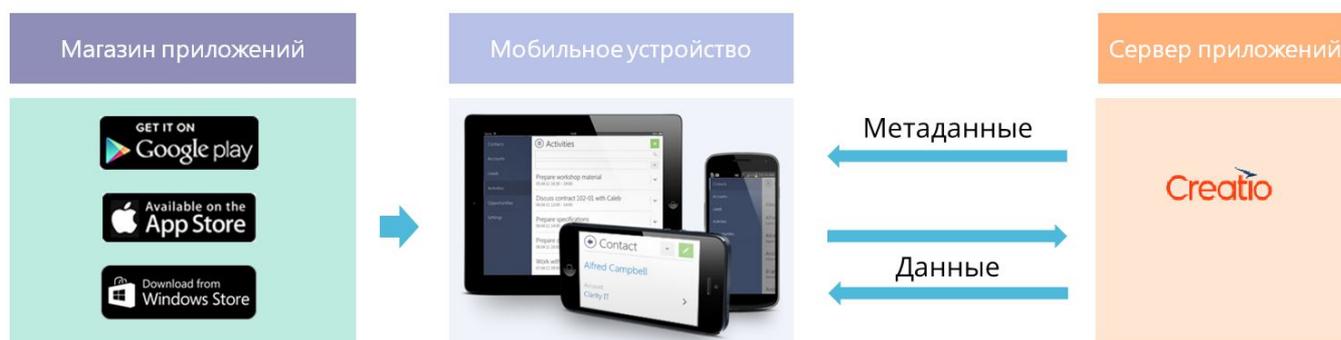
- манифест;
- схемы и настройки разделов.

Конфигурационные файлы приложение получает в ходе синхронизации с сервером Creatio и сохраняет локально в файловой системе устройства.

Общая 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 (рис. 2):

Рис. 2. — Общая схема работы мобильного приложения



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 может работать в двух режимах:

- с подключением к основному приложению (online);
- без подключения к основному приложению (offline).

Разница между этими режимами отображена в таблице 1:

Табл. 1. — Сравнение режимов работы мобильного приложения

Online

Необходимо наличие соединения с интернетом.

Offline

Наличие соединения с интернетом необязательно. Необходимо только для первичного импорта и синхронизации.

Пользователь работает напрямую с сервером Creatio.

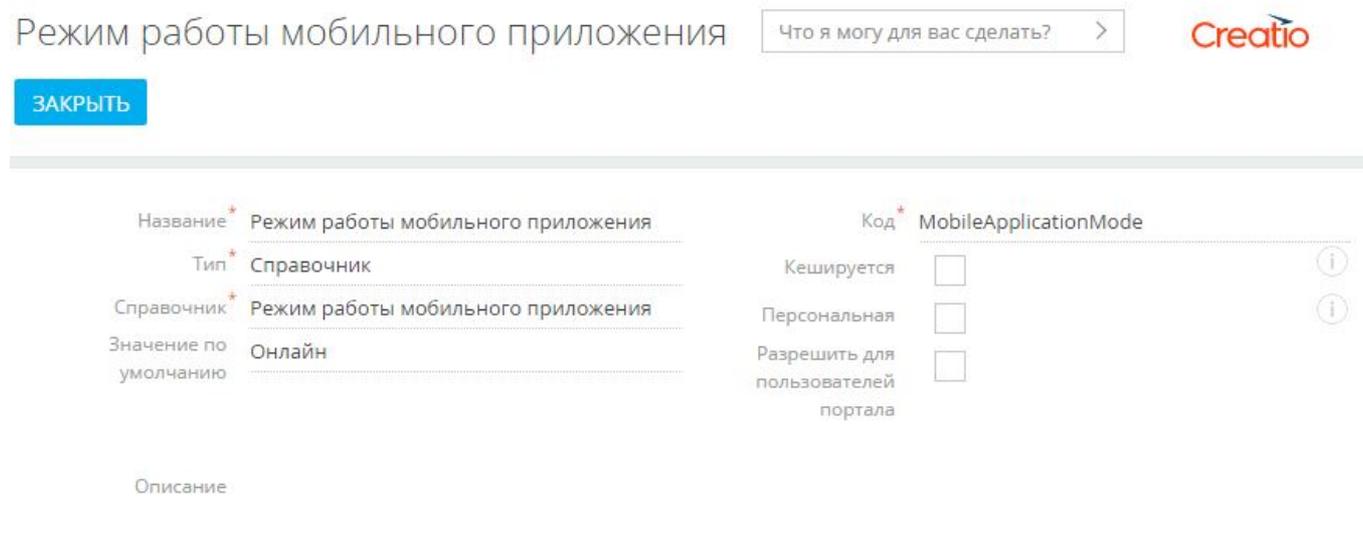
Синхронизацию необходимо выполнять только при конфигурационных изменениях (добавление или удаление колонки, изменение логики работы).

Данные сохраняются локально на мобильном устройстве.

Синхронизацию необходимо выполнять регулярно для актуализации данных и для получения конфигурационных изменений.

За режим работы мобильного приложения отвечает системная настройка [Режим работы мобильного приложения] в Creatio. Если нужно изменить режим работы одновременно для всех пользователей мобильного приложения, необходимо установить нужное значение этой настройки без установленного свойства [Персональная] (рис. 3). 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.

Рис. 3. — Системная настройка [Режим работы мобильного приложения]



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 получения обновлений, так и для передачи или получения измененных или новых данных. Общая схема синхронизации для offline-режима представлена на рис. 4:

Рис. 4. — Общая схема синхронизации (offline-режим)

Аутентификация
(logout и login)

Пакет разницы

Метаданные
(манифест, конфигурационные схемы)

Системные данные
(простые справочники, системные
настройки)

Экспорт данных

Импорт данных

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.

К СВЕДЕНИЮ

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`.

1.2 С чего начать разработку

Содержание

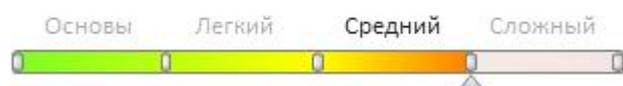


Отладка мобильного приложения (Section 1.2.1)

Во время разработки пользовательских решений для мобильного приложения Creatio необходимо многократно выполнять проверку правильности работы новой функциональности — отладку приложения. Как это сделать с помощью инструментов разработки современных браузеров читайте в этой статье.

1.2.1 Отладка мобильного приложения

Уровень сложности



Общие сведения

Во время разработки пользовательских решений для мобильного приложения Creatio необходимо многократно выполнять проверку правильности работы новой функциональности — *отладку приложения*.

Поскольку мобильное приложение Creatio является **приложением гибридного типа (Section 1.1.1)** — мобильным web-приложением, "упакованным" в native-оболочку, то существует возможность его отладки [средствами инструментов разработчика](#) браузера Google Chrome в [режиме мобильного устройства](#).

Подробнее об отладке клиентского кода приложения средствами инструментов разработчиков современных браузеров можно узнать в статье "[Отладка клиентского кода](#)" документации по разработке.

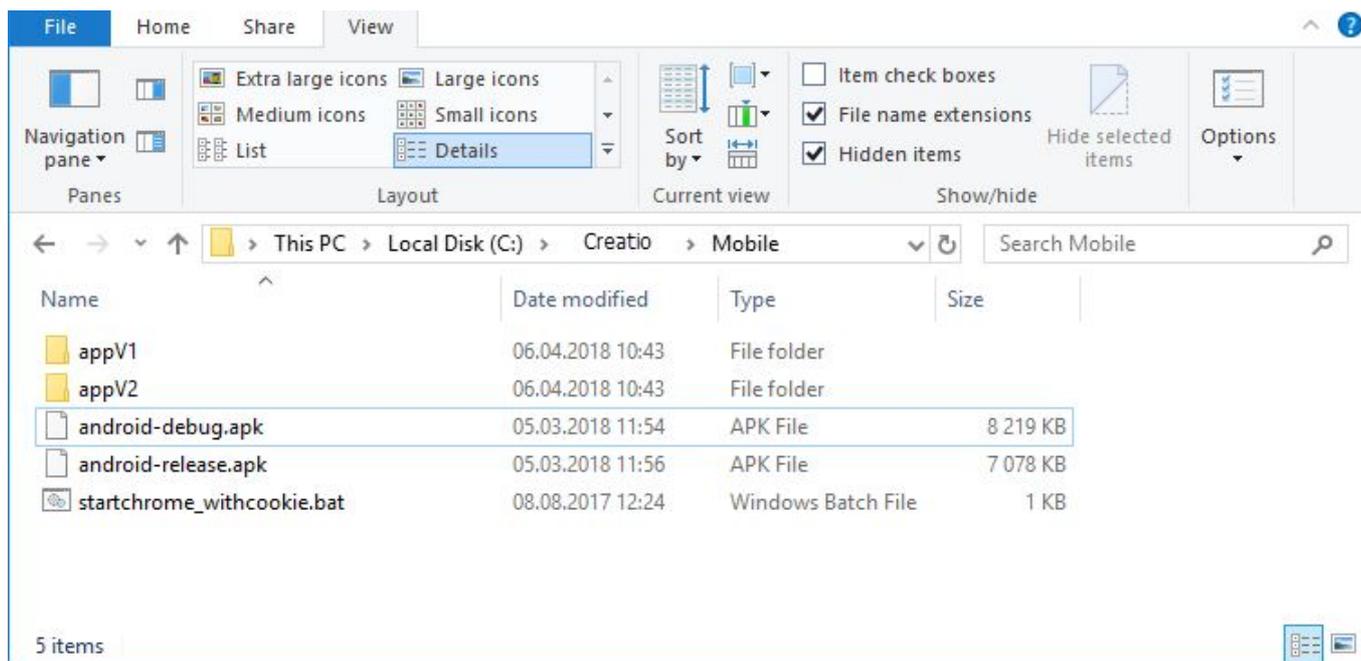
Последовательность запуска мобильного приложения в режиме отладки:

1. Получить необходимые для отладки мобильного приложения файлы.
2. Запустить пакетный файл `startchrome_withcookie.bat`.
3. В браузере Google Chrome перейти в режим отладки мобильного устройства.
4. Внести необходимые настройки и синхронизировать мобильное приложение Creatio.

Получение необходимых файлов

Для получения необходимых для отладки мобильного приложения файлов нужно обратиться в службу поддержки. Служба поддержки предоставит архив, содержащий нужные файлы. Архив необходимо распаковать в произвольный каталог, например, `C:\creatio\Mobile` (рис. 1).

Рис. 1. — Содержимое распакованного архива



Запуск пакетного файла `startchrome_withcookie.bat`

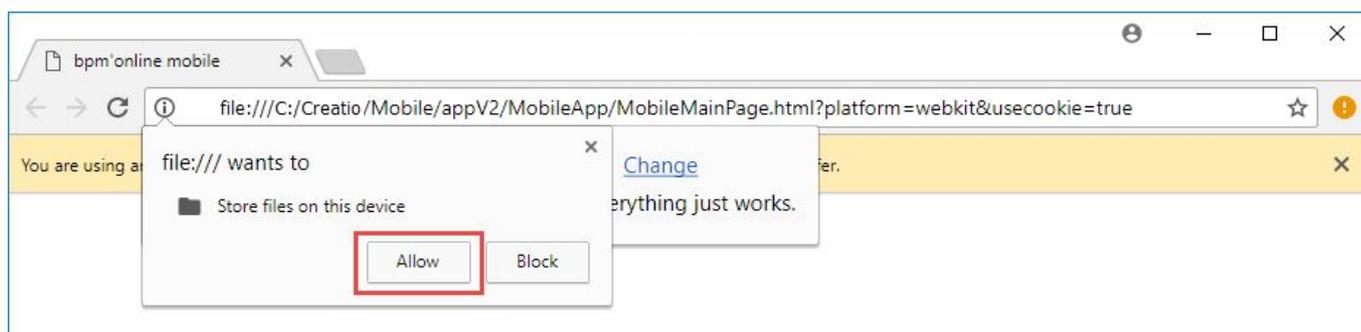
ВАЖНО

Прежде чем запустить пакетный файл `startchrome_withcookie.bat` на выполнение, необходимо закрыть все экземпляры браузера Google Chrome.

Пакетный файл `startchrome_withcookie.bat` размещен в корневом каталоге распакованного архива. После выполнения команд пакетного файла запустится браузер Google Chrome.

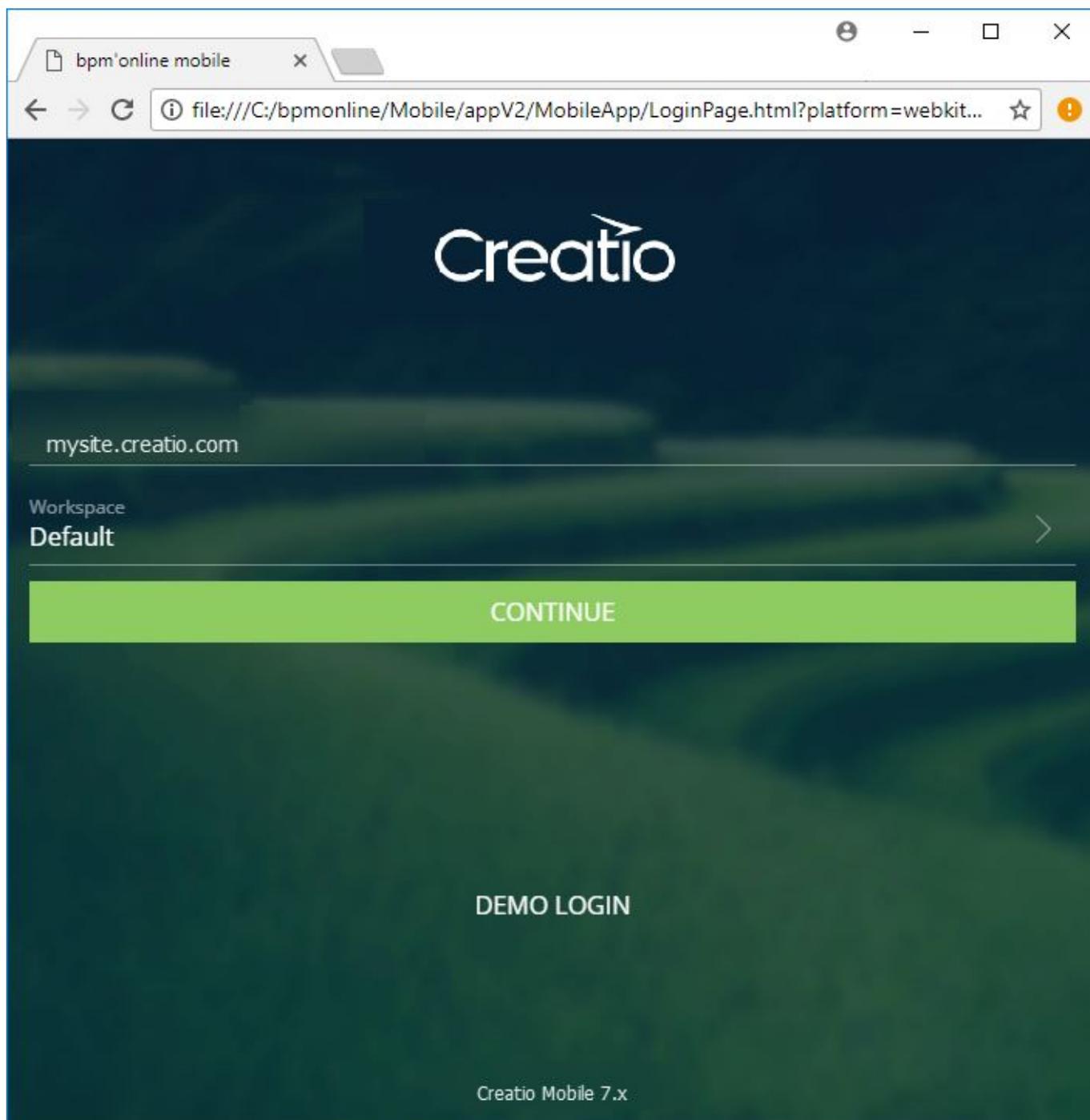
При первом запуске браузера с помощью пакетного файла отобразится информационное окно, предупреждающее о сохранении файлов в файловую систему (рис. 2). Нужно разрешить сохранение файлов. Также можно закрыть предупреждение о неподдерживаемом флаге `--disable-web-security` (рис. 2).

Рис. 2. — Информационное окно с предупреждением



После выполнения команд пакетного файла `startchrome_withcookie.bat` запустится браузер Google Chrome с открытой страницей настройки мобильного приложения Creatio (рис. 3).

Рис. 3. — Страница настройки мобильного приложения



Переход в режим отладки мобильного устройства

Чтобы вызвать инструментарий разработчика в браузере Google Chrome, необходимо нажать функциональную клавишу *F12* клавиатуры или комбинацию клавиш *Ctrl + Shift + I*. Отладку локальной версии мобильного приложения можно выполнять средствами браузера. Подробнее об отладке клиентского кода приложения средствами инструментов разработчиков современных браузеров можно узнать в статье "[Отладка клиентского кода](#)" документации по разработке.

К СВЕДЕНИЮ

После перехода в режим отладки мобильного устройства необходимо обновить страницу отображения, нажав клавишу *F5*.

Ввод настроек мобильного приложения и синхронизация

При первом входе в мобильное приложение на странице настроек необходимо ввести http-адрес приложения Creatio, для которого необходимо выполнить отладку, и нажать на кнопку [Далее] ([Continue]) (рис. 4). Затем необходимо ввести имя пользователя и пароль (рис. 5).

Рис. 4. — Страница настроек локальной версии мобильного приложения

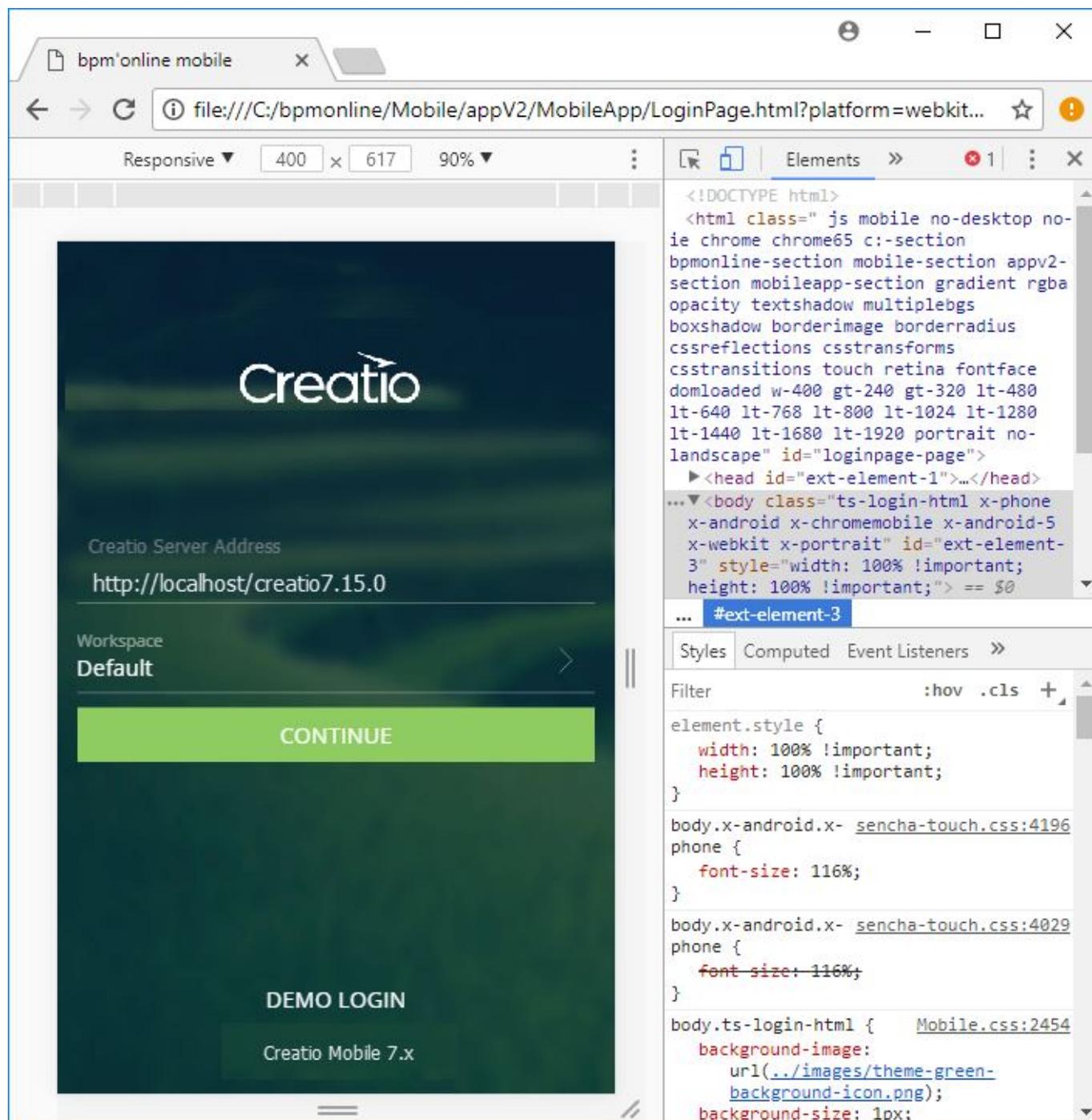
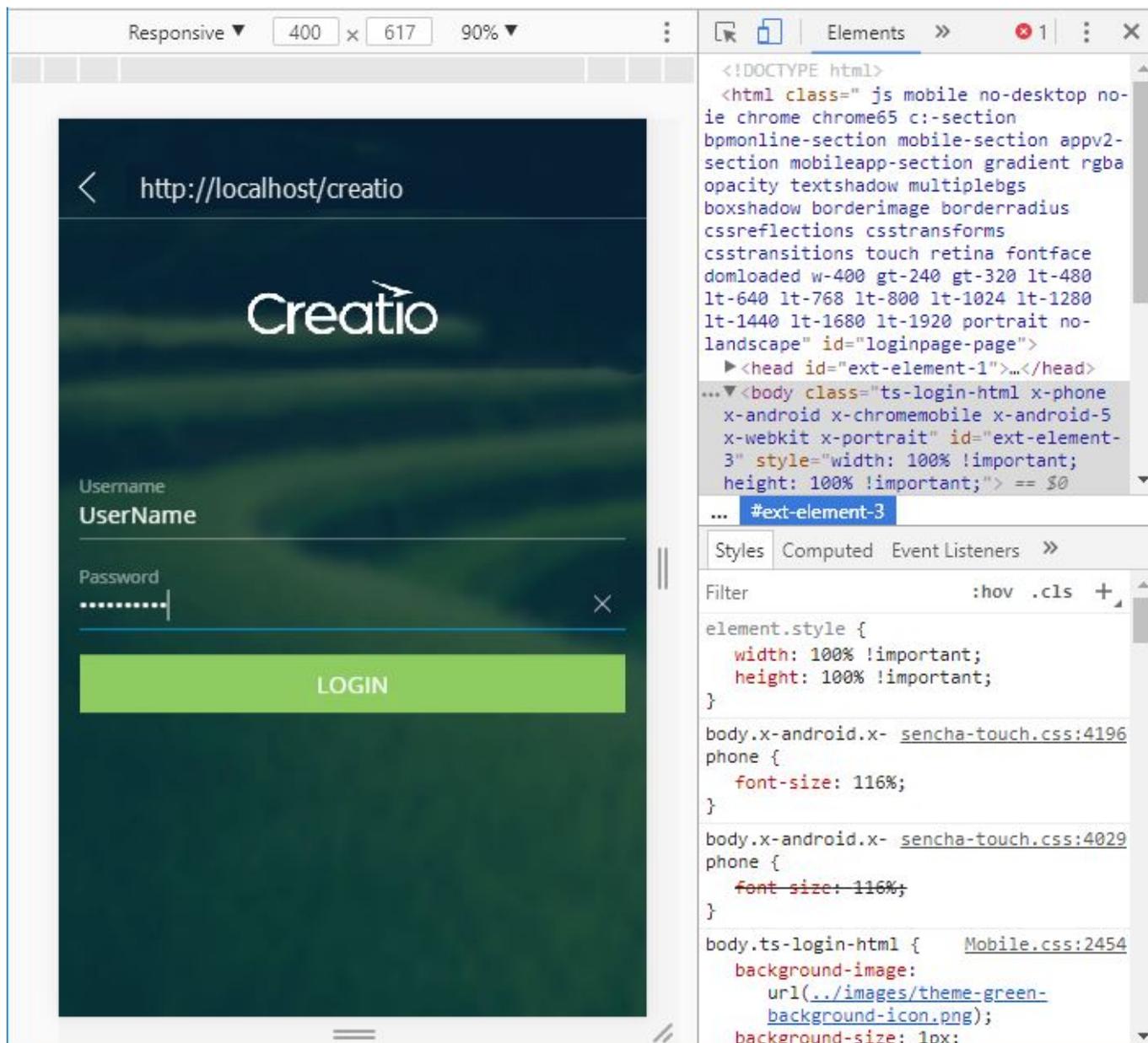


Рис. 5. — Страница авторизации



После выполнения настроек и входа в приложение локальная версия мобильного приложения будет вести себя аналогично приложению, только что установленному в мобильное устройство. При этом native-функции мобильного устройства, например, работа с камерой, загрузка файлов и др., поддерживаться не будут. О том, как работать с мобильным приложением Creatio, можно ознакомиться в документации [Mobile Creatio](#).

1.3 Описание платформы

Содержание



Манифест мобильного приложения (Section 1.3.1)

Манифест мобильного приложения описывает структуру всего мобильного приложения — его объекты и связи между ними.

- **Манифест. Свойства интерфейса приложения (Section 1.3.1.1)**
- **Манифест. Свойства данных и бизнес-логики (Section 1.3.1.2)**
- **Манифест. Свойства синхронизации приложений (Section 1.3.1.3)**

- Экспорт данных в пакетном режиме (Section 1.3.1.4)



Жизненный цикл страниц в мобильном приложении (Section 1.3.2)

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.



Фоновое обновление конфигурации в мобильном приложении (Section 1.3.3)

В мобильном приложении Creatio реализован механизм синхронизации структуры приложения, который может работать в автоматическом фоновом режиме.



Получение настроек и данных раздела [Итоги] (Section 1.3.4)

Функциональность получения настроек и данных по дашбордам реализована в сервисе *AnalyticsService* и в утилитном классе *AnalyticsServiceUtils* пакета *Platform*.



Автоматическое разрешение конфликтов при синхронизации (Section 1.3.5)

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

1.3.1 Манифест мобильного приложения

Уровень сложности



Общие положения

Манифест мобильного приложения описывает структуру всего мобильного приложения — его объекты и связи между ними. Базовая версия мобильного приложения Creatio описывается манифестом, который содержится в схеме *MobileApplicationManifestDefaultWorkspace* пакета *Mobile*.

В процессе доработки мобильного приложения пользователями создаются новые разделы и страницы. Все они должны быть зарегистрированы в манифесте для того, чтобы приложение могло с ними работать. Так как у сторонних разработчиков нет возможности вносить изменения в манифест базового приложения, то при регистрации пользовательских разделов и страниц при помощи мастера мобильных приложений система автоматически создает пользовательский манифест, в котором в заданном формате описаны все взаимосвязи созданных объектов. Название схемы манифеста формируется по маске *MobileApplicationManifest[Название рабочего места]*. Так, например, для рабочего места [Полевые продажи], система сформирует название схемы манифеста *MobileApplicationManifestFieldForceWorkspace*, а для рабочего места [Основное рабочее место] — название *MobileApplicationManifestDefaultWorkspace*.

Структура манифеста мобильного приложения

Манифест мобильного приложения — это конфигурационный объект, с помощью свойств которого описывается структура мобильного приложения. Перечень и назначение свойств конфигурационного объекта манифеста приведены в таблице 1.

Табл. 1. — Свойства конфигурационного объекта манифеста.

Свойство	Назначение
----------	------------

ModuleGroups	Содержит верхнеуровневую настройку групп главного меню.
Modules	Описывает свойства модулей мобильного приложения.
SyncOptions	Описывает параметры для настройки синхронизации данных.
Models	Содержит конфигурацию импортируемых моделей приложения.
PreferredFilterFuncType	Определяет операцию, которая будет использоваться при поиске и фильтрации данных.
CustomSchemas	Подключает к мобильному приложению дополнительные схемы.
Icons	Позволяет добавить в приложение пользовательские изображения.
DefaultModuleImageId	Устанавливает изображение по умолчанию для пользовательского интерфейса V1.
DefaultModuleImageIdV2	Устанавливает изображение по умолчанию для пользовательского интерфейса V2.

Все свойства конфигурационного объекта манифеста условно можно разделить на три группы (рис. 1):

- *Свойства интерфейса приложения* — содержит свойства, с помощью которых формируется интерфейс мобильного приложения. При помощи свойств этой группы происходит формирование разделов приложения, главного меню, настраиваются пользовательские изображения. Подробнее о свойствах, входящих в эту группу, можно узнать в статье "**Манифест. Свойства интерфейса приложения (Section 1.3.1.1)**".
- *Свойства данных и бизнес-логики* — содержит свойства, в которых описываются импортируемые данные, а также пользовательская бизнес-логика обработки этих данных в мобильном приложении. Подробнее о свойствах, входящих в эту группу, можно узнать в статье "**Манифест. Свойства данных и бизнес-логики (Section 1.3.1.2)**".
- *Свойства синхронизации приложений* — содержит единственное свойство настройки синхронизации данных с основным приложением. Подробнее об этом свойстве можно узнать в статье "**Манифест. Свойства синхронизации приложений (Section 1.3.1.3)**".

Рис. 1. — Группы свойств конфигурационного объекта манифеста



Смотрите также

- **Манифест. Свойства интерфейса приложения (Section 1.3.1.1)**
- **Манифест. Свойства данных и бизнес-логики (Section 1.3.1.2)**
- **Манифест. Свойства синхронизации приложений (Section 1.3.1.3)**
- **Экспорт данных в пакетном режиме (Section 1.3.1.4)**

1.3.1.1 Манифест. Свойства интерфейса приложения

Уровень сложности

**Общие положения**

Условная группа свойств конфигурационного объекта манифеста. Содержит свойства, с помощью которых формируется интерфейс мобильного приложения. При помощи свойств этой группы происходит формирование разделов приложения, главного меню, настраиваются пользовательские изображения. Подробнее о манифесте мобильного приложения и всех его свойствах можно узнать в статье "**Манифест мобильного приложения (Section 1.3.1)**".

Свойство ModuleGroups

Группы модулей приложения. Описывает верхнеуровневую настройку групп главного меню мобильного приложения. Для каждой группы меню задается списком именованных конфигурационных объектов с единственным возможным свойством Position (см. табл. 1).

Табл. 1. — Свойство конфигурационного объекта для настройки группы меню.

Свойство	Значение
Position	Позиция группы в главном меню. Начинается с 0.

Пример

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

```
// Группы 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
  }
}
```

Свойство Modules

Модули мобильного приложения. Модуль представляет собой раздел приложения. Каждый модуль в свойстве [Modules] конфигурационного объекта манифеста описывается конфигурационным объектом со свойствами, приведенными в таблице 2. Имя конфигурационного объекта раздела должно совпадать с названием модели, которая предоставляет данные раздела.

Табл. 2. — Свойства конфигурационного объекта раздела.

Свойство	Значение
Group	Группа меню приложения, в которой размещается раздел. Задается строкой с названием соответствующего раздела меню из свойства <i>ModuleGroups</i> конфигурационного объекта манифеста.
Model	Название модели, которая предоставляет данные раздела. Задается строкой с названием одной из моделей, объявленных в свойстве <i>Models</i> конфигурационного объекта манифеста.
Position	Позиция раздела в группе главного меню. Задается числовым значением, начиная с

	о.
Title	Заголовок раздела. Строка с названием локализованного значения заголовка раздела. Локализованное значение заголовка раздела должно быть добавлено в блок <code>[LocalizableStrings]</code> 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 (<i>true</i> — скрыт, <i>false</i> — отображается). Необязательное свойство. По умолчанию — <i>false</i> .

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".

```
// 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
  }
}

```

Свойство Icons

Свойство предназначено для подключения к мобильному приложению пользовательских изображений.

Задается массивом конфигурационных объектов, каждый из которых имеет свойства, приведенные в таблице 3.

Табл. 3. — Свойства конфигурационного объекта для подключения пользовательского изображения.

Свойство	Значение
ImageListId	Идентификатор списка изображений.
ImageId	Идентификатор подключаемого изображения из списка <i>ImageListId</i> .

Пример

```

// 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

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.

Пример

```

//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"

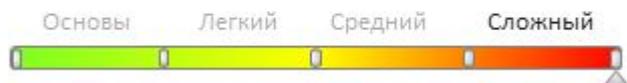
```

Смотрите также

- **Манифест. Свойства данных и бизнес-логики (Section 1.3.1.2)**
- **Манифест. Свойства синхронизации приложений (Section 1.3.1.3)**
- **Экспорт данных в пакетном режиме (Section 1.3.1.4)**

1.3.1.2 Манифест. Свойства данных и бизнес-логики

Уровень сложности



Общие положения

Условная группа свойств конфигурационного объекта манифеста. Содержит свойства, в которых описываются импортируемые данные, а также пользовательская бизнес-логика обработки этих данных в мобильном приложении. Подробнее о манифесте мобильного приложения и всех его свойствах можно узнать в статье "[Манифест мобильного приложения \(Section 1.3.1\)](#)".

Свойство Models

Содержит импортируемые модели приложения. Каждая модель в свойстве описывается конфигурационным объектом с соответствующим именем. Свойства конфигурационного объекта модели представлены в табл. 1.

Табл. 1. — Свойства конфигурационного объекта модели.

Свойство	Значение
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 редактирования 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.).

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.

Свойство *Models* конфигурационного объекта манифеста должно выглядеть следующим образом:

```
// Импортируемые модели.
"Models": {
  // Модель "Контакт"
  "Contact": {
    // Схема страницы реестра.
    "Grid": "MobileContactGridPage",
    // Схема страницы просмотра.
    "Preview": "MobileContactPreviewPage",
```

```

// Схема страницы редактирования.
"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"
    ]
}
}
}

```

Свойство PreferredFilterFuncType

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*. Перечень функций фильтрации приведен в таблице 2.

Табл. 2. — Функции фильтрации (*Terrasoft.FilterFunctions*)

Функция	Значение
<i>SubStringOf</i>	Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки <i>property</i> .
<i>ToUpper</i>	Приводит значения колонки, заданной в <i>property</i> , к верхнему регистру.
<i>EndsWith</i>	Проверяет, оканчивается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.
<i>StartsWith</i>	Проверяет, начинается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.
<i>Year</i>	Возвращает год по значению колонки <i>property</i> .
<i>Month</i>	Возвращает месяц по значению колонки <i>property</i> .
<i>Day</i>	Возвращает день по значению колонки <i>property</i> .
<i>In</i>	Проверяет вхождение значения колонки <i>property</i> в диапазон значений, переданных в качестве аргумента функции.
<i>NotIn</i>	Проверяет невхождение значения колонки <i>property</i> 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.

Пример

Для поиска данных использовать функцию поиска подстроки.

Свойство *PreferedFilterFuncType* конфигурационного объекта манифеста должно выглядеть следующим образом:

```
// 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.

Свойство CustomSchemas

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.

Пример

Подключить дополнительные пользовательские схемы регистрации действий и утилит.

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

Смотрите также

- **Манифест мобильного приложения (Section 1.3.1)**
- **Манифест. Свойства интерфейса приложения (Section 1.3.1.1)**
- **Манифест. Свойства синхронизации приложений (Section 1.3.1.3)**

1.3.1.3 Манифест. Свойства синхронизации приложений

Уровень сложности



Общие положения

Условная группа свойств конфигурационного объекта манифеста. Содержит единственное свойство, с помощью которого выполняются настройки синхронизации данных с основным приложением. Подробнее о манифесте мобильного приложения и всех его свойствах можно узнать в статье "**Манифест мобильного**

приложения (Section 1.3.1)".

Свойство SyncOptions

Описывает параметры для настройки синхронизации данных. Содержит конфигурационный объект со свойствами, представленными в таблице 1.

Табл. 1. — Свойства конфигурационного объекта для настроек синхронизации.

Свойство	Значение
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, приведенными в таблице 2.

Табл. 2. — Свойства конфигурационного объекта для настройки синхронизации модели.

Свойство	Значение
Name	Название модели (см. свойство <i>Models</i> конфигурационного объекта манифеста).
SyncColumns	Массив колонок модели, для которых импортируются данные. Помимо явно перечисленных колонок, при синхронизации в обязательном порядке будут импортироваться системные колонки (<i>CreatedOn</i> , <i>CreatedBy</i> , <i>ModifiedOn</i> , <i>ModifiedBy</i>) 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, представленными в таблице 3.

Табл. 3. — Свойства конфигурационного объекта фильтра модели.

Свойство	Значение
type	Тип фильтра. Задается значением перечисления <i>Terrasoft.FilterTypes</i> . Необязательное свойство. По умолчанию <i>Terrasoft.FilterTypes.Simple</i> . Типы фильтров (<i>Terrasoft.FilterTypes</i>): <i>Simple</i> — Фильтр с одним условием. <i>Group</i> — Групповой фильтр с несколькими условиями.
logicalOperation	Логическая операция объединения коллекции фильтров (для фильтров с типом <i>Terrasoft.FilterTypes.Group</i>). Задается значением перечисления <i>Terrasoft.FilterLogicalOperations</i> . Значение по умолчанию - <i>Terrasoft.FilterLogicalOperations.And</i> . Виды логических операций (<i>Terrasoft.FilterLogicalOperations</i>): <i>Or</i> — Логическая операция ИЛИ. <i>And</i> — Логическая операция И.
subfilters	Коллекция фильтров, применяемых к модели. Обязательное свойство для типа фильтра <i>Terrasoft.FilterTypes.Group</i> . Фильтры между собой объединяются

	логической операцией, указанной в свойстве <i>logicalOperation</i> . Каждый фильтр представляет собой конфигурационный объект фильтра.																				
property	Название колонки модели, по которой выполняется фильтрация. Обязательное свойство для типа фильтра <i>Terrasoft.FilterTypes.Simple</i> .																				
valueIsMacrosType	Признак, определяющий, является ли значение для фильтрации макросом. Необязательное свойство. Может принимать значения: <i>true</i> , если для фильтрации используется макрос, иначе — <i>false</i> .																				
value	Значение для фильтрации колонки, указанной в свойстве <i>property</i> . Обязательное свойство для типа фильтра <i>Terrasoft.FilterTypes.Simple</i> . Может задаваться непосредственно значением для фильтрации (в том числе, может быть <i>null</i>) либо макросом (для этого свойство <i>valueIsMacrosType</i> должно иметь значение <i>true</i>). Макросы, которые можно использовать в качестве значения свойства, содержатся в перечислении <i>Terrasoft.ValueMacros</i> . Макросы значения (<i>Terrasoft.ValueMacros</i>): <table> <tr> <td><i>CurrentUserContactId</i></td> <td>Идентификатор текущего пользователя.</td> </tr> <tr> <td><i>CurrentDate</i></td> <td>Текущая дата.</td> </tr> <tr> <td><i>CurrentDateTime</i></td> <td>Текущие дата и время.</td> </tr> <tr> <td><i>CurrentDateEnd</i></td> <td>Полная дата окончания текущей даты.</td> </tr> <tr> <td><i>CurrentUserContactName</i></td> <td>Имя текущего контакта.</td> </tr> <tr> <td><i>CurrentUserContact</i></td> <td>Идентификатор и имя текущего контакта.</td> </tr> <tr> <td><i>SysSettings</i></td> <td>Значение системной настройки. Имя системной настройки передается в свойстве <i>macrosParams</i>.</td> </tr> <tr> <td><i>CurrentTime</i></td> <td>Текущее время.</td> </tr> <tr> <td><i>CurrentUserAccount</i></td> <td>Идентификатор и имя контрагента текущего пользователя.</td> </tr> <tr> <td><i>GenerateUid</i></td> <td>Сгенерированный идентификатор.</td> </tr> </table>	<i>CurrentUserContactId</i>	Идентификатор текущего пользователя.	<i>CurrentDate</i>	Текущая дата.	<i>CurrentDateTime</i>	Текущие дата и время.	<i>CurrentDateEnd</i>	Полная дата окончания текущей даты.	<i>CurrentUserContactName</i>	Имя текущего контакта.	<i>CurrentUserContact</i>	Идентификатор и имя текущего контакта.	<i>SysSettings</i>	Значение системной настройки. Имя системной настройки передается в свойстве <i>macrosParams</i> .	<i>CurrentTime</i>	Текущее время.	<i>CurrentUserAccount</i>	Идентификатор и имя контрагента текущего пользователя.	<i>GenerateUid</i>	Сгенерированный идентификатор.
<i>CurrentUserContactId</i>	Идентификатор текущего пользователя.																				
<i>CurrentDate</i>	Текущая дата.																				
<i>CurrentDateTime</i>	Текущие дата и время.																				
<i>CurrentDateEnd</i>	Полная дата окончания текущей даты.																				
<i>CurrentUserContactName</i>	Имя текущего контакта.																				
<i>CurrentUserContact</i>	Идентификатор и имя текущего контакта.																				
<i>SysSettings</i>	Значение системной настройки. Имя системной настройки передается в свойстве <i>macrosParams</i> .																				
<i>CurrentTime</i>	Текущее время.																				
<i>CurrentUserAccount</i>	Идентификатор и имя контрагента текущего пользователя.																				
<i>GenerateUid</i>	Сгенерированный идентификатор.																				
macrosParams	Значения, которые передаются в макрос в качестве параметра. Необязательное свойство. В настоящее время используется только для макроса <i>Terrasoft.ValueMacros.SysSettings</i> .																				
isNot	Определяет, применяется к фильтру оператор отрицания. Необязательное свойство. Принимает значение <i>true</i> , если к фильтру применяется оператор отрицания, иначе — <i>false</i> .																				
funcType	Тип функции, которая применяется к колонке модели, заданной в свойстве <i>property</i> . Необязательное свойство. Может принимать значения перечисления <i>Terrasoft.FilterFunctions</i> . Значения аргументов для функций фильтрации задаются в свойстве <i>funcArgs</i> . Значение, с которым сравнивается результат функции, задается свойством <i>value</i> . Функции фильтрации (<i>Terrasoft.FilterFunctions</i>): <table> <tr> <td><i>SubStringOf</i></td> <td>Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки <i>property</i>.</td> </tr> <tr> <td><i>ToUpper</i></td> <td>Приводит значения колонки, заданной в <i>property</i>, к верхнему регистру.</td> </tr> <tr> <td><i>EndsWith</i></td> <td>Проверяет, оканчивается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.</td> </tr> <tr> <td><i>StartsWith</i></td> <td>Проверяет, начинается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.</td> </tr> <tr> <td><i>Year</i></td> <td>Возвращает год по значению колонки <i>property</i>.</td> </tr> <tr> <td><i>Month</i></td> <td>Возвращает месяц по значению колонки <i>property</i>.</td> </tr> </table>	<i>SubStringOf</i>	Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки <i>property</i> .	<i>ToUpper</i>	Приводит значения колонки, заданной в <i>property</i> , к верхнему регистру.	<i>EndsWith</i>	Проверяет, оканчивается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.	<i>StartsWith</i>	Проверяет, начинается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.	<i>Year</i>	Возвращает год по значению колонки <i>property</i> .	<i>Month</i>	Возвращает месяц по значению колонки <i>property</i> .								
<i>SubStringOf</i>	Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки <i>property</i> .																				
<i>ToUpper</i>	Приводит значения колонки, заданной в <i>property</i> , к верхнему регистру.																				
<i>EndsWith</i>	Проверяет, оканчивается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.																				
<i>StartsWith</i>	Проверяет, начинается ли значение колонки <i>property</i> значением, переданным в качестве аргумента.																				
<i>Year</i>	Возвращает год по значению колонки <i>property</i> .																				
<i>Month</i>	Возвращает месяц по значению колонки <i>property</i> .																				

<i>Day</i>	Возвращает день по значению колонки <i>property</i> .
<i>In</i>	Проверяет вхождение значения колонки <i>property</i> в диапазон значений, переданных в качестве аргумента функции.
<i>NotIn</i>	Проверяет невхождение значения колонки <i>property</i> в диапазон значений, переданных в качестве аргумента функции.
<i>Like</i>	Определяет, совпадает ли значение колонки <i>property</i> с заданным шаблоном.
funcArgs	Массив значений аргументов для функции фильтрации, заданной в свойстве <i>funcType</i> . Порядок значений в массиве <i>funcArgs</i> должен соответствовать порядку параметров функции <i>funcType</i> .
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 <i>Terrasoft.FilterOperation</i> . По умолчанию имеет значение <i>Terrasoft.FilterOperation.General</i> . Операции фильтрации (<i>Terrasoft.FilterOperation</i>): <i>General</i> Стандартная фильтрация. <i>Any</i> Фильтрация с применением фильтра <i>exists</i> .
compareType	Тип операции сравнения в фильтре. Необязательный параметр. Принимает значения из перечисления <i>Terrasoft.ComparisonType</i> . По умолчанию — <i>Terrasoft.ComparisonType.Equal</i> . Операции сравнения (<i>Terrasoft.ComparisonType</i>): <i>Equal</i> Равно. <i>LessOrEqual</i> Меньше или равно. <i>NotEqual</i> Не равно. <i>Greater</i> Больше. <i>GreaterOrEqual</i> Больше или равно. <i>Less</i> Меньше.

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.

Свойство *SyncOptions* конфигурационного объекта манифеста должно выглядеть следующим образом:

```
// Настройки синхронизации.
"SyncOptions": {
  // Количество страниц, импортируемых в одном потоке.
  "ImportPageSize": 100,
  // Количество потоков импорта.
  "PagesInImportTransaction": 5,
```

```

// Массив импортируемых системных настроек.
"SysSettingsImportConfig": [
    "SchedulerDisplayTimingStart", "PrimaryCulture", "PrimaryCurrency",
"MobileApplicationMode", "CollectMobileAppUsageStatistics",
"CanCollectMobileUsageStatistics", "MobileAppUsageStatisticsEmail",
"MobileAppUsageStatisticsStorePeriod", "MobileSectionsWithSearchOnly",
"MobileShowMenuOnApplicationStart", "MobileAppCheckUpdatePeriod",
"ShowMobileLocalNotifications", "UseMobileUIV2"
],
// Массив импортируемых системных справочников.
"SysLookupsImportConfig": [
    "ActivityCategory", "ActivityPriority", "ActivityResult",
"ActivityResultCategory", "ActivityStatus", "ActivityType", "AddressType",
"AnniversaryType", "InformationSource", "MobileApplicationMode",
"OppContactInfluence", "OppContactLoyalty", "OppContactRole", "OpportunityStage",
"SupplyPaymentDelay", "SupplyPaymentState", "SupplyPaymentType"],
// 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", "ActivityCategory", "Priority", "Owner", "Account", "Contact",
"ShowInScheduler", "Author", "Type"
        ]
    },
    // Модель ActivityType загружается полностью.
    {
        "Name": "ActivityType",
        "SyncColumns": []
    }
]
}

```

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 [службы DataService](#). Ранее для фильтрации данных использовалось свойство *SyncFilter*, а импорт выполнялся с помощью [OData \(EntityDataService\)](#).

ВАЖНО

Импорт данных с помощью службы DataService доступен только для платформ Android и iOS. Для платформы Windows используется OData (EntityDataService).

Формат фильтра *QueryFilter* представляет собой набор параметров в виде JSON-объекта, передаваемых в запросе к службе DataService. Описание параметров DataService можно найти в статье "[DataService. Фильтрация данных](#)" документации по разработке Creatio.

Пример exists-фильтра приведен ниже:

```
{
  "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,
                      "macroType": 2
                    }
                  }
                }
              }
            }
          }
        }
      }
    ]
  }
}
```

Смотрите также

- **Манифест мобильного приложения (Section 1.3.1)**
- **Манифест. Свойства интерфейса приложения (Section 1.3.1.1)**
- **Манифест. Свойства данных и бизнес-логики (Section 1.3.1.2)**

1.3.1.4 Экспорт данных в пакетном режиме

Уровень сложности



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.

1.3.2 Жизненный цикл страниц в мобильном приложении

Уровень сложности



Общие сведения

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.

Заккрытие страницы

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

Заккрытие 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.

Обработчики событий жизненного цикла

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

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.

Навигация страниц

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

forward(openingPageConfig)

Метод открывает страницу с учетом свойств конфигурационного объекта-параметра *openingPageConfig*. Основные свойства этого объекта представлены в таблице 1.

Табл. 1. — Свойства объекта *openingPageConfig*

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

Навигация с использованием маршрутов

Маршрутизация

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* — выполняют восстановление предыдущего маршрута при возврате.

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

add(name, config)

Добавляет маршрут. Параметры:

- *name* — уникальное имя маршрута. В случае повторного добавления, последний переопределяет 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:

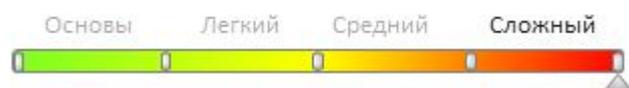
```
var mainPageController = Terrasoft.util.getMainController();
Terrasoft.Router.route("record", mainPageController, [{pageSchemaName:
"MobileActivityGridPage"}]);
```

back()

Выполняет закрытие текущего маршрута и восстановление предыдущего.

1.3.3 Фоновое обновление конфигурации в мобильном приложении

Уровень сложности



Общая информация

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] (рис. 1).

Рис. 1. — Системная настройка [Периодичность проверки обновлений]

Название *	Периодичность проверки обновлений	Код *	MobileAppCheckUpdatePeriod
Тип *	Целое число	Кешируется	<input checked="" type="checkbox"/>
Значение по умолчанию	10	Персональная	<input type="checkbox"/>
		Разрешить для пользователей портала	<input type="checkbox"/>
	Значение в часах		
Описание			

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];
- 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.

3. На платформе Windows 10 приложение при старте проверяет (не в фоновом режиме) наличие на сервере обновлений.

В случае наличия обновлений отобразится страница с соответствующей информацией.

1.3.4 Получение настроек и данных раздела [Итоги]

Уровень сложности



Общая информация

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

AnalyticsService

Класс, реализующий сервис *AnalyticsService*, содержит следующие публичные методы:

- *public Stream GetDashboardViewConfig(Guid id)* — возвращает настройки представления и виджетов на вкладке итогов по идентификатору страницы итогов.
- *public Stream GetDashboardData(Guid id, int timeZoneOffset)* — возвращает данные по всем виджетам на вкладке итогов по идентификатору страницы итогов.
- *public 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.

Примеры запросов к сервису AnalyticsService

HEADERS

Accept:application/json

Метод GetDashboardViewConfig()

URL

POST /0/rest/AnalyticsService/GetDashboardViewConfig

Содержимое запроса

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

Содержимое ответа

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

Метод GetDashboardData()

URL

POST /0/rest/AnalyticsService/GetDashboardData

Содержимое запроса

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

```
    "timeZoneOffset": 120
  }
}
```

Содержимое ответа

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

Метод GetDashboardItemData()

URL

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

Содержимое запроса

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

Содержимое ответа

```
{
  "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": []
}
```

}

1.3.5 Автоматическое разрешение конфликтов при синхронизации

Уровень сложности



Общая информация

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 дублирующей 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, представлен схематически на рис. 1:

Рис. 1. — Схема разрешения конфликта, возникшего в результате слияния дублей на сервере



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 (см. “Слияние 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.

1.4 Mobile SDK

Содержание



SDK реестра (Section 1.4.1)

Классы, методы и свойства реестра мобильного приложения Creatio.



Бизнес-правила мобильного приложения (Section 1.4.2)

Классы, методы и свойства бизнес-правил мобильного приложения Creatio.



Пользовательские бизнес-правила мобильного приложения (Section 1.4.3)

Классы, методы и свойства пользовательского бизнес-правила мобильного приложения Creatio.

1.4.1 SDK реестра

Уровень сложности



ВАЖНО

Эта статья актуальна для мобильного приложения версии 7.11.1 и выше.

Общие сведения

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

Методы Terrasoft.sdk.GridPage

setPrimaryColumn()

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

Сигнатура метода

```
setPrimaryColumn(modelName, column)
```

Параметры

modelName — название модели.

column — название колонки.

Пример вызова

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

setSubtitleColumns()

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

Сигнатура метода

```
setSubtitleColumns(modelName, columns)
```

Параметры

modelName — название модели.

columns — массив колонок или конфигурационных объектов колонок.

Пример вызова

Вариант 1

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

Вариант 2

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

setGroupColumns()

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

Сигнатура метода

```
setGroupColumns(modelName, columns)
```

Параметры

modelName — название модели.

columns — массив колонок или конфигурационных объектов колонок.

Пример вызова

Вариант 1

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

Вариант 2

```
Terrasoft.sdk.GridPage.setGroupColumns("Case", [
{
name: "Symptoms",
isMultiline: true, //Отображать как многострочное поле
label: "CaseGridSymptomsColumnLabel", //Имя локализованной строки
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.

Пример

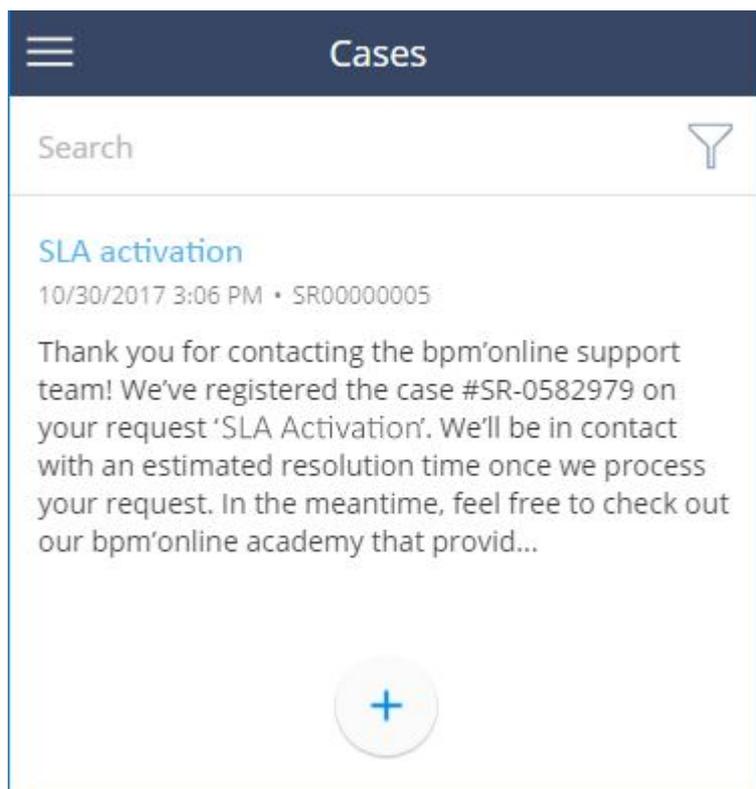
Необходимо настроить реестр раздела [Обращения] ([Cases]) таким образом, чтобы отображался заголовок с темой обращения, подзаголовок с датой регистрации и номером, а также описание обращения в виде многострочного поля.

Для настройки реестра необходимо использовать приведенный ниже исходный код:

```
// 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 так, как показано на рис. 1.

Рис. 1. — Настроенный реестр обращений



1.4.2 Бизнес-правила мобильного приложения

Уровень сложности



Общие сведения

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

ВАЖНО

Бизнес-правила работают только на страницах редактирования 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 редактирования, например, "Contact".
- *config* — объект, определяющий свойства бизнес-правила. Перечень свойств зависит от конкретного типа бизнес-правила.

Базовое бизнес-правило

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

Свойства конфигурационного объекта *config*, которые могут быть использованы наследниками базового бизнес-правила:

- *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*.

Перечисление *Terrasoft.BusinessRuleEvents* содержит следующие значения:

- *Terrasoft.BusinessRuleEvents.Save* — правило отработает перед сохранением данных.
- *Terrasoft.BusinessRuleEvents.ValueChanged* — правило отработает после изменения данных (при редактировании).
- *Terrasoft.BusinessRuleEvents.Load* — правило отработает при открытии страницы редактирования.

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

Определяет обязательность заполнения поля на странице редактирования. Используемые 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.

Перечисление *Terrasoft.RequirementTypes* содержит следующие значения:

- *Terrasoft.RequirementTypes.Simple* — проверка значения в одной колонке.
- *Terrasoft.RequirementTypes.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*)

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

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

Пример использования

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

```
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. Используемые 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.

Примеры использования

Пример фильтрации 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*.

```
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 фильтрации по значению другой колонки

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

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

Бизнес-правило [Взаимная фильтрация] (Terrasoft.RuleTypes.MutualFiltration)

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].

Используемые 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.

Примеры использования

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)

Выполняет проверку соответствия значения колонки регулярному выражению. Используемые 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"]
});

```

1.4.3 Пользовательские бизнес-правила мобильного приложения

Уровень сложности



Общие сведения

Бизнес-правила являются одним из механизмов Creatio, позволяющим настраивать поведение полей на странице редактирования записи. С их помощью можно, например, настроить обязательность и видимость полей, их доступность и т. п. Подробно бизнес-правила рассмотрены в статье "**Бизнес-правила мобильного приложения (Section 1.4.2)**".

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

Свойства конфигурационного объекта *config*

При добавлении пользовательского бизнес-правила с помощью метода `Terrasoft.sdk.Model.addBusinessRule(name, config)` можно использовать свойства конфигурационного объекта `config` базового бизнес-правила (см. "**Бизнес-правила мобильного приложения (Section 1.4.2)**"). Также дополнительно предусмотрено свойство `executeFn`.

Используемые 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`.
- `executeFn` — функция-обработчик, содержащая пользовательскую логику выполнения бизнес-правила.

Перечисление `Terrasoft.BusinessRuleEvents` содержит следующие значения:

- `Terrasoft.BusinessRuleEvents.Save` — правило сработает перед сохранением данных.
- `Terrasoft.BusinessRuleEvents.ValueChanged` — правило сработает после изменения данных (при редактировании).
- `Terrasoft.BusinessRuleEvents.Load` — правило сработает при открытии страницы редактирования.

Свойства функции-обработчика `executeFn`

Функция-обработчик, объявляемая в свойстве `executeFn` должна иметь следующую сигнатуру:

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

Параметры функции:

- `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`. Рекомендуются следующие 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.

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. Возможные 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.

Например, для 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"]
  }
});
```

Примеры пользовательского бизнес-правила

Пример 1

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

```
// Правило для страницы редактирования 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.Finished &&
            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]);
}
});

```

Пример 2

Добавление и удаление фильтрации по пользовательской логике.

```

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]);
    }
});

```

Пример 3

Пример логики сбрасывания 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);
    }
});

```

Пример 4

Пример генерации заголовка активности для решения FieldForce.

```

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.FieldForceTitlePrefix, accountName);
            record.set("Title", title, true);
        }
        Ext.callback(callbackConfig.success, callbackConfig.scope, [true]);
    }
});

```

1.5 Разработка Mobile Creatio на примерах

Содержание



Как добавить стандартную деталь с колонками (Section 1.5.1)

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.



Модификаторы доступа страницы в мобильном приложении (Section 1.5.3)

В мобильном приложении версий 7.11.0 добавилась полноценная возможность настройки модификаторов доступа раздела или стандартной детали. Например, можно запретить в разделе изменение, добавление и удаление записей для всех пользователей.



Добавление пользовательского типа виджета в мобильное приложение (Section 1.5.2)

Как добавить на страницу итогов мобильного приложения пользовательский виджет, отображающий текущие дату и время.



Добавление элементов управления на страницу раздела (Section 1.5.4)

Как добавить кнопку на страницу раздела.



Отображение страницы на планшете во весь экран (Section 1.5.5)

Как настроить отображение страницы на планшете во весь экран.

1.5.1 Как добавить стандартную деталь с колонками

Уровень сложности



Общие сведения

Для добавления детали в раздел мобильного приложения Creatio необходимо использовать мастер мобильного приложения. Как настроить деталь с помощью мастера мобильного приложения описывается в статье "[Настройка детали раздела](#)" документации Mobile Creatio.

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.

Описание примера

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

Исходный код

Пакет с реализацией примера можно скачать по [ссылке](#).

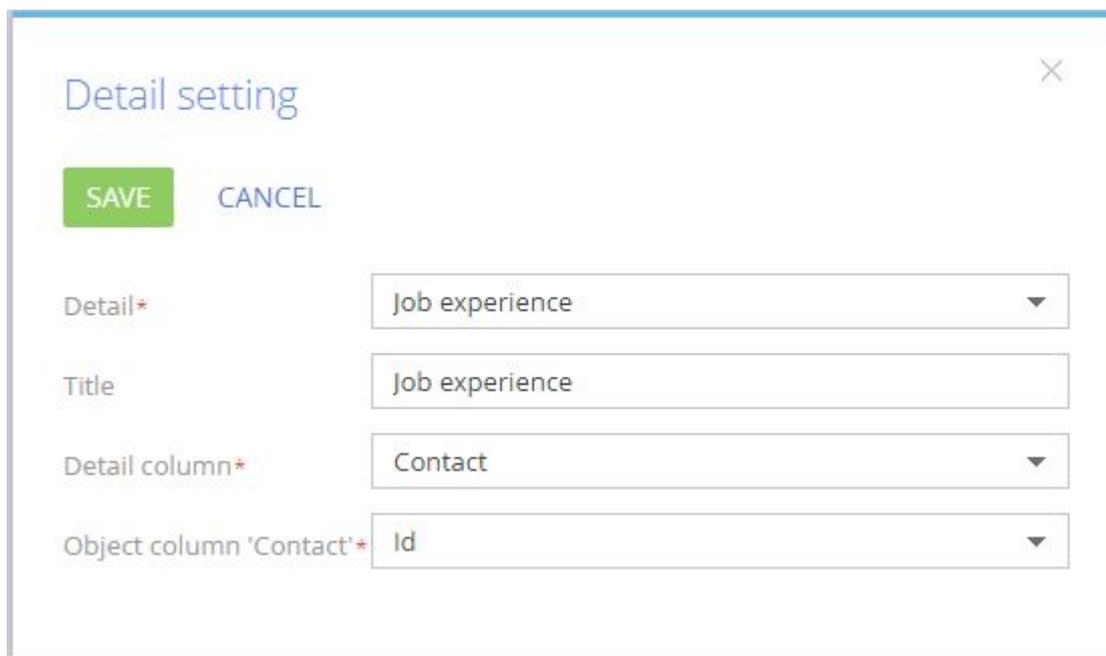
Алгоритм реализации кейса

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

Чтобы добавить деталь на страницу редактирования записи, используйте [мастер мобильного приложения](#). Для этого:

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

Рис. 1. — Настройка детали [Карьера] ([Job experience])



Detail setting

SAVE CANCEL

Detail* Job experience

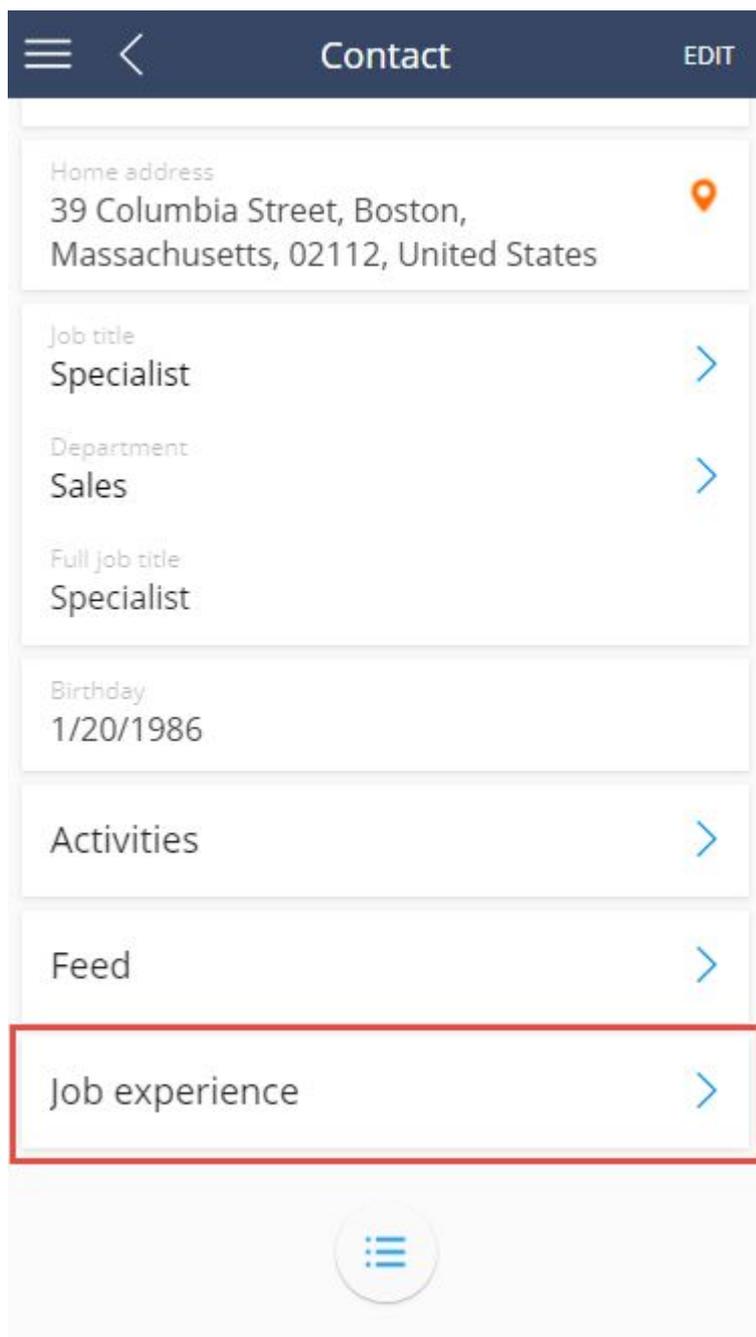
Title Job experience

Detail column* Contact

Object column 'Contact'* Id

После сохранения результатов настройки детали, раздела и рабочего места в мобильном приложении отобразится деталь [Карьера] ([Job experience]) (рис. 2).

Рис. 2. — Деталь [Карьера] ([Job experience]) на странице записи раздела [Контакты] ([Contacts])



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).

Рис. 3. — Отображение идентификатора связанной записи контакта

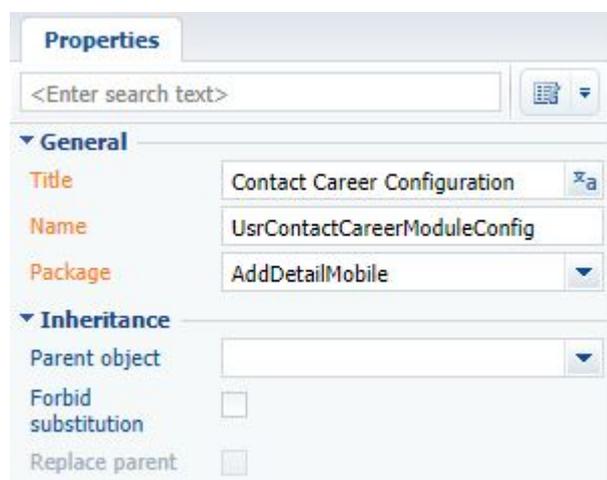


2. Создать схему модуля, в которой выполнить конфигурирование реестра детали

В разделе [\[Конфигурация\]](#) приложения Creatio в пользовательском пакете [создайте клиентский модуль](#) со следующими свойствами (рис. 4):

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

Рис. 4. – Свойства схемы модуля



Добавьте в схему модуля следующий исходный код:

```
// 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:

3.1. Откройте в дизайнера клиентского модуля схему манифеста мобильного приложения *MobileApplicationManifestDefaultWorkplace*. Эта схема создается в пользовательском пакете мастером мобильного приложения (см. **"Как добавить пользовательский раздел в мобильное приложение (on-line documentation)"**).

3.2. Добавьте модуль *UsrContactCareerModuleConfig* в секцию *PagesExtensions* модели *ContactCareer*:

```

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

```

3.3. Сохраните схему манифеста мобильного приложения

В результате выполнения примера на странице детали [Карьера] ([Job experience]) будут отображаться записи по колонке [Должность] (рис. 5).

Рис. 5. — Результат выполнения примера



ВАЖНО

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

1.5.2 Добавление пользовательского типа виджета в мобильное приложение

Уровень сложности



Общие сведения

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 (см. "**Получение настроек и данных раздела [Итоги] (Section 1.3.4)**"). Поддерживаются следующие типы блоков (виджетов) итогов — график, показатель, список и шкала.

Для добавления пользовательского типа виджета в мобильное приложение необходимо:

1. Реализовать интерфейс настройки виджета в приложении Creatio.
2. Добавить экземпляр реализованного пользовательского виджета в приложение.
3. Реализовать отображение виджета в мобильном приложении.

ВАЖНО

В данной статье описывается только реализация отображения виджета в мобильном приложении.

Чтобы пользовательский тип виджета отображался в мобильном приложении необходимо:

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

Описание кейса

На страницу итогов мобильного приложения добавить пользовательский виджет, отображающий текущие дату и время.

Алгоритм выполнения кейса

1. Реализация получения данных пользовательского типа виджета

Чтобы осуществить получение данных каждого пользовательского типа виджета 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 \[Исходный код\]](#).

Для пользовательского типа виджета *CustomDashboardItem* реализация класса получения данных будет следующей:

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

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

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*.

```
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 следующий исходный код:

```

Terrasoft.DashboardItemClassName.CustomDashboardItem =
"Terrasoft.configuration.controls.CustomDashboardItem";

```

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:

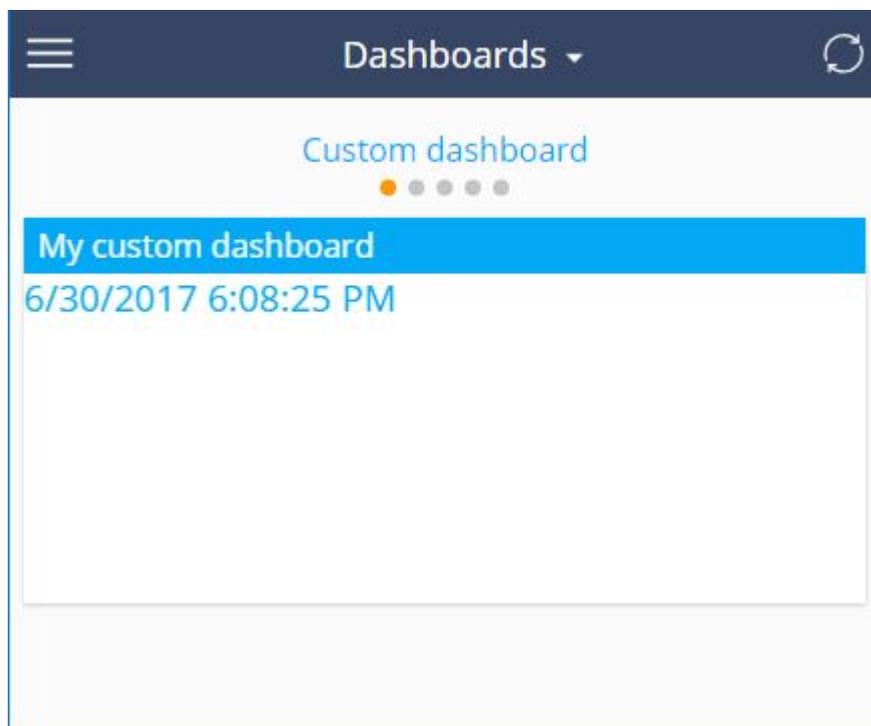
```

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

```

После сохранения всех изменений виджет будет отображаться в разделе [Итоги] мобильного приложения (рис. 1).

Рис. 1. — Результат выполнения кейса

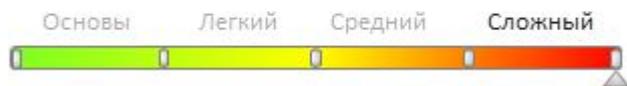


ВАЖНО

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

1.5.3 Модификаторы доступа страницы в мобильном приложении

Уровень сложности



В мобильном приложении версий 7.11.0 и выше добавилась полноценная возможность настройки модификаторов доступа раздела или стандартной детали. Например, можно запретить в разделе изменение, добавление и удаление записей для всех пользователей.

Для установки доступа в режим чтения необходимо в схему, название которой содержит значение "ModuleConfig", добавить следующий исходный код :

```
Terrasoft.sdk.Module.setChangeModes ("UsrClaim", [Terrasoft.ChangeModes.Read]);
```

Или для стандартной детали:

```
Terrasoft.sdk.Details.setChangeModes ("UsrClaim", "StandardDetailName", [Terrasoft.ChangeModes.Read]);
```

В результате на странице реестра будет удалена кнопка добавления, а на странице просмотра – кнопка редактирования. Также на странице просмотра будут отсутствовать кнопки действий [Добавить], [Удалить], [Добавить запись] для работы с записями раздела на встроенной детали др.

Модификаторы доступа можно комбинировать. Например, если нужно добавлять и изменять записи, но удалять их нельзя, то необходимо добавить следующий исходный код:

```
Terrasoft.sdk.Module.setChangeModes ("UsrClaim", [Terrasoft.ChangeModes.Create, Terrasoft.ChangeModes.Update]);
```

Все модификаторы доступа приведены в перечислении *Terrasoft.ChangeModes*.

1.5.4 Добавление элементов управления на страницу раздела

Общие сведения

На страницу раздела можно добавлять следующие элементы управления:

- группы колонок *Terrasoft.ColumnSet*;
- встроенные детали *Terrasoft.EmbeddedDetail*;
- стандартные детали;
- компоненты-наследники класса *Terrasoft.RecordPanelItem*.

Для добавления первых трех типов элементов управления рекомендуется использовать мастер настройки мобильного приложения.

Чтобы добавить на страницу раздела компонент-наследник класса *Terrasoft.RecordPanelItem* необходимо:

1. Создать пользовательский класс, расширяющий класс *Terrasoft.RecordPanelItem*, в котором описать конфигурационный объект компонента, а также методы, определяющие его функциональность.
2. Создать схему настроек раздела (*Mobile[Раздел]ModuleConfig*), в котором реализовать добавление созданного компонента на страницу раздела с помощью метода *addPanelItem()* класса *Terrasoft.sdk.RecordPage*.
3. Добавить созданные схемы в манифест мобильного приложения.

Описание примера

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

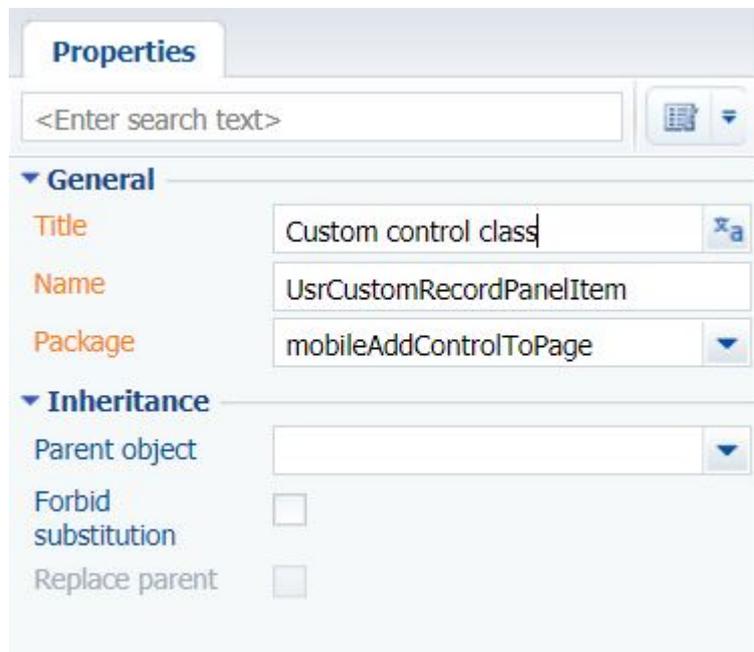
Алгоритм реализации кейса

1. Создать пользовательский класс-наследник *Terrasoft.RecordPanelItem*

В разделе [Конфигурация] приложения Creatio в пользовательском пакете [создайте клиентский модуль](#) со следующими свойствами (рис. 1):

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

Рис. 1. — Свойства схемы модуля



В модуль добавьте исходный код:

```
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:

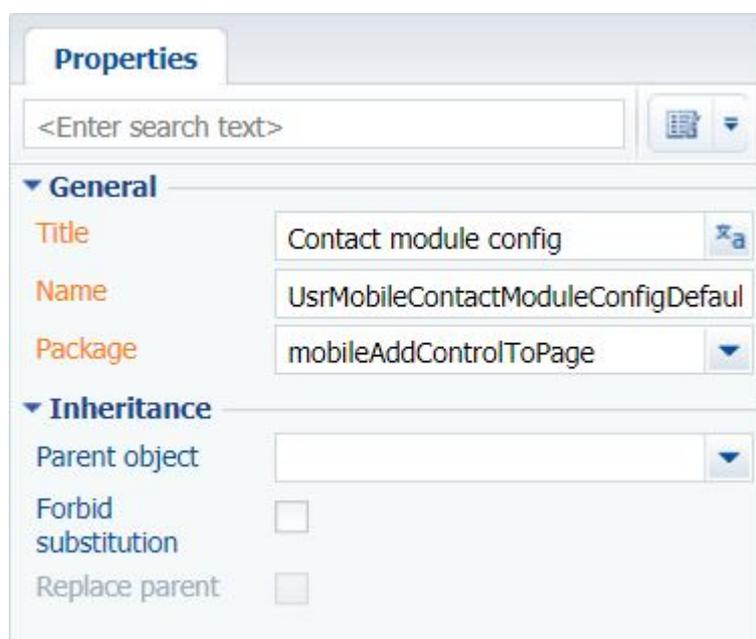
- `initialize()` - метод-обработчик события нажатия кнопки;
- `onClickMeButtonClick()` - метод, который инициализует созданный элемент и присваивает событию нажатия на кнопку ссылку на метод-обработчик.

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

В [разделе \[Конфигурация\]](#) приложения Creatio в пользовательском пакете [создайте клиентский модуль](#) со следующими свойствами (рис. 2):

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

Рис. 2. — Свойства схемы модуля



Добавьте в схему модуля следующий исходный код:

```
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:

3.1. Откройте в дизайнера клиентского модуля схему манифеста мобильного приложения `MobileApplicationManifestDefaultWorkplace`. Эта схема создается в пользовательском пакете мастером мобильного приложения (см. ["Как добавить пользовательский раздел в мобильное приложение \(on-line documentation\)"](#)).

3.2. Добавьте модуль `UsrCustomRecordPanelItem` в секцию `CustomSchemas`, а модуль `UsrContactCareerModuleConfig` в секцию `PagesExtensions` модели `Contact`:

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

3.3. 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(рис. 3, 4).

Рис. 3. — Результат выполнения кейса. Добавление кнопки

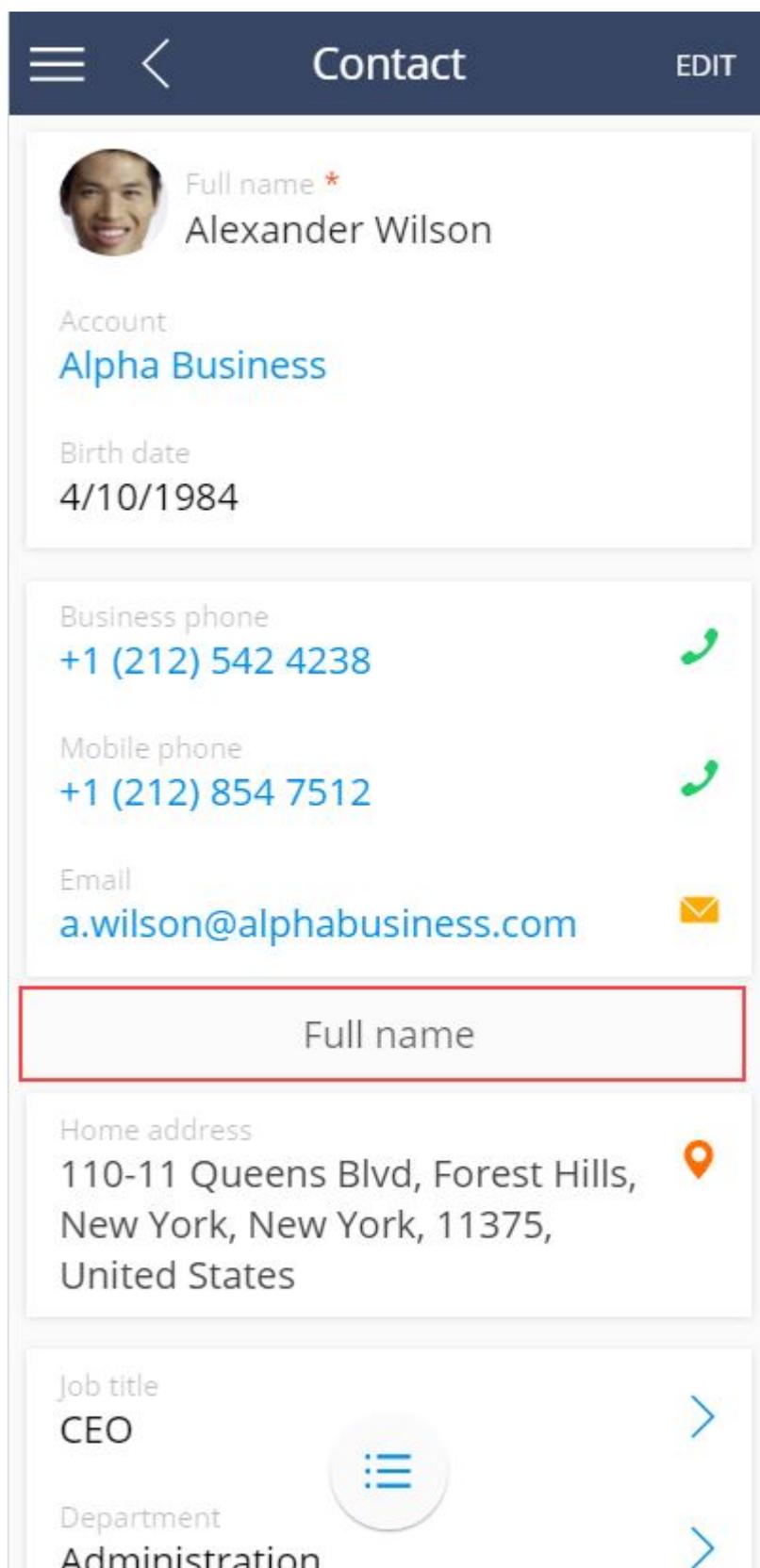
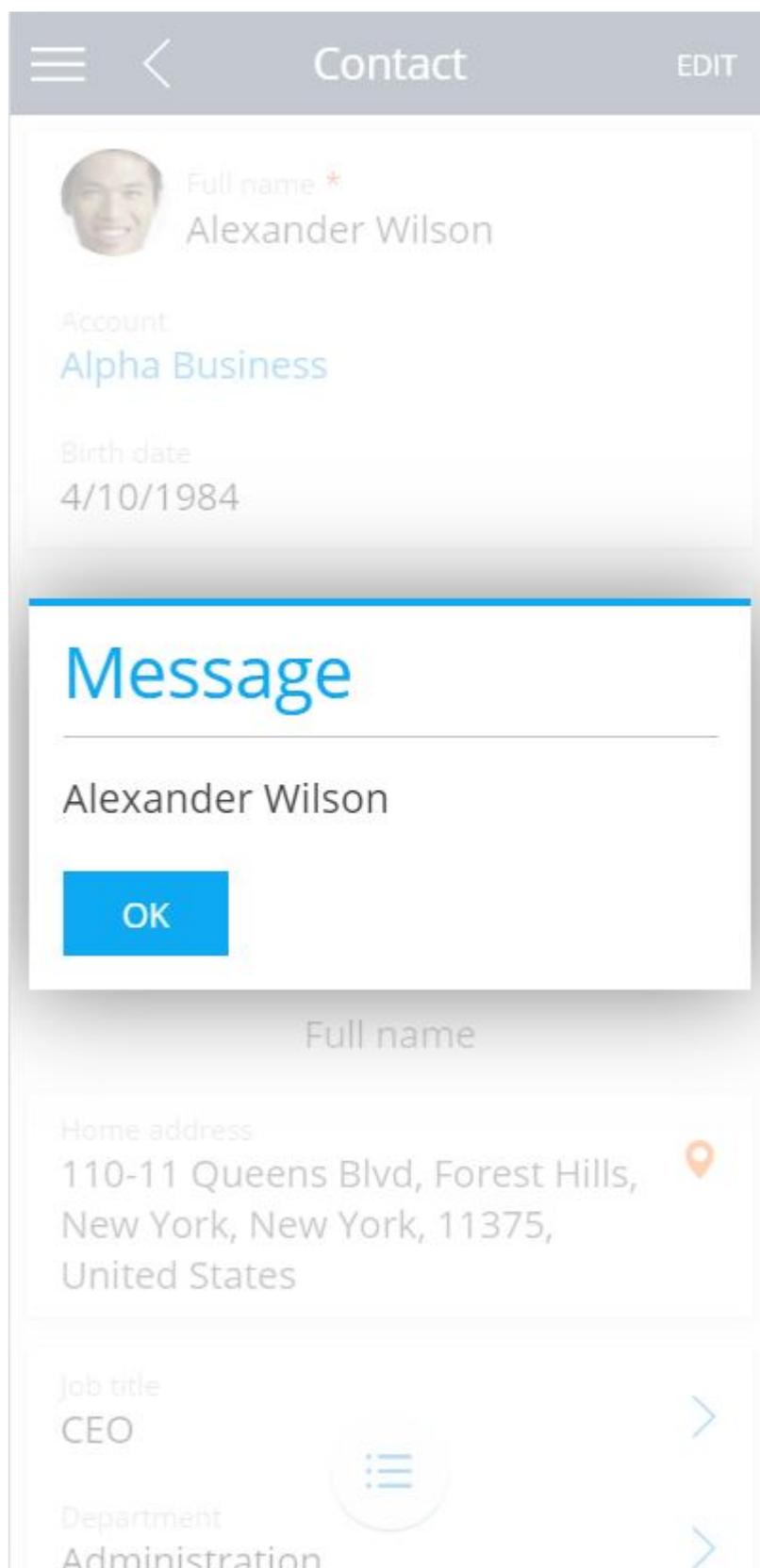
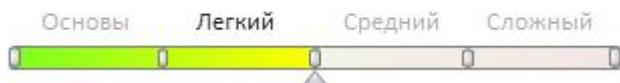


Рис. 4. — Результат выполнения кейса. Нажатие на кнопку



1.5.5 Отображение страницы на планшете во весь экран

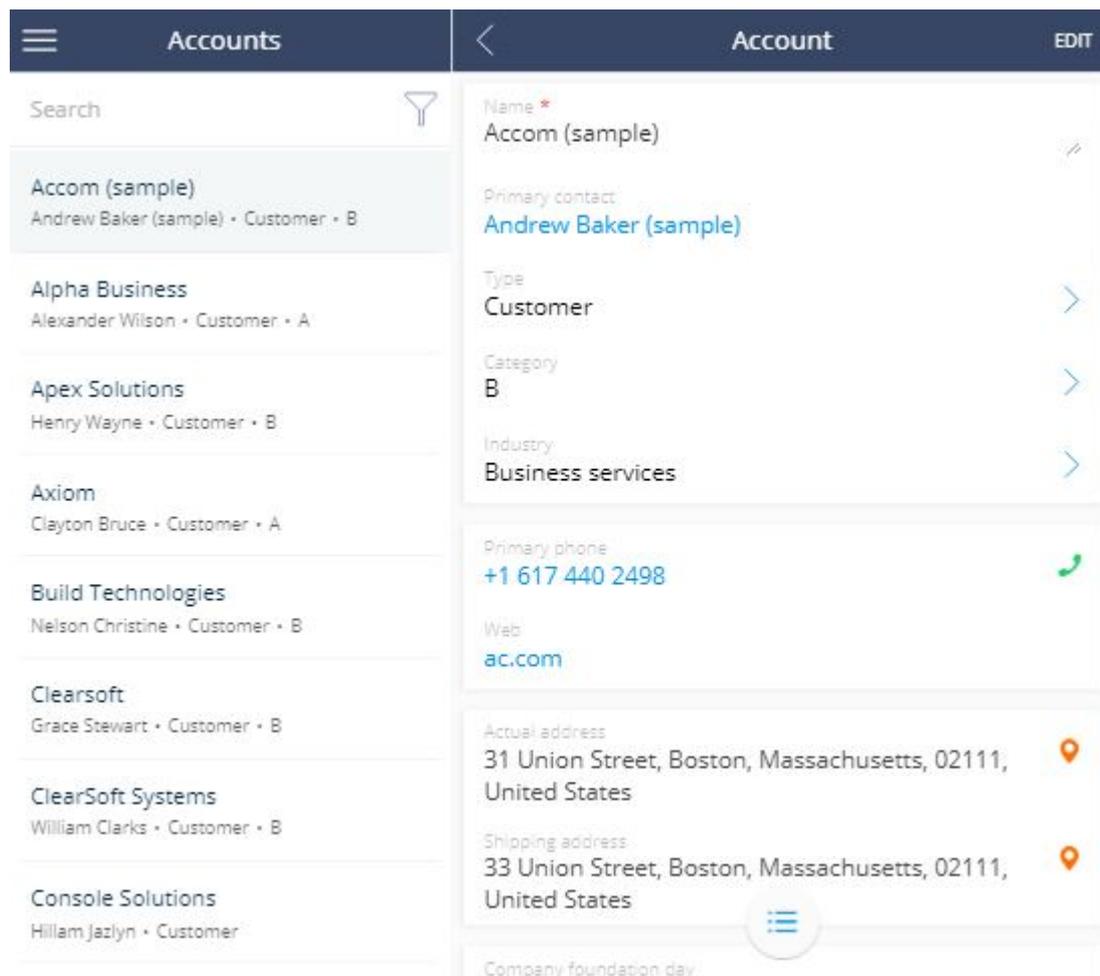
Уровень сложности



Общие сведения

При просмотре страницы раздела мобильного приложения Creatio на планшете по умолчанию срабатывает режим отображения реестра раздела в левой области экрана (рис. 1).

Рис. 1. — Отображение страницы раздела на планшете по умолчанию



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

Исходный код манифеста

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

После сохранения схемы и перезагрузки мобильного приложения страница раздела на планшете будет

отображаться во весь экран (рис. 2).

Рис. 2. — Отображение страницы раздела на планшете во весь экран

Account

EDIT

Name *
Accom (sample)

Primary contact
Andrew Baker (sample)

Type
Customer

Category
B

Industry
Business services

Primary phone
+1 617 440 2498

Web
ac.com

Actual address
31 Union Street, Boston, Massachusetts, 02111, United States

Shipping address
33 Union Street, Boston, Massachusetts, 02111, United States

Company foundation day
8/20/2007