ООО «АйТи-Альянс»
http://www.it-alnc.ru/
ИНН 7718755096 / КПП 772501001
ОГРН 1097746108745

Описание функциональных характеристик модуля ITA DESIGNER программного обеспечения ITA::FORMS («Дизайнер форм и бизнес-процессов»)

Москва, 2025

IT-ALLIANCE

Оглавление

1. Аннотация

Документ содержит описание функциональных характеристик модуля ITA Designer («Дизайнер процессов и форм») интеграционной платформы ITA::FORMS.

Модуль ITA Designer предназначен для быстрой настройки бизнес – процессов и экранных форм пользовательских интерфейсов.

2. Термины и определения

Программное обеспечение / ПО	Программа или множество программ, используемых для управления компьютером
Модуль	Составная часть программного обеспечения, обладающая самостоятельным функционалом
Приложение	Программное обеспечение «Дизайнер форм и бизнес-процессов»
web-приложение, веб-приложение	Прикладное программное обеспечение, которое работает на веб-сервере, в отличие от компьютерных программ, которые запускаются локально в операционной системе (ОС) устройства
БД, база данных	Совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных
Drag-and-drop	Способ оперирования элементами интерфейса в путем «захвата» элемента мышкой и перемещения в другое место для изменения расположения.
Control, контрол, поле	Интерфейсная компонента в разметке экранной формы
Handler, хендлер	Некоторая типовая процедура, которая используется в форме для получения данных, выполнения определенных действий, запуска других процедур.
Форма	Совокупность полей на экране.
Виджет	Вариант формы, у которой дополнительно определен собственный источник данных.
Бизнес-процесс	Набор действий, выполняемых внутри организации или между организациями. В спецификации BPMN бизнес-процесс изображается в качестве диаграммы, представляющей собой совокупность действий и шлюзов, влияющих на порядок выполнения этих действий.

3. Основные функции модуля ITA Designer

Модуль ITA Designer поддерживает следующие функции:

- Настройка процессов в нотации BPMN в удобном графическом редакторе;
- Удобное конструирование экранных форм и пользовательских задач для процессов в drag-and-drop редакторе;
- Быстрая и эффективная интеграция процессов с ИТ системами организации;
- Предоставление метаданных пользовательских форм для интерфейса решений на платформе ITA::FORMS или других приложений пользователя;
- Предоставление метаданных настроек выполнения сервисных задач в бизнес-процессах на платформе ITA::FORMS.

Меню модуля ITA Designer содержит следующие разделы:

- **Старт процессов** – позволяет вручную запустить доступные бизнес – процессы;
- **Редактор процессов** – позволяет создавать и редактировать схемы бизнес – процессов;
- **Редактор форм** – позволяет создавать и редактировать экранные формы разной сложности;
- **Множественная загрузка форм** – позволяет загружать в систему / выгружать из системы шаблоны форм в виде файлов .txt или .json; переносить формы и бизнес-процессы между стендами в среде заказчика.
- **Описание объектов DataVault** – позволяет просматривать структуру бизнес-объектов в формате DataVault (хабов, сателлитов и связей).
- **Редактор бизнес**-объектов – позволяет создавать модель данных

3.1.Редактор форм

Данный функционал позволяет создавать экранные формы, которые в дальнейшем можно вызывать как в бизнес-процессах – в пользовательских задачах User tasks, так и вне процессов. Drag-and-drop интерфейс редактора позволяет перетаскивать элементы дизайна на веб-странице и изменять их положение или порядок.

Базовые функции:

- **Поиск форм** – позволяет найти в списке доступных форм нужную форму, открыть найденную форму для последующего редактирования.
- **Сохранение форм** – сохраняет внесенные изменения; допустимо как сохранение основной формы, так и сохранение всей цепочки вложенных форм.
- **Действия с формой** – позволяет загрузить форму из файла для последующего редактирования, сохранить форму в файл (для последующей загрузки), скопировать форму, привязать форму к пользовательской задаче бизнес-процесса.
- **Редактор формы** – drag-and-drop интерфейс редактора, в котором выполняется настройка формы.
- **Предварительный просмотр** – позволяет оценить, как будет выглядеть форма по мере добавления на нее элементов дизайна.
- **Паттерн формы** – позволяет описать структуру данных, которые будут отображаться на форме.
- **Хендлеры формы** – позволяет настроить вызов определенных процедур в форме, которые будут выполняться при наступлении определенных событий – триггеров.
- **Параметры формы** – позволяет задать некоторые глобальные параметры формы.
- **Отчеты (репорты)** – позволяет добавить на форму печатную форму формата .doc, .pdf ; в качестве шаблонов печатных форм допустимо использовать шаблоны birt.
- **Объекты** – позволяет создавать на форме свое собственное меню (справа) для перехода в другие формы.

3.1.1. Функция поиска формы

Данная функциональность позволяет найти нужную форму для последующего редактирования. Поиск выполняется по уникальному коду - имени формы, либо названию - лейблу формы. Функция доступна в редакторе посредством поисковой строки.

3.1.2. Функция открытия формы

Данная функциональность позволяет открыть на редактирование найденную в БД форму. Функция доступна в редакторе посредством кнопки «Открыть», становится активной, если заданная в поисковой строке форма найдена.

3.1.3. Функция сохранения формы

Данная функциональность позволяет сохранять в БД изменения, выполненные на форме. Данная функциональность доступна в редакторе в виде двух кнопок:

- Сохранить основной шаблон
- Сохранить

Кнопка «Сохранить основной шаблон» выполнит сохранение основной формы, при этом не будет сохранять изменения в шаблонах таблиц, виджетов и других контролов, использующих внутри себя формы-шаблоны.

Кнопка «Сохранить» сохранит всю цепочку вложенных форм.

3.1.4. Действия с формой

В рамках данной функциональности поддерживаются следующие действия с формой:

- **Загрузить из файла** – позволяет загрузить в форму сохраненный ранее в виде файла .txt / .json шаблон
- **Сохранить в файл** – позволяет сохранить открытую текущую форму в виде файла txt / .json
- **Скопировать форму** – позволяет скопировать форму со всеми вложенными формами (кроме форм, открываемых хендлерами). Получившаяся копия не сохранена в БД, поэтому её нужно сохранить чтобы не потерять изменения.
- **Привязать к процессу** – позволяет связать форму с пользовательской задачей в некотором бизнес – процессе.

Данная функциональность представлена в редакторе в виде выпадающего меню.

3.1.5. Функционал редактирования формы

Данная функциональность позволяет создавать новые формы и редактировать существующие.

Функционал доступен в закладке «Редактор» и представляет собой drag-and-drop интерфейс.

Обязательные атрибуты для создания формы:

- **Name** – уникальный код формы
- **Label** – Наименование формы

После того как форма создана, ее нужно наполнить элементами дизайна, или контролами. Наполнение формы контролами осуществляется путем перетаскивания мышкой нужного элемента из перечня доступных элементов. Для вновь добавленного элемента необходимо задать атрибуты.

Обязательными атрибутами являются:

- **Attribute name** – уникальный идентификатор поля формы; указывается путь к данным внутри HLS-данных в формате *hub.{link[x].hub.}sat.attr*. Путь должен однозначно соответствовать единственному значению в наборе данных, полученному из источника данных. Пример доступа к данным в простом поле input:

  hub_MPShowcase.link_MPVersion_MPShowcase[0].hub_MPVersion.sat_MPVersion_Main.Description

- **Ширина (в связке с Настройка ширины)** – ширина отображения поля на странице;
- **Control type** – тип контрола.

Другие атрибуты могут стать обязательными в сочетании с какими-то определенными полями. (Например, при создании таблицы необходимо заполнять и поле Template name).

- **Label** – отображаемое название поля формы (то, что будет отображаться на экране в качестве подписи к полю);
- **Переменная** – переменная Camunda, связанная с полем формы, которая передается в бизнес-процесс;
- **Placeholder** – текстовая подсказка-заполнитель, для полей вида input, text-area;
- **Test formulae** – для контрола text позволяет задать формулу преобразования значения (можно указать значение другого поля в соответствии с правилами построения выражений);
- **Color** – для кнопок action, позволяет выбрать цвет и рамку кнопок из списка;
- **Hint/icon** - позволяет назначать иконки некоторым полям, например, кнопкам; требуется указать код иконки; поддерживаются как коды из Angular Material Icons, так и кастомные из собственной библиотеки;
- **Items** – опции, позволяет задавать константные значения для контролов, предполагающих выбор из нескольких значений;

- **Disabled** – позволяет сделать поле неактивным / нередактируемым;
- **key field** – признак ключевого поля;
- **Template name** – указывается имя шаблона, который вызывается в экранной форме (для таблиц, массивов, аккордеонов, некоторых кастомных контролов);
- **Title** – здесь задается формула для заголовка элемента аккордеона;
- **Description** – здесь задается формула для описания элемента аккордеона;

Также у каждого контрола есть свои дополнительные параметры, расширяющие его функциональность. Дополнительные параметры определяются типом контрола.

Для контролов доступен функционал удаления и копирования. Функционал доступен с помощью соответствующих кнопок.

При копировании контрола у этой копии назначается новый attribute_name, поскольку не может быть двух контролов на форме с одинаковыми attribute_name, это уникальное значение.

3.1.5.1. Типы контролов

Все контролы в редакторе форм условно сгруппированы в несколько разделов:

- Поля ввода
- Контейнеры
- Таблицы
- Ссылки и файлы
- Сложные контролы
- Другое

Поля ввода

В данном меню редактора форм собраны простые поля, предназначенные как для просмотра, так и для ввода / редактирования данных

- **Input** - простое текстовое поле для ввода и отображения данных; текст отображается одной строкой
- **Input-number** - поле для ввода числовых значений, значение форматируется с разделителями разрядов и дробной части
- **Text-area** - текстовое поле для ввода и отображения данных; позволяет ввести длинный текст, который отображается в несколько строк

- **Check-box** - чек-бокс. Используется для различных признаков, подразумевающих значение «Да» или «Нет»
- **Slide-toggle** - переключатель, по смыслу это чек-бокс
- **Radio-button** - для различных признаков, вариантов. Варианты можно указать с помощью опций в поле Items, тогда если значение поля соответствует одному из вариантов в Items, то автоматически будет устанавливаться в этом варианте.
- **Solid-radio-button** - как Radio-button, но внешний вид - панель
- **Slider** - позволяет выбирать значение из диапазона с определенным шагом, перемещая ползунок
- **Select** - выбор значения из списка (список выпадающий). Список можно задать:
    - С помощью опций в атрибуте «Items»,
    - С помощью хендлера - получить данные из каких-либо справочников ITA Info,
    - С помощью парамеров listPath, listKey, listValue – выбрать массив из контекста данных или из переменной бизнес-процесса.
- **Date-picker** - для отображения и задания даты без времени
- **Datetime-picker** - для отображения и задания даты со временем
- **Date-picker-range** - для задания диапазона дат

Контейнеры

В данном меню редактора форм собраны контролы, подразумевающие группировку полей на форме.

- **Accordion** - аккордеон. Позволяет отображать вложенные массивы.
- **Array** – массив. Позволяет отображать на форме множественные значения данных.
- **Container** - элемент дизайна; предназначен для красивой группировки полей на форме.
- **Expantion-panel** - элемент дизайна; также предназначен для красивой группировки полей на форме, при этом группу можно свернуть или развернуть.
- **Modal** - контейнер для полей, которые будут отображены в модальном окне. Используется в связке с хендлером "Вывод модального окна" - чтобы открыть модальное окно нужно создать этот хендлер, указать соответствующий тип модального окна и указать нужное модальное окно. Далее этот хендлер можно выполнить в любом удобном месте, например - привязать к кнопке.

- **Tab** - элемент tab-панели – одна из закладок горизонтального меню
- **Tablist** - Tab-панель – это горизонтальное меню закладок

Таблицы

В данном меню редактора форм собраны различные виды таблиц, для которых источником данных является паттерн формы.

- **Simple-table** - простая таблица с данными.
- **Multiple-table-select** - для множественного выбора данных, которые отображаются в виде таблицы (таблица с чек-боксом).
- **Object-table** – сложная таблица, в которой можно проваливаться в содержимое строки. Детальное содержимое строки открывается поверх таблицы с возможностью возврата назад.
- **Address-table** – специальная таблица для отображения адресов клиента

Ссылки и файлы

- **Link** - позволяет отображать значение в виде ссылки с открытием другой формы в отдельном экране (модальное окно или перекрытие текущего экрана) по клику.
- **File** - контрол, позволяющий пользователю выбрать файл. Доступ к выбранному файлу осуществляется с помощью модификатора get-file: {{ my-file | get-file }}

Сложные контролы

В данном меню редактора форм собраны кастомные контролы со специально разработанным сложным функционалом.

- **Ita-object-list** - контрол, предназначенный для реализации реестровых таблиц. Отбирает объекты по заданным критериям. Фильтрацию для отбора можно либо настроить в интерфейсе, либо в редакторе с помощью параметра "ita-filter". Значениями фильтров могут быть выражения. Если не заданы параметры объекта (object-template, object-boclass), то открытие формы объекта будет невозможно. В шаблоне строки таблицы отбора должно присутствовать поле с параметром role: id - значение этого поля будет использовано в качестве идентификатора объекта при двойном клике по строке. Для того чтобы по двойному клику по строке форму объекта можно было открыть, нужно добавить соответствующие параметры - object-template и object-boclass.

- **Ita-object-history-list** - таблица истории изменений. Самостоятельный реестр, которому для получения данных необходимо указать имя сателлита и идентификатор объекта. Под объектом подразумевается родительский для нужного хаб. В этом реестре отображаются состояния объекта после каждого изменения, изменённые поля подсвечены цветом #B5E61D. Таблица истории изменений не имеет отдельного шаблона - шаблон строится прямо на лету по данным от ITAObjects. По этой причине изменение шаблона в редакторе отключено.
- **Ita-object-search** - позволяет пользователю отобрать объект по заданным критериям. Фильтрацию для отбора можно либо настроить в интерфейсе, либо в редакторе с помощью параметра "ita-filter". Значениями фильтров могут быть выражения. Если не заданы параметры объекта (object-template, object-boclass), то открытие формы объекта будет невозможно В шаблоне строки таблицы отбора должны присутствовать поле с параметром role: id и поле с параметром role: label - значения этих полей будут использованы в качестве идентификатора и имени объекта при выборе объекта соответственно
- **Ita-object-tree** - представляет собой иерархический список элементов. Обязательным параметром является object-tree-levels, который представляет собой набор параметров для каждого уровня дерева. На данный момент существует 2 вида узлов: Статические узлы - узлы, которые не обращаются к операциям и, соответственно, не получают данные извне. Имеют простую структуру key + value; Динамические узлы - узлы, которые при клике обращаются к операции с id выбранного узла. Каждый узел имеет полный набор данных, пришедших от операции.
- **Widget** - блок с динамическим контентом для отображения данных объекта: по заданным типу (object-boclass) и идентификатору хаба (object-bocode) загружаются данные объекта и привязываются к полям шаблона (object-template). При изменении связанных объектов виджет самостоятельно обновляет данные шаблона в соответствии с выбранной политикой обновления.
- **Ita-mass-upload** - кастомный контрол, разрабатывался для реализации функционала Массовых операций.

Другое

- **Action** - кнопки, предназначено для запуска операций или выполнения специальных действий. Является триггером в хендлерах.

- **Dual-select** - для множественного выбора данных; позволяет перемещать данные из одного окошка в другое.
- **Chips-select** - для множественного выбора данных; позволяет выбирать множество данных, которые будут отображаться отдельными «чипами».
- **Divider** - задает линию-разделитель, может быть вертикальным и горизонтальным.
- **Hidden** - скрытое поле, может использоваться как в разметке, так и для задания скрытых значений для вычислений.
- **Image** - позволяет добавлять картинку в форму.
- **Input-address** - поле для задания адреса строкой (в некоторых проектах взаимодействие со справочником ФИАС),
- **Multiple-select** - для множественного выбора данных; данные выбираются из выпадающего списка и отображаются строкой через запятую.
- **Stepper** - элемент дизайна, позволяет задавать текущий шаг процесса; может быть горизонтальным и вертикальным.
- **Text** - простой текст; можно задать в поле «Text formulae» как константу, так и более сложное выражение с помощью модификаторов. По умолчанию отображается значение атрибута attribute_name.
- **Dynamic-field** - для задания динамической таблицы или динамических полей. Сложный кастомный контрол, реализован для отображения таблиц тарифов.
- **Json-table** - сложный кастомный контрол, реализован для функционала настройки уведомлений.

3.1.5.2. Поддержка выражений и модификаторов

Функционал выражений и модификаторов предназначен для более гибкого управления отображением данных на форме.

В выражении можно указать значение другого поля в соответствии с правилами построения выражений: {{some_control}} , где some_control – это attribute_name другого контрола на форме.

В выражениях поддерживаются специальные ключи, которые дают доступ к данным из других источников, например:

- SELECTED_OBJECT - идентификатор объекта, выбранного в таблице/списке

- user.* - данные о текущем пользователе. Доступны такие поля: id, name, firstName, middleName, surName, login, office, office_id, position, phone, roles, token, email
- ROOT_ID – id корневого хаба

В выражениях разрешено использовать различные модификаторы. Модификаторы преобразуют вывод значения поля. Модификаторы указываются через символ "|" после имени поля, например: {{some_control | number}}. При значении some_control = '1234567' вывод будет такой: '1 234 567.00'.

3.1.5.2.1. Виды модификаторов

Условия:

is-not-empty - проверяет является ли значение не пустым - если да, то вернётся 1, иначе - 0

Если field1 не существует, то *{{ field1 | is-not-empty }}* вернёт 0; если field2 === null, то *{{ field2 | is-not-empty }}* вернёт 0; если field3 === "", то *{{ field3 | is-not-empty }}* вернёт 0; если field4 === "some-value", то *{{ field4 | is-not-empty }}* вернёт 1

is-set - проверяет существует ли значение у указанного контрола - если да, то вернётся 1, иначе - 0

Если field1 не существует, то *{{ field1 | is-set }}* вернёт 0; если field2 === null, то *{{ field2 | is-set }}* вернёт 0; если field3 === "some-value", то *{{ field3 | is-set }}* вернёт 1

if - простейший условный оператор: принимает на вход 3 значения - условие, значение 1 и значение 2. Если условие истинно/существует, то возвращает значение 1; иначе - возвращает значение 2. this - полученное модификатором значение.

Если value === 3, то *{{ value | if:2:"value is equal to two!":"value is not equal to two..." }}* вернёт "value is not equal to two..."

or - возвращает STATEMENT если значение falsy: Если выполняется над нулевым значением (null, undefined, '') или над выражением с ошибкой (типа "x + undefined = z", по которому видно что из-за ошибки одно значение не удалось получить), то возвращает STATEMENT.

Если field1 не существует, то *{{ field1 | or:'error' }}* вернёт "error"; если field2 === null, то *{{ field2 | or:'error' }}* вернёт "error"; если field3 === "some-value", то *{{ field3 | or:'error' }}* вернёт "some-value"

switch - комплексный условный оператор: принимает на вход список кейсов в виде пар условие-значение, включая кейс default, представляющий ситуацию, когда ни одно из перечисленных условий не выполнилось.

Выражение *{{ x | switch:1:"uno":2:"dos":default:"extraviado" }}* вернёт "uno" если x === 1; "dos" если x === 2; в остальных случаях - "extraviado"

Даты:

date - преобразовывает дату в удобный для просмотра вид. Если передать true, то вернёт дату без времени.

Если my_date === "2014-02-27T10:00:00", то *{{ my_date | date }}* вернёт "27.02.2014 10:00", а *{{ my_date | date:true }}* вернёт "27.02.2014"

addPeriod - добавляет период к дате по коду.

Если my_date === "2014-02-27T10:00:00", то *{{ my_date | addPeriod:12M | date }}* вернёт "2015-02-27T10:00:00"

moment_get - получает для даты количество единиц, указанных в unit - годы, месяцы, часы и тд.

Например, если date === 2023-04-21T13:51:27.677, то выражение *{{ date | moment_get:year }}* вернёт 2023, а *{{ date | moment_get:month }}* - 4.

moment_add / moment_subtract - добавить к дате/отнять от даты указанное количество единиц. Например, если my_date === "2023-04-21T13:51:27.677", то *{{ my_date | moment_add:2:days | date}}* вернёт "23.04.2023 13:51"

Строки:

displayPeriod - функция для отображения периода в читабельном виде, принимает на вход код периода вида 20D, 3M, 12M.

Если period_code === '12M', то выражение *{{ period_code | displayPeriod }}* вернёт "12 мес."

trim - удаляет пробелы в начале и конце строки. Если value === " x ", то *{{ value | trim }}* вернёт "x"

split - разбивает строку по разделителю. В качестве разделителя можно указать RegExp.

Если my_string === "строка, с, запятыми", то *{{ my_string | split:',' }}* вернёт ["строка", " с", " запятыми"], а *{{ my_string | split:'\s?,\s?' }}* вернёт ["строка", "с", "запятыми"]

replace - ищет соответствия и заменяет их. Для поиска соответствия используется регулярное выражение. Например, если x === "aclm14110", то выражение *{{ x | replace:1:2 }}* вернёт 'aclm24220', а выражение *{{ x | replace:\\d+:2 }}* вернёт 'aclm2'. Регулярное выражение создаётся сразу с ключом g - global; спецсимволы должны быть экранированы.

includes - возвращает true/false если множества A и B пересекаются, то есть все или несколько элементов совпадают. Например, если a = '123', b = '234' и c = '987', то *{{ a | split:"" | includes:({{ b | split:"" }}) }}* вернёт true, а *{{ a | split:"" | includes:({{ c | split:"" }}) }}* вернёт false.

number - приводит строку представляющую число к стандартному формату отображения чисел. По умолчанию количество знаков после запятой - 2. Также принимает на вход количество знаков после запятой.

Если x === "12345.14159127369", то *{{ x | number }}* вернёт "12 345.14", а *{{ x | number:0 }}* вернёт "12 345"

to-number - приводит строку к числу. Умеет парсить float.

Если x === "3.14", то *{{ x | to-number }}* вернёт 3, а *{{ x | to-number:true }}* вернёт 3.14

Списки / массивы / таблицы:

selection - возвращает выбранные в таблице строки. На данный момент актуально только для object-list и multiple-table-select.

length - возвращает количество элементов списка/массива. Также можно применить и к строке.

Если в таблице my_table 17 элементов, то *{{ my_table | length }}* вернёт "17"

sumBy - суммирует все строки списка/массива по конкретному полю.

for-each - выполняет выражение для каждого элемента массива. this - текущий элемент.

Если array === ['1', '2', '3'], то *{{ array | for-each:(={{this}}=) }}* вернёт ['_-=1=-_', '_-=2=-_', '_-=3=-_']

get-item - получает из списка/массива элемент по индексу.

Пример: *{{ table | get-item:0 }}* вернёт первую строку.

pick - собирает все значения конкретного поля из каждого элемента списка/массива и возвращает в виде списка примитивных значений.

Если в строке таблицы table есть поле id, то *{{ table | pick:id }}* вернёт

[

{значение поля id из первой строки},

{значение поля id из второй строки},

...

].

map - собирает нужные значения из каждого элемента списка/массива и возвращает в виде списка объектов. Нужен для того чтобы отобрать из списка/массива конкретные поля. this - текущий элемент списка.

Пример: *{{ table | map:id:hub_Client.id:label:hub_Client.Name }}* вернёт

[

{

id: {значение поля hub_Client.id из первой строки},

label: {значение поля hub_Client.Name из первой строки},

},

{

id: {значение поля hub_Client.id из второй строки},

label: {значение поля hub_Client.Name из второй строки},

}

...

]

find - ищет элемент, соответствующий критерию, то есть возвращает элемент списка, для которого переданное выражение-условие вернуло true. this - текущий элемент списка.

Пример: если в таблице clients перечислены клиенты, для каждого из которых существует таблица счетов accounts, то например с помощью find можно проверить есть ли такой клиент у которого нету счетов - *{{ clients | find:(this | get-property:accounts | length | if:0:true:false) | is-set }}* вернёт 1 если в таблице clients есть клиенты без счетов.

filter - ищет элементы, соответствующие критерию, то есть возвращает элементы списка для которых переданное выражение-условие вернуло true. Аналогичен модификатору find, только возвращает множество элементов, а не один элемент. this - текущий элемент списка.

Пример: если pin === 1000008, то *{{ pin | split:'' | filter:(this | if:'0':true:false) | length }}* вернёт 5. То есть берём строку, делим на символы, получаем список символов, и фильтруем по критерию "если элемент равен '0', то true, иначе false", .

join - объединяет элементы массива в строку, между элементами будет помещен указанный пользователем разделитель.

Если array === [1, 2, 3], то *{{ array | join:'|' }}* вернёт "1|2|3"

Объекты:

get-property - возвращает значение запрашиваемого атрибута объекта. Чаще всего используется в связке с get-item: get-item отбирает нужный объект из списка, а get-property отбирает у него нужный атрибут.

Если в первом элементе массива arr есть свойство id со значением 1234, то *{{ arr | get-item:0 | get-property:id }}* вернёт "1234"

Специальные:

get-file - возвращает объект файла, выбранный в контроле file. Файл имеет такие поля:

- name - имя файла
- size - размер, в байтах
- type - MIME-тип файла
- content - содержимое файла в виде строки

Чтобы получить, например, содержимое файла нужно написать такое выражение: *{{ my-file | get-file | get-property:content }}*

get-policy-decision - возвращает true/false если объект доступен/не доступен пользователю. Объекты разбиты на группы - можно запросить разрешение объекта из той или иной группы, на данный момент доступная группа только одна - forms. Например, если значение контрола test === 'form_JL', то выражение *{{ test | get-policy-decision:forms }}* вернёт true если форма form_JL доступна пользователю.

get-status - возвращает один из статусов формы. Существующие статусы:

- changed
- unchanged
- valid
- invalid

Например, если текущая форма валидна и не изменена пользователем, то *{{ this | get-status:valid }}* и *{{ this | get-status:changed }}* вернут true и false.

Самостоятельные:

get-context - возвращает текущий контекст формы: FL или UL (реализовывалось для раздельных БД физических и юридических лиц).

Пример: выражение *{{ get-context }}* вернёт текущий контекст, если он у формы задан.

3.1.6. Предварительный просмотр формы

Данная функциональность позволяет просматривать получаемый результат по мере добавления различных элементов дизайна на форму без специального запуска формы в приложении или запуска бизнес-процесса.

При предварительном просмотре форма будет отображаться без данных, если у контролов не заданы константные значения через специальные параметры.

Функционал доступен в редакторе в закладке «Предварительный просмотр».

3.1.7. Хендлеры

Данный функционал позволяет с помощью типовых процедур получать данные на форме или выполнять определенные действия на форме. Такие процедуры выполняются при наступлении событий, перечисленных в списке триггеров; в качестве триггера можно использовать одно из фиксированных событий или имя поля. Если триггер - имя поля, то хендлер выполнится при изменении пользователем этого поля (при условии, что значение поля установлено).

Для вызова хендлера необходимо выбрать вид хендлера из списка возможных, задать триггер (также из списка), параметры хендлера, при необходимости – условия выполнения хендлера.

Фиксированные события триггера:

- **Сборка формы** – событие сборки формы, это самое раннее событие, происходящее в процедуре обработки формы, до любого другого;
- **Инициализация формы** – событие инициализации формы, выполняется после сборки формы и подхватывает значения, заданные контролам в форме;
- **Возврат к форме** – событие возврата к форме;
- **Нажатие кнопки «Назад»** - нажали на кнопку Назад

Контролы, которые могут выступать в качестве триггера:

- Action
- Select
- Dynamic-field
- (Другие изменяемые контролы)

Параметр «Имя хендлера» для всех видов хендлеров опциональный и задается тогда, когда данный хендлер нужно вызвать из другого хендлера. Другие параметры хендлеров задаются в зависимости от вида хендлеров. Также, любой из параметров хендлера может быть выражением.

Условия выполнения – это дополнительные условия, при выполнении которых хендлер будет выполняться. Этот параметр позволяет более гибко управлять запуском хендлеров.

3.1.7.1. Виды хендлеров

Хендлеры вне групп

Вывод сообщения – выводит на экран сообщение. Если не задан текст сообщения, то сообщение выведено не будет, а в консоль будет выведено соответствующее предупреждение

Параметры:

- Тип – тип сообщения, поддерживается 5 типов сообщений: success, error, warning, info, show
- Текст – текст сообщения
- Заголовок – заголовок окошка, в котором выводится текст сообщения

Вывод модального окна - отображает на экране модальное окно, которое в зависимости от указанного типа ждёт от пользователя того или иного действия. От действия пользователя зависит дальнейшая обработка: если этот хендлер используется перед выполнением операции и пользователь изъявил отказ, то целевой хендлер выполнение операции выполнен не будет. Специальный тип custom позволяет использовать собственный шаблон для модального окна, для чего нужно предварительно создать контейнер modal с требуемым наполнением. В качестве заголовка модального окна используется заголовок, указанный в хендлере или заголовок указанного контрола modal. Соответственно, если нет ни того, ни другого, то у модального окна не будет заголовка. Для закрытия модального окна в ней нужно создать кнопки с соответствующими ролями - Кнопка согласия и Кнопка отказа. Клик вне области модального окна также закроет её с отрицательным решением.

Параметры:

- Тип – тип модального окна, поддерживается 3 типа: alert, confirm и custom
    - Alert – информационное окно с одной кнопкой «ОК»;
    - Confirm – окно с текстом (вопроса) и двумя кнопками «Да» и «Нет»;
    - Custom – модальное окно, которое для отображения использует контрол типа modal.
- Заголовок – заголовок окна
- Текст – текст, который будет отображаться в модальном окне (для alert и confirm)
- Модальное окно – (для custom) имя контрола modal из формы

Модификация и сохранение данных в форму для дальнейшего использования другими контролами – используется в связке с выполнением пользовательского скрипта.

Параметры:

- Имя создаваемого контрола, куда будут сложены данные
- Модификация входных значений

Выполнение пользовательского скрипта – позволяет выполнять скрипты, написанные пользователями, для какого-либо преобразования данных, которое невозможно реализовать простыми средствами редактора.

Параметры:

- Скрипт – окно редактирования пользовательского скрипта. Поддерживается JavaScript.

Прокрутка страницы – прокручивает страницу до указанного поля

Параметры:

- Целевое поле – имя поля, до которого будет прокручена страница. После прокрутки это поле будет видно пользователю
- Выравнивание по вертикали – тип выравнивания, возможно 4 типа:
  - Элемент будет расположен в центре (center)
  - Элемент будет расположен в конце страницы, в самом низу (end)
  - Докрутить до ближайшего края элемента (nearest)
  - Элемент будет расположен в начале страницы, в самом верху (start)

Группа «Формы»

В данном разделе объединены хендлеры, предназначенные для взаимодействия между формами.

Открытие формы – открывает форму объекта по заданным параметрам. Работает с контролом action и пока не работает с object- list и link.

Параметры:

- **Идентификатор объекта** – идентификатор объекта, который будет использован для загрузки данных в форму. Поддерживает выражения. Если не задан, то шаблон не будет обогащен данными объекта.
- **Шаблон** – имя шаблона, который будет использован в качестве новой формы.
- **Способ открытия формы** – на данный момент существует три способа открытия формы:
  - В качестве самостоятельной формы (по умолчанию, также можно задать как _form)
  - В модальном окне (задается через _modal)
  - В контейнере (указывается имя контейнера, созданного в данной форме; контейнеры появятся в выпадающем списке)
- **Экспорт значений** – в открываемую форму можно перенести значения из текущей формы, для этого надо добавить нужные поля в список для экспорта и назначить им имя в целевой форме. Например, если выбрать для экспорта поле test-field и назначить ему имя imported-test-field, то при открытии целевой формы в ней появится скрытое поле imported-test- field с тем значением, которое было у поля test-field в предыдущей форме. Если при импорте поля окажется что такое поле в форме уже

есть, то его значение будет перезаписано, а в консоль будет выведено оповещение.

- **Таблица-источник** – для того чтобы открыть форму объекта из таблицы, нужно указать целевую таблицу. Это нужно только для того чтобы создать линк для нового объекта - если в таблице нет линка для открываемого объекта, то он будет создан, и его можно будет впоследствии сохранить стандартными средствами.
- **Строка таблицы** - если указана таблица-источник, то также нужно указать какую именно строку открываем. В качестве значения может быть только выражение. Если строка не найдена или не указана, то при открытии формы будет создан линк, который можно будет сохранить стандартными средствами.
- **ID хаба** - для того чтобы создать линк, нужно знать идентификаторы двух хабов, но в общем случае их нельзя узнать сразу, так как один из хабов, возможно, будет собран позже. Поэтому нужен способ указать переменный ID хаба. Как раз для этого и нужен этот параметр - указывается выражение, а при сохранении второй хаб нашего линка получит тот ID который получился в результате обработки выражения.
- **Имя хендлера** - устанавливается для того, чтобы можно было сослаться на конкретный хендлер. Опционально.

Обновление формы – обновляет форму. На данный момент варианта всего 2 можно обновить эту форму, ту в которой создан хендлер, или можно обновить текущую форму в цепочке форм.

Параметры:

- Целевая форма – два типа целевой формы:
  - Эта форма (_self)
  - Текущая форма в цепочке открытых форм (_current)

Переход на форму / страницу – совершает переход: либо на одну из страниц, перечисленных в меню; либо на одну из предыдущих форм в текущей цепочке форм; либо закрывает активные модальные окна; либо по конкретному URL, достроенному после текущего base. Если указан переход на предыдущую форму, но указанная форма не найдена, то переход осуществлён не будет, а в консоль будет выведено соответствующее сообщение.

Параметры:

- Тип перехода – поддерживается три типа перехода:
    - Предыдущая форма (previous-form) – переход к одной из предыдущих форм из цепочки форм;
    - Закрытие модального окна (close-modal) – при указании данного типа 1) не обязательно заполнять target и 2) закрыты будут все активные модальные окна
    - Навигация (url) – переход по заданному адресу
- Экспорт значений - в открываемую форму можно перенести значения из текущей формы, для этого надо добавить нужные поля в список для экспорта и назначить им имя в целевой форме. Например, если выбрать для экспорта поле test-field и назначить ему имя imported-test-field, то при открытии целевой формы в ней появится скрытое поле imported-test- field с тем значением, которое было у поля test-field в предыдущей форме. Если при импорте поля окажется что такое поле в форме уже есть, то его значение будет перезаписано, а в консоль будет выведено оповещение.
- URL строка (с поддержкой выражений) – можно задать путь до конкретной формы, например: 'audit-ui/audit/operations/{{Client_Id}}'
- Целевая форма/страница – для типа перехода url значение '_blank' будет открывать ссылку в новой вкладке, пустое - в текущей.

Группа «Операции»

В данном разделе объединены хендлеры, запускающие какие-либо операции.

Выполнение операции – выполняет указанную в параметрах операцию

Параметры хендлера:

- **Класс** – класс BOCLass объекта из модели данных ITA Objects. В выпадающем списке отображаются все доступные классы
- **Операция** – доступные для выбранного класса объекта операции.
- **Идентификатор объекта** – идентификатор объекта, над которым выполняется операция.
- **Аргументы для операции** – набор входных аргументов для операции. В качестве значения любому аргументу можно указать выражение, по которому при выполнении операции будет рассчитано значение аргумента. Если при расчёте значения аргумента произойдет ошибка (например, в случае отсутствия поля, использованного в выражении), то это поле будет исключено из аргументов операции.
- **Перед выполнением** – перед выполнение операции можно выполнить те или иные действия, представленные набором хендлеров. Ошибка или отрицательный результат при обработке этих хендлеров остановит

цепочку выполнения: ни последующие хендлеры, ни сама операция не будут выполнены.

- **Критерии вывода на успех/ошибку** – после выполнения операции, её ответ не всегда представляет собой положительный результат. Иногда он может быть вполне успешным, но говорящим о том, что на процессе есть ошибка и, как пример, должен не давать возможность пропустить процесс далее. Этот параметр служит как раз регулятором того, на какой случай мы должны наткнуться «успех / ошибка / другое» в случае выполнения условия. Пример условия *{{ response | get- property: clientStateList | find:(this | get- property: error)}}* - говорит о том, что в ответе операции есть массив clientStateList, который ищет поле еггог в любом из его элементов. Если такое поле находится, то ивент бросается в либо успех, либо ошибку, либо другое
- **После выполнения** – после выполнения операции, в зависимости от результата, можно выполнить те или иные действия, представленные набором хендлеров. Для этого нужно задать хендлерам имена и потом указать их в поле требуемого результата «успех», «ошибка» или другого. Например, можно вывести сообщение об успешном выполнении операции и вернуться на заданную форму: для этого нужно создать два соответствующих хендлера и указать их последовательно в данном параметре.

Всем хендлерам, выполняемым в рамках этой операции, ответ операции доступен по ключу response. Например, если в ответе есть поле message, то оно доступно по ключу response.message.

READER – операция чтения данных и сборки формы. Для этого хендлера не нужны триггеры, но он выполняется при сборке формы. Если на форме в разделе «Паттерн» не задан паттерн, то будет использован паттерн текущего шаблона.

Параметры хендлера:

- Класс – класс BOCLass объекта из модели данных ita-objects. В выпадающем списке отображаются все доступные для чтения классы
- Операция – список доступных операций для выбранного класса. Нужно выбирать операцию, которая возвращает в ответе json в структуре HLS
- Идентификатор объекта – идентификатор объекта над которым выполняется операция. Если не задан, то будет использован идентификатор, с которым была вызвана форма. Например, в случае с виджетом идентификатор указывается в параметре object-bocode

самого поля widget, и при сборке формы виджета READER получит этот идентификатор

- Аргументы для операции – набор входных аргументов для операции. В качестве значения любому аргументу можно указать выражение, по которому при выполнении операции будет рассчитано значение аргумента. Если при расчёте значения аргумента произойдет ошибка (например, в случае отсутствия поля, использованного в выражении), то это поле будет исключено из аргументов операции. Поддержка выражений – такая же, как и в хендлере Выполнение операций
- После выполнения – функционал действий в зависимости от результата выполнения взят из хендлера Выполнения операций.

SAVER – Операция сохранения данных и сборки формы. В случае, если параметры не заполнены, будут использованы значения по умолчанию.

Параметры хендлера:

- Класс – по умолчанию Project, но можно также выбрать класс из доступных BOClass модели
- Операция – по умолчанию DVSaver, но также можно выбрать операцию
- Идентификатор – идентификатор объекта, над которым выполняется сохранение

Группа «Справочники»

В данном разделе объединены хендлеры для работы со справочниками. Под справочниками подразумеваются справочники, содержащиеся в модуле ITA Info, который является составной частью ITA FORMS.

Наполнение селекта из справочника – загружает для указанного поля опции из справочников. Допустимые типы поля для данного хендлера: select, multiple-select, dual-select, radio-button, solid-radio-button, chips-select, stepper

Параметры:

- **Справочник** – код справочника из справочной системы ITA Info. Можно вписать выражение, по которому будет рассчитан код справочника.
- **Лейбл** – имя атрибута справочной записи, значение которого будет использовано в качестве лейбла, отображаемого на форме. Выбор в

выпадающем списке происходит из атрибутов справочников в справочной системе ITA Info. По умолчанию выбран label

- **Ключ** – имя атрибута справочной записи, значение которого будет использовано в качестве ключа (сохраняемое и передаваемое значение). Выбор в выпадающем списке происходит из атрибутов справочников в справочной системе ITA Info. По умолчанию выбран refKey
- **Фильтр** – задание условия для фильтра данных из справочника. Можно выбрать атрибут справочника и указать критерий
- **Установить значением первый элемент** – если установлен, то первая запись справочника будет автоматически выбрана в поле (если нет – то поле будет пустым, пока что-нибудь не выберем)
- **Атрибут для сортировки** – имя атрибута справочной записи, по которому будет производиться сортировка. По умолчанию выбран атрибут showOrder
- **Сортировать по убыванию** – по умолчанию, когда данный параметр не установлен, сортировка выполняется по возрастанию. Ее можно изменить, установив данный параметр в «Да».
- **Поле** – имя поля на форме, для которого будут загружены опции.

Наполнение контрола значением из справочника – загружает в указанное поле значение из справочника. Данный хендлер нужен, чтобы для ключевого значения (например, какого-то кода) найти читабельную расшифровку в справочнике и отобразить на форме. Для данного хендлера допустим тип поля input.

Параметры:

- Справочник – код справочника из справочной системы ITA Info. Можно вписать выражение, по которому будет рассчитан код справочника.
- Атрибут для вывода – имя атрибута справочной записи, значение которого будет выведено в целевой контрол. По умолчанию выбран label
- Идентификатор записи – идентификатор записи, то есть на самом деле ее refKey. То есть здесь указывается имя контрола (в {{}}), значение которого будет являться refKey в заданном справочнике, и по которому будет выполняться поиск атрибута для вывода.
- Целевое поле – имя поля, в которое будет загружено значение

Таблицы

В данном разделе объединены хендлеры для манипуляций с данными в таблицах, которые невозможно выполнить стандартными способами в редакторе

Добавление объектов – добавляет объекты в таблицу из списка объектов. Используется в multiple-table- select, simple-table, object-table, address-table

Хендлер принимает на вход имя целевой таблицы и имя формы отбора: при выполнении этого хендлера будет отображена форма отбора, пользователь сможет выбрать объекты и нажать на кнопку отбора, после чего выбранные объекты будут добавлены в таблицу.

Параметры:

- Целевая таблица – таблица, в которую будут добавлены объекты. Важно: так как для получения данных реестра используется паттерн, то в шаблоне целевой таблицы должны быть использованы атрибуты именно этого паттерна, а не атрибуты хаба. Иначе нужно указать связь полей паттерна с полями HLS-объекта
- Форма отбора – имя шаблона, который будет использоваться в качестве формы отбора
- Маппинг полей – так как для получения данных используется паттерн, то для успешного создания HS-объекта нужно указать с каким атрибутом HLS-объекта связан атрибут паттерна: pattern -> HLS. Нужно только если поля целевой таблицы непосредственно представляют атрибуты HS-объекта.
- Источник объектов – таблица или массив, из которой будут получены объекты. Специфическая настройка, нужная для тех случаев, когда список объектов полученный из целевого ita-object-list нуждается в пользовательской обработке перед добавлением

Виджеты

В данном разделе сгруппированы хендлеры для взаимодействия с виджетами.

Обновление данных виджета – обновляет данные виджета. Список объектов (ita-object-list) в том числе является виджетом, поэтому и его можно обновить при желании. Также возможно обновить родительский виджет, в случае применения этой функции будет найден ближайший родитель-виджет и обновлен.

Параметры:

- Виджеты – список доступных полей-виджетов на форме. Можно выбрать из выпадающего списка, или в некоторых случаях написать вручную
- Обновить родительский виджет – при установке в «Да» происходит поиск и обновление ближайшего виджета-родителя.

Импорт данных в файл – выгружает данные контрола в файл. На данный момент хендлер применим только к ita-object-list.

Параметры:

- Имя контрола – имя контрола ita-object-list, данные из которого который предполагается импортировать в файл
- Формат выгрузки – на данный момент возможна только выгрузка в csv-файл

Удаление бизнес-объекта – проставляет бизнес-объекту все атрибуты, соответствующие удалённому состоянию. Если аргументы не заданы, то проставляет атрибуты корневому для текущей формы бизнес-объекту. Для фактического удаления после проставления атрибутов требуется сохранить объект.

Параметры:

- Класс – по умолчанию Project
- Операция – по умолчанию DVSaver
- Идентификатор объекта – идентификатор объекта, над которым выполняется операция
- Перед выполнением – перед выполнением операции можно выполнить те или иные действия, представленные набором хендлеров. Ошибка или отрицательный результат при обработке этих хендлеров остановит цепочку выполнений ни последующие хендлеры, ни сама операция выполнены не будут.

3.1.8. Паттерн формы

Данный функционал позволяет задать источник данных для определенной формы.

Паттерн – это описание структуры данных в формате DataVault (HLS - хабов / сателлитов / линков). В такой структуре есть один главный хаб. Для работы паттерна ID этого главного хаба должен быть каким-то образом задан: можно задать константой, а можно передать динамически.

Пример паттерна с указанием hub_id в параметре BOCode:

```
{
 "Pattern": {
      "hub_MPShowcase": {
        "sat_MPShowcase_Main": {},
        "link_MPVersion_MPShowcase": {
          "hub_MPVersion": {
            "sat_MPVersion_Main": {},
            "link_MPField_MPVersion": {
              "hub_MPField": {
                "sat_MPField_Main": {}
              }}}}} ,
  "BOCode": "32b37788-2ff2-4431-9931-eca3c9d7879e"
}
```


3.1.9. Параметры формы

Данный функционал позволяет дополнительно установить для формы глобальные параметры.

В настоящий момент существуют следующие глобальные параметры:

- **Лейбл для кнопки «Назад»** - позволяет подменять название стандартной кнопки «Назад» на другое произвольное название.
- **Лейбл для breadcrumbs** – этот лейбл будет использоваться для отображения имени формы в строке пути - breadcrumbs. При его отсутствии будет использован основной лейбл шаблона формы.
- **Контекст обращений ФЛ / ЮЛ** – специальный параметр для запросов в отдельные базы данных для физических лиц или юридических лиц. Все хендлеры, виджеты, реестры добавляют в запрос этот параметр, указывающий на контекст - ФЛ или ЮЛ.
- **Использовать данные родительской формы** – используется, когда данные родительской формы еще не хранятся на сервере.
- **Использовать Id корневого объекта родительской формы** – используется, когда данные родительской формы еще не хранятся на сервере.

Добавление и удаление глобальных параметров на форме выполняется с помощью соответствующих функциональных кнопок.

3.2.Редактор процессов

Функционал редактора процессов позволяет:

- создавать новый бизнес-процесс в нотации BPMN,
- редактировать существующий бизнес-процесс,
- настраивать свойства процесса и задач, необходимые для технического исполнения,
- загружать BPMN-схему бизнес-процесса, созданную в каком – либо другом редакторе,
- связывать экранные формы, созданные с помощью редактора форм, с пользовательскими задачами в бизнес-процессе,
- выкладывать схему бизнес-процесса напрямую на сервер Camunda.

Функционал доступен в разделе «Редактор процессов» из меню приложения.

3.3.Старт процессов

Данный функционал позволяет после создания бизнес – процесса запустить выполнение этого процесса для проверки работоспособности.

Бизнес-процесс может быть запущен как вручную, так и автоматически, если он встроен в какой-нибудь другой процесс.

При запуске бизнес – процесса для него автоматически генерируется уникальный идентификатор businesskey, далее по этому идентификатору можно отслеживать выполнение.

3.4.Множественная загрузка форм

Данный функционал предназначен для групповой загрузки форм на разные стенды, массовой выгрузки форм и схем бизнес – процессов для последующей загрузки на другой стенд, поиска форм.

Поддерживаются следующие возможности:

- **Глобальный поиск** – данная функциональность позволяет осуществлять поиск по всем зарегистрированным формам. Поиск осуществляется по всем атрибутам формы - по имени, лейблу, содержимому полей и т. д. Кликните по имени формы чтобы открыть

ее в редакторе; по клику на имя поля форма также откроется в редакторе - поле будет подсвечено.

- **Загрузить форму / формы** – данная функциональность позволяет загружать форму или несколько форм из файлов .json или .txt
- **Скачать форму / формы** – данная функциональность позволяет выгружать форму или несколько форм в файлы .json или .txt для последующей загрузки на другом стенде.
- **Загрузить процесс / процессы с формами и скриптами** – данная функциональность позволяет загружать bpmn-схему бизнес – процесса вместе с файлами форм одним zip-архивом.
- **Скачать процесс / процессы с формами и скриптами** – данная функциональность позволяет скачивать в виде единого zip-архива bpmn-схемы бизнес – процессов вместе с файлами форм.

3.5. Описание объектов DataVault

Данный функционал предоставляет возможность просматривать бизнес-объекты системы в виде модели DataVault.

Функционал доступен в разделе «Объекты DataVault» из меню приложения.

3.6. Редактор бизнес – объектов

Функционал редактирования бизнес-объектов позволяет описать модель данных.

Функционал доступен в разделе «Редактор бизнес-объектов».

4. Программно-аппаратные требования для установки и функционирования модуля ITA Designer

Для использования приложения требуется следующее программное обеспечение на клиентском рабочем месте:

Операционная система:

- Windows (7 и выше)
- Linux

- Mac OS X

Интернет-браузер:

- Google Chrome 23 и выше (рекомендуется)
- Mozilla Firefox 18 и выше
- Safari 6 и выше
- Mozilla Firefox 17 ESR, 24 ESR, 31 ESR, 38 ESR, 45 ESR
- Internet Explorer 10 и выше, Edge
- Opera 16 и выше

5. Программное обеспечение, используемое для создания модуля ITA Designer

Языки программирования: TypeScript

Используемая платформа разработки: Angular 14.2

Используемые сторонние компоненты ПО: bpmn.io

6. Режим функционирования программного обеспечения

Основной режим работы Системы 24x7 и не подразумевает профилактических, либо иных плановых остановок.