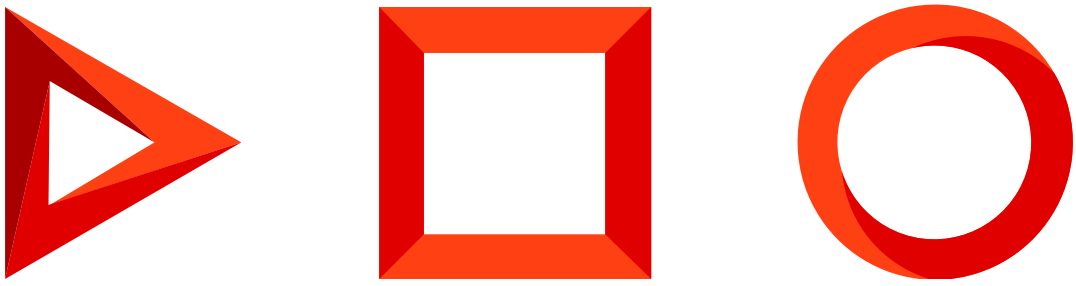
Creatio Academy

# Манифест мобильного приложения

Версия 8.0



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

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

# Содержание

# Манифест мобильного приложения

Сложный

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

## Свойства конфигурационного объекта для настроек синхронизации

`ImportPageSize`

Количество страниц, импортируемых в одном потоке.

`PagesInImportTransaction`

Количество потоков импорта.

`SysSettingsImportConfig`

Массив импортируемых системных настроек.

`SysLookupsImportConfig`

Массив импортируемых системных справочников.

`ModelDataImportConfig`

Массив моделей, для которых будут загружаться данные при синхронизации.

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

## Свойства конфигурационного объекта для настройки синхронизации модели

`Name`

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

---

`SyncColumns`

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

---

`SyncFilter`

Фильтр, накладываемый на модель при импорте модели.

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

# Свойства конфигурационного объекта фильтра модели

---

`type`

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

Возможные значения ( `Terrasoft.FilterTypes` )

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

---

`logicalOperation`

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

Возможные значения ( `Terrasoft.FilterLogicalOperations` )

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

---

`subfilters`

Коллекция фильтров, применяемых к модели. Обязательное свойство для типа фильтра

`Terrasoft.FilterTypes.Group`. Фильтры между собой объединяются логической операцией, указанной в свойстве `logicalOperation`. Каждый фильтр представляет собой конфигурационный объект фильтра.

---

`property`

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

---

`valueIsMacrosType`

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

---

`value`

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

Возможные значения (`Terrasoft.ValueMacros`)

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

---

`macrosParams`

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

---

`isNot`

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

---

`funcType`

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

Возможные значения (`Terrasoft.FilterFunctions`)

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

`funcArgs`

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

`name`

Имя фильтра или группы фильтров. Необязательное свойство.

`modelName`

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

`assocProperty`

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

`operation`

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

Возможные значения (`Terrasoft.FilterOperation`)

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

`compareType`

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

Возможные значения (`Terrasoft.ComparisonType`)

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

## Свойство SyncOptions.ModelDataImportConfig.QueryFilter

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

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

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

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

**Пример exists-фильтра**

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

```
                        "logicalOperation": 0,
                        "filterType": 6,
                        "rootSchemaName": "ActivityParticipant",
                        "items": {
                           "ParticipantFilter":{
                              "filterType": 1,
                              "comparisonType": 3,
                              "leftExpression": {
                                 "expressionType": 0,
                                 "columnPath": "Participant"
                              },
                              "rightExpression": {
                                 "expressionType": 1,
                                 "functionType": 1,
                                 "macrosType": 2
                              }
                           }
                        }
                     }
                  }
               }
            }
         }
      ]
   }
}
```

# Настроить меню мобильного приложения

Сложный

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

## Реализация примера

**Свойство** `ModuleGroups`

```
// Групы модулей мобильного приложения.
"ModuleGroups": {
    // Настройка группы основного меню.
    "main": {
```

```
        // Позиция группы в главном меню.
        "Position": 0
    },
    // Настройка группы меню [Продажи].
    "sales": {
        // Позиция группы в главном меню.
        "Position": 1
    }
}
```

# Настроить стартовую страницу и разделы меню мобильного приложения

Сложный

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

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

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

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

## Реализация примера

**Свойство** `Modules`

```
// Модули мобильного приложения.
"Modules": {
    // Раздел "Контакт".
    "Contact": {
        // Группа меню приложения, в которой размещается раздел.
        "Group": "main",
        // Название модели, которая предоставляет данные раздела.
        "Model": "Contact",
        // Позиция раздела в группе меню.
        "Position": 0,
        // Заголовок раздела.
        "Title": "ContactSectionTitle",
```

```
        // Подключение пользовательского изображения к разделу.
        "Icon": {
            // Уникальный идентификатор изображения.
            "ImageId": "4c1944db-e686-4a45-8262-df0c7d080658"
        },
        // Подключение пользовательского изображения к разделу.
        "IconV2": {
            // Уникальный идентификатор изображения.
            "ImageId": "9672301c-e937-4f01-9b0a-0d17e7a2855c"
        },
        // Признак отображения в меню.
        "Hidden": false
    },
    // Раздел "Контрагент".
    "Account": {
        // Группа меню приложения, в которой размещается раздел.
        "Group": "main",
        // Название модели, которая предоставляет данные раздела.
        "Model": "Account",
        // Позиция раздела в группе меню.
        "Position": 1,
        // Заголовок раздела.
        "Title": "AccountSectionTitle",
        // Подключение пользовательского изображения к разделу.
        "Icon": {
            // Уникальный идентификатор изображения.
            "ImageId": "c046aa1a-d618-4a65-a226-d53968d9cb3d"
        },
        // Подключение пользовательского изображения к разделу.
        "IconV2": {
            // Уникальный идентификатор изображения.
            "ImageId": "876320ef-c6ac-44ff-9415-953de17225e0"
        },
        // Признак отображения в меню.
        "Hidden": false
    }
}
```

# Настроить конфигурацию моделей

 Сложный

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

1. `Контакт` — указать названия схем страниц реестра, просмотра и редактирования, требуемые модели, модули расширения модели и страниц модели.

2. `Адрес контакта` — указать только модуль расширения модели.

## Реализация примера

**Свойство** `Models`

```
// Импортируемые модели.
"Models": {
    // Модель "Контакт"
    "Contact": {
        // Схема страницы реестра.
        "Grid": "MobileContactGridPage",
        // Схема страницы просмотра.
        "Preview": "MobileContactPreviewPage",
        // Схема страницы записи.
        "Edit": "MobileContactEditPage",
        // Названия моделей, от которых зависит модель "Контакт".
        "RequiredModels": [
            "Account", "Contact", "ContactCommunication", "CommunicationType", "Department",
            "ContactAddress", "AddressType", "Country", "Region", "City", "ContactAnniversary",
            "AnniversaryType", "Activity", "SysImage", "FileType", "ActivityPriority",
            "ActivityType", "ActivityCategory", "ActivityStatus"
        ],
        // Расширения модели.
        "ModelExtensions": [
            "MobileContactModelConfig"
        ],
        // Расширения страниц модели.
        "PagesExtensions": [
            "MobileContactRecordPageSettingsDefaultWorkplace",
            "MobileContactGridPageSettingsDefaultWorkplace",
            "MobileContactActionsSettingsDefaultWorkplace",
            "MobileContactModuleConfig"
        ]
    },
    // Модель "Адреса контактов".
    "ContactAddress": {
        // Страницы реестра, просмотра и редактирования сгенерированы автоматически.
        // Расширения модели.
        "ModelExtensions": [
            "MobileContactAddressModelConfig"
        ]
    }
}
```

# Использовать функцию поиска подстроки для поиска данных

Сложный

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

## Реализация примера

**Свойство** `PreferedFilterFuncType`

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

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

# Загрузить данные моделей при синхронизации

Сложный

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

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

## Реализация примера

**Свойство** `SyncOptions`

```
// Настройки синхронизации.
"SyncOptions": {
```

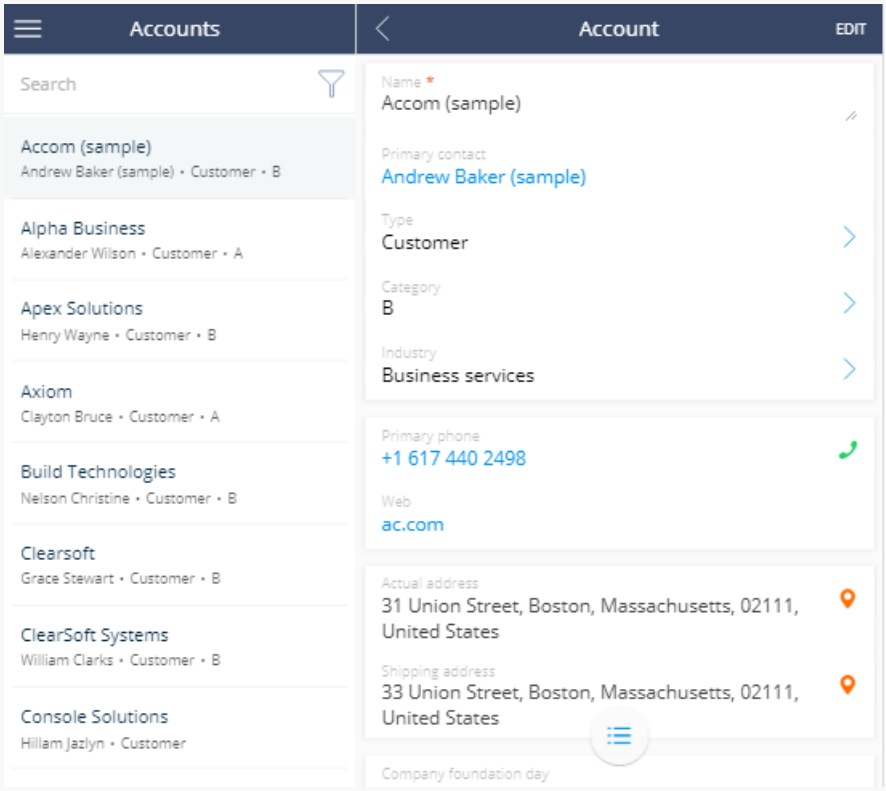
```
    // Количество страниц, импортируемых в одном потоке.
    "ImportPageSize": 100,
    // Количество потоков импорта.
    "PagesInImportTransaction": 5,
    // Массив импортируемых системных настроек.
    "SysSettingsImportConfig": [
        "SchedulerDisplayTimingStart", "PrimaryCulture", "PrimaryCurrency",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

# Отобразить страницу на планшете во

# весь экран

Легкий

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




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

## Реализация примера

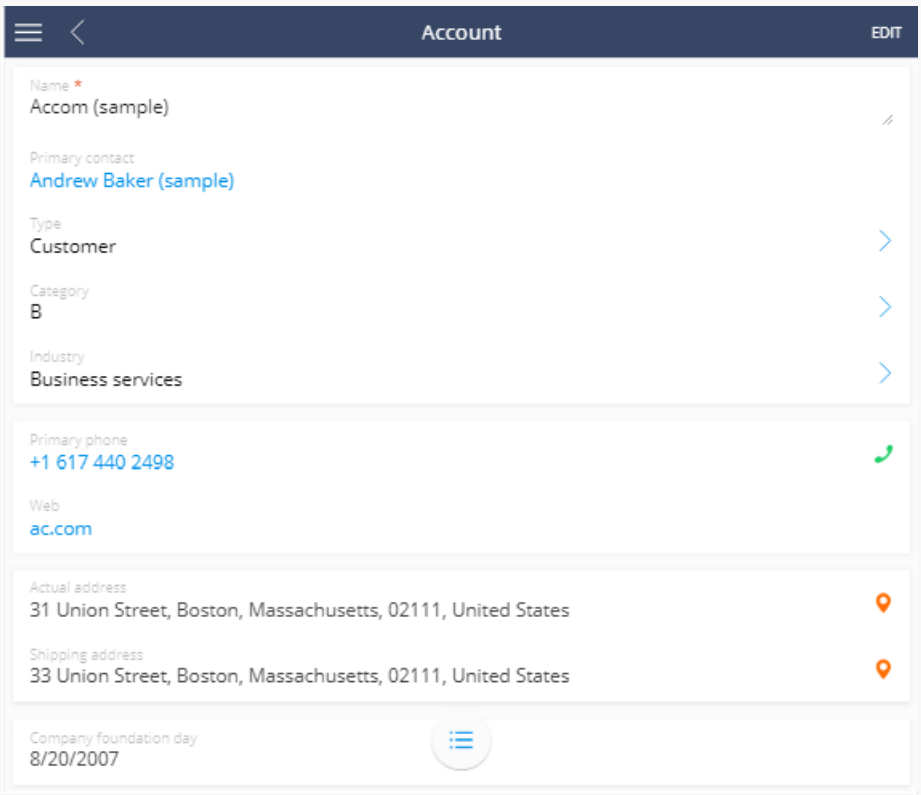
Для отображения страницы раздела во весь экран необходимо внести изменения в манифест мобильного приложения, добавив в него свойство `TabletViewMode` со значением "`SinglePage`".

**Свойство** `TabletViewMode`

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

```
    "Modules": {},
    "Models": {}
}
```

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




# Добавить стандартную деталь с колонками

Сложный

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

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

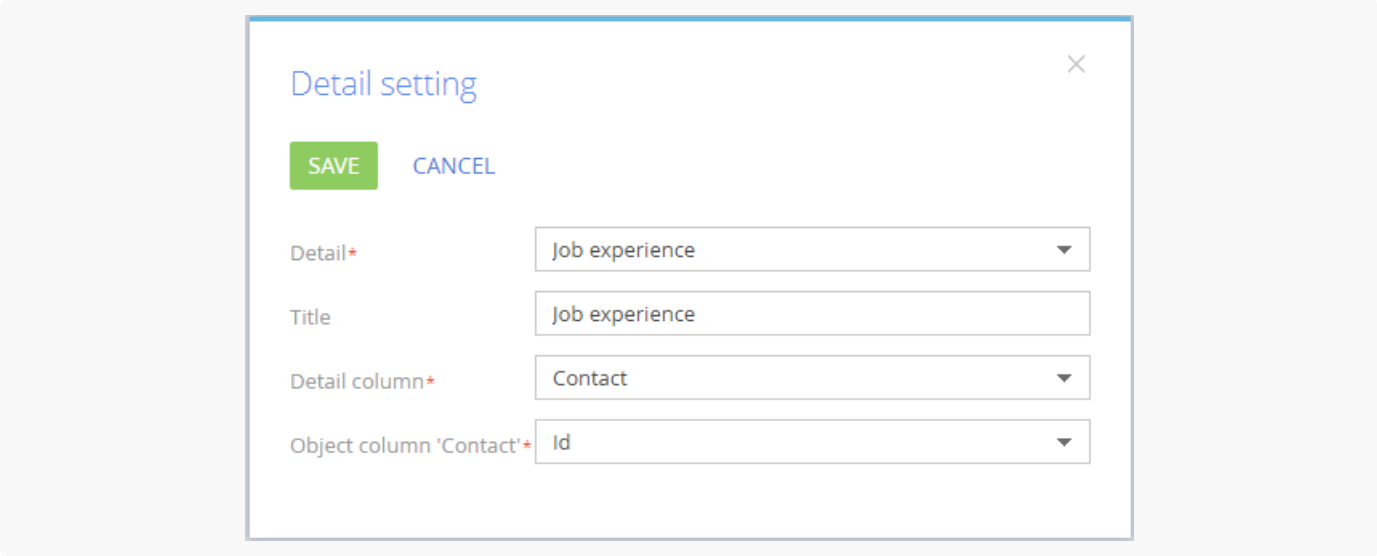
**Пример.** На страницу записи раздела [ *Контакты* ] мобильного приложения добавить деталь [ *Карьера* ]. В качестве основной отображать колонку [ *Должность* ].

# Алгоритм реализации примера

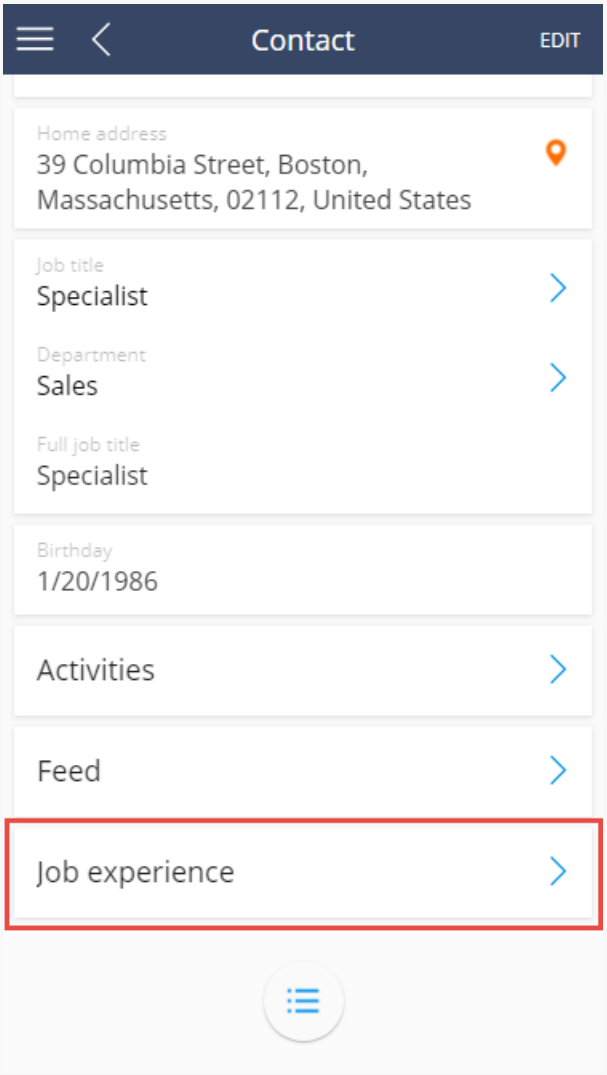
## 1. Добавить деталь [ *Карьера* ] с помощью мастера мобильного приложения

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

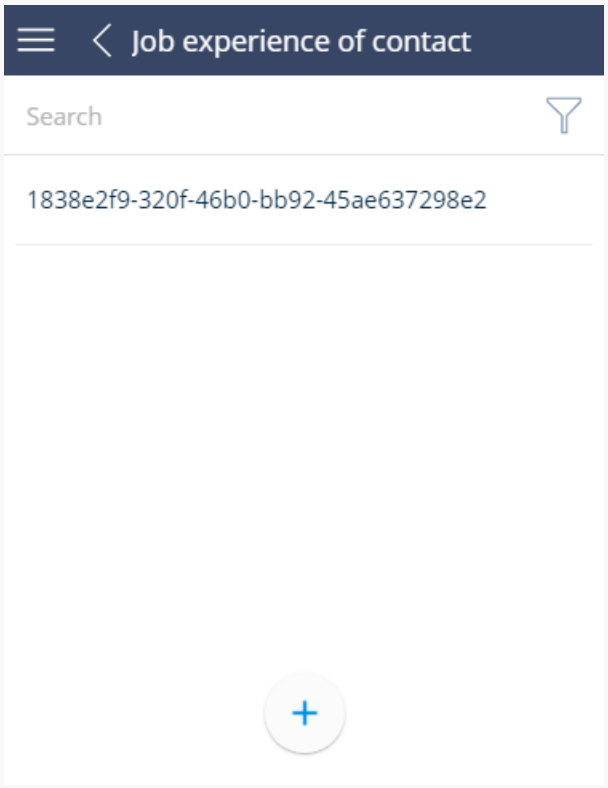
1. Откройте нужное рабочее место, например, [ *Основное рабочее место* ] ([ *Main workplace* ]) и нажмите кнопку [ *Настроить разделы* ] ([ *Set up sections* ]).
2. Выберите раздел [ *Контакты* ] и нажмите кнопку [ *Настроить детали* ] ([ *Details setup* ]).
3. Настройте деталь [ *Карьера* ] ([ *Job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




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



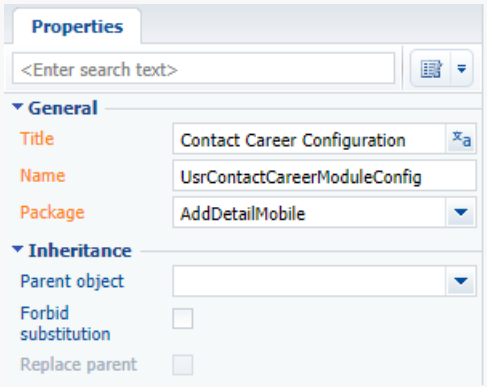

Однако, поскольку объект детали [ *Карьера* ] не является объектом раздела мобильного приложения Creatio, то на странице детали отображается значение основной колонки [ *Контакт* ] (идентификатор связанной записи контакта).




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

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

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




Добавьте в схему модуля исходный код.

```
UsrContactCareerModuleConfig
```

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

Здесь:

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

## 3. Подключить схему модуля в манифесте мобильного приложения

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

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

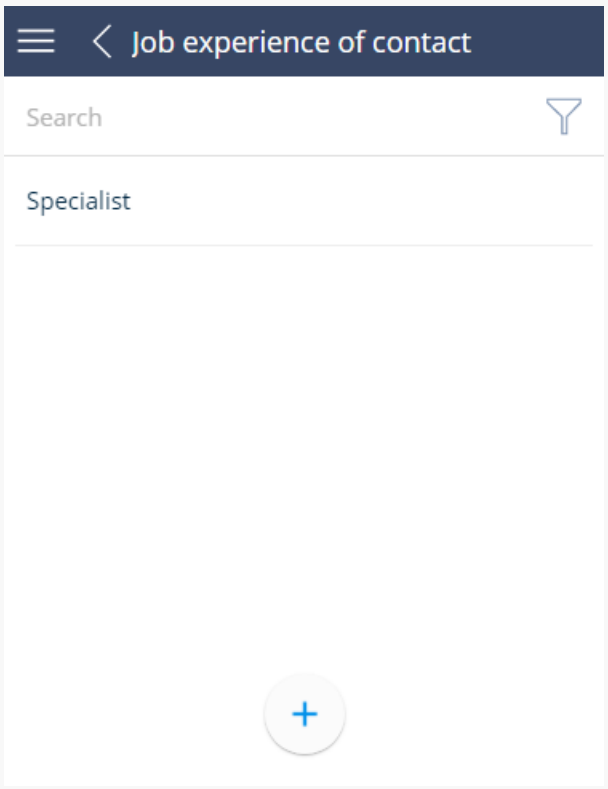
**ContactCareer**

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

```
        }
    }
```

3. Сохраните схему манифеста мобильного приложения.

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




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

# Добавить пользовательский дашборд в мобильное приложение

Сложный

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

Для добавления пользовательского типа дашборда в мобильное приложение необходимо:

1. Реализовать интерфейс настройки дашборда в приложении Creatio.

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

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

Чтобы пользовательский тип дашборда отображался в мобильном приложении необходимо:

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

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

# Алгоритм реализации примера

## 1. Реализация получения данных пользовательского типа дашборда

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

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

**CustomDashboardItem**

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

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

        }
```

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

## 2. Реализация отображения пользовательского типа дашборда

### 2.1. Добавить класс отображения данных

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

**UsrMobileCustomDashboardItem**

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

});
```

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

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

**CustomDashboardItem**

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

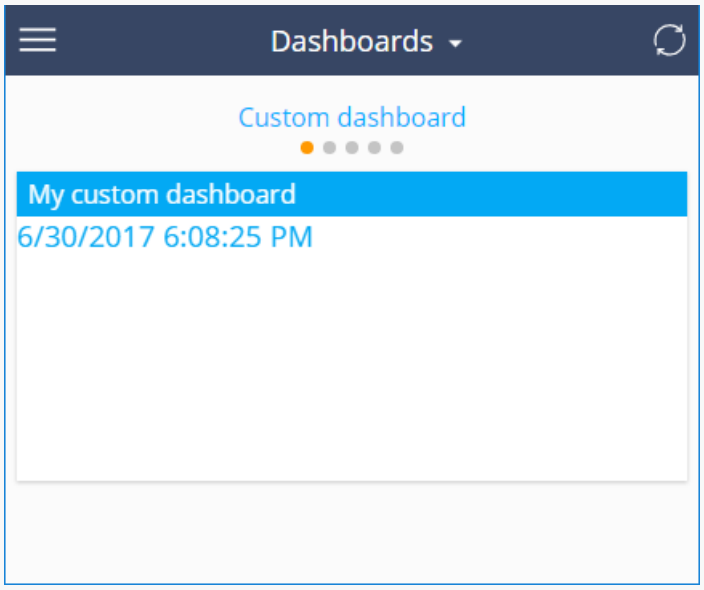
### 2.3. Добавить название созданной клиентской схемы в манифест мобильного приложения

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

`CustomSchemas`

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

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




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

# Добавить кнопку для отображения имени контакта

Сложный

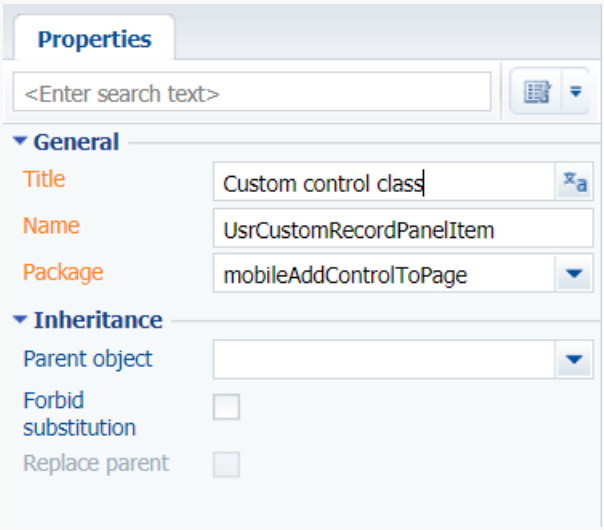
**Пример.** На страницу записи раздела [ *Контакты* ] мобильного приложения добавить кнопку, нажатие на которую будет показывать сообщение с полным именем контакта.

# Алгоритм реализации примера

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

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

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




В модуль добавьте исходный код.

**CustomRecordPanelItem**

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

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

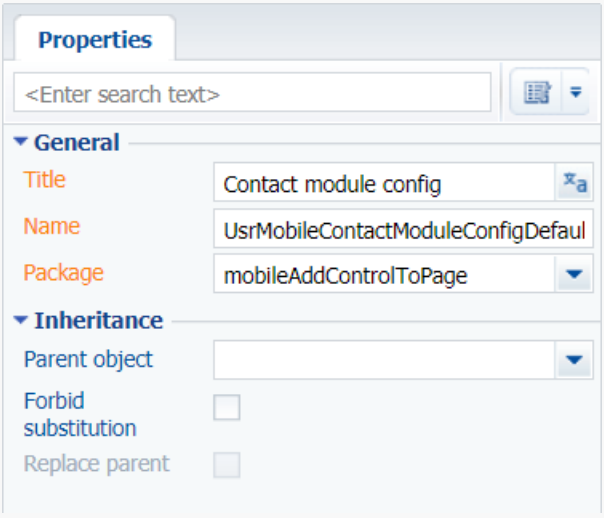
В классе описан конфигурационный объект созданного элемента управления и два метода:

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

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

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

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




Добавьте в схему модуля исходный код.

**UsrMobileContactModuleConfigDefaultWorkplace**

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

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

## 3. Подключить схемы модулей в манифесте мобильного приложения

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

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

   **Добавление модулей**

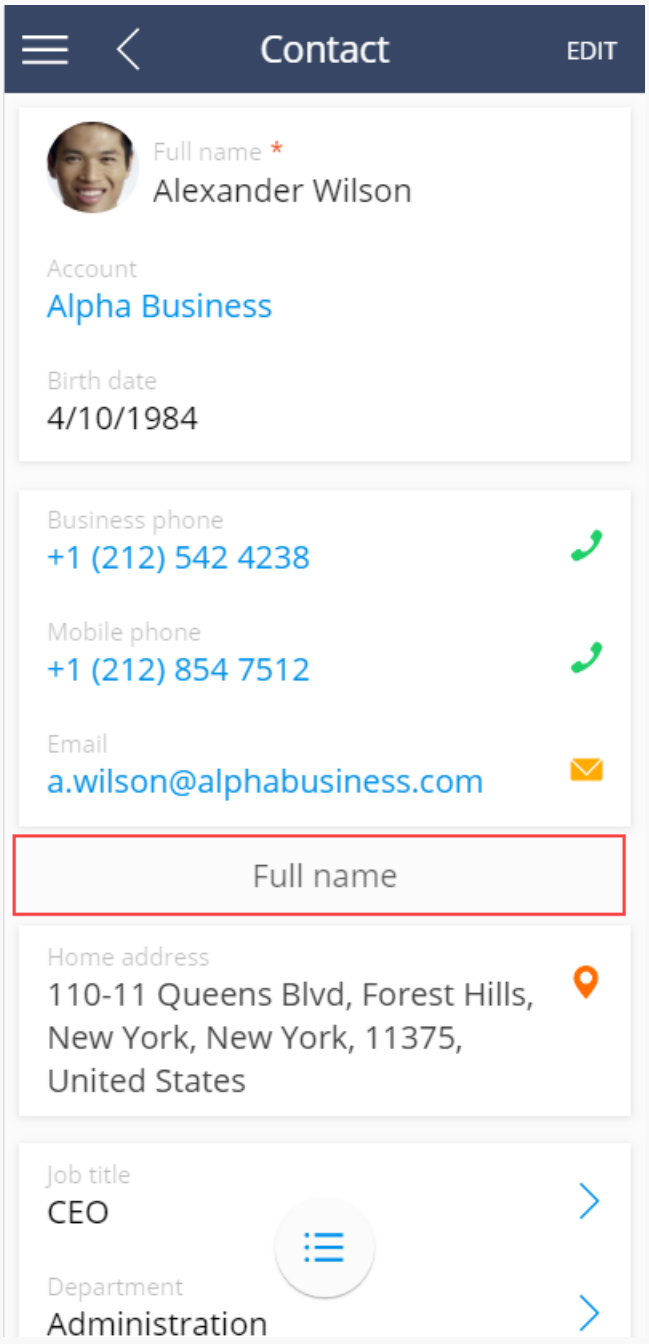
   ```
   {
       "CustomSchemas": [
           "UsrCustomRecordPanelItem.js"
       ],
       "SyncOptions": {},
       "Modules": {},
       "Models": {
           "Contact": {
               "RequiredModels": [],
               "ModelExtensions": [],
               "PagesExtensions": [
            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

3. Сохраните схему манифеста мобильного приложения.

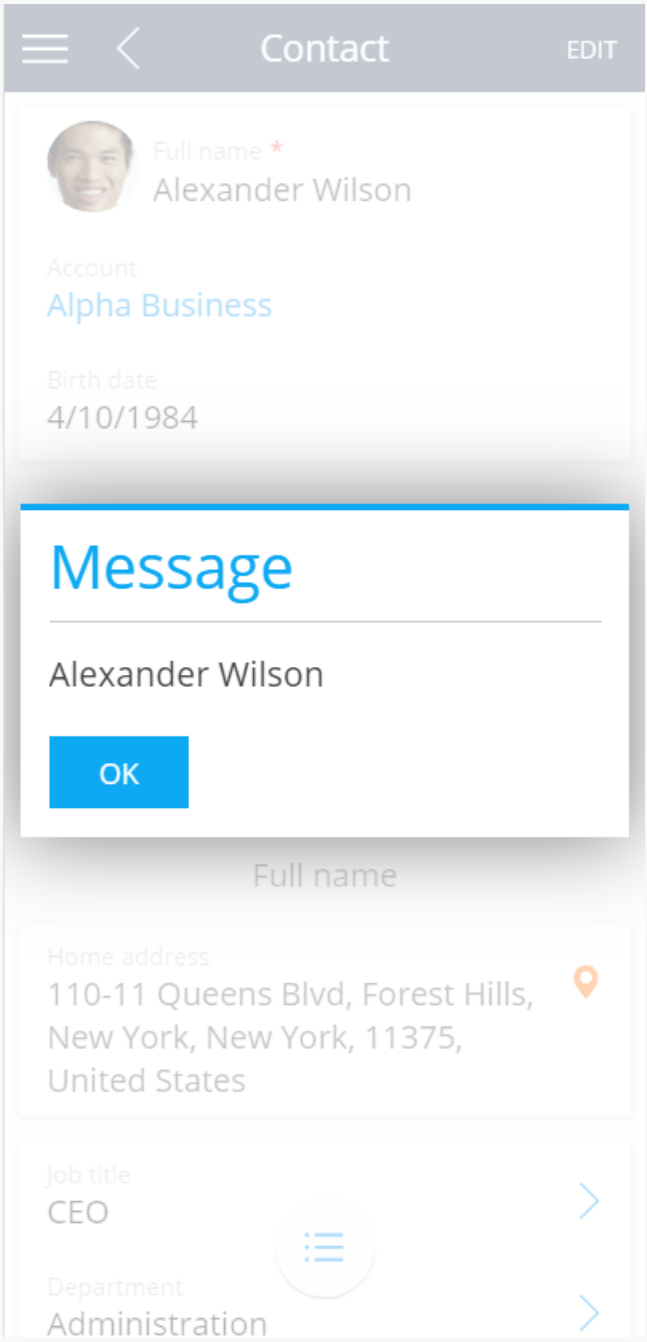
В результате выполнения примера на странице контакта появится элемент управления, при нажатии на

который отобразится сообщение с полным именем контакта.

Результат выполнения кейса. Добавление кнопки




Результат выполнения кейса. Нажатие на кнопку




# Свойство ModuleGroups JS

Сложный

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

## Свойство конфигурационного объекта

`Position`

Позиция группы в главном меню. Начинается с 0.

# Свойство Modules JS

Сложный

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

## Свойства конфигурационного объекта

---

`Group`

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

---

`Model`

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

---

`Position`

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

---

`Title`

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

---

`Icon`

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

---

`IconV2`

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

---

`Hidden`

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

# Свойство Icons JS

Сложный

Свойство предназначено для подключения к мобильному приложению пользовательских изображений.

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

## Свойства конфигурационного объекта

---

`ImageListId`

Идентификатор списка изображений.

---

`ImageId`

Идентификатор подключаемого изображения из списка `ImageListId`.

**Подключение пользовательских изображений**

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

# Свойства DefaultModuleImageId и DefaultModuleImageIdV2 JS

Сложный

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

**Установка уникальных идентификаторов изображений**

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

# Свойство Models JS

Сложный

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

## Свойства конфигурационного объекта

---

`Grid`

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

---

`Preview`

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

---

`Edit`

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

---

`RequiredModels`

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

---

`ModelExtensions`

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

`PagesExtensions`

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

# Свойство PreferedFilterFuncType JS

Сложный

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

## Функции фильтрации (Terrasoft.FilterFunctions)

`SubStringOf`

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

`ToUpper`

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

`EndsWith`

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

`StartsWith`

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

`Year`

Возвращает год по значению колонки `property`.

`Month`

Возвращает месяц по значению колонки `property`.

`Day`

Возвращает день по значению колонки `property`.

---

`In`

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

---

`NotIn`

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

---

`Like`

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

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

# Свойство CustomSchemas JS

Сложный

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

Значение свойства задается массивом с именами подключаемых пользовательских схем.

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

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

# Свойство SyncOptions JS

Сложный

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

## Свойства конфигурационного объекта для настроек синхронизации

`ImportPageSize`

Количество страниц, импортируемых в одном потоке.

`PagesInImportTransaction`

Количество потоков импорта.

`SysSettingsImportConfig`

Массив импортируемых системных настроек.

`SysLookupsImportConfig`

Массив импортируемых системных справочников.

`ModelDataImportConfig`

Массив моделей, для которых будут загружаться данные при синхронизации.

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

## Свойства конфигурационного объекта для настройки синхронизации модели

`Name`

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

`SyncColumns`

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

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

---

`SyncFilter`

Фильтр, накладываемый на модель при импорте модели.

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

## Свойства конфигурационного объекта фильтра модели

---

`type`

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

Возможные значения ( `Terrasoft.FilterTypes` )

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

---

`logicalOperation`

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

Возможные значения ( `Terrasoft.FilterLogicalOperations` )

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

---

`subfilters`

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

---

`property`

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

---

`valueIsMacrosType`

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

---

`value`

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

Возможные значения (`Terrasoft.ValueMacros`)

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

---

`macrosParams`

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

---

`isNot`

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

---

`funcType`

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

Возможные значения (`Terrasoft.FilterFunctions`)

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

---

`funcArgs`

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

---

`name`

Имя фильтра или группы фильтров. Необязательное свойство.

---

`modelName`

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

---

`assocProperty`

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

---

`operation`

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

Возможные значения (`Terrasoft.FilterOperation`)

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

---

`compareType`

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

Возможные значения (`Terrasoft.ComparisonType`)

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

# Свойство SyncOptions.ModelDataImportConfig.QueryFilter

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

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

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

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

**Пример exists-фильтра**

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

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