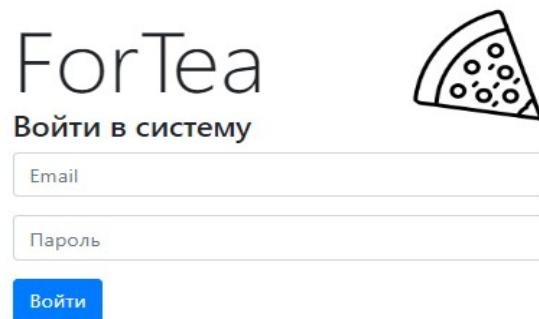


<b>Страница авторизации.....</b>	<b>3</b>
<b>Главный экран.....</b>	<b>3</b>
<b>Конструкторы.....</b>	<b>5</b>
<b>Деревья.....</b>	<b>11</b>
<b>Объекты.....</b>	<b>13</b>
<b>История.....</b>	<b>15</b>
<b>Работа с формулами.....</b>	<b>21</b>
<b>Константы.....</b>	<b>25</b>
<b>Массивы (array).....</b>	<b>28</b>
<b>Словари (Dictionary).....</b>	<b>30</b>
<b>Справочник по формулам.....</b>	<b>32</b>
<b>Функция create_object.....</b>	<b>46</b>
<b>Функция edit_object.....</b>	<b>47</b>
<b>Функция remove_object.....</b>	<b>48</b>
<b>Функция remove_object_list.....</b>	<b>49</b>
<b>Функция get_object.....</b>	<b>50</b>
<b>Функция get_object_list.....</b>	<b>50</b>
<b>Функция get_object_hist.....</b>	<b>52</b>
<b>Функция «create_class».....</b>	<b>54</b>
<b>Функция «edit_class».....</b>	<b>55</b>
<b>Функция «remove_class».....</b>	<b>56</b>
<b>Функция «create_class_param».....</b>	<b>56</b>
<b>Функция «edit_class_param».....</b>	<b>58</b>

<b>Функция «get_class».....</b>	<b>59</b>
<b>Функция «get_class_list».....</b>	<b>59</b>
<b>Функция «Выполнить вычисление».....</b>	<b>60</b>
<b>API.....</b>	<b>61</b>
<b>    create-object.....</b>	<b>61</b>
<b>    edit-object.....</b>	<b>62</b>
<b>    remove-object.....</b>	<b>63</b>
<b>    remove-object-list.....</b>	<b>63</b>
<b>    get-object.....</b>	<b>65</b>
<b>    get-object-list.....</b>	<b>66</b>
<b>    create-class.....</b>	<b>67</b>
<b>    edit-class.....</b>	<b>67</b>
<b>    remove-class.....</b>	<b>68</b>
<b>    create-class-param.....</b>	<b>68</b>
<b>    edit-class-param.....</b>	<b>68</b>
<b>    get-class.....</b>	<b>68</b>
<b>    get-class-list.....</b>	<b>69</b>
<b>    run-eval.....</b>	<b>70</b>
<b>Контракты.....</b>	<b>71</b>
<b>    Описание Тех. процесса.....</b>	<b>72</b>
<b>    Исследование Delay/Fact.....</b>	<b>82</b>
<b>Глоссарий.....</b>	<b>120</b>
<b>Примеры готовых конфигураций.....</b>	<b>120</b>

# Страница авторизации



ссылка <http://42.hq.f-trade.ru>

Тестовый доступ: admin1

Пароль: 12345

Проект имеет кодовое название 42. Был «40 (ForTea)» — но текущая реализация — уже вторая.

## Главный экран

Рабочее окно системы общий вид:

Справочники Контракты Задачи (0) История Добрый день, Алексей Титаренко. Users Выход

Курс Доллара 90,3846

Курс Евро 100,6562

График курсов за последнюю неделю

May 28 2023 Jun 4 Jun 11 Jun 18 Jun 25 Jul 2 Jul 9 Jul 16

Курс доллара Курс Евро

Список пользователей

6	robot dummy
7	Филипп Шулика
8	Алексей Титаренко
9	Сергей Александрович
10	Максим Юрковец
11	Виктор Свирь

Copyright © 2023 f-trade.ru. Все права защищены. Инструкция

Пока в прототипе окно после авторизации содержит только ссылки на объекты системы —

такие как Справочники, Контракты, Задачи, имя пользователя, ссылка на админку «Django» для управления списком пользователей и Выход.

Стартовое окно используется для формирования Дашборда для пользователя. (Дашборд — это набор ссылок на наиболее часто используемые объекты, набор статистических данных, которые оперативно требуются для сотрудника в его роли, часто используемые функции, в качестве примера в демо — используем график курсов валют и список пользователей).

dashboard (Константы главной страницы) размещен в системных данных. Справочник с ID 847 Пример:

Дерево справочников

Найти

- Личные данные ID: 218
- Деньги ID: 219
- Дерево тестов ID: 253
- Системный данные ID: 335
  - Типы данных ID: 144
  - Классы ID: 338
  - Константы главной страницы ID: 847
- Фонды ID: 430
- Клиенты и поисковые ID: 575
- Изделия складские ID: 636
- ИТ инфраструктура ID: 656
- Гран ID: 1172
- новая константа ID: 361
- поправочные коэффициенты ID: 622
- Годы услуг ID: 939

Управление классами

ID: 847

Название: Константы главной страницы

Тип: Константа

ID каталога: 335

Новый Сохранить Удалить

Управление параметрами константы

ID	Название	Формула
872	Курс Доллара	[[table.131.2.115]]
873	Курс Евро	[[table.131.3.115]]
874	график курсов за последнюю неделю	<pre>1 import plotly.graph_objs as go 2 from app.models import RegistratiorLog 3 backs_hist = list(RegistratiorLog.objects.filter(reg_name_id=1, json_class_id=111, 4 json_code=1, json_name=135)) 5 .order_by('id')[127] 6 euro_hist = list(RegistratiorLog.objects.filter(reg_name_id=1, json_class_id=111, 7 json_code=1, json_name=135)) 8 .order_by('id')[127] 9 x = [bh.date_update for bh in backs_hist] 10 e = [bh.json['value'] for eh in euro_hist] 11 y = [bh.json['value'] for bh in backs_hist] 12 fig = go.Figure() 13 # fig['layout']['yaxis']['autorange'] = "reversed" 14 fig.add_trace(go.Scatter(x=x, y=y, name='курс доллара')) 15 fig.add_trace(go.Scatter(x=x, y=e, name='курс Евро')) 16 result = fig.to_html()</pre>

1029 Список пользователей

Новый Сохранить Удалить

Условные обозначения

- обязательный параметр
- параметр отображается в таблице
- Для использования планирование значений для данного параметра

План — реализация частичная:

Это наши обычные константы — в котором мы можем для каждого пользователя настроить свой dashboard через eval поля.

Его и будем показывать при входе в систему — при рендрере главной страницы.

Конструктивно — нам не хватает возможности прав на основе CRUD для каждого реквизита. Понадобится еще один **системный справочник**, в котором мы будем задавать права на реквизиты справочников и контрактов в формате: создавать, читать, обновлять, удалять, потом понадобится возможность группировать эти права в роли, и роли привязывать к пользователю. В первом приближении можем воспользоваться и структурой типа alias (константы), но для управления видимостью не хватает функционала прав. Поэтому **dashboard сейчас работает для всех пользователей одинаково**.

Обойти это ограничение сейчас можно через проверку в коде поля, на id пользователя.

# Конструкторы

Переходим по ссылке Справочники.

The screenshot shows the 'Constructors' application interface. On the left, there's a tree view titled 'Dictionary' with various categories like 'Personal data', 'Contracts', 'Tasks', and 'History'. A search bar is at the top left. On the right, there's a 'Catalog management' section with fields for 'ID' (set to 218), 'Name' (set to 'Personal data'), 'Type' (set to 'Catalog'), and a 'Catalog ID' field. Buttons for 'New', 'Save', and 'Delete' are at the bottom. Below the main area, there's a legend for symbols used in the dictionary tree.

В прототипе по умолчанию открывается конструктор классов справочников.

В этом интерфейсе мы занимаемся описанием структур данных справочников и других объектов, включенных в дерево справочников. Здесь мы можем описать виды справочников, порядок их объединения в группы, объединение групп в иерархию и порядок использования таких структур как деревья.(\* расшифровка групп и деревьев или ссылка на определения).

Когда далее мы говорим об объектах системы, то мы ограничиваемся только теми, с которыми работаем в этом интерфейсе. Это — справочники, константы, такие системные объекты, как справочник прав пользователей. Мы не рассматриваем контракты и историю.

Когда мы работаем на странице конструктора справочников, то:

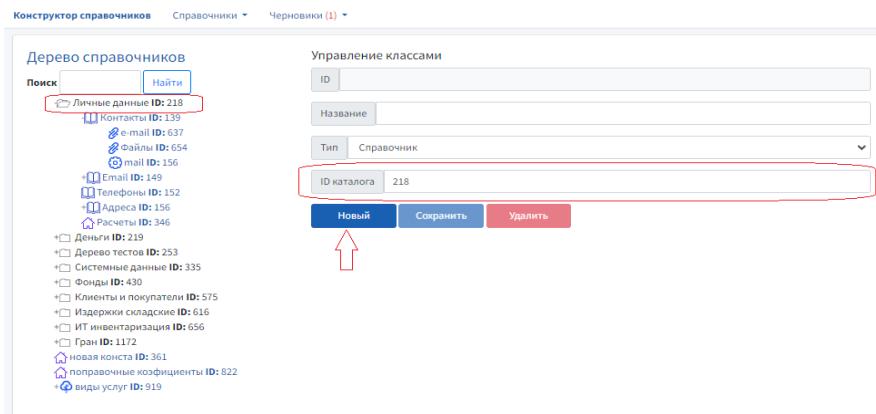
Левая часть — уже сформированные объекты системы(описания классов объектов системы). Правая - редактор текущего объекта(класса). По умолчанию сразу открывается первый сверху. В данном случае видим редактор каталога для удобства формирования и группировки классов. Также каталог позволяет управлять меню выбора объектов (справочников)(классов справочников)

This screenshot is similar to the previous one but includes a context menu that has been opened over the 'Personal data' node in the 'Dictionary' tree. The menu lists several sub-categories such as 'Contacts', 'Email', 'Phones', 'Addresses', 'Calculations', 'Clients and buyers', 'Warehouses', 'Inventory', 'Grants', 'New account', 'Corrective coefficients', and 'Service types'. This demonstrates how the application allows users to quickly navigate and manage different components of the dictionary structure.

На этом примере видно — нажатие на ссылку [Справочники](#) открывает меню с выбором справочников так — как это указано в конструкторе. С выбором данных объектов справочников и других данных, размещенных в дереве справочников.

Вернемся к интерфейсу «Конструктора справочников».

Создадим тестовый класс справочника *типа* «Каталог» — нажимаем кнопку Новый

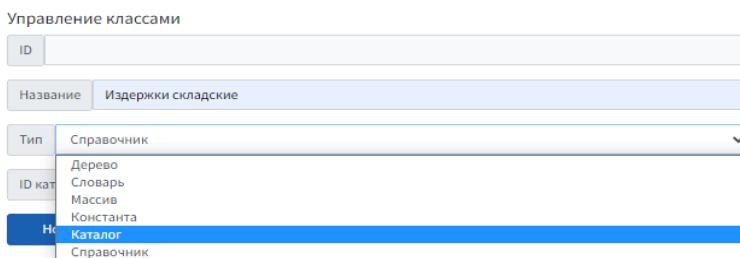


Шаблон страницы очищается — поля ID, Название. ID Каталога заполняется кодом родительской группы — берем его по текущей группе из «Дерева справочников» слева. В поле Тип — появилось слово «Справочник» - это значение берется по умолчанию, как первое в списке из справочника с ID 336 (Системные данные → Классы).

ID — формируется автоматически, после сохранения.

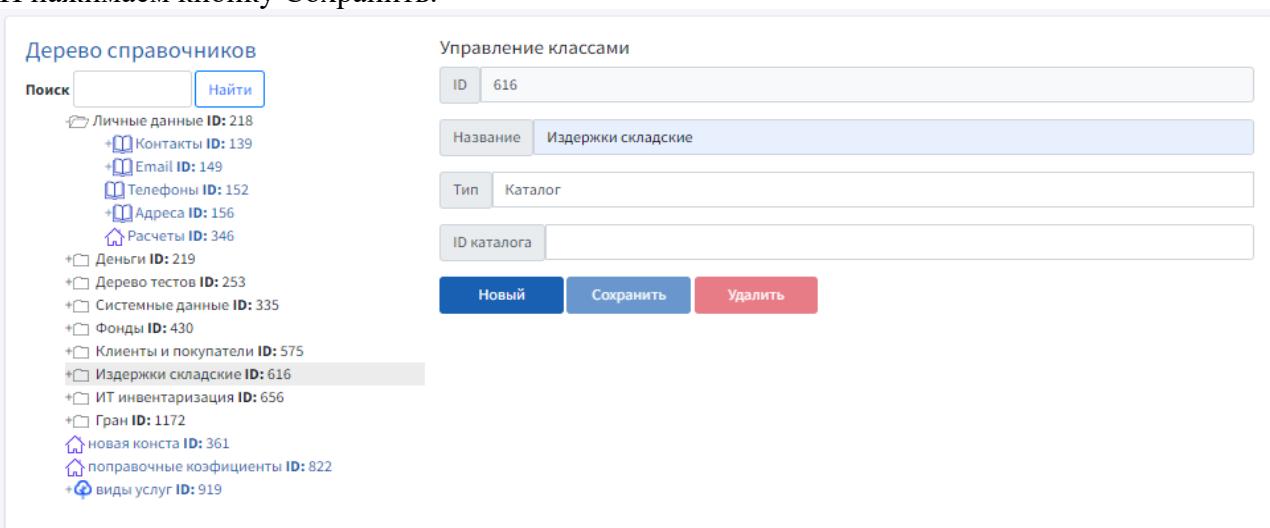
Название — задаем в произвольной форме — для примера создадим каталог «Издержки складские»

Указываем Тип: «Каталог» из выпадающего списка. Понятие Каталога определено на странице (\*)



И определяем этот каталог в корне дерева — значит(для этого) поле «ID каталога» оставляем пустым.

И нажимаем кнопку Сохранить.



Появился каталог с названием Издержки складские с ID 616

Перейдем в каталог и создадим там еще один класс справочника.

The screenshot shows the 'Dictionary Constructor' interface. On the left, there's a tree view of existing dictionaries. On the right, the 'Управление классами' (Class Management) section is displayed. It includes fields for 'ID' (set to 616), 'Название' (set to 'Складское оборудование'), 'Тип' (set to 'Справочник'), and 'ID каталога' (set to 616). Below these are three buttons: 'Новый' (New), 'Сохранить' (Save), and 'Удалить' (Delete).

Действуем аналогично созданию каталога  
в поле ID каталога набираем 616 — это только что созданный каталог «Издержки складские»  
Нажимаем кнопку Сохранить.

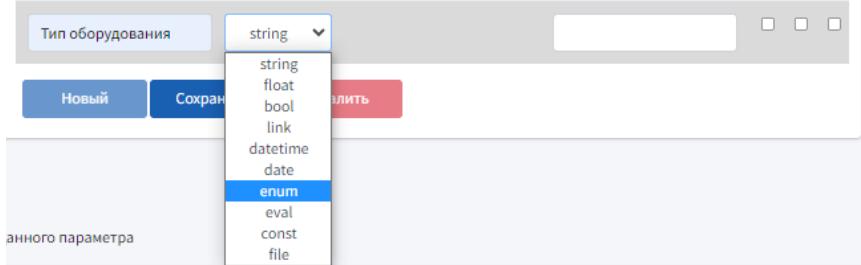
The screenshot shows the 'Dictionary Constructor' interface again. The 'Название' field now contains 'Складское оборудование'. The 'Управление классами' section shows the newly created class with ID 616. The 'Складское оборудование' class is listed under the 'Издержки складские' catalog. The 'Складское оборудование' class has its own tree view on the left.

Появился справочник с кодом 617. Перейдем в него по ссылке.

The screenshot shows the 'Управление параметрами справочника' (Dictionary Parameters Management) interface. It lists a single parameter: 'Наименование' (Name) with value 'Складское оборудование', 'Тип' (Type) set to 'string', and checkboxes for 'По умолчанию' (Default) and 'Обяз.' (Mandatory) both checked. Below this is a table with columns: ID, Название, Тип, Код значения, По умолчанию, Обяз., Видим. The first row shows the parameter with checked checkboxes for 'По умолчанию' and 'Обяз.'. At the bottom are three buttons: 'Новый' (New), 'Сохранить' (Save), and 'Удалить' (Delete).

В новых справочниках(В новом классе справочника) всегда есть обязательное поле —  
Наименование  
добавим новую колонку (реквизит) справочника — Тип оборудования — пусть это будет  
перечисление — в котором три значения - Стационарное оборудование, Расходные

инструменты, Движущаяся техника. Нажимаем кнопку Новый — внизу страницы. (названия кнопок надо бы переименовать — тяжело объяснять удаленному пользователю)



В названии пишем «Тип оборудования»

Тип определяем enum (перечисление) и заполняем его так:

ID	Название	Тип	Код значения	По умолчанию	O	T	D
618	Наименование	string			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
619	Тип оборудования	enum	1 Стационарное оборудование 2 Расходные инструменты 3 Движущаяся техника		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Отмечаем галку «T» - это позволит нам увидеть колонку в справочнике в общем списке — это чуть позже покажем, когда будем добавлять значение в сам справочник. Легенда подписана внизу страницы:

Условные обозначения  
 — обязательный параметр  
 Т — параметр отображается в таблице  
 D — Delay. Использовать планирование значений для данного параметра

Если не отмечать — то реквизит будет виден в значениях — когда перейдем на карточку элемента, но в таблице справа не будет «включена».

Нажимаем Сохранить.

Сразу аналогично добавим реквизит инвентарный номер — тип string, дату инвентаризации — тип datetime, Состояние удовлетворительное — тип bool, Остаточная стоимость — float, и Поставщик оборудования — link.

Добавить расшифровку типов из примера (\*)

Последнее нужно подробнее описать

В конструкторе уже есть созданный справочник Юридические лица с кодом 576

Дерево справочников

- + Личные данные ID: 218
- + Деньги ID: 219
- + Дерево тестов ID: 253
- + Системные данные ID: 335
- + Фонды ID: 430
- + Клиенты и покупатели ID: 575
  - Юридические лица ID: 576
- + Издержки складские ID: 616
- + Инн ID: 550
- + новая конста ID: 361

Управление классами

ID	576
Название	Юридические лица
Тип	Справочник
ID каталога	575

Новый Сохранить Удалить

Управление параметрами справочника

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
577	Наименование	string		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
578	ИНН	string		<input type="checkbox"/>	<input type="checkbox"/>	
579	Зарубежный	bool		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
586	Контроль ИНН	eval	def inn_ctrl_summ(nums, type): inn_ctrl_type = {	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

и мы хотим чтобы у нас появилась возможность из него выбирать значения. Поэтому делаем так: выбираем тип данных реквизита link, вид классов объектов — Справочник, а в качестве значения поля рядом(«Код значения») указываем ID класса справочника Юридические лица

— 576.

#### Управление параметрами справочника

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
618	Наименование	string			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
619	Тип оборудования	enum	Стационарное оборудование Расходные инструменты Движущаяся техника	Движущаяся техника	<input type="checkbox"/>	<input checked="" type="checkbox"/>
620	Инвентарный номер	string	PS:00		<input type="checkbox"/>	<input checked="" type="checkbox"/>
621	Дата инвентаризации	datetime	дд.мм.гггг --::--		<input type="checkbox"/>	<input type="checkbox"/>
622	Состояние удовлетворительное	bool	<input checked="" type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>
623	Остаточная стоимость	float			<input type="checkbox"/>	<input checked="" type="checkbox"/>

В поле код значения — устанавливаем код ссылки на справочник. Также отметим видимость для колонок Остаточная стоимость и инвентарный номер. Для колонки Инвентарный номер — я также установил значение по умолчанию «PS:00» — оно появится(будет появляться), когда мы создадим карточку(объект) этого справочника.

#### Описание работы с новым типом данных «file»

*Обновление от 3.06.22*

*Добавлен тип данных file.* (В нашей системе есть тип данных «file», предназначенный для хранения файлов внутри нашей системы).

Для нашего Справочника «Складское оборудование» - добавим реквизит с этим типом.

635	Акт приемки	file		<input type="checkbox"/>	<input type="checkbox"/>
-----	-------------	------	--	--------------------------	--------------------------

Посмотрим результат нашего добавления- как выглядят объекты «Складского оборудования» после добавления нового реквизита:

Справочник "Складское оборудование" ID: 617

Код	Наименование	Тип оборудования	Остаточная стоимость	Ост. стоим. в евро
2	Шланговый инструмент (лотата)	Расходные инструменты	100.0	0.99
1	Погрузчик	Движущаяся техника	756000.0	7510.71

Страница 1 из 10

Найти Export list Export object

Версия Сейчас  
От 24.06.2023 17:17:20 До 24.07.2023 17:17:20 Обновить таймлайн

Код 1

Наименование \* Погрузчик

Тип оборудования Движущаяся техника

Инвентарный номер PS:000015

Дата инвентаризации 27.05.2022 14:18:00

Состояние удовлетворительное

Остаточная стоимость 756000

Ост. стоим. в евро 7510.71

Поставщик оборудования

Акт приемки

Выберите файл Обзор

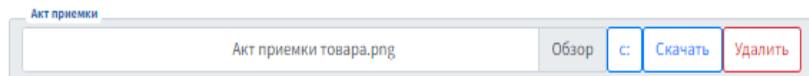
Новый Сохранить В черновик Удалить

Подсказки: \* — поля, обязательные к заполнению

Когда значение файла пустое — реквизит имеет одну кнопку «Обзор». Выбираем файл.

наименование поменяется, но элемент справочника еще не сохранен, и файл на сервер для хранения еще не передан. Сохраняем. Файл отправляется на хранение на сервер.

Выбранный реквизит после сохранения должен выглядеть так



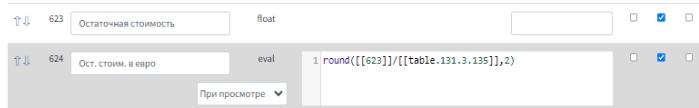
По кнопке Скачать — получаем файл, который мы загрузили до этого. По кнопке Удалить — удаляем файл с сервера (с хранения). Кнопка с невразумительным названием «с:» - копирует относительный путь файла на сервере в буфер обмена.

#### Тип значения реквизита «eval».

Есть еще один тип значения — который называется Eval - это реквизит который вычисляет формулы. Сделаем это поле — чтобы посчитать Остаточную стоимость(ID 623) по курсу Евро.

В системе уже создан справочник (Системное имя table) Валюта — код у него 131. В самом справочнике есть реквизит Курс. Код элемента Евро — 3. ID этого реквизита курс 135.

Формула расчета будет выглядеть так: [[623]]/[[table.131.3.135]]. Но так — как тут деление, а хочется видеть ее в формате денег — вызываем функцию python round – round([[623]]/[[table.131.3.135]],2). Подробней о формулах читайте в разделе «Error: Reference source not found Справочник по формулам» (\* указать номер страницы)



Отдохнем от нашего тестового примера. Вернемся к нему позже, когда будем рассматривать объекты. Нужно рассмотреть структуру данных — дерево.

# Деревья

Добавлен новый тип данных — Дерево. Это иерархический тип данных, описывающий подчиненность элементов. Оперирует двумя сущностями: Ветка (branch) и Куст (cluster).

Ветка — это элемент(узел,лист) дерева без подчиненных ему структура, а Куст — это как раз-таки как сам элемент, так и все подчиненные ему структуры. Когда мы говорим о рассчитываемых характеристиках групп объектов подчиненного класса, помещенных внутри дерева, то в формуле расчета характеристики мы можем конкретизировать рассчитывать эту характеристику по объектам(подчиненного класса) которые размещены только на Ветке или же включать в расчет объекты размещенные на Кусте.

Справочники Контракты История

Добрый день, Алексей Титаренко. Выход

Конструктор контрактов Контракты Черновики (0)

Дерево контрактов

Поиск по ID Найти

+ тесты контрактные ID: 4  
+ контракты с покупателями ID: 82  
+ тесты для истории ID: 171  
+ скрипты здесь ID: 31  
+ второй ID: 33  
+ третий ID: 35  
+ валюты контрактов ID: 37  
+ новый! ID: 77  
+ вид ID: 541  
+ 14741 ID: 551  
+ перезапись ID: 367  
+ четвертый ID: 73  
+ скажи куст ID: 377

Управление контрактами

ID: 367  
Название: Третье дерево  
Тип: Дерево  
ID каталога

Новый Сохранить Удалить

Управление параметрами дерева

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
368	Правильное дерево	bool		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
406	podr	string		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
407	its ok	bool		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
408	Количество контрактов	eval	1 [[contract.73.all.271 2 {{branch agr_fun='count'}} 3 ]] При просмотре	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
547	Test	eval	1 [[class_id]] При просмотре	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Новый Сохранить Удалить

Дерево существует как отдельный тип данных класса, но в сочетании с «подчиненными» классами справочников енравочниками позволяет фильтровать данные интерактивно.

Пример интерфейса Дерева

Конструктор контрактов Контракты Черновики (0)

Дерево "Третье дерево" ID: 367

Поиск Найти

Ветка	podr	its ok	Количество контрактов	Test	Код	3
- 1 куст1		✓	1	367	Название	ромашка
2 картофель		✓		367	Родитель	1
3 ромашка	11	✓		367	podr	11
4 партизан		✓		367	its ok	<input checked="" type="checkbox"/>
5 Бересклет	ii	✓		367	Количество контрактов	
- 5 второй кустарник	22	✓		367	Test	367
6 урок		✓	1	367		
7 кизил	poi	✓	1	367		
8 уроки	88	✓	3	367		
11 pop	cvbcb	✓		367		

Новый Сохранить Удалить

Дереву могут быть подчинены несколько справочников.



Деревья позволяют интерактивно фильтровать подчиненные справочники, а также поддерживают расчет элементов как на ветках, так и на кустах. Подробнее про расчеты — в разделе «справочник по формулам».

The screenshot shows the 'Справочник "услуги"' (Service Catalog) interface. At the top, there's a navigation bar with 'Справочники', 'Контракты', 'Задачи (0)', 'История', 'Добрый день, Алексей Титаренко.', 'Users', and 'Выход'. Below the navigation is a breadcrumb trail: 'Конструктор справочников > Справочники > Черновики (1)'. The main area displays a table of service items with columns: Ветка (Branch), комментарий (Comment), уровень вложенности (Level of nesting), чиселко (Count), выкл/вкл (Enabled/Disabled), and таймштамп (Timestamp). One row is expanded to show its children: '0 виды услуг' (0 types of services), '42 Все услуги' (All services), '2 Основные' (2 Main), '19 Дополнительные' (19 Additional), and '6 Одноразовые' (6 One-time). To the right of the table are several input fields: 'Версия' (Version) set to 'Сейчас' (Now), timestamp range from '24.06.2023 18:49:33' to '24.07.2023 18:49:33', a 'Код' (Code) field with value '4', a 'Ветка' (Branch) field with value '20', a 'Нº' (No.) field with value '999', and a 'Название' (Name) field with placeholder 'до свидания!'. Below these are three tabs: 'коммент' (Comment), 'Fact', and 'delay', each with a date input field 'дд.мм.гггг ::--:: delay-date'. At the bottom are buttons for 'Новый' (New), 'Сохранить' (Save), 'В черновик' (To Draft), and 'Удалить' (Delete). A note at the bottom left says: 'Подсказки: \* — поле, обязательное к заполнению'.

Еще один пример с обновленным шаблоном:

### Альтернативное рассмотрение дерева.

В процессе работы с системой у нас появляются потребности не связанные непосредственно с занесением в нее информации из системной области. Можно сказать, что эти потребности связаны с отображением информации. Из таких соображений возникает такой вид классов как «Каталог» (\*может ли он быть в контрактах)

Из похожих соображений не связанных непосредственно с регистрацией информации о предметной области возникает и тип классов «Дерево». Этот тип может быть использован как в классах справочников, так и в классах контрактов.

Дерево — это тип класса, позволяющий организовать объекты другого класса в иерархию типа дерева. Еще одним свойством дерева является то, что для кустов и ветвей можно задать дополнительные вычисляемые параметры. Можно сказать, что дерево решает задачу группировки объектов, позволяет выстроить эти группы в иерархию и, для этой иерархии, задать дополнительные поля, вводящие дополнительные сводные характеристики для каждой группы.

Тип класса «Дерево» может быть привязан к одному или нескольким другим классам.

Каждый объект справочника «над» которым построено «Дерево» подчинено определенному узлу.

# Объекты

Вернемся к нашему примеру со складским оборудованием.

The screenshot shows the 'Builder of reference lists' interface. On the left, there is a tree view of reference lists: 'Личные данные', 'Деньги', 'Дерево тестов', 'Системные данные', 'Фонды', 'Клиенты и покупатели', 'Издержки складские', 'qq', and 'новая конста'. A context menu is open over the 'Складское оборудование' node, with options like 'Управление', 'Название', 'Справка', and 'ID каталога'. The main area shows the details of the selected item: ID 617, Name 'Складское оборудование', and Catalog ID. A 'Новый' (New) button is visible at the bottom right.

Пустой только что созданный класс справочника

The screenshot shows the 'Warehouse equipment' reference list creation form. At the top, it says 'Справочник "Складское оборудование" ID: 617'. Below that are search fields for 'Поиск' and 'Найти'. The main table has columns: Код, Наименование, Тип оборудования, Остаточная стоимость, Ост. стоим. в евро, and Код. There is a dropdown for 'Страница 1 из 1'. The form fields include: Наименование (Name), Тип оборудования (Equipment type, set to 'Стационарное оборудование'), Инвентарный номер (Inventory number), Дата инвентаризации (Inventory date, dd.mm.yyyy), Состояние (Condition, checked), Остаточная стоимость (Remaining value), and Ост. стоим. в евро (Remaining value in euros). At the bottom are buttons for 'Новый' (New), 'Сохранить' (Save), and 'Удалить' (Delete).

Здесь и далее объекты класса справочника мы будем называть элементами.

Создадим несколько элементов — Кнопка Новый. (Если передумали создавать, просто обновите страницу — f5 (в некоторых случаях еще на тестах — так как иногда копирует элемент) )

Заполняем Наименование, тип оборудования, инвентарный номер, дату инвентаризации, состояние, и остаточную стоимость. Поля с формулами вычисляются после сохранения (В платформе реализован контроль, чтобы объект не сохранялся — если в функции ошибки. Позволяет сделать контроль над правильностью заполнения реквизитов)

Жмем сохранить.

Справочники Контракты Задачи История

Конструктор справочников Справочники Черновики (1) Выход

Добрый день, Алексей Титаренко.

Помощь Найти Export list Export object

Справочник "Складское оборудование" ID: 617

Код	Наименование	Тип оборудования	Остаточная стоимость	Ост. стоим. в евро
2	Шанцовый инструмент (лопата)	Расходные инструменты	100.0	0.99
1	Погрузчик	Движущаяся техника	756000.0	7510.71

Страница 1 из 10

Версия Сейчас  
От 25-06-2023 11:17:24 До 25-07-2023 11:17:24 Обновить галереи

Код 2

Наименование Шанцовый инструмент (лопата)

Тип оборудования Расходные инструменты

Инвентарный номер PS:000018

Дата инвентаризации Дд.Мм.Гггг

Состояние удовлетворительное

Остаточная стоимость 100

Ост. стоим. в евро 0.99

Поставщик оборудования 1 ООО "АВ Сталь-Мет"

Акт приемки товара/предмета Обзор Скачать Удалить

Новый Сохранить В черновик Удалить

Подсказки:  
\* — поля, обязательные к заполнению

Переход по элементам слева, редактор реквизитов справа. Дизайн справочников обновили, добавили управление временем (Time line). Для сравнения со старым дизайном — смотри ниже.

Упустил что не сохранил реквизит Поставщик оборудования — добавим теперь и его. Вернемся в справочник — и посмотрим на результат (Скриншот из старого дизайна. Оставил пока для истории).

Код	Наименование	Тип оборудования	Остаточная стоимость	Ост. стоим. в евро
2	Шанцовый инструмент (лопата)	Расходные инструменты	100	1.55
1	Погрузчик	Движущаяся техника	756000	11731.99

Страница 1 из 10

Наименование Шанцовый инструмент (лопата)

Тип оборудования Расходные инструменты

Инвентарный номер PS:000018

Дата инвентаризации Дд.Мм.Гггг

Состояние удовлетворительное

Остаточная стоимость 100

Ост. стоим. в евро 1.55

Поставщик оборудования

Поставщик оборудования 1

Поставщик оборудования 1 ООО "АВ Сталь-Мет"

Поле появилось — укажем поставщика — пишем его код — в справочниках он сейчас всего 1, и код его тоже 1.

Поставщик оборудования

1
ООО "АВ Сталь-Мет"

Появился, сохраним.

# История

Здесь и далее реквизиты справочника мы можем называть полями.

Каждое поле в справочнике — исторический реквизит. Каждый раз — когда происходит сохранение — данные об этом попадают в историю. Данные ([истории?](#)) можно использовать для аналитики, безопасности и прочих событий в системе. Если нажать на ссылку — Поставщик оборудования — попадем в историю — как это поле изменилось.

История изменений						
Выберите период	01.01.1970 00:00	27.05.2022 11:48	ID регистратора	ID класса	Код Объекта	2
ID	reg ID	Имя регистратора	Обновлен	user ID	Пользователь	Данные
2415	391	Редактирование объекта класса ID:617. Название: Складское оборудование	27.05.2022 11:47:57	8	Алексей Титаренко	входящие данные:[{"code":2,"parent_structure":617,"625":{"id":1181,"name":"Поставщик оборудования","type":"link","value":"1"}] данные:[{"code":2,"parent_structure":617,"625":{"id":1181,"name":"Поставщик оборудования","type":"link","value":"1"}]}
2414	391	Редактирование объекта класса ID:617. Название: Складское оборудование	27.05.2022 11:47:54	8	Алексей Титаренко	входящие данные:[{"code":2,"parent_structure":617,"625":{"id":1181,"name":"Поставщик оборудования","type":"link","value":"1"}] данные:[{"code":2,"parent_structure":617,"625":{"id":1181,"name":"Поставщик оборудования","type":"link","value":"1"}]}
2413	391	Редактирование объекта класса ID:617. Название: Складское оборудование	27.05.2022 11:45:29	8	Алексей Титаренко	входящие данные:[{"code":2,"parent_structure":617,"621":{"id":1178,"name":"Дата инвентаризации","type":"datetime","value":"2022-05-24T14:33"}] данные:[{"code":2,"parent_structure":617,"621":{"id":1178,"name":"Дата инвентаризации","type":"datetime","value":""}],"625":{"id":1181,"name":"Поставщик оборудования","type":"link","value":"1"}]}

Свойство истории — часть платформы. Любые действия — приводят к появлению к событию в истории. В справочниках и в контрактах — всегда находятся актуальные данные на текущий момент. Если нужна информация — как так получилось — нужно подключать куб отчетов — для разбора этих «логов».

Каждый реквизит объекта имеет `datetime` последнего изменения. Это позволяет посмотреть на объект на временной шкале и получить информацию как выглядел этот объект с момента его создания, и кто вносил изменения.

У каждого объекта в интерфейсе есть «ползунок» времени (Time line), который можно передвигать от события в прошлом, до события в будущем — если на реквизитах установлен будущий `Delay`. Если у будущего `delay` не установлена дата — такое вполне возможно, если контракт находится в состоянии `Delay Wait for result` (в работе), когда ждем ответ на запрос, то данный `delay` игнорируется.

Пример:

Версия 24.07.2023 03:31:02 robot dummy

От 25.06.2023 11:22:57 До 25.07.2023 11:22:57 Обновить таймлайн

Код 3

Наименование \* Евро

Цифровой код \* State 978 Fact delay delay-date ДД.ММ.ГГГГ --:--

Символный код \* EUR

Курс 100,66

Дата обновления 24.07.2023 03:31:00

коэффициент 1,10

Внутренний курс 110.7218200000001

Тест Ошибка. Некорректно задан параметр формулы 135 {{agr\_fun = "count" date\_from = "26.09.2022"}}

**Новый Сохранить В черновик Удалить**

и смотрим историю двигая time line:

Версия 22.07.2023 03:31:02 robot dummy

От 25.06.2023 11:22:57 До 25.07.2023 11:22:57 Обновить таймлайн

Код 3

Наименование \* Евро

Цифровой код \* State 978 Fact delay delay-date ДД.ММ.ГГГГ --:--

Символный код \* EUR

Курс 100,66

Дата обновления 22.07.2023 03:31:00

коэффициент 1,10

Внутренний курс 110.7218200000001

Тест Ошибка. Некорректно задан параметр формулы 135 {{agr\_fun = "count" date\_from = "26.09.2022"}}

**Новый Сохранить В черновик Удалить**

Особенность работы с данными типа файл. Удалить «исторические» файлы невозможно. Обратите внимание на элемент «второй файл».

Версия	Сейчас			
От	25.08.2022 11:31:12	До	25.07.2023 11:31:12	Обновить таймлайн
Код	52			
Наименование *				
птица счастья				
актуальность				
<input checked="" type="checkbox"/>				
файл				
Выберите файл		Обзор	Удалить	
комментарий				
Коммент от 13.04				
второй файл *				
IMG_20220428_153244.jpg		Обзор	c: Скачать Удалить	
Службы доставки				
№	название	наложка	код склада	складское хранение
1	Альфа	✓	1	01.01.2002 01:03
3	гамма	✓	1	01.01.2002 01:03
4	дельта		1	19.07.2022 11:29

Здесь кнопка удалить файл — доступна.

Версия	13.04.2023 15:04:17 Филипп Шулика			
От	25.08.2022 11:31:12	До	25.07.2023 11:31:12	Обновить таймлайн
Код	52			
Наименование *				
птица счастья				
актуальность				
<input checked="" type="checkbox"/>				
файл				
Выберите файл		Обзор	Удалить	
комментарий				
коммент от 13.04				
второй файл *				
IMG_20220428_153244.jpg		Обзор	c: Скачать	
Службы доставки				
№	название	наложка	код склада	складское хранение
1	Альфа	✓	1	01.01.2002 01:03
3	гамма	✓	1	01.01.2002 01:03

Двигаем time line.

А тут этой кнопки нет. Возможность удалять файлы в прошлом заблокирована.

Такая работа с интерфейсом предполагает наличие возможности получать временной срез объекта в прошлом в настоящем и в планируемом будущем. В будущих релизах в планах сделать временной срез классов и с формулами. В текущем релизе при запросе в прошлое — формулы используются только текущие, структура данных тоже только текущая. Реализовали возможность посмотреть что находилось в массивах с учетом истории. На рисунках выше, в элементе «Службы доставки» это можно увидеть.

Замечание:

По умолчанию Time line в шаблоне загружается на **месяц назад** от текущего времени.  
**Загрузка данных Time line происходит после загрузки текущих данных**, чтобы не заставлять пользователя ждать.

Пример формулы с временем

```
[[contract.135.84.363 {{version='2022-03-27'}}]]
```

Такой же пример без истории

1. [[table.144.12.234]] или 2. [[table.144.12.234 {{version='2022-03-27'}}]]

в варианте 1 — мы никогда не будем использовать историю в eval полях — работает как и сейчас — берем реквизит на текущий момент

в варианте 2.

Берем информацию из истории. Если в поле timestamp время не установлено, то берем на текущий момент, если данные в timestamp установлены — то на позицию времени.

Пример:

При подборе товара в контракт с помощью link поля с id 12 — получаем ссылку на объект товар.

У товара есть поле Цена с id = 232

Формула которая возвращает цену на текущую цену выглядит так [[contract.12.232]].

Сохраняем контракт. В во всех полях текущего контракта появляется timestamp. Если формулу оставить без изменений, то при просмотре контракта — цена может обновится на новую. Чтобы это не допустить — формула должна была выглядеть так [[contract.12.232 {{version='timestamp'}}]]. Подробнее в «[Справочнике по формулам](#)»

Читаем как: **Взять объект из поля контракта с id = 12 и получить из Logs его реквизит с id 232 на позицию времени в timestamp. Если в timestamp – null, то в логи не лезем, а берем текущие данные — отрабатываем также как и раньше. [[contract.12.232]]**

Рассмотрим пример:

Создаем контракт на поставку товара под заказ. Подбираем товар в спецификацию. И теперь в такой простой ситуации у нас возникает вопрос, на какое время брать цену? У нас есть время создания контракта, у нас есть время текущее... предположим мы берем цену на текущее время, записываем контракт — отправляем заявку в снабжение по закупке этого товара и спокойно работаем над другими задачами. Через некоторое время поступает информация от нашего клиента, что ему нужно в этом контракте увеличить количество. Но буквально минуту назад менеджер по снабжению изменил цену на эту позицию (так как — при заказе получил новую информацию о повышении закупочной цены или других издержек), и если мы добавим этот товар — он добавит новую позицию такого же товара, но уже с новой ценой. Если мы изменим количество в товаре, который уже добавлен - то у нас есть еще несколько вариантов:

1. Используем прошлое время, через delay — изменяя количество, как если бы мы это количество изначально и подбирали по той — прошлой цене. Delay автоматически позволяет передать в снабжение корректировку заказа.
2. Усредняем цену, фактически добавляя такой же товар с новой ценой и объединяя позиции со старой ценой.
3. Обновляем цену на новую и в уже добавленной позиции, фактически удаляя позицию и добавляя ее снова с новым количеством по новой цене.

Когда мы добавляем товар в таком примере — мы можем работать с текущим временем, с

прошлым или будущим. Этот выбор либо должен быть предопределен контрактом (реквизитом контракта), либо это должно быть интерактивным выбором пользователя.

По умолчанию поведение такое:

Если в подобранным товаре еще нет timestamp (подобран, но еще не было записи), то используется текущее фактическое значение его реквизитов.

Если у товара в реквизите уже есть timestamp – с установленным временем, то его время используется как опорное. И следовательно пользователь — который «просто» изменил количество в товаре — не получит обновленную цену товара, так как работает с объектом в прошлом, с тем его состоянием, когда он подбирался. Естественно — если для этого реквизита установлено свойство delay – то редактирование в прошлом пойдет через него.

Если не установлено — то мы информацию о редактировании объекта запишем в историю, и изменение применим сейчас.

Разницу этих двух подходов можно объяснить так (подробнее см в разделе контракты).

Без Delay

Количество\Дата	Now-2	Now-1	Now
Fact	2	2	3

Получение данных по дате без delay

```
{time_position:'Now-2',fact_value:2,state_value:2}
```

С Delay (-2 дня)

Количество\Дата	Now-2	Now-1	Now
Fact	2	2	3
Delay	(+1)	(+1)	

Используя срез времени (интерфейсно — ползунок Time line) реквизиты с delay будут показаны по-разному, хотя на текущий момент разницы нет. Естественно получая информацию со значением в котором установлен delay – возвращается json структура типа

```
{time_position:'Now-2',fact_value:2,delay_value:1,state_value:3}
```

Из — за такого подхода формула может иметь разный вариант использования с временем  
[[contract.12.232 {{version='timestamp'}}]] – вернет State\_value (или просто value) – которая является суммой fact\_value и delay\_value

[[contract.12.232.fact {{version='timestamp'}}]] – вернет fact\_value – **на разработке**

[[contract.12.232.delay {{version='timestamp'}}]] – вернет delay\_value — **на разработке**

## Куб используется от WebDataRocks.

Отчет Web-Data-Rocks

Connect Open Save Export Format Options Fields Fullscreen

1	2	3	4	5
ID	TYPE	LOCATION	PARAM ID	PARAM VALUE
2	450	table		757 // коменты from app.models import ContractCells if [code] == 3: objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name') o = next o for o in objects if o.name_id == 20 o.value = int([452]) edit_objects = [] edit_objects.append(o) ContractCells.objects.bulk_update(edit_objects, ['value']) result = 'Ok' else: result = 'not applicable'
3	450	table		757 // коменты from app.models import ContractCells objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name') o = next o for o in objects if o.name_id == 20 o.value = int([452]) edit_objects = [] edit_objects.append(o) ContractCells.objects.bulk_update(edit_objects, ['value']) result = 'Ok'
4	450	table		757 // коменты from app.models import ContractCells objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name') o = next o for o in objects if o.name_id == 20 o.value = int([452]) edit_objects = [] edit_objects.append(o) ContractCells.objects.bulk_update(edit_objects, ['value']) result = int([452])
5	450	table		757 from app.models import ContractCells objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name') o = next o for o in objects if o.name_id == 20 o.value = int([452]) edit_objects = [] edit_objects.append(o) ContractCells.objects.bulk_update(edit_objects, ['value']) result = int([452])
6	450	table		757 from app.models import ContractCells objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name') o = next o for o in objects if o.name_id == 20 o.value = int([756]) edit_objects = [] edit_objects.append(o) ContractCells.objects.bulk_update(edit_objects, ['value']) result = int([452])
7	450	table		758 (blank)
8	450	table		757 from app.models import ContractCells objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name') o = next o for o in objects if o.name_id == 20 o.value = int([452]) edit_objects = [] edit_objects.append(o) ContractCells.objects.bulk_update(edit_objects, ['value']) result = int([452])
9	747	(blank)		(blank)
10				
11				
12				
13				

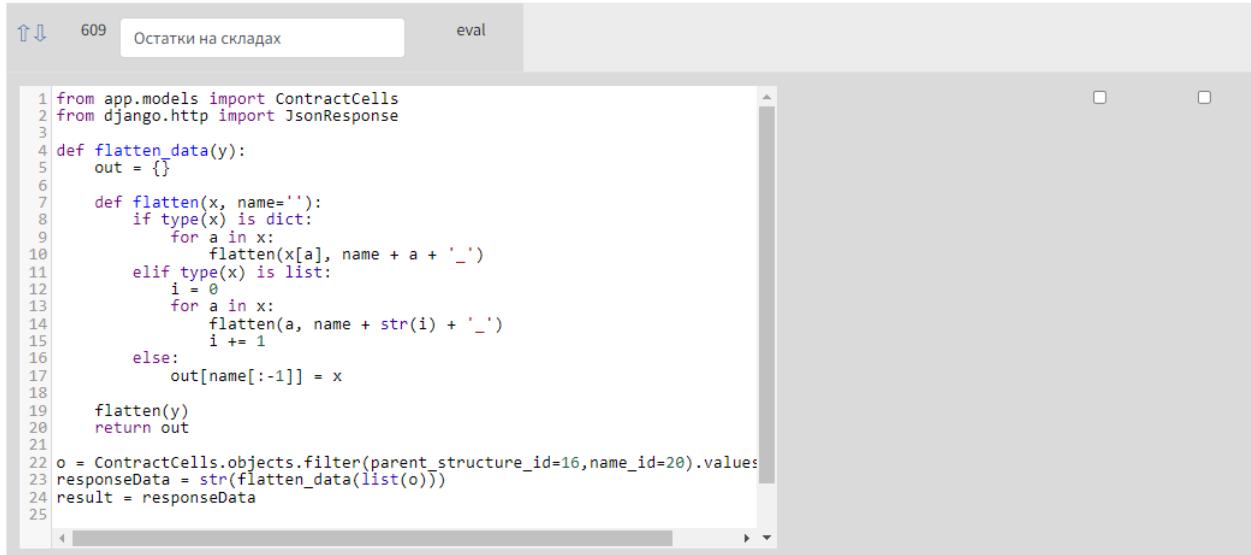
История изменений

Выберите период   ID регистратора  ID класса  Код Объекта

ID	reg ID	Имя регистратора	Обновлен	user ID	Пользователь	Данные
195	10	Редактирование параметра класса	21.06.2022 09:07:35	8	Алексей Титаренко	ходящие данные("id":450, "type": "table", "location": "table", "param": {"id": 757, "value": "# коменты"},\nfrom app.models import ContractCells\nn objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name') o = next o for o in objects if o.name_id == 20 o.value = int([452])  edit_objects = [] edit_objects.append(o)  nContractCells.objects.bulk_update(edit_objects, [\nvalue])  result = 'Ok'\n\n"p":\n"данные("id":450, "type": "table", "location": "table", "param": {"id": 757, "value": "# коменты"},\nfrom app.models import ContractCells\nn if [code] == 3:\n objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name')\n to = next o for o in objects if o.name_id == 20 \no.value = int([452])  edit_objects = [] edit_objects.append(o)  n ContractCells.objects.bulk_update(edit_objects, [\nvalue])  n result = 'Ok'\n else:\n result = 'not applicable'\n"\n\n

# Работа с формулами.

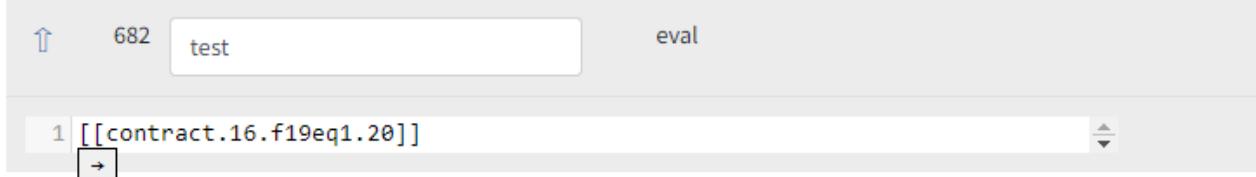
В поле справочника можно почти без ограничений использовать язык Python.



```
1 from app.models import ContractCells
2 from django.http import JsonResponse
3
4 def flatten_data(y):
5     out = []
6
7     def flatten(x, name=''):
8         if type(x) is dict:
9             for a in x:
10                 flatten(x[a], name + a + '_')
11         elif type(x) is list:
12             i = 0
13             for a in x:
14                 flatten(a, name + str(i) + '_')
15                 i += 1
16         else:
17             out[name[:-1]] = x
18
19     flatten(y)
20     return out
21
22 o = ContractCells.objects.filter(parent_structure_id=16, name_id=20).values()
23 responseData = str(flatten_data(list(o)))
24 result = responseData
25
```

Для подсветки кода и для удобства редактирования пользуемся библиотекой CodeMirror.

Добавили возможность перехода по ссылкам



```
1 [[contract.16.f19eq1.20]]
```

При наведении на ссылку макроподстановки обрамленную [[452]] – появляется стрелка с возможностью перехода на страницу контракта, объекта, откуда эта ссылка взята.

К примеру вот функция контроля ИНН — пример есть в справочнике «Юридические лица»

```
def inn_ctrl_summ(nums, type):
    inn_ctrl_type = {
        'n2_12': [7, 2, 4, 10, 3, 5, 9, 4, 6, 8],
        'n1_12': [3, 7, 2, 4, 10, 3, 5, 9, 4, 6, 8],
        'n1_10': [2, 4, 10, 3, 5, 9, 4, 6, 8],
    }
    n = 0
    l = inn_ctrl_type[type]
    for i in range(0, len(l)):
        n += nums[i] * l[i]
    return n % 11 % 10

def inn_check(inn):
    sinn = str(inn)
    nums = [int(x) for x in sinn]
    if len(sinn) == 10:
```

```

n1 = inn_ctrl_summ(nums, 'n1_10')
return n1 == nums[-1]
elif len(sinn) == 12:
    n2 = inn_ctrl_summ(nums, 'n2_12')
    n1 = inn_ctrl_summ(nums, 'n1_12')
    return n2 == nums[-2] and n1 == nums[-1]
else:
    return False

if inn_check([[578]]):
    result = 'Ok'
else:
    raise ValueError('ИИН некорректен')

```

Примечание: Если для формулы поставлена `is_required` (обяз.), то формула проверяется на ошибки до сохранения объекта. И объект не сохраняется если есть ошибка. Побочная проблема — Объект теряет информацию о том — что было исправлено, и приходится заново все набирать. Это решается методом `Draft` — подробнее в разделе Контракты

Отмечу — что для возврата результата используется конструкция `result = «Результат»` а для входящих параметров если это конкретные объекты пользуемся конструкциями типа `[[578]]` — две квадратные кавычки — со ссылками на реквизиты объектов справочника или контракта.

### **Временное решение для обработки задач связанных с разными расчетами на форме.**

Для таких вещей как расчет площади, или курсовых расчетов

Пока не работает режим `on_edit`, можно поступать так:

Создаем поле `Eval` и в нем пишем код, который возвращает на форму строку `html`

```

result = '<script> \
    let elem = document.querySelector("#ta_155"); \
    elem.addEventListener("input", function() { \
        var elem_t = document.querySelector("#ta_153"); \
        console.log(elem_t.value); \
        elem_t.value = elem.value; \
    }); \
</script> \
'

```

где `#ta_155` — это поле объекта которое будем слушать, а `#ta_153` — это поле в которое в данном примере будем дублировать значение.

Плохо — что такое поле (`div`) будет «мусорным» - его в таком же коде надо делать `hidden`, он не несет визуальной информации для пользователя. А также можно получить странный результат, если `on_view`, или `on_update` — имеют другую формулу вывода. В итоге — при просмотре будет один результат, при редактировании другой. Ну и возможный минус — в дублировании алгоритмов, один будет написан на `python`, а другой на `javascript`.

также можно делать более сложные запросы по базе данных

```
from app.models import ContractCells
```

```
from django.http import JsonResponse

o = ContractCells.objects.filter(parent_structure_id=16, name_id=20).values_list()
responseData = str(list(o))
result = responseData
```

Такой код выводит список всех контрактов складское хранение (ID – 16) поля количество (ID – 20).

но можно воспользоваться и языком формул — например суммирование всех объектов по количеству — из контракта можно записать так [[contract.16.All.20]]

А просуммировать остаток товара с кодом 1 — например так [[contract.16.f19eq1.20]] — в поле 19 — ссылка на товар, в поле 20 — количество.

Читается так — взять все контракты с кодом 16, с фильтром по полю 19(товар) eq(равным) 1 и просуммировать значения из поля 20.

Ошибки:

Есть в справочнике формула Сумма \* Количество с ID 895

```
result = [[773]]*[[774]]
```

При попытке сделать итог по сумме (По всем объектам справочника) — получаем ошибку  
[[table.661.all.895]]

В справочнике эти поля по ошибке завели как строковые, и парсер сейчас тоже не работает с вычисляемыми полями — пришлось сделать так

```
from app.models import Objects
```

```
import pandas as pd
```

```
import numpy as np
```

```
q_all = list(Objects.objects.filter(parent_structure_id=661, name_id__in=[773]).values('value'))
s_all = list(Objects.objects.filter(parent_structure_id=661, name_id__in=[774]).values('value'))
```

```
se = pd.json_normalize(q_all, max_level=3)
```

```
df = pd.json_normalize(s_all, max_level=3)
```

```
df['q'] = se.value
```

```
df['q'] = pd.to_numeric(df['q'], errors='coerce')
```

```
df['value'] = pd.to_numeric(df['value'], errors='coerce')
```

```
df = df.replace(np.nan, 0, regex=True)
```

```
df.eval('S = (value * q)', inplace=True)
```

```
result = df['S'].sum()
```

Но как временный обход — можно использовать.

Интерфейсные замечания. Перенести в описание справочников.

Также отмечу что порядком полей в справочнике также можно управлять — особенно если нужно разместить рядом реквизиты близкие по смыслу.

Управление параметрами справочника					
ID	Название	Тип	Код значения	По умолчанию	O T D
618	Наименование	string		<input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	
619	Тип оборудования	enum	1 Стационарное оборудование 2 Расходные инструменты 3 Движущаяся техника	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	
620	Инвентарный номер	string	PS:00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
621	Дата инвентаризации	datetime	дд.мм.гггг --::--	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
622	Состояние удовлетворительное	bool		<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
623	Остаточная стоимость	float		<input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	
Настройки планирования значений		Ответственный	8	Автоподтверждение задачи <input checked="" type="checkbox"/>	
624	Ост. стоим. в евро	eval	1 round([[623]]/[table.131.3.135],2)	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	
При просмотре					
625	Поставщик оборудования	link	Тип: Справочник ID: 576 Название: Юридические лица	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
635	Акт приемки	file		<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
<a href="#">Новый</a>		<a href="#">Сохранить</a>	<a href="#">Удалить</a>		

В левой части видны стрелки — вверх и вниз.

# Константы

Есть еще три объекта, которые нужно описать. Мы видели их ранее

The screenshot shows the 'Constructors' application interface. In the top navigation bar, there are links for 'Справочники' (Dictionary), 'Контракты' (Contracts), 'Задачи' (Tasks), and 'История' (History). On the right, it says 'Добрый день, Алексей Титаренко.' (Good day, Alexey Titarenko.), 'Users', and 'Выход' (Logout). Below the navigation, there's a breadcrumb trail: 'Конструктор справочников' (Dictionary Constructor) > 'Справочники' (Dictionary) > 'Черновики (1)' (Drafts (1)).

The main area is titled 'Управление классами' (Class Management). It has fields for 'ID' and 'Название' (Name). A dropdown menu for 'Тип' (Type) is open, showing options: 'Справочник' (Dictionary), 'Дерево' (Tree), 'Словарь' (Dictionary), 'Массив' (Array), 'Константа' (Constant), 'Каталог' (Catalog), and 'Справочник' (Dictionary) again, with 'Справочник' being the selected option.

To the left, there's a tree view titled 'Дерево справочников' (Dictionary Tree). It lists various objects with their IDs: 'Личные данные ID: 218', 'Контакты ID: 139', 'Email ID: 149', 'Телефоны ID: 152', 'Адресса ID: 156', 'Расчеты ID: 346', 'Деньги ID: 219', 'Дерево тестов ID: 253', 'Системные данные ID: 335', 'Фонды ID: 430', 'Клиенты и покупатели ID: 575', 'Издерки складские ID: 616', 'Складское оборудование ID: 617', 'ИТ инвентаризация ID: 656', 'Гран ID: 1172', 'новая конста ID: 361', 'поправочные коэффициенты ID: 822', and 'виды услуг ID: 919'. There are also '+' and '-' icons next to some entries.

At the bottom left, there's a section titled 'Условные обозначения' (Conditional Expressions) with three items:

- О - обязательный параметр (O - mandatory parameter)
- Т - параметр отображается в таблице (T - parameter is displayed in the table)
- Д - Delay. Использовать планирование значений для данного параметра (D - Delay. Use scheduling for this parameter's values)

Каталог и справочник — мы рассмотрели  
У нас еще есть Словарь, Массив, Константа

Начнем с конца.

Константы — это справочник с самостоятельными значениями, когда нужно сформировать список значений, с разными типами для дальнейших использований в формулах.

Например — в системе есть справочник валют, но нам нужен курс, который умножен на какой — то процент. Назовем это Алиас — вроде тот, же объект, можно воспользоваться и справочником, но константы поддерживают формулы.

Так вот алиасы (константы) будут выглядеть так

## Управление классами

ID	339
Название	Внутренние валюты
Тип	Константа
ID каталога	219
<b>Новый</b> <b>Сохранить</b> <b>Удалить</b>	

## Управление параметрами константы

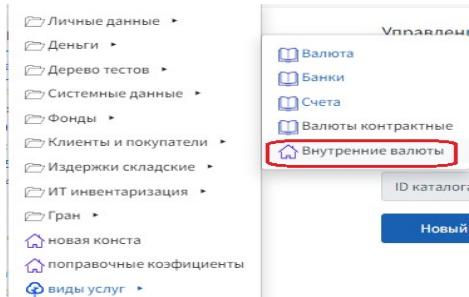
ID	Название	Формула
340	оптимистичный рубль	<code>[[contract.37.9.40]] / 1.8</code>
363	пессимистичный рубль	<code>[[contract.37.9.40]] / 1.5</code>

**Новый**

**Сохранить**

**Удалить**

## А в Справочнике



Константы отмечаются символом

И в справочниках имеет другую форму вывода

Константа "Внутренние валюты" ID: 339		
ID	Название	Значение
340	оптимистичный рубль	38.88888888888886
363	пессимистичный рубль	46.66666666666664

Из этой формы — добавить новую константу нельзя, только из режима конструктора.

Примечание:

Если нужны валюты типа Доллар + 1.5% - то формула получит вид  
`[[static.131.2.135]]*(1+1.5/100)`

Вообще константы — это таблица, в которой нужно держать значения, которые редко изменяются во времени и используются в разных справочниках или контрактах, чтобы не исправлять формулы каждый раз. А также константы доступны для добавления только в конструкторе справочника. **В дальнейшем пользователям (кроме администраторов) будет закрыт доступ в конструктор.**

# Массивы (array).

Массив в системе — это подчиненный справочник, со своей структурой, но с обязательным реквизитом «Собственник» (он же Parent). Собирается по тем же правилам, что и обычный Справочник (Reference book)...

Предназначен для оперативного доступа к элементам связанного справочника через интерфейс Собственника.

Например (новый дизайн)

The screenshot shows the 'Groups of characteristics' array interface. At the top, there is a navigation bar with links for 'Справочники', 'Контракты', 'Задачи', 'История', 'Добрый день, Алексей Титаренко.', 'Users', and 'Выход'. Below the navigation bar, the page title is 'Справочник "Группы характеристик" ID: 521'. There are search and export buttons: 'Поиск', 'Найти', 'Export list', and 'Export object'. A table lists items with columns 'Код' and 'Наименование'. Item 8 is selected ('Весы лабораторные'). On the right, a modal window shows the details for item 8, including a timestamp from 25.06.2023 11:48:25 to 25.07.2023 11:48:25, a code field (8), and a note field ('Весы лабораторные'). A sub-modal window titled 'Характеристики группы' displays two items: '1 Размер платформы (ШГ), мм' and '2 Диаметр платформы, мм'. At the bottom, there are buttons for 'Новый', 'Сохранить', 'В черновик', and 'Удалить'.

## Попадаем в массив

The screenshot shows the 'Characteristics of group' array interface. At the top, there is a navigation bar with links for 'Справочники', 'Контракты', 'Задачи', 'История', 'Добрый день, Алексей Титаренко.', 'Users', and 'Выход'. Below the navigation bar, the page title is 'Массив "Характеристики группы" ID: 563'. There are search and export buttons: 'Поиск', 'Собственник 8', 'Найти', 'Export list', and 'Export object'. A table lists items with columns 'Код', 'Собственник', 'Характеристика', and 'Основная'. Item 2 is selected ('Весы лабораторные'). On the right, a modal window shows the details for item 2, including a timestamp from 25.06.2023 13:18:16 to 25.07.2023 13:18:16, a code field (2), and a note field ('Весы лабораторные'). A sub-modal window displays the characteristic details: 'Характеристика' (4) and 'Основная'. At the bottom, there are buttons for 'Новый', 'Сохранить', and 'Удалить'.

Или что более часто используется для таких связанных справочников (старый дизайн):

Код	Наименование	Дата рождения	Актуальность	
4	контакт	dd.mm.yyyy	<input checked="" type="checkbox"/>	Наименование контакт
3	Паша техник	05.05.1995	<input checked="" type="checkbox"/>	Фамилия неизвестно
1	начальник космозо	12.12.1912	<input type="checkbox"/>	Имя неизвестно

Страница 1 из 10

Наименование	контакт
Фамилия	неизвестно
Имя	неизвестно
Отчество	
Дата рождения	dd.mm.yyyy
Комментарий	анонимный контакт. Для сделок без данных
Актуальность	<input checked="" type="checkbox"/>
Тест	2022-06-03

e-mail	Редактировать
Код	E-Mail
2	b@b1.com
1	a@a1.com

Через отмеченную кнопку — попадаем в интерфейс редактирования

Справочник "e-mail" ID: 637

Поиск	Собственник 4	Найти

Код	Собственник	E-Mail	Код	
2	контакт	b@b1.com	2	Собственник 4 контакт
1	контакт	a@a1.com		E-Mail b@b1.com

Страница 1 из 1 10

**Новый** **Сохранить** **Удалить**

Здесь видно — что обязательным реквизитом является «Собственник». И есть возможность вернуться в родительский элемент.

# Словари (Dictionary).

Название вытекло из описания: Словарь — это источник данных, информация в котором упорядочена с помощью разбивки на небольшие статьи (в нашем случае — на типизированные значения)

Словарь это составной реквизит Справочника. Требуется, когда для одного элемента Справочника нужно установить несколько разнотипных по структуре данных значений. Примером может служить — например данные о характеристиках.

The screenshot shows a list of items in a dictionary:

ID	Наименование	Проверка	Описание
34	ручка	<input type="checkbox"/>	его нет
33	кирпич	<input type="checkbox"/>	его нет
32	Dell-980	<input checked="" type="checkbox"/>	его нет
31	Acer-aspire 48 0	<input checked="" type="checkbox"/>	его нет IMG_2022042 8_122443.jpg
30	HP E-530	<input checked="" type="checkbox"/>	его нет

To the right, a detailed view of item #32 (Dell-980) is shown:

Категория	Параметр	Значение
общие характеристики	объем	0,002
	вес	0,02
	тип	хозя

Buttons: **Удалить**, **Добавить**.

В данном примере — есть элемент «ручка», у которой есть общие характеристики, но нет данных о компьютерных характеристиках. Но если понадобится ручка-ноутбук — пользователь может добавить характеристики и «общие характеристики», и «компьютеры» А вот для компьютера Dell-980 мы не указали общих характеристик, но нам были существенны характеристики для компьютера

The screenshot shows a list of items in a dictionary:

ID	Наименование	Проверка	Описание
32	Dell-980	<input checked="" type="checkbox"/>	его нет
31	Acer-aspire 48 0	<input checked="" type="checkbox"/>	его нет IMG_2022042 8_122443.jpg
30	HP E-530	<input checked="" type="checkbox"/>	его нет

To the right, a detailed view of item #32 (Dell-980) is shown:

Категория	Параметр	Значение
компьютеры	оперативка	12
	HDD	21
	videokarta	poi
	ноутбук	<input type="checkbox"/>

Buttons: **Удалить**, **Добавить**.

Структура словаря добавляется также как и справочники.

## Управление классами

ID	37
Название	компьютеры
Тип	Словарь
ID каталога	626
<b>Новый</b> <b>Сохранить</b> <b>Удалить</b>	

## Управление параметрами словаря

ID	Название	Тип	По умолчанию	Видим.
41	оперативка	float		<input checked="" type="checkbox"/>
42	HDD	float		<input checked="" type="checkbox"/>
43	videокарта	string		<input checked="" type="checkbox"/>
44	ноутбук	bool	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

**Новый**    **Сохранить**    **Удалить**

Добавление новых словарей ограничено — только Администраторы могут добавлять новые словари. Пользователь может только управлять видимостью доступных словарей.

# Справочник по формулам

Замечание для разработки: **В текущем релизе граф ссылок отключен.** Формула вычисляется один раз и хранится в структуре данных для быстрого выполнения запроса (данные кэшируются). За ее обновление отвечает граф ссылок. Он анализирует формулы на предмет изменения и хранится в отдельной структуре данных. Это позволяет снизить нагрузку на вычисления, и обновляется в то случае, если изменились данные или формула. **Также граф ссылок позволяет не допускать удаление реквизитов, если они используются в формулах, и обновлять кэш в связанных ссылками и формулами реквизитах.**

1. **Общий принцип работы вычисляемого поля.** Поле «eval» представляет собой код, обрабатываемый интерпретатором языка Python. В результате выполнения кода интерпретатор возвращает нам значение переменной «result».

Примеры:

Код	Результат
result = 3 * 8	24
result = 5 / 2 a = 3 + 4	2.5

Однако, не обязательно прописывать переменную «result». Если нам необходимо выполнить простое вычисление в рамках одной строки, то достаточно просто написать выражение для вычисления. Если программа не обнаруживает переменной «result» в коде, она добавляет ее в начале кода. Примеры:

Код	Результат	Примечание
int(10 / 3)	3	1. Итоговый код будет выглядеть так: result = int(10/3) 2. int – означает целое число. Можно использовать этот оператор для округления дробных чисел
10 + 5 11 - 8	15	Итоговый код будет выглядеть так: result = 10 + 5 11 — 8 Программа выведет нам значение переменной result, проигнорировав вторую строку.

Если в формуле будет допущена синтаксическая ошибка. То в результате будет возвращена строка, начинающаяся со слова «Ошибка», после чего будет указан вид ошибки.

Код	Результат	Примечание
10 / 0	Ошибка - division by zero	Ошибка очевидна: на нуль делить нельзя.
3 * 15%	Ошибка - invalid syntax (, line 1)	Здесь на первый взгляд неочевидно. Но не забываем, что речь идет об интерпретаторе Python. В

		Пайтоне нет операции взятия процента. Корректно написать будет так: $3 * 0.15$
--	--	--

Кроме исполнения кода на Python программа понимает собственные внутренние теги — FortCode. Это команды (или теги), заключенные в двойные квадратные скобки. Например: `[[57]], [[contract.369]]`. Их можно использовать совместно как со знаками арифметических действий (`[[57]] * [[contract.369]]`), так и в коде Python. Важно понимать приоритет задач. Система вначале вычислит значение всех форткодов, а затем выполнит общее вычисление. Разные варианты поведения форткода описываются при помощи атрибутов. Атрибуты представляют собой параметры, заключенные в двойные фигурные скобки внутри тега. Например: `[[ 57 {{version_number = 1}} ]]`

**2. Вызов значения поля.** Поле eval работает в контексте объекта, следовательно ему доступны все значения реквизитов данного объекта. Чтобы вызвать значение любого реквизита данного (текущего) объекта достаточно указать в форттеге его [заголовок](#).

Синтаксис: `[[ <header_id> ]]`, где:

`header_id` – ID заголовка класса.

Для примера создадим класс «Товары» в меню справочников со следующей структурой:

Управление параметрами справочника

ID	Название	Тип	Код значения	По умолчанию	O	T	D
17	Наименование	string			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
18	артикул	string			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
22	Категория	link	Тип: Справочник ID: 20  Название: Категории товаров		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
44	Цена	float			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Новый
Сохранить
Удалить

Предположим, нам необходимо создать вычисляемое поле «Оптовая цена». Для оптовых клиентов мы будем делать скидку 20%. Следовательно, чтобы получить оптовую цену достаточно указать следующее: `[[ 44 ]] * 0.8`

**Примечание.** Если мы укажем `[[ 18 ]] * 0.8`, то в результате будет ошибка, т. к. система не может умножить строку на число.

**3. Внутренняя ссылка.** В системе есть тип данных — ссылка. При помощи форткодов можно вытаскивать данные реквизитов объекта по ссылке. Для этого необходимо указать список заголовков через точку.

Синтаксис: `[[ <link_id>[.<link_field_id>,] ]]`, где:

`link_id` – ID реквизита текущего объекта. Тип данных у указанного реквизита должен быть link. В противном случае система вернет ошибку.

`link_field_id` – ID заголовка класса-родителя (по ссылке). Параметр взят в квадратные скобки чтобы показать его неограниченную вложенность.

Пример: Ранее мы указали класс-справочник «Товары». В нем есть ссылочный параметр «Категории товаров». Покажем класс-справочник, на который ссылается этот параметр:

## Управление параметрами справочника

ID	Название	Тип	Код значения	По умолчанию	O	T	D
21	Наименование	string			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
45	Артикул по ОКПД	enum	1 56 1000 2 56 1001 3 56 1100		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

[Новый](#)

[Сохранить](#)

[Удалить](#)

Допустим нам нужно, чтобы вычисляемое поле «код ОКПД» выводило по ссылке поле 45 «Артикул по ОКПД». Для этого необходимо в формуле набрать [[18.45]].

**Примечание.** Обратите внимание, что первое число — ID заголовка первого класса. Далее, проваливаясь в класс-родитель, мы указываем ID заголовка необходимого нам реквизита.

**Примечание 2.** Внутренняя ссылка работает только с типом данных — link. При попытке использовать ее с другим типом данных будет возвращать ошибку.

**Примечание 3.** Данный тип форткода поддерживает неограниченное количество родителей. Если все звенья, кроме последнего, будут типа данных link – то ссылка будет корректно выводить данные и делать с ними необходимые вычисления.

**4. Внешняя ссылка.** Этот тип форкода похож на предыдущий (внутренняя ссылка), с той разницей, что вне зависимости от текущего объекта будет вызван один и тот же реквизит указанного объекта. Ссылка указывается на стороннее поле стороннего объекта без связи с текущим объектом.

Синтаксис: [[ <location>.<class\_id>.<code>[.<field\_id>, ] ]], где:

location – Расположение объекта - «table», «contract» или «tp»;

class\_id – ID класса, целое число;

code — код объекта, целое число;

field\_id — ID поля, целое число.

Пример: [[table.1889.1562.8796 ]]. Снова рассмотрим класс «Товары» с ID 19. Если нам необходимо отображать цену товара с кодом 1234, то мы получим: [[ table.19.1234.44 ]]

**Примечание 1.** Конструкция [.<field\_id>, ] означает, что поддерживается неограниченная вложенность подобно внутренней ссылке.

**Примечание 2.** Если последнее звено цепочки из чисел указывает на число, строку, константу или формулу, то появляются следующие возможности работы с такими объектами:

4.1. Суммирование всех объектов класса. Если необходимо сложить (или выполнить другую агрегатную функцию) все объекты класса, то вместо кода объекта необходимо указать ключевое поле «all».

Пример: [[table.19.all.44]]. В данном случае система сложит цены всех товаров и вернет результат. Суммарная цена, конечно же, не имеет практической ценности. Но, как мы писали выше, мы можем выполнить и другие агрегатные функции, о чем подробней читайте в соответствующем разделе. Если нам необходимо получить самый дешевый товар в системе и посмотреть его цену, то для этого добавим optionalный параметр «Агрегатная функция», в итоге получим:  
[[ table.19.all.44 {{ agr\_fun = min }} ]]

4.2. Суммирование по фильтрам. Если необходимо сложить (или выполнить другую агрегатную функцию) какой-то реквизит не для всех объектов, а лишь для тех, которые удовлетворяют определенному условию (набору условий), то вместо кода

объекта мы будем использовать фильтр.

Синтаксис фильтра: фильтр начинается с ключевого символа «f» в поле «Код». Далее идет ID заголовка, знак сравнения, сравниваемое значение, точка с запятой, после которой вводим второе условие и т. д.

Знак сравнения представляет собой два символа. Приведем знаки и их расшифровку:

- eq – равно (equal)
- ne – неравно (non equal)
- gt – больше (great than)
- lt – меньше (less than)
- ge – больше или равно (great than or equal)
- le – меньше или равно (less than or equal)
- lk – содержит (like). Применяется к строкам. Регистронезависимая подстрока.

Сравниваемое значение может быть целым числом, строкой, датой, датой-временем или логическим значением(true / false).

Если условий несколько, то они могут быть соединены либо знаком «ИЛИ», либо знаком «И». По умолчанию используется знак «И». Подробности в п. 8.6 — Логический знак.

### Примеры.

Создадим новый справочник — Валюта. ID: 47. Пусть у него будет следующая структура:

ID	Название	Тип
48	Наименование	string
49	курс	float
50	код	string

Если нам необходимо получить средний курс всех валют от 50 до 500.

Результат: [[ table.47.f49ge50;44le500.49 {{agr\_fun=avg}} ]]

Если нам необходимо получить самый высокий курс валют, в названии которых встречается слово «доллар»: [[ table.47.f48lk'доллар';49 {{agr\_fun=max}} ]]

Подробный разбор агрегатных функций смотрите п. 8.5

Таблица знаков сравнения и типов данных, к которым они применимы

Знак	Типы данных
eq	Все
ne	Все
gt, lt, ge, le	float, date, datetime, eval, const
lk	String, eval (если в результате строка), const (если в результате строка)

Таблица агрегатных функций и поддерживающих их типов данных для выводимого в расчет поля

Функция	Типы данных
count	Все
Max, min	float, eval, const, date, datetime

Sum, avg	float, eval, const
----------	--------------------

**Работа с нулевыми или пустыми значениями.** Незаполненные поля — предмет неоднозначных подходов. Кто-то считает, что пустое числовое поле можно приравнять к нулю, а кто-то не согласен с этим. Когда мы фильтруем таблицу, то по умолчанию мы игнорируем пустые поля. Но булевое поле является исключением. Пустые поля по умолчанию приравниваются к заполненным со значением False. Если необходимо учесть пустое поле (мы знаем, что не все объекты заполнены), то необходимо в сравниваемое значение добавить «+null». Например, если нам необходимо посчитать, сколько имеется валют с курсом ниже 30, включая незаполненные значения курсов, то формула будет иметь вид:

```
[[ table.47.f49lt50+null.49 {{ agr_fun=count }} ]].
```

Если нам необходимо вывести только пустые, или наоборот, только непустые поля, то используем конструкцию «eqnull / nenull». Пример: Посчитаем валюты, у которых задан курс: [[ table.47.f49eqnull.49 {{ agr\_fun=count }} ]].

**5. Короткая внешняя ссылка.** Этот форткод аналогичен предыдущему, но используется в том случае, когда нам известен ID реквизита объекта — т. е. Место его непосредственного размещения в базе данных. Из интерфейса системы его нельзя получить, т. е. пользователь должен иметь доступ непосредственно к базе данных. Ссылка состоит из двух компонентов — ключевого слова — 'table' или 'contract' – и ID одной из таблиц БД - app\_contractcells или app\_objects

Синтаксис ссылки: [[ <location>.<cell\_id>]], где:

location – расположение объекта. Значения: 'table' или 'contract';

class\_id – ID ячейки, целое число.

Пример: [[table.392]].

**6. Ссылка на константу.** Константа — это одна из разновидностей классов в системе. Она представляет собой набор внешних ссылок. Но иногда есть необходимость сделать ссылку на константу. Этот вид ссылки отличается от предыдущих тем, что форткод состоит из трех составляющих.

**Синтаксис:** [[ <location>.<const\_id>.<header\_id> ]]

- 1) location – локация константы — table, contract.
- 2) const\_id – ID константы.
- 3) header\_id – ID параметра константы.

**Пример:** У нас есть константа с ID 8. Она состоит из следующих реквизитов:

ID	Название	Формула
9	Какой сегодня день недели	
1	<pre>from datetime import datetime dow = datetime.today().strftime('%A') dict_days = {'Monday': 'Понедельник', 'Tuesday': 'Вторник', 'Wednesday': 'Среда', 'Thursday': 'Четверг', 'Friday': 'Пятница', 'Saturday': 'Суббота', 'Sunday': 'Воскресенье'} result = dict_days[dow]</pre>	
10	Который час	<pre>from datetime import datetime result = datetime.strftime(datetime.today(), '%H:%M:%S')</pre>
12	Лучший регион	<pre>objs = [[table.4.f11eq[[table.4.all.11 {{agr_fun = 'max' }} ]]]] result = objs[0][5]['value']</pre>

[Новый](#) [Сохранить](#) [Удалить](#)

Если мы хотим вывести ссылку на формулу «Лучший регион», то получим:  
[[ table.8.12 ]]

**7. Ссылка, возвращающая массив данных.** Все предыдущие ссылки возвращали одно значение. Эта же ссылка возвращает JSON-объект в виде списка словарей (в терминологии языка Python).

Синтаксис ссылки: [[ <location>.<class\_id>.params]], где:

location – расположение объекта — table или contract.

class\_id – ID класса, список объектов которого нам необходимо получить.

params – список параметров, фильтрующий выводимые объекты.

Синтаксис полностью аналогичный параметрам статической ссылки (см п. 4.1, 4.2).

Формула поддерживает следующие опциональные параметры (см. п. 8): logical\_sign, branch, cluster, order, lifo, version, version\_after.

Также форткод работает с группами опциональных параметров, например: {{logical\_sign='or' version='20.02.2023' branch }}. В указанном примере данные будут отфильтрованы с применением логического «или» между фильтрами, объекты отберутся только те, что принадлежат текущей ветке, при этом будет показана не текущая версия объектов, а версия по состоянию на 20.02.2023 00:00:00.

Не поддерживается связка опциональных параметров version version\_after + order lifo.

Т.е. данные истории не сортируются. Это придется делать вручную после получения массива.

Результат использования: Каждый словарь из списка содержит три обязательных поля — code, parent\_structure, type. Это, соответственно, код объекта и ID класса и тип класса. Остальные поля в ключе содержат ID заголовка, а в значении — словарь с данными о характеристиках реквизита. Пример результата:

```
[{"code": 3, "parent_structure": 131, "type": "table", "id": 394, "name": "Наименование", "type": "string", "value": "Евро"}, {"code": 3, "parent_structure": 131, "type": "table", "id": 395, "name": "Символьный код", "type": "string", "value": "EUR"}, {"code": 3, "parent_structure": 131, "type": "table", "id": 396, "name": "Цифровой код", "type": "float", "value": 978.0}, {"code": 3, "parent_structure": 131, "type": "table", "id": 397, "name": "Курс", "type": "float", "value": 76.9564}, {"code": 3, "parent_structure": 131, "type": "table", "id": 398, "name": "Дата обновления", "type": "datetime", "value": "2023-02-03T03:31"}, {"code": 3, "parent_structure": 131, "type": "table", "id": 527, "name": "коэффициент", "type": "float", "value": 1.1}, {"code": 3, "parent_structure": 131, "type": "table", "id": 528, "name": "Внутренний курс", "type": "eval", "value": 84.65204000000001}, {"code": 2, "parent_structure": 131, "type": "table", "id": 389, "name": "Наименование", "type": "string", "value": "Доллар США"}, {"code": 2, "parent_structure": 131, "type": "table", "id": 390, "name": "Символьный код", "type": "string", "value": "USD"}, {"code": 2, "parent_structure": 131, "type": "table", "id": 391, "name": "Цифровой код", "type": "float", "value": 840.0}, {"code": 2, "parent_structure": 131, "type": "table", "id": 392, "name": "Курс", "type": "float", "value": 70.0414}, {"code": 2, "parent_structure": 131, "type": "table", "id": 393, "name": "Дата обновления", "type": "datetime", "value": "2023-02-03T03:31"}, {"code": 2, "parent_structure": 131, "type": "table", "id": 529, "name": "коэффициент", "type": "float", "value": 0.7}, {"code": 2, "parent_structure": 131, "type": "table", "id": 530, "name": "Внутренний курс", "type": "eval", "value": 49.02898}], ]
```

**8. Опциональные параметры.** Некоторые форткоды имеют вариативное поведение, которое описывается при помощи опциональных параметров. Опциональные параметры помещаются внутри форткода после его основной части и заключаются в двойные фигурные скобки. Синтаксис опциональных параметров рассмотрим на примере:

Пример: [[contract.135.84.363 {{version='2022-03-27'}}]]

Как видно из примера в двойных фигурных скобках указываем название опционального параметра ставим знак «равно» а затем значение параметра.

**Примечание.** Если форткод поддерживает несколько опциональных параметров, то их необходимо размещать в одних скобках без знаков препинания, разделяя пробелом или переводом строки.

**8.1 «version».** Указывая этот параметр, система возьмет не текущее значение реквизита объекта, а значение из прошлого на момент даты и времени, указанных в параметре. Общий синтаксис параметра {{ version = <timestamp> }}, где:

timestamp – дата и время в строковом формате. Параметр принимает как формат даты-времени, так и даты; как в формате ISO, так и в российской локали. Дату-время (значение параметра) обязательно заключать в кавычки.

Пример: Ниже указана дата и время во всех доступных системе форматах — 8 марта 2022 г 15:47:23

'2022-03-08T15:47:23', '2022-03-08 15:47:23', '08.03.2022T15:47:23', '08.03.2022 15:47:23'.

Также система в качестве значения параметра принимает дату. Время в этом случае автоматически ставится «00:00:00». Примеры даты: «2022-08-03», «03.08.2022»

Примеры использования:

[[146 {{ version='2021-08-16'}}]] - реквизит текущего объекта

[[ table.131.2.135 {{ version='05.01.2020T16:01:00'}}]] – реквизит внешнего внешнего объекта

**Примечание.** Поскольку тип данных eval не протоколируется в истории, то указанный параметр для типов данных eval вернет ошибку. Также не рекомендуем применять этот параметр для типа данных — const. Он вернет верное значение

параметра константы. Но расчитываться формула константы будет по современным значениям, т. е. Данные, возможно, будут неактуальными.

**8.2 «version\_after».** Полностью идентичный параметру version, с той лишь разницей, что система возьмет первое состояние объекта после указанной даты. Синтаксис и примеры применения смотрите в п.8.1 version.

**8.3 «version\_number».** Параметр, аналогичный п.8.1 и 8.2. Показывает состояние объекта на указанное количество изменений назад. В качестве значения параметра указывается целое число. Если мы указали число 2, значит формула вернет значение реквизита два изменения назад.

**Примеры:**

```
[[146 {{ version_number=2 } } ]]  
[[ table.131.2.135 {{ version_number = 4 } } ]]
```

**8.4 date\_to, date\_from.** Если Вам необходимо вычислить агрегатную функцию не ряда объектов, а набора значений одного и того же реквизита объекта в заданном временном интервале, то используйте эти optionalные параметры. Например, можно найти максимальную цену товара в заданном промежутке времени.

Синтаксис: [[ <header\_id> {{ date\_from='<datetime1>' date\_to = '<datetime2>' } } ]], где: header\_id – ID заголовка. Поддерживается как заголовок текущего объекта, так и заголовок внешнего объекта.

Datetime1, datetime2 – дата и время в строковом формате. Подробное описание формата смотрите в п. 8.1 — Версия.

Пример: Рассмотрим созданный ранее справочник «Товары». Предположим, нам необходимо видеть среднюю цену товаров за 2023 год:

```
[[ 34 {{ date_from='2023-01-01' date_to='2024-01-01' agr_fun=avg } } ]]- форткод для поля внутри класса «Товары»  
[[ table.16.1.34 {{ date_from='2023-01-01' date_to='2024-01-01' agr_fun=avg } } ]]- форткод для получения средней цены товара с кодом 1 за 2023 год.
```

**8.5 «agr\_fun».** Вычисляет агрегатную функцию по отношению к реквизиту или группе реквизитов. Система поддерживает все основные агрегатные функции — сумма, среднее арифметическое, максимум, минимум, счет.

Синтаксис параметра: {{ agr\_fun = <a\_fun> } }, где:

a\_fun – название агрегатной функции. Ее значение должно быть одним из следующего списка: [sum, avg, max, min, count]. Если параметр не указан, по умолчанию вычисляется сумма.

**Примеры:**

Рассмотрим пример со справочником «Валюта», описанным в п. 4. Допустим нам необходимо вычислить минимальный курс валюты с кодом 2 за период 2022 года.

```
[[ table.47.2.49 {{ date_from='01.01.2022' date_to='01.01.2023' agr_fun=min } } ]]
```

**8.6. «logical\_sign».** Если Вы используете форткод для суммирования по фильтру, то по умолчанию все условия объединяются логическим «И», т. е. фильтрация осуществляется по выполнению всех условий. Однако иногда необходимо выполнение любого из условий фильтраций, т. е. связующее логическое «ИЛИ». Для этого используется optionalный параметр logical\_sign

В этом случае и применяем этот параметр. У него может быть два значения [or, and].

Пример: [[ table.156.f160ge70;161le4.165 {{logical\_sign=or} } ]]

**8.7. «branch».** Используется в работе с деревьями. В параметре дерева укажите суммирующую формулу, либо дополните ее агрегатной функцией. В суммирующей

формуле укажите подчиненный класс, а в optionalном параметре пропишите ключевое слово - «branch». Теперь для каждой ветки этого дерева будет выполняться указанная Вами агрегатная функция (суммирование, подсчет количества и пр.) не всех объектов указанного класса, а только являющихся дочерними для данной ветки. Рассмотрим пример. Создадим структуру данных из дерева «Виды услуг» и подчиненного справочника «Услуги» как показано на рисунке ниже.

- виды услуг ID: 919

 услуги ID: 836

Для дерева создадим вычисляемое поле с формулой:

`[[table.836.all.992 {{branch agr_fun=count}} ]]`

На странице управления ветками или объектами теперь будет видно, сколько услуг приходится на каждую ветку.

Ветка	комментарий	количество услуг
-  0 виды услуг 		
-  2 Основные 		
 9 скидки 		
 10 срочные 		
 12 внутренние 	код - 00	1
-  19 Дополнительные 		1
 6 Одноразовые 		1
 20 прочее 	код - 33	3

8.8. «cluster». Кластером называется ветка со всеми дочерними ветками. Параметр очень схож по смыслу с параметром branch (см. п. 8.7), но в отличие от него агрегатная функция будет вычисляться не только для дочерних объектов данной ветки, но и для дочерних объектов всех дочерних ветвей по отношению к текущей ветке. Если мы рассмотрим пример из пункта 8.7, но в качестве optionalного параметра укажем cluster, то результат будет отличаться: `[[table.836.all.992 {{cluster agr_fun=count}} ]]`

Ветка	комментарий	количество
		услуг
- <input type="checkbox"/> 0 виды услуг <a href="#">/</a>		
- <input type="checkbox"/> 2 Основные <a href="#">/</a>		1
<input type="checkbox"/> 9 скидки <a href="#">/</a>		
+ <input type="checkbox"/> 10 срочные <a href="#">/</a>		
<input type="checkbox"/> 12 внутренние <a href="#">/</a>	код - 00	1
- <input type="checkbox"/> 19 Дополнительные <a href="#">/</a>		5
+ <input type="checkbox"/> 6 Одноразовые <a href="#">/</a>		1
<input type="checkbox"/> 20 прочее <a href="#">/</a>	код - 33	3
Код	Наименование	колво
5	доставка	4

Ветка 19 «Дополнительные» показывает количество услуг, равное 5. У нее есть собственная дочерняя услуга, а также одна услуга дочерней ветки 6 «Одноразовые» и три услуги дочерней ветки 20 «Прочее».

8.9 «**order**». Упорядочивает данные в массиве по заданным параметрам. Работает с формулой, возвращающей массив данных.

Пример использования `{ { order = 11+, 14, 45- } }`, где:

`11+, 14, 45-` - ID заголовков параметров, по которым будет выполняться упорядочивание. Если после числа идет знак «+», значит сортировка будет идти от меньшего значения к большему. Если знак «-» - наоборот, от большего к меньшему. Если не указать знак, то по умолчанию сортировка выполняется по возрастанию. Если параметров больше одного, значит сортировка выполняется только для тех объектов, которые по предыдущему параметру уже отсортированы и занимают одну позицию.

8.10 «**lifo**». Работает с формулой, возвращающей массив данных. Разворачивает массив, сортируя его т.о. от новых к старым. Синтаксис `{ {lifo} }`

9. Условие. В системе есть функция, выполняющая вычисление или отображающая какие-либо данные в зависимости от выполнения или не выполнения условия. Ее синтаксис схож с аналогичной функцией в MS Excel. Функция «условие» - не является форт-кодом, т. е. заключать в двойные кавычки ее не следует.

В общем виде он выглядит следующим образом:

`if(<условие>; <истина>;<ложь>)`

где:

`<условие>` - блок условия. Оно состоит из логического выражения. Общая схема логического выражения:

`<левая часть> <знак> <правая часть>`

где:

`<левая часть>, <правая часть>` - число, строка, арифметическое или строковое выражение.

**<знак>** - один из следующих знаков: `>`, `<`, `=`, `<>`, `<=`, `>=`. Знаки означают соответственно: больше, меньше, равно, неравно, больше либо равно, меньше либо равно.

Примеры блоков условий:

```
a + b = 10  
[[table.3975]] < [[table.3977.1485]] * 1,5  
[[contract.14598.468.45]] <> 'много'
```

Как видно из примеров, в условиях можно использовать ссылки на соседние поля, ссылки на внешние ключи, статические ссылки.

Блок условия обязательно должен следовать указанной схеме, т. е. Состоять из трех составляющих — левой, правой частей и знака между ними. В случае отсутствия какой-либо составляющей блока условий на выходе получим ошибку вида «Ошибка синтаксиса условия»

**<истина>** - возвращаемая часть при выполнении условия. Она может быть числом, строкой, арифметическим выражением. Примеры:

```
a * b  
[[table.3975]]  
[[table.3977.1485]] / 1,5  
[[contract.14598.468.45]]  
'Работает'  
'В ремонте'
```

Обращаю внимание, если используем строковые данные — то наличие кавычек обязательно. В противном случае функция выдаст ошибку.

**<ложь>** - возвращаемая часть при невыполнении условия. Формат и примеры аналогичны блоку «Истина».

Примечание. Вложенные условия не поддерживаются.

10. Получение кода текущего объекта `[[code]]`.
11. Получение ID текущего класса — `[[class_id]]`.
12. Получение ID текущего пользователя — `[[user_id]]`
13. Получение текущего класса в контексте родительского дерева — `[[child_class]]`
14. Получение уровня вложенности объекта - `[[ level ]]`. где 0 — корневая папка.
15. Формулы с пользовательскими данными. Иногда при вычислении формулы могут потребоваться какие-либо пользовательские данные от пользователя. При работе с объектами это не требуется. Однако при работе с константами такая необходимость может возникнуть, например, если мы используем константу для вывода отчета, то данные для фильтрации результатов обязательно понадобятся. После ввода данных пользователю предлагается нажать на кнопку, расположенную там же в форме ввода данных. После нажатия кнопки под формой ввода появится результат вычисления.

Для этого используется форт-код: **`user_data`**.

Синтаксис: `[[ user_data_<number> {{ <optional params> }} ]]`,

где:

**number** – номер пользовательской переменной. Необходимо указать произвольное целое положительное число. Главное, чтобы эти числа не повторялись в рамках одного поля eval.

**optional params** – дополнительные опциональные параметры:

**label** – наименование переменной. По умолчанию пользователь увидит сообщение вида: «Пользовательская переменная № `<number>`».

Синтаксис: `label = '<label_name>'`, где `label_name` – непосредственно наименование переменной. В качестве наименования необходимо указать строку из чисел, символов, пробелов или знака `«_»`. Обращаем внимание, что наименование переменной необходимо заключить в кавычки.

**type** – тип данных. Форт-код понимает следующие типы данных: (string, number, bool, date, datetime, link, table). По умолчанию принимается тип «string». В зависимости от указанного типа пользователю для ввода данных будет показан тег `input` с соответствующим типом тела. Например, если указан тип данных `checkbox`, то на экран будет выведен тег `<input type=checkbox>`. Это упрощает ввод данных, снижает возможность допущения ошибки пользователем.

Синтаксис: `type = <data_type>`, где `data_type` – непосредственно указанный тип.

Необходимо ввести одно из указанных значений из списка выше. Значение указывается без кавычек.

Если Вы указали тип данных `link`, то необходимо добавить еще один опциональный параметры — `link_class` и `link_location`.

`link_class` – ID класса (целое число), на который ведет ссылка.

`link_location` – расположение класса — контракты или справочники (символ «t» или «c»).

`clean` – параметр, отвечающий за самоочистку поля ввода. Укажите его среди параметров, если Вам нужно, чтобы данное поле очищалось после нажатия кнопки внизу формы ввода.

Пример. Если необходим форткод, выводящий форму для ввода ссылки на описанный ранее справочник с ID 45, то его формула будет выглядеть так:

```
[[ user_data_1 {{ type = link link_class = 45 link_location = t }} ]]
```

**button** – заголовок кнопки, при нажатии которой производится вычисление формулы с пользовательскими данными. По умолчанию система подпишет кнопку «Рассчитать». Синтаксис параметра: `button = '<button_name>'`, где `button_name` – непосредственный заголовок кнопки. Разрешается вводить символы, числа, пробелы и нижнее подчеркивание.

Пример 1: Снова рассмотрим класс «Товары» с ID 19. Допустим, нам необходимо отображать цену товара с кодом, который укажет пользователь. Напишем подсказку для пользователя «Укажите код товара», также уточним, что тип данных будем вводить числовой (чтобы лишить пользователя возможности ввести символы, отличные от чисел).

Кроме этого подпишем кнопку «Вывести». Формула приобретет следующий вид:

```
[[ table.19.[[user_data_1 {{ label = 'Укажите код товара' type = number button = 'Вывести' }} ]].44 ]]
```

Обратите внимание, форт-код поддерживает переводы строк. Это удобно при использовании большого количества параметров.

**Табличный тип данных.** Иногда возникает необходимость работать с большим массивом данных неограниченной длины. Нам может быть неизвестно, сколько именно переменных пользователь введет. Для этого подходит пользовательский форткод с типом `table`. Как работать с этим типом данных:

1. В параметрах укажите тип `table`: `type=table`

2. Добавьте параметр `structure`, в котором опишите структуру Вашей таблицы.

3. Синтаксис параметра `structure`. `structure = [ {name = '<name>' type=<type> clean} ]`, где: `name` – наименование столбца таблицы. Если не указать, то по умолчанию будет отображать «Заголовок №...». `<name>` - значение наименования. Поддерживает буквы, цифры, пробелы, знак подчеркивания. Значение наименования обязательно заключать в одинарные или двойные кавычки.

`type` – тип данных. Внутри таблицы форткод работает со всеми типами данных, описанных выше. Для типа данных `link` также работают дополнительные два параметра `link_class` и

link\_location.

clean – параметр самоочищения поля ввода.

4. Выводимый результат. Система после обработки данных форкод заменит двумерным списком.

Пример. Вывести для пользователя форму предварительного расчета заказа. Поля: ID товара, количество, скидка.

```
my_table = table = [[user_data_1 {{label = 'Предзаказ' type = table structure = [
    {name = 'ID товара' type = link link_location = t link_class = 1}
    {name = 'Количество' type = number} {name = 'Скидка' type = bool}
    ]}}]]
```

Внешний вид такой таблицы:

Предзаказ		
ID товара	Количество	Скидка
		<input type="checkbox"/>
<a href="#">+</a>		

[Рассчитать](#)

Пример:

Изначально таблица покажет одну строку. Кнопка в левом нижнем углу позволяет добавлять дополнительные строки. Заполним таблицу тремя строками.

Предзаказ		
ID товара	Количество	Скидка
1	4	<input type="checkbox"/>
2	5	<input checked="" type="checkbox"/>
3	6	<input type="checkbox"/>
<a href="#">+</a>		

[Рассчитать](#)

**16. Состояние выводимых данных по отношению ко времени.** Мы выделяем три состояния данных: Общее, фактическое и отложенное. Или по английски: state, fact, delay. Фактическое — это текущее состояние параметра на данный момент. Отложенное — это значение параметра объекта, уже известное сейчас, но еще не вступившее в силу. Общее — это сумма фактического и отложенное. Общее состояние можно назвать будущим.

Т.о. общая парадигма состояний выражается формулой state = fact + delay.

Система позволяет выводить любое состояние объекта. Синтаксис следующий:

[[<param>.<status>]], где:

**param** – это данные, параметр объекта, выводимого форт-кодом. Здесь можно указать внутренний параметр, в т.ч. внутреннюю ссылку; либо внешний параметр объекта.

**status** – это одно из состояний. Если его не указать, то поумолчанию будет показывать общее состояние (state). В большинстве свойств большинства объектов, как правило отключены отложенные значения. В этом случае общее состояние равно фактическому. status может принимать одно из четырех значений: [state, fact, delay, delta], где:

**delta** – разность между текущим фактическим (fact) значением и предыдущим фактическим состоянием. Дельта работает только с одним типом данных — с числовым (float). Можно также выводить дельту значения, работая с историческими данными, т. е. Дельта корректно работает в комбинации с опциональными параметрами version, version\_after и version\_number.

**Пример:** Рассмотрим объявленный ранее пример с классом 131 — Валюты с параметром «Курс» id = 135. Валюта «доллар США» имеет код 2. Допустим на сегодняшний день курс доллара равен 100 рублей. Но в системе уже есть информация о новом курсе величиной в 95, вступающим в силу завтра в 00:00:00. Также нам известно, что вчера курс доллара был 103 рубля.

Формула	Результат
[[ table.131.2.135.fact ]]	100
[[ table.131.2.135.delay ]]	-5
[[ table.131.2.135.state ]]	95
[[table.131.2.delta]]	-3

Также работает и с внутренними параметрами. Добавим в класс «Валюта» два вычисляемых поля. В первом покажем, насколько изменился курс валюты со вчерашнего дня, а во втором, насколько изменится курс к завтрашнему дню.

В первом поле нам необходимо указать [[135.delta]]. Во втором - [[ 135.delay ]]

Если нам необходимо знать, насколько изменилась валюта неделю назад, то воспользуемся опциональным параметром version\_number: [[135.delta {{version\_number = 7}}]].

Аналогично для вызова снаружи [[ table.131.2.135.delta {{ version\_number = 7 }} ]].

В результате нажатия на кнопку «Рассчитать» переменная table будет равна следующему списку:

table = [[1,4,False],[2,5,True],[3,6,False]]

Далее админ системы обрабатывает полученные данные в соответствии со своими задачами. Напомню, на экран всегда выводится содержимое переменной result.

### Управление параметрами справочника

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
1004	Наименование	string		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
1005	Файл	file		<input type="checkbox"/>	<input type="checkbox"/>	
1006	тест линк	eval	1 [[1005]] При просмотре ▾		<input type="checkbox"/>	<input type="checkbox"/>
1007	Обработчик файлов	eval	1 filename = [[1005]] 2 with open(filename, "r+") as f: 3     data = f.read() 4 result = data При просмотре ▾		<input type="checkbox"/>	<input type="checkbox"/>
1008	Скрипт	eval	При просмотре ▾		<input type="checkbox"/>	<input type="checkbox"/>
			1 result = "<a href=[[1005]]> <img src=[[1005]] width='500' height='600' /> </a>"		<input type="checkbox"/>	<input type="checkbox"/>

Новый

Сохранить

Удалить

Результат будет такой

Версия 24.01.2023 13:36:46 Алексей Титаренко

Код 3

Наименование \* Тест файлов

Файл 8195134 (1).jpg Обзор c: Скачать Удалить

тест линк database\_files\_history/table\_1003/2023-01-24/240120231336468195134 (1).jpg

Обработчик файлов Ошибка - 'utf-8' codec can't decode byte 0xff in position 0: invalid start byte

Скрипт

```

class df {
    public static void main() {
        String str = "Hello, World!";
        System.out.println(str);
    }
}

public class Main {
    public static void main() {
        System.out.println("Hello, World!");
    }
}

```

## Пример работы с html . Добавим несколько справочников



### рассмотрим справочник JS

Справочник "js" ID: 879

Поиск	Найти	Export list	Export object
<b>Код</b>	<b>Наименование</b>		
3	utils		
2	script		
1	map		

Страница 1 из 1 [10]

Версия 14.10.2022 12:17:16 Алексей Титаренко

Код 3

Наименование \* utils

js utils.js Обзор c: Скачать Удалить

Новый Сохранить В черновик

Подгружаем файлы нужные для html шаблона. JS, CSS, HTML. (Пример нужен другой, на релизе заменить в документации )

В справочнике, где нужно использовать внешнюю форму делаем так.

Управление классами

ID	888
Название	test_html
Тип	Справочник
ID каталога	878

**Новый** **Сохранить** **Удалить**

Управление параметрами справочника

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
889	Наименование	string		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
890	run	eval	<input type="button" value="При просмотре"/>			
<pre>1 filename = "database_files_history/table_885/2022-10-14/14102022122719index.html" 2 with open(filename, "r+") as f: 3     data = f.read() 4 result = data</pre>						

**Новый** **Сохранить** **Удалить**

При получении страницы будет такой пример.

Версия	14.10.2022 12:21:35 Алексей Титаренко
Код	1
Наименование	тест

Способ получения товара

**Самовывоз** **Доставка по городу**

**Самовывоз**

Первомайская, 42 Покровка, 31 Затонная, 2 Обручева, 30А

При работе с шаблоном — нужно пути к js, css переопределить на ссылки файлов внутри системы. Пути формируются платформой. Получить его можно через интерфейс кнопкой копирования ссылки, или программно через `[[“номер реквизита”]]`

Пример пути: « database\_files\_history/table\_885/2022-10-14/14102022122719index.html »

**Пример плохой формулы** срабатывающего On\_view (при загрузке контента на страницу) в данном случае мы берем данные из текущего объекта с кодом 3 и устанавливаем в контракте 16 (складское хранение) в элементе с кодом 3, в поле 20 — значение из поля 452 текущего класса.

```
from app.models import ContractCells # модуль контрактов
if [[code]] == 3:
    objects = ContractCells.objects.filter(code=3, parent_structure_id=16).select_related('name')
    o = next(o for o in objects if o.name_id == 20)
    o.value = int([[452]])
    edit_objects = []
    edit_objects.append(o)
    ContractCells.objects.bulk_update(edit_objects, ['value'])
    result = 'Ok'
else:
    result = 'not applicable'
```

Такое редактирование - «грязное», оно не вызывает записи в истории. Следовательно — такие «финты» нужно избегать. Для корректной работы с объектами в редакторе кода используйте специальные функции: `create_object`, `edit_object`, `remove_object`.

## Функция `create_object`

**Назначение:** Создание объекта и сохранение в истории сведений о созданном объекте.  
Интерфейс вызова:

```
from app.functions.api_funs import create_object
new_object = create_object(class_id, user_id, location='table', source=None, **params)
```

где:

**class\_id** – ID класса создаваемого объекта.

**user\_id** – ID пользователя, под которым будут сохранены эти данные. Рекомендуется передавать значение `request.user.id`.

**location** – физическое расположение объекта в базе данных. Все объекты расположены либо в справочниках, либо в контрактах, либо в словарях. Соответственно в параметр можно передавать только строковые значения «table», «contract» или «dict». Другое значение вернет сообщение об ошибке. Если параметр не задан, то по умолчанию (как видно из интерфейса) система примет значение «table».

**source** – источник. Принимает словарь с тремя ключами 'class\_id', 'code', 'location'. Необходим для отслеживания источников работы с объектами из истории. Если параметр не задан, то по умолчанию передает значение None.

**\*\*params** – параметры. Словарь с неограниченным количеством пар. В ключах передаются ID параметров класса. Их можно посмотреть в конструкторе справочников / контрактов. **ВАЖНО!!!** Ключи — числовые значения, но передавать их необходимо в виде строк, т. е. заключать ключи в кавычки. В значениях указываем непосредственно значения полей.

Также в словарь «params» передаются следующие параметры:

**owner** – владелец. Обязательный параметр для типов данных «array» и «dict». В значении необходимо указать код владельца (родительского объекта).

**parent** – родитель. Указывается для объектов, подчиненным деревьям. В значении

необходимо указывать код родительской ветки дерева.

**timestamp** – таймштамп (данные о дате и времени) совершения операции. Если проигнорировать параметр, то по умолчанию передастся текущий момент.

**parent\_transact** – ID родительской транзакции, которая вызывает данное изменение. Если таковой транзакции нет, то по умолчанию значение остается пустым.

Например:

```
params = {'163': 12, '199': True, '228': '8-992-357-96-87'}
```

**Примечание 1.** Для создания объекта словаря или массива необходимо указать обязательный параметр `owner`. Он указывает код объекта-родителя. Например:

```
params = {'163': 12, '199': True, '228': '8-992-357-96-87', 'owner': 116}
```

**Примечание 2.** При создании объекта можно передать отложенное значение реквизита, поддерживающего отложенные значения. К ID реквизита добавим `«_delay»`. Для корректной работы отложенного значения обязательно также передать дату выполнения отложенного значения. Без указания этой даты отложенное значение будет проигнорировано и не добавлено в объект. Для передачи даты выполнения добавьте к параметру `«_delay_date»`. Дату необходимо указать в одном из следующих форматов: YYYY-mm-dd, YYYY-mm-ddTHH:MM, YYYY-mm-dd HH:MM, dd.mm.YYYYTHH:MM, dd.mm.YY HH:MM.

Например, необходимо передать отложенное значение для параметра с ID 163, равное 200. Дата срабатывания — 20 ноября 2030 г 15:45. В параметрах необходимо указать следующее:

```
params = {'163_delay': 50, '163_delay_date': '20.11.2030 15:45'}
```

**new\_object** – переменная, возвращаемое значение. В случае успешного создания объекта возвращается queryset со всеми реквизитами объекта. Каждый реквизит является отдельной записью этого списка. Если объект не создался, то возвращается строка с текстом ошибки.

## Функция `edit_object`

**Назначение:** Редактирование полей объекта и сохранение в истории

Интерфейс вызова:

```
from app.functions.api_funs import edit_object  
edited_object = edit_object(class_id, code, user_id, location='table', source=None, **params)
```

где:

**class\_id** – ID класса редактируемого объекта.

**code** – код редактируемого объекта.

**user\_id** – ID пользователя, под которым будут сохранены эти данные. Рекомендуется передавать значение `request.user.id`.

**location** – физическое расположение объекта в базе данных. Все объекты расположены либо в справочниках, либо в контрактах, либо в словарях. Соответственно в параметр можно передавать только строковые значения `«table»`, `«contract»`, `«dict»` или `«tr»`. Другое значение вернет сообщение об ошибке. Пропустить этот параметр, то он примет значение по умолчанию - `«table»`.

**source** – источник. Принимает словарь с тремя ключами `'class_id'`, `'code'`, `'location'`. Необходим

для отслеживания источников работы с объектами из истории. Пропустить этот параметр, то он примет значение по умолчанию - None.

**\*\*params** – параметры. Словарь с неограниченным количеством пар. В ключах передаются ID параметров класса. Их можно посмотреть в конструкторе справочников / контрактов. В значения указываем непосредственно значения полей.

**ВАЖНО!!!** Ключи — числовые значения, но передавать их необходимо в виде строк, т. е. заключать ключи в кавычки.

Также в словарь «Params» передаются два параметра — timestamp и parent\_transact.

**timestamp** – таймштамп (данные о дате и времени) совершения операции. Если проигнорировать параметр, то по умолчанию передастся текущий момент.

**parent\_transact** – ID родительской транзакции, которая вызывает данное изменение. Если таковой транзакции нет, то по умолчанию значение остается пустым.

**edited\_object** – переменная, возвращаемое значение. В случае успешного внесения изменений в объект возвращается queryset со всеми реквизитами объекта. Каждый реквизит является отдельной записью этого списка. Если объект не создался, то возвращается строка с текстом ошибки.

Например:

```
params = {'163': 12, '199': True, '228': '8-992-357-96-87'}
```

**Примечание 1.** При редактировании словаря нет смысла указывать параметр owner, поскольку подчиненные объекты не меняют своего родителя.

**Примечание 2.** При редактировании объекта можно не только непосредственно изменять значение реквизитов объекта, но и задать отложенное значение. Для передачи отложенного значения необходимо передать два параметра — значение и время исполнения. Для передачи отложенного значения добавьте «\_delay» к заголовку реквизита. Для передачи даты выполнения к заголовку реквизита добавьте «\_delay\_date». Дату необходимо указать в одном из следующих форматов: YYYY-mm-dd, YYYY-mm-ddTHH:MM, YYYY-mm-dd HH:MM,

Например: нам необходимо передать отложенное значение для параметра с ID 163, равное 200. Дата срабатывания — 20 ноября 2030 г 15:45. В параметрах необходимо указать следующее:

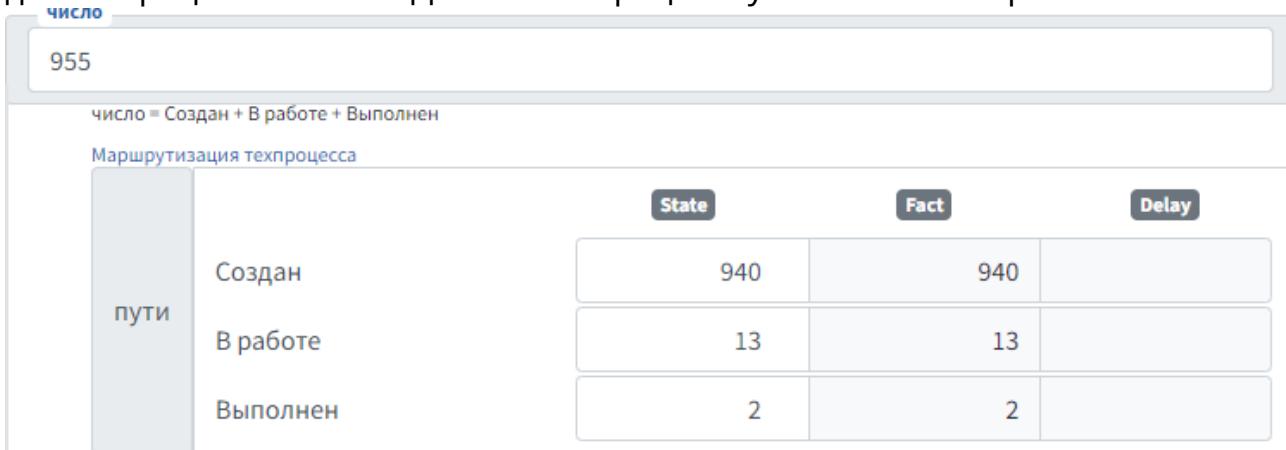
```
params = { '163_delay': 50, '163_delay_date': '20.11.2030 15:45'}
```

**Примечание 3.** При редактировании техпроцесса обязательно указать изменения всех стадий в одной операции. Перед сохранением проверяется целостность данных. Если она не будет соблюдена, то изменения объекта не сохранятся.

Пример. Допустим у нас есть техпроцесс с ID 260 и следующей структурой:

ID	Название	Ответственный	Потомки
264	Создан		267 В работе  -
267	В работе	7 admin@admin.ru Филипп Шулика	268 Выполнен  -
268	Выполнен		 +

Также у нас есть объект с кодом 9 контракта с ID: 355, являющегося родителем для техпроцесса с ID 260. Данные техпроцесса указаны на изображении:



Нам необходимо переместить 20 единиц со стадии «Создан» в стадию «В работе».

Вызов функции будет выглядеть следующим образом:

```
from app.functions.api_funs import edit_object
params = { '264': -20, '267': '20'}
edited_tp = edit_object(260, 9, request.user.id, 'tp', **params)
```

## Функция remove\_object

**Назначение:** Удаление объекта и сохранение сведений об удаленном объекте в истории  
**Интерфейс вызова:**

```
from app.functions.api_funs import remove_object
result = remove_object(class_id, code, user_id, location='table', source=None, forced=False)
```

где:

**class\_id** – ID класса удаляемого объекта.

**code** – код удаляемого объекта.

**user\_id** – ID пользователя, под которым будут сохранены эти данные. Рекомендуется передавать значение **request.user.id**.

**location** – физическое расположение объекта в базе данных. Все объекты расположены либо в справочниках, либо в контрактах, либо в словарях. Соответственно в параметр можно передавать только строковые значения «table», «contract» или «dict». Другое значение вернет

сообщение об ошибке.

**source** – источник. Принимает словарь с тремя ключами 'class\_id', 'code', 'location'. Необходим для отслеживания источников работы с объектами из истории.

**forced** – Принудительное удаление. Булевое значение, опциональная переменная, по умолчанию передается ложь. Перед удалением объекта производится проверка, есть ли другие объекты в системе, использующие данный объект в качестве ссылки. Если таковые объекты есть — значит объект не будет удален. При включенном параметре forced=True проверка не производится, объект удаляет в любом случае.

**result** – строковая переменная, возвращаемое значение. Содержит извещение об ошибке или об успешном удалении объекта.

Также в словарь «Params» передаются два параметра — timestamp и parent\_transact.

**\*\*params** – параметры. Словарь с параметрами. Передаются два параметра.

**timestamp** – таймштамп (данные о дате и времени) совершения операции. Если проигнорировать параметр, то по умолчанию передастся текущий момент.

**parent\_transact** – ID родительской транзакции, которая вызывает данное изменение. Если таковой транзакции нет, то по умолчанию значение остается пустым.

## Функция remove\_object\_list

**Назначение:** Удаление объекта и сохранение сведений об удаленном объекте в истории  
Интерфейс вызова:

```
from app.functions.api_funs import remove_object_list  
result = remove_object_list(class_id, user_id, **params)
```

где:

**class\_id** – ID класса удаляемого объекта.

**user\_id** – ID пользователя, под которым будут сохранены эти данные. Рекомендуется передавать значение **request.user.id**.

**\*\*params** – опциональные параметры в виде словаря Python.

Рассмотрим все ключи params:

**location** – физическое расположение объекта в базе данных. Все объекты расположены либо в справочниках, либо в контрактах, либо в словарях. Соответственно в параметр можно передавать только строковые значения «t», «c» или «d». Другое значение вернет сообщение об ошибке. По умолчанию принимает значение «t».

**interval\_codes** – интервал кодов. Стока в виде двух целых чисел и тире между ними. Также принимается строка с пропуском верхнего или нижнего предела интервала, т.е. строка вида «15-» или «-998». В таких случаях будут соответственно удалены все товары до или после указанного кода. Указанные коды всегда включаются в список удаляемых товаров.

**list\_codes** – список кодов. Тип данных: «Список». Используется, если необходимо удалить конкретные объекты.

**Примечание по параметрам list\_codes и interval\_codes:** Если пропустить оба параметра, то будут удалены все объекты указанного класса.

**source** – источник. Принимает словарь с тремя ключами 'class\_id', 'code', 'location'. Необходим для отслеживания источников работы с объектами из истории.

**forced** – Принудительное удаление. Булевое значение, опциональная переменная, по умолчанию передается ложь. Перед удалением объекта производится проверка, есть ли другие объекты в системе, использующие данный объект в качестве ссылки. Если таковые объекты есть — значит объект не будет удален. При включенном параметре forced=True проверка не производится, объект удаляет в любом случае.

**timestamp** – таймштамп (данные о дате и времени) совершения операции. Если

проигнорировать параметр, то по умолчанию передастся текущий момент.  
**parent\_transact** – ID родительской транзакции, которая вызывает данное изменение. Если таковой транзакции нет, то по умолчанию значение остается пустым.

**Возвращаемые данные:** Возвращается список из двух переменных:  
Первая – список кодов удаленных объектов.  
Вторая – комментарии. Они возникают, если какие-то объекты не были удалены.

**Пример запроса:**

```
list_del_codes, comments = remove_object_list(131, 8, location='t', interval_codes='16-32')
```

## Функция get\_object

**Назначение:** Получение объекта в формате джейсон.

```
from app.functions.api_funs import get_object  
my_object = get_object(class_id, code, location='table')
```

где:

**class\_id** – ID класса требуемого объекта.

**code** – код объекта

**my\_object** – возвращаемое значение. Объект в формате JSON, где ключами являются ID параметров класса данного объекта, а значениями – словари со всей сопутствующей информацией об указанном параметре.

**Пример результата:**

```
{"code": 1,  
"parent_structure": 131,  
"type": "table",  
"132": {"id": 385, "name": "Наименование", "type": "string", "value": "Российский рубль"},  
"133": {"id": 386, "name": "Символьный код", "type": "string", "value": "RUB"},  
"134": {"id": 387, "name": "Цифровой код", "type": "float", "value": "643"},  
"135": {"id": 388, "name": "Курс", "type": "float", "value": "1"},  
"198": {"id": 531, "name": "коэффициент", "type": "float", "value": "1"}}
```

## Функция get\_object\_list

**Назначение:** Получение списка объектов в формате джейсон.

```
from app.functions.api_funs import get_object_list  
my_object_list = get_object_list(class_id, *conditions, **params)
```

где:

**class\_id** – ID класса требуемого объекта.

**conditions** – условия. Переменная типа list, где каждое значение — строка. У элементов списка есть обязательный синтаксис:

<ID\_параметра\_класса><знак\_логического\_сравнения><сравниваемое\_значение>

**ID\_параметра\_класса** — целое число.

**знак\_логического\_сравнения** — символы. Далее приведены эти символы и их расшифровка:

eq – равно, ne – неравно, gt – больше, lt – меньше, ge – больше либо равно, le – меньше либо равно

**сравниваемое\_значение** — число или строка. С этим значением будут сравниваться значения параметров объектов и т.о. отбираться в результирующий список.

**\*\*params** – дополнительные параметры в формате словаря. Среди них:

**location** – физическое расположение объектов — table, contract, dict. По умолчанию задается значение table. Пример:

```
my_object_list = get_object_list(class_id, *conditions, location='contract')
```

**logic\_connector** – знак логической операции - «ИЛИ» либо «И». По умолчанию все условия объединены логическим «И». Но можно задать значение «ИЛИ». Знак задается один раз и служит операцией для всех передаваемых условий. Значения — 'or' или 'and'. Пример:

```
my_object_list = get_object_list(class_id, *conditions, logic_connector='or')
```

**num\_objects** – количество возвращаемых объектов. Если список большой, то его можно ограничить, выводя лишь первые n объектов. Пример:

```
my_object_list = get_object_list(class_id, *conditions, num_objects=10)
```

**page** – номер страницы. Работает только в связке с параметром num\_object. Если пользователь ограничивает количество выводимых объектов, то список условно разбивается на страницы, на каждой из которых находятся **num\_objects** объектов. Page уточняет, какую именно страницу пользователю необходимо указать.

**my\_object** – возвращаемое значение. Объект в формате JSON. Список объектов, каждый из которых является словарем, у которого ключами являются ID параметров класса данного объекта, а значениями – словари со всей сопутствующей информацией об указанном параметре.

**Пример результата:**

```
[{"code": 9, "parent_structure": 131, "type": "table", "132": {"id": 1554, "name": "Наименование", "type": "string", "value": "йота"}, "133": {"id": 1555, "name": "Символьный код", "type": "string", "value": "YTA"}, "134": {"id": 1912, "name": "Цифровой код", "type": "float", "value": "951"}, "135": {"id": 1556, "name": "Курс", "type": "float", "value": "16"}, "137": {"id": 1667, "name": "Дата обновления", "type": "datetime", "value": "2022-06-30T11:23"}}, {"code": 7, "parent_structure": 131, "type": "table", "132": {"id": 1551, "name": "Наименование", "type": "string", "value": "зайчик"}, "133": {"id": 1552, "name": "Символьный код", "type": "string", "value": "ZCK"}, "134": {"id": 1782, "name": "Цифровой код", "type": "float", "value": "34"}, "135": {"id": 1553, "name": "Курс", "type": "float", "value": "88"}, "137": {"id": 1783, "name": "Дата обновления", "type": "datetime", "value": "1966-03-23T00:58"}}, {"code": 5, "parent_structure": 131, "type": "table", "132": {"id": 1548, "name": "Наименование", "type": "string", "value": "тугрик"}, "133": {"id": 1549, "name": "Символьный код", "type": "string", "value": "TRK"}, "135": {"id": 1550, "name": "Курс", "type": "float", "value": "18"}]
```

В листе разработки — **нужны формулы типа On\_click**. Это поле типа Eval, но вместо вычисляемого поля — button (кнопка). Нужна тогда, когда формулу нужно выполнить только если Пользователю нужно что-то запустить. Например вывести отчет на печать (чек на ФР), или получить какой-нибудь отчет, или загрузить данные из внешнего источника (заполнить объекты значениями в пакетной обработке)

**Также нужна формула On\_update.** Заменили на специальные поля (Условие выполнения (business rule или BR), Link map (Карта ссылок - LM) и Триггер (Trigger или TR)). В Справочниках от них отказались. Нужен в контрактах, когда изменяется какой-либо контракт, и нужно внести изменения в другой

Пример: Исправляем документ Приходная накладная от поставщика, и нужно внести изменения в связанный с ним контракт Складское хранение.

*Возвращаемые результаты eval (формул) могут быть в виде HTML блоков. <Input> – заменен на <Div>*

Пример:

К проекту подключен Pandas и можно делать такие запросы к базе

Из контракта Складское хранение получить все объекты и показать их в элементе

```
from app.models import ContractCells  
import pandas as pd
```

```
o = ContractCells.objects.filter(parent_structure_id=16, name_id=20).values()  
df = pd.json_normalize(o, max_level=3)  
result = df.to_html(classes='table table-striped')
```

Результат может выглядеть так.

Остатки на складах		id	code	parent_structure_id	name_id	value
	0	194	1	16	20	1
	1	198	2	16	20	-4
	2	360	3	16	20	70
	3	589	4	16	20	5

## Функция get\_object\_hist

**Назначение:** Получение списка версий одного объекта в формате джейсон.

```
from app.functions.api_FUNS2 import get_object_hist  
my_object_hist = get_object_hist(class_id, code, date_from, date_to **params):
```

где:

**class\_id** – ID класса требуемого объекта (целое число)

**code** – код требуемого объекта (целое число),

**date\_from** – Дата и время, с которого необходимо начинать отчет истории объекта (формате datetime),

**date\_to** – Дата и время, по которое необходимо собрать отчет истории объекта (формат datetime). Если его не указать, то по умолчанию примет текущее время.

**params** – набор дополнительных необязательных условий (тип — dictionary). Среди них:

**location** – физическое расположение объектов — table, contract, dict, tp. По умолчанию задается значение table.

**children** – опция, позволяющая отключить вывод дочерних объектов. Логическая (булевая) переменная — True/False. По умолчанию примет значение True, т. е.

Выведет все дочерние объекты.

### Пример:

```
from datetime import datetime
from dateutil.relativedelta import relativedelta
from app.functions.api_funs2 import get_object_hist

date_from = datetime.today() - relativedelta(days=2)
my_object_hist = get_object_list(131, 18, date_from, location='contract')
```

Данный пример покажет все изменения контракта № 131 с кодом объекта 18 за последние 2 дня от текущего момента.

**my\_object\_hist** – возвращаемое значение. Объект в формате JSON. Список объектов, каждый из которых является словарем. Ключами у данного словаря служат ID параметров класса данного объекта, а значениями – словари со всей сопутствующей информацией об указанном параметре.

Пример результата:

```
[{"code": 3, "parent_structure": 131, "type": "table", "132": {"type": "string", "value": null, "delay": null}, "135": {"type": "float", "value": 100.2625, "delay": null}, "137": {"type": "datetime", "value": "2024-03-15T03:31", "delay": null}, "134": {"type": "float", "value": null, "delay": null}, "133": {"type": "string", "value": null, "delay": null}, "198": {"type": "float", "value": null, "delay": null}, "date_update": "2024-03-15T03:31:02"}, {"code": 3, "parent_structure": 131, "type": "table", "132": {"type": "string", "value": null, "delay": null}, "135": {"type": "float", "value": 99.9718, "delay": null}, "137": {"type": "datetime", "value": "2024-03-16T03:31", "delay": null}, "134": {"type": "float", "value": null, "delay": null}, "133": {"type": "string", "value": null, "delay": null}, "198": {"type": "float", "value": null, "delay": null}, "date_update": "2024-03-16T03:31:03"}, {"code": 3, "parent_structure": 131, "type": "table", "132": {"type": "string", "value": null, "delay": null}, "135": {"type": "float", "value": 99.9718, "delay": null}, "137": {"type": "datetime", "value": "2024-03-17T03:31", "delay": null}, "134": {"type": "float", "value": null, "delay": null}, "133": {"type": "string", "value": null, "delay": null}, "198": {"type": "float", "value": null, "delay": null}, "date_update": "2024-03-17T03:31:05"}, {"code": 3, "parent_structure": 131, "type": "table", "132": {"type": "string", "value": null, "delay": null}, "135": {"type": "float", "value": 99.9718, "delay": null}, "137": {"type": "datetime", "value": "2024-03-18T03:31", "delay": null}, "134": {"type": "float", "value": null, "delay": null}, "133": {"type": "string", "value": null, "delay": null}, "198": {"type": "float", "value": null, "delay": null}, "date_update": "2024-03-18T03:31:03"}]
```

## ФУНКЦИЯ «create\_class»

**Назначение:** Создание нового класса.

```
from app.functions.api_funs import create_class  
is_done, text = create_class(user_id, class_type, class_name, parent_id=None, **params)
```

где:

*a) Входные данные:*

**is\_done** – булевая переменная, флаг результата. Если класс был создан – возвращает True, если была допущена ошибка – False

**text** – сопровождающий текст. В случае ошибки содержит текст ошибки. Если класс создан – то в тексте содержится информация по созданному классу.

*b) Входные данные – обязательные параметры:*

**user\_id** – ID пользователя, создавшего класс. Целое число

**class\_type** – тип (разновидность) класса. Стока. Все классы строго типизированы. Поэтому обязательно необходимо указать один из существующих типов. Список типов: folder, tree, table, contract, array, dict, alias, techpro

**class\_name** – Наименование класса. Стока. Не допускаются одинаковые имена в рамках одной области видимости, т.е. подчиненные одному родительскому классу.

**parent\_id** – ID родительского класса. Целое число.

*c) Входные данные - опциональные параметры:*

**location** – расположение класса. Один из символов – ['с', 'т']. По умолчанию равен «т». Все классы относятся либо к меню справочников, либо к меню контрактов. Если класс будет расположен в меню контрактов – следует указывать «с», в противном случае «т» или опустить параметр.

**parent\_transact** – ID родительской транзакции. Стока. Если создание класса – единичная транзакция – проигнорируйте этот параметр. Если же создание класса – часть каскада операций, то укажите ID родительской транзакции.

**timestamp** – временная метка. Дата-время. По умолчанию берет текущий момент. Указывать непосредственно необходимо только в случае, если транзакция не единичная, а часть каскада операций.

**business\_rule** – Бизнес-правило. Стока. Применяется для некоторых типов классов. По умолчанию – пустая строка. Заполнять при необходимости.

**link\_map** – Линкмап. Стока. По умолчанию пустая строка. Передавать только в случае необходимости для требуемого типа класса. Примечание: для технического процесса линкмап – это список из целых чисел, где каждое число – это ID пользователя, ответственного за соответствующую стадию техпроцесса.

**trigger** – Триггер. Стока. По умолчанию – пустая строка. Указывать только при необходимости для некоторых типов.

**stages** – Стадии. Список строк. По умолчанию задается одна стадия – «Создан». Список соответственно выглядит так: ['Создан', ]. Используется только для одного типа – техпроцесс. Для других типов игнорируйте параметр.

**control\_field** – Контрольное поле. Целое число. По умолчанию – None (пусто). Используется только для типа «техпроцесс», при этом для этого типа параметр является обязательным. Если его не указать, то техпроцесс не будет создан.

Примеры:

```
is_done, message = create_class(41, 'contract', 'учет персонала', location='c')
```

Эта команда создаст класс типа «Контракт» в корне древовидной структуры (поскольку не указан parent\_id).

```
stages = ['на складе', 'в пути', 'доставлен']  
link_map = [None, 41, 56]  
is_done, message = api_funs.create_class(request.user.id, 'techpro', 'перемещение товара', 847, loca-
```

```
tion='c',
stages=stages, link_map=link_map, control_field=849)
```

Данная команда создаст технический процесс.

Обратите внимание, на переменные stages и link\_map. Это два списка. Один перечисляет стадии техпроцесса, второй – ответственных пользователей за работу на этих стадиях. Во-первых, количество элементов должно быть одинаково. Если стадий будет больше, чем элементов link\_map, то стадии, не получившие ответственного, будут выполняться автоматически, без создания заявок сотруднику. Если в линкмапе элементов будет больше, то все лишние элементы будут проигнорированы.

Во-вторых, если на некоторых стадиях не нужны подтверждения ответственных сотрудников, то на соответствующих позициях необходимо поставить пустоту, как это сделано в нашем случае. Первая стадия – «На складе» будет принимать новые значения без создания задач пользователю.

## Функция «edit\_class»

**Назначение:** Редактирование существующего класса. Функция позволяет лишь переименовывать класс и менять его родительский класс, т.е. перемещать его в структуре данных. Если необходимо изменить настройки класса, воспользуйтесь функциями create class param или edit class param.

```
from app.functions.api_funs import edit_class
text = edit_class(user_id, class_id, class_type, location='t', **params)
```

где:

*a) Выходные данные:*

**text** – Функция возвращает строку. В случае ошибки она содержит текст ошибки. Если операция выполнена успешно, то функция вернет ‘ok’.

*b) Входные данные – обязательные параметры:*

**user\_id** – ID пользователя, редактирующего класс. Целое число

**class\_id** – ID редактируемого класса. Целое число.

**class\_type** – тип (разновидность) класса. Стока. Все классы строго типизированы. Поэтому обязательно необходимо указать один из существующих типов. Список типов: folder, tree, table, contract, array, dict, alias, techpro

**location** – расположение класса (меню Справочников или контрактов). Символ ‘t’ или ‘c’. По умолчанию принимает значение ‘t’, т.е. ищет класс в меню справочников

*c) Входные данные - опциональные параметры:*

**parent\_transact** – ID родительской транзакции. Стока. Если создание класса – единичная транзакция – проигнорируйте этот параметр. Если же создание класса – часть каскада операций, то укажите ID родительской транзакции.

**timestamp** – временная метка. Дата-время. По умолчанию берет текущий момент. Указывать непосредственно необходимо только в случае, если транзакция не единичная, а часть каскада операций.

**class\_name** – название класса. Стока. Указывается только в том случае, если необходимо переименовать класс.

**parent\_id** – ID родительского класса. Целое число. Указывать необходимо только в случае перемещения класса

**Примеры:**

```
result = edit_class(25, 855, 'alias', 'c', class_name='Новое имя класса')
```

Данный пример переименует класс.

```
result = api_funs.edit_class(25, 855, 'alias', 'c', parent_id=77)
```

А этот пример переносит класс к новому родителю. Можно также в одной операции выполнить оба вида редактирования.

## Функция «remove\_class»

**Назначение:** Удаление существующего класса.

```
from app.functions.api_funs import remove_class  
text = remove_class(user_id, class_id, class_type, location='t', **params)
```

где:

*a) Выходные данные:*

**text** – Функция возвращает строку. В случае ошибки она содержит текст ошибки.

Если операция выполнена успешно, то функция вернет ‘ok’.

*b) Входные данные – обязательные параметры:*

**user\_id** – ID пользователя, удаляющего класс. Целое число

**class\_id** – ID удаляемого класса. Целое число.

**class\_type** – тип (разновидность) класса. Стока. Все классы строго типизированы.

Поэтому обязательно необходимо указать один из существующих типов. Список типов: folder, tree, table, contract, array, dict, alias, techpro

**location** – расположение класса (меню Справочников или контрактов). Символ ‘t’ или ‘c’. По умолчанию принимает значение ‘t’, т.е. ищет класс в меню справочников.

*c) Входные данные - optionalные параметры:*

**parent\_transact** – ID родительской транзакции. Стока. Если создание класса – единичная транзакция – проигнорируйте этот параметр. Если же создание класса – часть каскада операций, то укажите ID родительской транзакции.

**timestamp** – временная метка. Дата-время. По умолчанию берет текущий момент.

Указывать непосредственно необходимо только в случае, если транзакция не единичная, а часть каскада операций.

**parent\_id** – ID родительского класса. Целое число. Указывать необходимо только в случае перемещения класса

**Пример:**

```
result = remove_class(25, 855, 'alias', 'c')
```

## Функция «create\_class\_param»

**Назначение:** Создание параметра (реквизита) класса.

**Примечание:** У классов есть пользовательские и системные параметры. Системные параметры нельзя создавать. Они создаются при создании класса. Примеры системных параметров (реквизитов): LinkMap, BusinessRule, Trigger.

```
from app.functions.api_funs import create_class_param  
text = create_class_param(user_id, class_id, class_type, param_name, param_type, location='t',  
**params)
```

где:

*a) Выходные данные:*

**text** – Функция возвращает строку. В случае ошибки она содержит текст ошибки.

Если операция выполнена успешно, то функция вернет данные о созданном реквизите.

*b) Входные данные – обязательные параметры:*

**user\_id** – ID пользователя, создавшего класс. Целое число

**class\_id** – ID редактируемого класса. Целое число.

**class\_type** – тип (разновидность) класса. Стока. Все классы строго типизированы.

Поэтому обязательно необходимо указать один из существующих типов. Список типов: folder, tree, table, contract, array, dict, alias, techpro, **param\_name** – название параметра. Стока. В пределах класса нельзя дублировать названия.

**param\_type** – тип данных. Стока. Список типов данных можно посмотреть в справочнике «Типы данных».

**location** – расположение класса (меню Справочников или контрактов). Символ ‘t’ или ‘c’. По умолчанию принимает значение ‘t’, т.е. ищет класс в меню справочников

#### **в) Входные данные - опциональные параметры:**

**parent\_transact** – ID родительской транзакции. Стока. Если создание класса – единичная транзакция – проигнорируйте этот параметр. Если же создание класса – часть каскада операций, то укажите ID родительской транзакции.

**timestamp** – временная метка. Дата-время. По умолчанию берет текущий момент. Указывать непосредственно необходимо только в случае, если транзакция не единичная, а часть каскада операций.

**value** – код значения. Стока или список. Обязательный параметр для следующих типов данных: eval, enum, const, link. У словарей отсутствует этот атрибут.

Как правильно заполнять свойство **value**:

- Для типа eval: - в соответствии с синтаксисом языка программирования Python. Поддерживаются также внутренние формулы, обернутые в двойные квадратные скобки и внутренние api-функции.
- Для типов const, link необходимо указать строку вида <location>.<link\_id>, где location – расположение – «contract» либо «table»; link\_id – ID класса, на который ссылается данный параметр. Синтаксис идентичен. Различие только в том, что для типа данных «const» необходимо ссылаться на класс типа «alias», а для типа данных «link» нужно указывать класс типов «contract» или «table». Пример: contract.558 или table.1137
- Для типа «enum» необходимо указать массив строк, например: для поля «возраст» можно передать массив: [«ребенок», «юноша», «взрослый», «пожилой»].

**default** – значение по умолчанию. Его тип соответствует типу данных создаваемого параметра. Исключение: Если тип класса – словарь (dict), а тип параметра – ссылка (link), то в поле default вносится вся информация о ссылке. Это будет строка вида:

<location.link\_id.default\_code>, где:

**location** – расположение родительского класса. Стока с типизированным значением – «contract» или «table».

**link\_id** – ID родительского класса (на который ссылается объект). Целое число.

**default\_code** – код родительского объекта по умолчанию. Его можно не указывать. Если не желаете указывать код по умолчанию, то ссылка будет иметь вид:

<location.link\_id.>.

**Примеры:** table.131., contract.88.

**is\_visible** – Видимость. Булевый тип данных. True – параметр отображается в таблице, False – соответственно нет.

**is\_required** – обязательный для заполнения параметр. Булевый тип. У словарей отсутствует этот атрибут.

**delay** – использовать отложенные значения. Булевый тип. Этот атрибут есть только у классов, расположенных в меню справочников. Все контракты поддерживают отложенные значения по умолчанию.

**Примеры:**

```
val = 'if ([[contract.788.3.355]] > 10; 0.8; 0.65)'  
result = api_funs.create_class_param(25, 855, 'alias', 'поправочный коэффициент', 'eval', 'c', value=val)  
  
result = api_funs.create_class_param(25, 837, 'contract', 'цена', 'float', 'c', default=1)
```

## Функция «edit\_class\_param»

Назначение: редактирует параметр (реквизит) класса

```
edit_class_param(user_id, class_type, param_id, **params)
```

где:

**user\_id** – ID пользователя, который будет отмечен в истории, как выполнивший изменение. Целое число.

**class\_type** – тип класса. Стока. Примеры: folder, contract, array, techpro

**params** – опциональные (необязательные) параметры, свойства реквизита.

**location** – расположение класса (справочники, контракты). Символ «t» или «с». По умолчанию параметр равен символу «t».

**name** – наименование. Стока. Этот, как и последующие опциональные параметры, при указании изменяет существующие свойства заданного реквизита и, соответственно, при не указании – оставляет их нетронутыми.

**value** – код значения. Его используют только следующие типы данных: enum, eval, const, link. У реквизитов классов типа dict данное свойство отсутствует.

**default** – значение по умолчанию. Свойство используется всеми типами данных, кроме следующих: enum, eval, file. Примечание: Если тип класса – словарь (dict), а тип параметра – ссылка (link), то в поле default вносится вся информация о ссылке. Это будет строка вида:

**<location.link\_id.default\_code>**, где:

**location** – расположение родительского класса. Стока с типизированным значением – «contract» или «table».

**link\_id** – ID родительского класса (на который ссылается объект). Целое число.

**default\_code** – код родительского объекта по умолчанию. Его можно не указывать. Если не желаете указывать код по умолчанию, то ссылка будет иметь вид:

**< location.link\_id.>**.

**Примеры:** table.131., contract.88.

**is\_visible** – видимость реквизита в таблице на странице управления объектами. Булевый тип.

**is\_required** – обязательность реквизита. Булевый тип. При игнорировании реквизита пользователем заполняется из реквизита default. Свойство отсутствует у типа dict.

**delay** – отложенные значения. Свойство работает только для классов, расположенных в справочниках. Тип dict не обладает данным свойством.

**location** – расположение класса (справочники, контракты). Символ «t» или «с». По умолчанию параметр равен символу «t».

**timestamp** – временная метка. Дата-время. Необходима для регистрации события в истории. По умолчанию равна текущему моменту.

**parent\_transact** – ID родительской транзакции. Стока. Необходима для регистрации события в истории. По умолчанию – None.

**Пример:**

```
from app.functions.api_funs import edit_class_param
```

```
result = api_funcs.edit_class_param(7, 'contract', 365, location='c', name='Новое имя', is_visible=False)
```

В данном примере реквизит с ID 365 изменит свое имя и параметр видимости.

## Функция «get\_class»

Назначение: возвращает класс со всеми реквизитами в формате JSON

```
its_ok, result = get_class(class_id, class_type, location='t'):
```

где:

**class\_id** – ID искомого класса. Целое число.

**class\_type** – тип класса. Страна. Например: folder, contract, tree, dict.

**location** – расположение класса. Символ «t» или «с». По умолчанию возьмет параметр, равный «t».

Возвращаемые параметры:

**its\_ok** – флаг результата. Булевая переменная. Если функция выполнилась без ошибки – переменная будет равна True, если во время выполнения произошла ошибка – вернет False.

**result** – непосредственно результат. В случае корректного выполнения функции – класс в формате JSON. Если возникает ошибка – строка с текстом ошибки

Пример результата:

```
{"id": 558,
"name": "Баланс",
"formula": "tree",
"parent_id": 4,
"559": {
  "name": "is_right_tree", "formula": "bool", "value_str": null, "parent_id": 558, "is_required": true,
  "default": null, "is_visible": false, "priority": 1, "value": true, "delay": null, "system": false},
"560": {"name": "name", "formula": "string", "value_str": null, "parent_id": 558, "is_required": true,
  "default": null, "is_visible": false, "priority": 2, "value": null, "delay": null, "system": false},
"561": {"name": "parent", "formula": "link", "value_str": null, "parent_id": 558, "is_required": true,
  "default": null, "is_visible": false, "priority": 3, "value": null, "delay": null, "system": false}
}
```

## Функция «get\_class\_list»

Назначение: возвращает класс со всеми реквизитами в формате JSON

```
its_ok, result = get_class_list(parent_id=None, location='t'):
```

где:

**parent\_id** – ID родительского класса. Вернет все дочернее дерево классов в виде списка. Если не указать этот параметр – вернет все дерево классов заданной локации (справочников или контрактов)

**location** – расположение класса. Символ «t» или «с». По умолчанию возьмет параметр, равный «t».

Возвращаемые параметры:

**its\_ok** – флаг результата. Булевая переменная. Если функция выполнилась без ошибки – переменная будет равна True, если во время выполнения произошла ошибка – вернет False.

**result** – непосредственно результат. В случае корректного выполнения функции – список классов, где каждый элемент – словарь со структурой {id, parent\_id, name, type}. Если возникает ошибка – строка с текстом ошибки

Пример запроса:

```
its_ok, result = get_class_list(82, 'c'):
```

## Пример результата (содержание переменной result):

```
[{"id": 5, "parent_id": 82, "name": "просто контракт", "type": "contract"}, {"id": 88, "parent_id": 82, "name": "Договор с покупателем", "type": "contract"}, {"id": 98, "parent_id": 82, "name": "Сделки - расход", "type": "contract"}, {"id": 99, "parent_id": 82, "name": "Сделки - доход", "type": "contract"}, {"id": 158, "parent_id": 82, "name": "покупательские константы", "type": "alias"}, {"id": 195, "parent_id": 82, "name": "Чек на продажу", "type": "contract"}, {"id": 230, "parent_id": 82, "name": "Тест истории", "type": "contract"}, {"id": 274, "parent_id": 82, "name": "Передача товара", "type": "contract"}, {"id": 720, "parent_id": 82, "name": "Счет", "type": "contract"}, {"id": 11, "parent_id": 5, "name": "просто массив", "type": "array"}, {"id": 197, "parent_id": 195, "name": "Товары", "type": "array"}, {"id": 199, "parent_id": 195, "name": "Деньги", "type": "array"}, {"id": 208, "parent_id": 195, "name": "Условия контракта", "type": "array"}, {"id": 278, "parent_id": 274, "name": "Товары в счете", "type": "array"}, {"id": 280, "parent_id": 274, "name": "Товары переданные по счету", "type": "array"}, {"id": 726, "parent_id": 720, "name": "Спецификация счета", "type": "array"}]
```

## ФУНКЦИЯ «ВЫПОЛНИТЬ ВЫЧИСЛЕНИЕ»

**Назначение:** Выполнить код на питоне, формулу системы, или код на питоне с использованием системных формул.

```
run_eval(user_id, eval)
```

где:

**user\_id** – ID пользователя, выполняющего вычисление. Целое число.

**eval** – непосредственно текст кода в виде строки.

### Примеры:

```
result = run_eval(7, '2 * 6')
```

```
# result = 12
```

```
result = run_eval(7, '[[table.131.3.135]])
```

```
# result = 89,167
```

```
result = run_eval(7, '[[table.387]])
```

```
# result = 643
```

## И Plotly



```
846 График тест eval
При просмотре ▾

1 import pandas as pd
2 import plotly.graph_objects as go
3
4
5 df=pd.DataFrame({'Value1': [1,4,8,2,3,7,10,2], 'Value2': [4,9,5,2,2,6,5,15]})
6 date = pd.date_range('2022-01-25', periods=6, freq='H')
7 date_df=pd.DataFrame(date,columns=['Date'])
8 df=df.concat([date_df, axis=1])
9
10 fig = go.Figure()
11 fig.add_trace(go.Scatter(x=df['Date'],y=df['Value1'],name="Value1"))
12 fig.add_trace(go.Scatter(x=df['Date'],y=df['Value2'],name="Value2"))
13 fig.update_layout(hovermode="x unified")
14 fig.update_layout(
15     title="Test",
16     xaxis_title="Time",
17     yaxis_title="Values [-]",
18     legend=dict(
19         yanchor="top",
20         y=0.99,
21         xanchor="left",
22         x=0.01))
23 #fig.show()
24 result = fig.to_html()
```

Тестовый пример.

```
import pandas as pd
import plotly.graph_objects as go

df=pd.DataFrame({'Value1': [1,4,8,2,3,7,10,2], 'Value2': [4,9,5,2,2,6,5,15]})  
date = pd.date_range('2022-01-25', periods=6, freq='H')  
date=pd.DataFrame(date,columns=['Date'])  
df=pd.concat([date,df], axis=1)

fig = go.Figure()  
fig.add_trace(go.Scatter(x=df["Date"],y=df["Value1"],name="Value1"))  
fig.add_trace(go.Scatter(x=df["Date"],y=df["Value2"],name="Value2"))  
fig.update_layout(hovermode="x unified")  
fig.update_layout(  
    title="Test",  
    xaxis_title="Time",  
    yaxis_title="Values [-]",  
    legend=dict(  
        yanchor="top",  
        y=0.99,  
        xanchor="left",  
        x=0.01))  
result = fig.to_html()
```

## API

API представляет собой набор функций, вызываемых непосредственно из веб-интерфейса.  
Все адреса вызовов функций АПИ находятся по адресу: /api/

### login

Функция авторизации. Принимает два параметра — login, password. Возвращает csrf-токен сессии.

Пример:

<http://42.hq.f-trade.ru/api/login?login=admin&password=admin>

### create-object

Адрес вызова функции:

<http://42.hq.f-trade.ru/api/create-object>

Передаваемые параметры:

а. Обязательные параметры:

**class\_id** – класс вызываемого объекта,

**location** – размещение вызываемого объекта в БД. Для справочников необходимо указать table, для контрактов — contract, для словарей — dict.

**owner** – параметр, передаваемый только при создании словаря. Он указывает код объекта — родителя словаря. Параметр обязательный.

б. Опциональные параметры:

В списке параметров также указываются реквизиты создаваемого объекта.

В качестве ключа необходимо указывать ID реквизита.

Перед отправкой запроса убедитесь, что Вы отправили все обязательные реквизиты. Это можно увидеть на страницу конструктора класса. В противном случае система заполнит обязательные реквизиты значениями по умолчанию. Кроме того, для справочников необходимо указать название, для массивов — собственника. В противном случае объект не будет создан.

**Пример запроса:**

[http://42.hq.f-trade.ru/api/create-object?class\\_id=111&112=ooo&location=dict&owner=7](http://42.hq.f-trade.ru/api/create-object?class_id=111&112=ooo&location=dict&owner=7)

При создании объекта можно сразу задать отложенные значения для реквизитов. Для этого необходимо передать два параметра: значение реквизита и дата выполнения реквизита.

Синтаксис отложенных значений следующий:

<req\_id>\_delay=<delay\_value>&<req\_id>\_delay\_date=<delay\_date>

где:

req\_id – ID реквизита, для которого мы задаем отложенное значение,

delay\_value – отложенное значение реквизита,

delay\_date – дата и время выполнения отложенного значения. Указываются в одном из следующих форматов: YYYY-mm-dd, YYYY-mm-ddTHH:MM, YYYY-mm-dd HH:MM, dd.mm.YYYYTHH:MM, dd.mm.YY HH:MM.

Пример: Предположим, у нас есть справочник с ID 222 с единственным параметром 230. Отправим запрос, который создаст объект данного справочника со значением реквизита 230 равным 10, и в этом же запросе зададим отложенное значение для реквизита 230. Пусть реквизит 230 с 01.01.2025 03:35 станет равным 100

<http://42.hq.f-trade.ru/api/create-object?>

[class\\_id=222&230=10&230\\_delay=100&230\\_delay\\_date=01.01.2025T03:35](http://42.hq.f-trade.ru/api/create-object?class_id=222&230=10&230_delay=100&230_delay_date=01.01.2025T03:35)

## edit-object

**Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/edit-object>

**Передаваемые параметры:**

а. Обязательные параметры:

**class\_id** – класс вызываемого объекта,  
**code** – код вызываемого объекта,  
**location** – размещение вызываемого объекта в БД. Для справочников необходимо указать **table**, для контрактов — **contract**, для словарей — **dict**.

б. Опциональные параметры:

В списке параметров также указываются реквизиты редактируемого объекта.

В качестве ключа необходимо указывать ID реквизита.

Указывать необходимо только те реквизиты, которые подлежат изменению. Для записей контракта нет смысла указывать параметр «Дата и время записи», т. к. он не редактируется пользователем. Для словарей также нет смысла указывать параметр **owner**, поскольку нельзя менять родителя словаря.

**Пример запроса:**

[http://42.hq.f-trade.ru/api/edit-object?class\\_id=111&112=uuu&location=dict&code=8](http://42.hq.f-trade.ru/api/edit-object?class_id=111&112=uuu&location=dict&code=8)

При помощи запроса **edit-object** можно передать отложенное значение реквизита, поддерживающего отложенные значения. Синтаксис и пример см. в п. [create-object](#)

## **remove-object**

**Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/remove-object>

**Передаваемые параметры:**

а. Обязательные параметры:

**class\_id** – класс вызываемого объекта,

**location** – размещение вызываемого объекта в БД. Для справочников необходимо указать **table**, для контрактов — **contract**, для словарей — **dict**.

б. Опциональные параметры:

**forced** – Принудительное удаление. Укажите этот параметр только в том случае, когда во что бы то ни стало нужно удалить объект. С указанием этого параметра выполняется небезопасное удаление. Система удалит объект вне зависимости от того, есть ли у него дочерние элементы. Под дочерними элементами полагают другие объекты, ссылающиеся на него. Подчиненные словари к таковым не относятся. Они являются наборами свойств объекта, поэтому в любом случае каскадно удаляются. Значение параметра не играет роли, его даже можно оставить пустым. Если система видит параметр forced, то выполняет небезопасное удаление.

**Пример запроса:**

[http://42.hq.f-trade.ru/api/remove-object?class\\_id=111&location=dict&code=8&forced=1](http://42.hq.f-trade.ru/api/remove-object?class_id=111&location=dict&code=8&forced=1)

## **remove-object-list**

**Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/remove-object-list>

## **Передаваемые параметры:**

а. Обязательные параметры:

**class\_id** – класс вызываемого объекта,

б. Опциональные параметры:

**location** – размещение вызываемого объекта в БД. Для справочников необходимо указать **t**, для контрактов — **c**, для словарей — **d**.

**interval\_codes** – интервал кодов. Стока вида «**a-b**», где **a** и **b** – целые числа – коды объектов. Система удалит все объекты между **a** и **b**, включая **a** и **b**.

**list\_codes** – список кодов. Стока вида «**a,b,c**», где **a**, **b**, **c** – целые числа, коды объектов. Система удалит только перечисленные объекты.

**forced** – Принудительное удаление. Укажите этот параметр только в том случае, когда во что бы то ни стало нужно удалить объект. С указанием этого параметра выполняется небезопасное удаление. Система удалит объект вне зависимости от того, есть ли у него дочерние элементы. Под дочерними элементами полагают другие объекты, ссылающиеся на него. Подчиненные словари к таковым не относятся. Они являются наборами свойств объекта, поэтому в любом случае каскадно удаляются. Значение параметра не играет роли, его даже можно оставить пустым. Если система видит параметр forced, то выполняет небезопасное удаление.

**timestamp** – временная метка в формате даты, дата-времени, указанной в российской локали, либо в формате ISO.

**parent\_transact** – ID родительской транзакции при регистрации в истории.

**Возвращаемое значение:** Вернутся две переменные – список кодов успешно удаленных объектов и строка с комментариями по ошибкам удаления. Две переменные будут преобразованы в строку, выводимую в html-формате.

## **Пример запроса:**

[http://42.hq.f-trade.ru/api/remove-object-list?class\\_id=111&location=c&interval\\_codes=5-98](http://42.hq.f-trade.ru/api/remove-object-list?class_id=111&location=c&interval_codes=5-98)

## **Пример ответа:**

[5, 8, 12, 44], “

В указанном примере все объекты успешно удалены.

## **get-object**

Возвращает объект в формате JSON со всеми существующими дочерними объектами — словарями, массивами.

**Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/get-object>

**Передаваемые параметры:**

а. Обязательные параметры:

**class\_id** – класс вызываемого объекта,

**code** – код вызываемого объекта

б. Опциональные параметры:

**location** – размещение вызываемого объекта в БД. Для справочников необходимо указать **table**, для контрактов — **contract**, для словарей — **dict**. Если параметр не указан — то будет использовано значение по умолчанию - **table**

**Пример запроса:**

[http://42.hq.f-trade.ru/api/get-object?code=33&class\\_id=626](http://42.hq.f-trade.ru/api/get-object?code=33&class_id=626)

## **get-object-list**

Возвращает объект в формате JSON со всеми существующими дочерними объектами — словарями, массивами.

**Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/get-object-list>

**Передаваемые параметры:**

а. Обязательные параметры:

**class\_id** — класс вызываемого объекта,

б. Опциональные параметры:

**location** — размещение вызываемого объекта в БД. Для справочников необходимо указать table, для контрактов — contract, для словарей — dict. Если параметр не указан — то будет использовано значение по умолчанию — table.

**condition** — условие отбора объектов. Строковое значение с обязательным синтаксисом:

**<ID\_параметра\_класса><знак\_логического\_сравнения><сравниваемое\_значение>**

**ID\_параметра\_класса** — целое число.

**знак\_логического\_сравнения** — символы. Далее приведены эти символы и их расшифровка:

eq — равно, ne — неравно, gt — больше, lt — меньше, ge — больше либо равно,

le — меньше либо равно

**сравниваемое\_значение** — число или строка. С этим значением будут сравниваться значения параметров объектов и т.о. отбираться в результирующий список.

Если Вам необходимо передать несколько условий, то передавайте в виде отдельных параметров condition. Система учитет их все.

**logic\_connector** — знак логической операции между условиями - «or» либо «and». Указать необходимо один раз. Он применится ко всем условиям. По умолчанию ставится знак «and».

**num\_objects** - количество возвращаемых объектов. Если список большой, то его можно ограничить, выводя лишь первые n объектов.

**page** — номер страницы. Работает только в связке с параметром num\_object. Если пользователь ограничивает количество выводимых объектов, то список условно разбивается на страницы, на каждой из которых находятся num\_objects объектов. Параметр page уточняет, какую именно страницу пользователю необходимо указать.

**Пример запроса:**

`api/get-object-list?class_id=131&condition=135le18&condition=135gt80&logic_connector=or&num_objects=5&page=2`

**Пример результата ответа:**

```
[{"code": 9, "parent_structure": 131, "type": "table", "132": {"id": 1554, "name": "Наименование", "type": "string", "value": "йота"}, "133": {"id": 1555, "name": "Символьный код", "type": "string", "value": "YTA"}, "134": {"id": 1912, "name": "Цифровой код", "type": "float", "value": "951"}, "135": {"id": 1556, "name": "Курс", "type": "float", "value": "16"}, "137": {"id": 1667, "name": "Дата обновления", "type": "datetime", "value": "2022-06-30T11:23"}},
```

```
{"code": 7, "parent_structure": 131, "type": "table", "132": {"id": 1551, "name": "Наименование", "type": "string", "value": "зайчик"}, "133": {"id": 1552, "name": "Символьный код", "type": "string", "value": "ZCK"}, "134": {"id": 1782, "name": "Цифровой код", "type": "float", "value": "34"}, "135": {"id": 1553, "name": "Курс", "type": "float", "value": "88"}, "137": {"id": 1783, "name": "Дата обновления", "type": "datetime", "value": "1966-03-23T00:58"}}, {"code": 5, "parent_structure": 131, "type": "table", "132": {"id": 1548, "name": "Наименование", "type": "string", "value": "тугрик"}, "133": {"id": 1549, "name": "Символьный код", "type": "string", "value": "TRK"}, "135": {"id": 1550, "name": "Курс", "type": "float", "value": "18"}}, {"code": 1, "parent_structure": 131, "type": "table", "132": {"id": 385, "name": "Наименование", "type": "string", "value": "Российский рубль"}, "133": {"id": 386, "name": "Символьный код", "type": "string", "value": "RUB"}, "134": {"id": 387, "name": "Цифровой код", "type": "float", "value": "643"}, "135": {"id": 388, "name": "Курс", "type": "float", "value": "1"}, "198": {"id": 531, "name": "коэффициент", "type": "float", "value": "1"}}]
```

## create-class

Создает класс заданного типа. Возвращает основные сведения о класса – ID, название, расположение.

**Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/create-class>

**Передаваемые параметры:**

Смотрите в соответствующих разделах функции [«Создать класс»](#)

Пример:

[http://42.hq.f-trade.ru/api/create-class?class\\_name=crm-справочники&class\\_type=folder&parent\\_id=549](http://42.hq.f-trade.ru/api/create-class?class_name=crm-справочники&class_type=folder&parent_id=549)

## edit-class

Редактирует заданный класс. Под редактированием следует понимать смену родительского ID или переименование. Для изменения реквизитов класса воспользуйтесь функциями create-class-param, edit-class-param, remove-class-param.

**Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/edit-class>

**Передаваемые параметры:**

Смотрите в соответствующих разделах функции [«Редактировать класс»](#)

Пример:

[http://42.hq.f-trade.ru/api/edit-class?class\\_id=855&class\\_name=crm-справочники&class\\_type=folder&parent\\_id=549&location=c](http://42.hq.f-trade.ru/api/edit-class?class_id=855&class_name=crm-справочники&class_type=folder&parent_id=549&location=c)

## remove-class

Удаляет заданный класс.

## **Адрес вызова функции:**

<http://42.hq.f-trade.ru/api/remove-class>

#### **Передаваемые параметры:**

Смотрите в соответствующих разделах функции [«Редактировать класс»](#)

## Пример:

[http://42.hq.f-trade.ru/api/remove-class?class\\_id=855&class\\_type=folder&location=c](http://42.hq.f-trade.ru/api/remove-class?class_id=855&class_type=folder&location=c)

## create-class-param

Создает параметр (реквизит) класса.

## Адрес вызова функции:

<http://42.hq.f-trade.ru/api/create-class-param>

**Передаваемые параметры:** Смотрите в соответствующих разделах функции «Создать параметр класса»

## Пример:

<http://42.hq.f-trade.ru/api/create-class-param?>

class

s\_id=837&class\_type=array&location=c&param\_type=float&param\_name=Стоимость&default=100

## **edit-class-param**

Редактирует заданный параметр класса.

**Адрес вызова функции:**

[api/edit-class-param?param\\_id=ID параметра класса&class\\_type=Тип класса&param\\_type=тип передаваемого параметра&location=\[t, c\]](#)

## Передаваемые параметры:

см. функцию «edit class param»

Также обязательным является параметр `param_type` – тип передаваемого параметра.

**Возвращаемый параметр:** В случае корректного выполнения функции – вернет текст «ok». В случае ошибки во время выполнения – текст ошибки

Пример:

[api/edit-class-param?param\\_id=1315&class\\_type=table&param\\_type=string&name=Новое имя](#)

## get-class

Получает класс в формате JSON, преобразованным в строку.

## Адрес вызова функции:

[api/get-class?class\\_id=ID класса&class\\_type=Тип класса&location=\[t,c\]](#)

## Передаваемые параметры:

см. функцию [«get\\_class»](#)

Возвращаемый параметр: класс в формате JSON, развернутый в строку. В случае ошибки – текст ошибки

## Пример:

[api/get-class?class\\_id=131&class\\_type=table](#)

## Пример результата:

```
{"id": 558, "name": "Баланс", "formula": "tree", "value_str": null, "parent_id": 4, "is_required": false, "default": null, "is_visible": false, "priority": 0, "value": null, "delay": null, "system": false, "559": {"name": "is_right_tree", "formula": "bool", "value_str": null, "parent_id": 558, "is_required": true, "default": null, "is_visible": false, "priority": 1, "value": true, "delay": null, "system": false}, "560": {"name": "name", "formula": "string", "value_str": null, "parent_id": 558, "is_required": true, "default": null, "is_visible": false, "priority": 2, "value": null, "delay": null, "system": false}, "561": {"name": "parent", "formula": "link", "value_str": null, "parent_id": 558, "is_required": true, "default": null, "is_visible": false, "priority": 3, "value": null, "delay": null, "system": false}}
```

## get-class-list

Получает список класс в формате JSON, преобразованным в строку, где каждый класс – словарь структуры {id, parent\_id, name, type}.

## Адрес вызова функции:

[api/get-class?parent\\_id=ID родительского класса &location=\[t,c\]](#)

## Передаваемые параметры:

см. функцию [«get\\_class»](#)

Возвращаемый параметр: класс в формате JSON, развернутый в строку. В случае ошибки – текст ошибки.

## Пример запроса:

[api/get-class?parent\\_id=131&location=t](#)

## Пример результата:

```
[{"id": 5, "parent_id": 82, "name": "просто контракт", "type": "contract"}, {"id": 88, "parent_id": 82, "name": "Договор с покупателем", "type": "contract"}, {"id": 98, "parent_id": 82, "name": "Сделки - расход", "type": "contract"}, {"id": 99, "parent_id": 82, "name": "Сделки - доход", "type": "contract"}, {"id": 158, "parent_id": 82, "name": "покупательские константы", "type": "alias"}, {"id": 195, "parent_id": 82, "name": "Чек на продажу", "type": "contract"}, {"id": 230, "parent_id": 82, "name": "Тест истории", "type": "contract"}, {"id": 274, "parent_id": 82, "name": "Передача товара", "type": "contract"}, {"id": 720, "parent_id": 82, "name": "Счет", "type": "contract"}, {"id": 11, "parent_id": 5, "name": "просто массив", "type": "array"}, {"id": 197, "parent_id": 195, "name": "Товары", "type": "array"}, {"id": 199, "parent_id": 195, "name": "Деньги", "type": "array"}, {"id": 208, "parent_id": 195, "name": "Условия контракта", "type": "array"}, {"id": 278, "parent_id": 274, "name": "Товары в счете", "type": "array"}, {"id": 280, "parent_id": 274, "name": "Товары переданные по счету", "type": "array"}, {"id": 726, "parent_id": 720, "name": "Спецификация счета", "type": "array"}]
```

## **run-eval**

Выполняет вычисление, переданные в параметре eval.

**Адрес вызова функции:**

[api/run-eval?eval=текст формулы](http://42.hq.f-trade.ru/api/run-eval?eval=текст%20формулы)

**Передаваемые параметры:**

**eval** – непосредственно текст кода или формулы в виде строки.

Примеры:

[http://42.hq.f-trade.ru/api/run-eval?eval=2\\*8](http://42.hq.f-trade.ru/api/run-eval?eval=2*8)

# 16

[http://42.hq.f-trade.ru/api/run-eval?eval=\[\[table.131.3.135\]\]](http://42.hq.f-trade.ru/api/run-eval?eval=[[table.131.3.135]])

# 89,167

## **upload-file**

Загружает файл через апи. Это единственная функция, выполняемая через POST-запрос. В теле запроса необходимо передать следующие параметры:

file – непосредственно файл;

header\_id – ID заголовка, или ID реквизита класса с типом данных — file,

code – код объекта

location – расположение файла — символ «t» или «c». Расположение класса — контракты или справочники. По умолчанию примет значение «t», т. е.

Попытается найти объект среди справочников.

csrfmiddlewaretoken – токен сессии. Уникальный 32-знаковый ключ сессии. Его можно получить из функции login.

# Контракты

Контракты — это объект системы неразрывно связанный с понятием тех процесса и бизнес-правилом. Обеспечивает манипуляцию объектами учета по алгоритму бизнес-процесса.

В части полей (реквизитов) — обладает теми же свойствами что и обычный Справочник.

Основным отличием является то — что заголовок контракта несет несколько дополнительных реквизитов - Business rule, link map, trigger. Предназначены они для автоматизации работы со связанными контрактами.

Пример:

The screenshot shows the configuration of a contract. At the top left, there's a sidebar with a tree view of objects, including 'osobye kontakty ID: 28' and 'новый временный контракт ID: 169'. The main area has tabs for 'Новый' (New), 'Сохранить' (Save), and 'Удалить' (Delete). The first tab is active.

**Управление системными параметрами контракта**

Название	Значение
Условие выполнения	<pre>if([[515]] == [[514]]; True; False)</pre>
Link Map	<pre>contract = 477 code = 4 params = { id = 483 value = [[515]] sign = -, id = 482 value = [[513.495]] * [[512]] sign = +}; contract = 477 code = [[table.497.F504eq[[code]]]] params = {id = 505 value = [[512]] sign = -}</pre>

**Триггер**

ID	Название	Тип	Код значения	По умолчанию	Образ.	Видим.
510	Дата и время записи	datetime			<input checked="" type="checkbox"/>	<input type="checkbox"/>
512	количество	float			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
513	товар	link	Тип: Контракт ID: 468 Название: мед-товары	2023-01-12T14:40 1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
514	стоимость	eval	<pre>[[512]] * [[513.495]]</pre>	При просмотре	<input type="checkbox"/>	<input checked="" type="checkbox"/>
515	оплата	float			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

At the bottom are buttons for 'Новый' (New), 'Сохранить' (Save), and 'Удалить' (Delete).

Текущая реализация BR, Link map и Trigger выглядит так.

Разберем пример. Условием срабатывания контракта является то, что Поле 515 (оплата) = Стоимости подобранныго товара Поле (514) в поле 513 (товар).

Поэтому BR будет выглядеть так: if([[515]] == [[514]]; True; False). Поле BR – булево.

Link map бросает данные в контракты с ID 477 — в существующий контракт с кодом 4, передает параметры

id = 483 value = [[515]] sign = -

и

id = 482 value = [[513.495]] \* [[512]] sign = +

Пользовательский триггер не установлен. (Обычно используется для передачи данных между разными сайтами или системами. Это обычное для системы Eval поле)

**При создании нового контракта — link map отработает полностью. При редактировании — будет кидать только дельту изменений. Работает на числовых значениях.**

В разработке интерфейс Link map, для удобства администрирования.

Если нужно что-то делать со строками, датами и прочим — для этих целей — есть поле trigger.

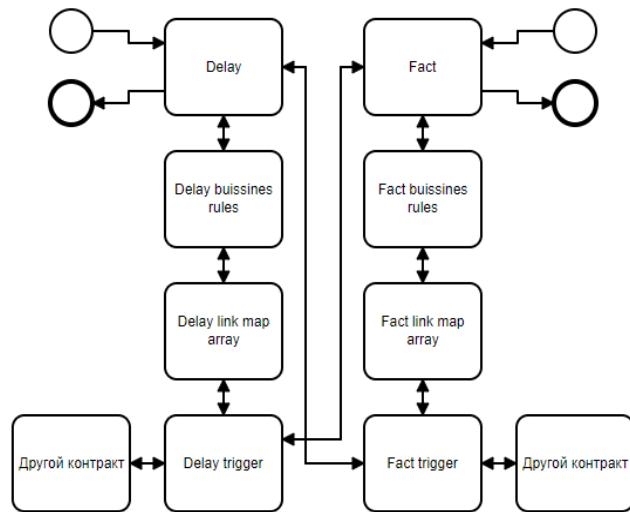
Подробное объяснение — ниже.

## Описание Тех. процесса.

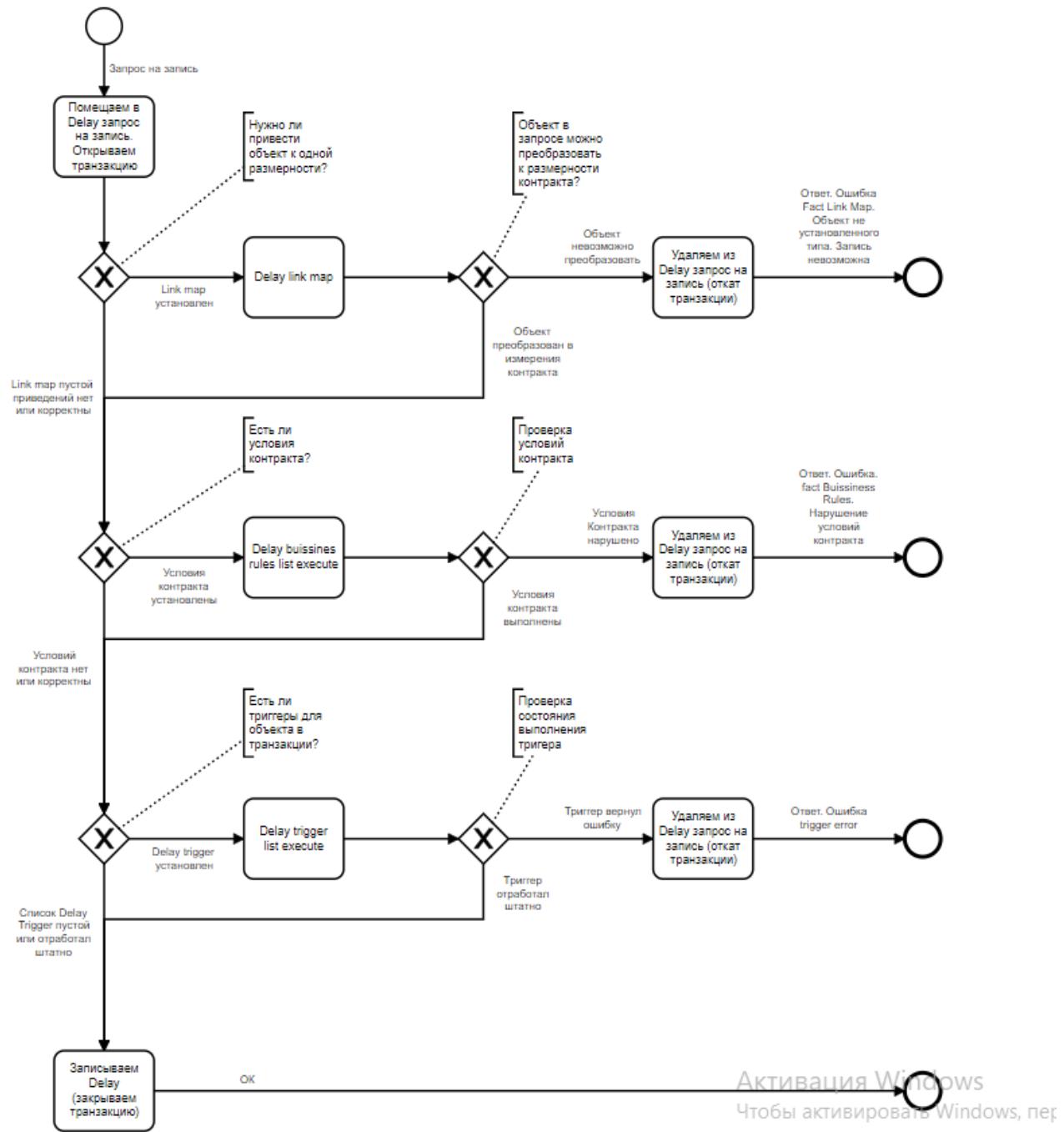
Тех. процесс обычно изображается как ящик, с входящими данными — модулем обработки и выходными данными.



В нашей системе элементарный тех процесс (контракт) выглядит так. (BR и link map нужно переставить местами.) Следующая диаграмма — правильная.



Подробнее обработка Delay выглядит так.



Fact – выполняется аналогично.

Delay и Fact – это место хранения объектов. Объекты, попадающие в Delay или Fact стараются перейти по техпроцессу дальше из Delay в Fact и наоборот, если такие свойства контракта определены.

Для того чтобы перейти к проверке условий контракта (Business Rules), нужно привести объект к той же размерности, какая указана в контракте. К примеру мы не можем проверить условие Деньги<Товары без уточнений. Для этих целей используется Link Map (таблица маршрутизации преобразований).

Если условие в Business Rules (Бизнес-правило, оно же условие контракта — выполняется), то объект может отобразить свои данные с помощью триггера на другие контракты. Если не выполняется — то процесс идет назад и на выход отправляется ошибка с данными о причинах невозможности выполнения. Причем, в истории должна появится информация — о

причинах невозможности выполнения, как и информация о успешном выполнении тригера.

Объект не разрушается, просто он проходя по системе должен оставлять следы. Смысл в том, чтобы объекты попадая в контракт — становились видимыми «замороженными» во времени, до того момента, когда не придет событие. Это можно представить себе как мы берем шарик для пин-понга — это наш единичный объект, а потом кладем его в какую-то «пластиковую» корзину. Вытаскиваем шарик из корзины. Фиксируем время на блокноте прихода и ухода шарика. В данном случае Корзина — это наш контракт. Это часть называется Fact. Что такое Delay? Запишем себе в ежедневник — положить завтра шарик в корзину, а завтра следуя напоминанию — кладем его туда. Delay — это задача которую надо выполнить — запрос, другими словами. Fact — это ответ при завершении задачи. Тут вроде все понятно. Но вот странно конечно выглядят ситуации — когда ты без задания кидаешь шарик в корзину, но в качестве развлечения — это нормально (переносим в качестве оптимизации товары по складу, потому что так удобнее искать). Мы породили Fact того, что шарик там. Что же отправить в delay? В Delay мы отправляем информацию — о том что был запрос на помещение шарика в корзину, который случился после fact. Потому как желание швырнуть шарик было — и это был запрос, который через систему не проходил, но раз факт появился — значит и запрос был.

Контракт имеет разные варианты существования. Контракт может быть **постоянным** (условно) — как та же корзина, существует пока пластик не рассыпется, и не придет в негодность. Может существовать **кратко**, например мы наш шарик завернули в упаковочную бумагу, а потом после разворачивания — бумагу выбросили. Может быть **ограниченным по условию**, когда мы корзину взяли — но обещали вернуть, например через время.

Примеры постоянного контракта — В компании есть сейф. И он без форс-мажора не может быть уничтожен. Сейф используется для хранения. Его можно выбирать в интерфейсе постоянно, без опасения того — что он завтра не окажется в архиве, пустой и разобранный.

**Сейчас в системе работает только этот вид контракта.**

Пример Ограниченнего по условию контракта. Есть договор на складское помещение, условием которого является ограничение по дате, без возможности пролонгирования. Мы должны будем вывезти все — что складировали до этой даты. Также в условии могут быть прописаны разные экзотические условия — типа не хранить очень тяжелые предметы, или чтобы они не высypались из окон.

При Наступлении условий контракта (**формула должна иметь булевое значение**) — вход в контракт блокируется. Работает только на выход. Контракт будет ждать своего закрытия автоматически, до тех пор, пока все объекты учета контракта не уйдут.

Пример краткого контракта — мы отдали курьеру товар. Курьер довез товар до адреса и отчитался — о доставке. Контракт исполнился, товар переместился клиенту. Контракт сразу был передан в архив для истории (аналитики).

Рассмотрим пример контракта который часто используется при продаже товара (Чек — мгновенная продажа). Введем только несколько упрощений. Товар у нас есть на в торговом зале, оплата будет наличными через кассу.

Use case:

1. клиент берет с полки неоплаченный товар, и (возможно) запоминает цену,
2. приносит его на кассу,
3. кассир подтверждает стоимость
4. формирует документ об одновременной оплате и отгрузке (чек)
5. Клиент уходит с оплаченным товаром.

Разберем пример. Тут как минимум два техпроцесса — складской и кассовый. Со склада товар ушел, и его денежный аналог пришел в кассу. Есть параллельный техпроцесс получения прибыли, но в рамках данного примера рассматривать его не будем. Есть еще тех процесс формирования чека на кассовом аппарате, и возможно еще... но рассмотрим два явных.

Тех Процесс Чека можно описать формулой объектов **деньги - товар = 0**. (в общем виде формула конечно сложнее, но нам это сейчас не так интересно)

Конечно нельзя два разнотипных объекта так вычитать, поэтому для таких техпроцессов нужно вводить либо метрику оценки, либо типизировать деньги как товары (для этих целей как раз и нужен link map с возможностью не только жесткого соответствия но и с eval функциями).

Первый вариант предполагает наличие у товара есть реквизит Цена. И тогда можно умножив количество на цену — получить число — и также поступить и с деньгами.

Второй вариант — вводить у денег реквизит Товар — и тогда можно получить список оплаченных товаров.

Создадим такой пример в нашей системе.

ID	Название	Тип	Код значения	По умолчанию	Образ.	Видим.
198	Собственник	link	Тип: Контракт ID: 195 Название: Чек на продажу	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
201	Товар	link	Тип: Справочник ID: 431 Название: Товары	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
202	Цена	float		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
203	Количество	float		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
206	Сумма	eval	$[[202]] * [[203]]$	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

#### Управление контрактами

ID	199
Название	Деньги
Тип	Массив
ID каталога	195
<input type="button" value="Новый"/> <input type="button" value="Сохранить"/> <input type="button" value="Удалить"/>	

#### Управление параметрами контракта

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
↓ 200	Собственник	link	Тип: Контракт ID: 195 Название: Чек на продажу	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
↑↓ 204	Время получения	datetime	дд.мм.гггг ::--	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
↑ 205	Сумма	float		<input type="checkbox"/>	<input checked="" type="checkbox"/>	
<input type="button" value="Новый"/> <input type="button" value="Сохранить"/> <input type="button" value="Удалить"/>						

Создадим новый объект контракта... Добавим товары — которые нужно списать, и деньги которые нам оплатили

Контракт "Чек на продажу" ID: 195

Посик	<input type="button" value="Найти"/>																									
Код	Дата и время записи	Код	1																							
1	2022-07-14T13:53	Дата и время записи	14.07.2022 13:53																							
Страница 1 из 10		<table border="1"> <thead> <tr> <th>Товары</th> <th>Редактировать</th> </tr> </thead> <tbody> <tr> <td>Код</td> <td>Товар</td> <td>Цена</td> <td>Количество</td> <td>Сумма</td> </tr> <tr> <td>1</td> <td>Товар1</td> <td>100</td> <td>2</td> <td>200</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Деньги</th> <th>Редактировать</th> </tr> </thead> <tbody> <tr> <td>Код</td> <td>Время получения</td> <td>Сумма</td> </tr> <tr> <td>2</td> <td>14.07.2022 17:01</td> <td>100</td> </tr> <tr> <td>1</td> <td>14.07.2022 17:00</td> <td>100</td> </tr> </tbody> </table>		Товары	Редактировать	Код	Товар	Цена	Количество	Сумма	1	Товар1	100	2	200	Деньги	Редактировать	Код	Время получения	Сумма	2	14.07.2022 17:01	100	1	14.07.2022 17:00	100
Товары	Редактировать																									
Код	Товар	Цена	Количество	Сумма																						
1	Товар1	100	2	200																						
Деньги	Редактировать																									
Код	Время получения	Сумма																								
2	14.07.2022 17:01	100																								
1	14.07.2022 17:00	100																								
<input type="button" value="Новый"/> <input type="button" value="Сохранить"/> <input type="button" value="Удалить"/>																										

Не хватает Типа контракта - Ограничен по условию. Обычно возвраты могут происходить в течении кассового дня. Или вообще Краткосрочности такого контракта. Заработало через BR.

В таких контрактах — деньги это другой объект и поэтому условие контракта можно записать так.

Добавим его как реквизит контракта

```
if [[contract.197.f198eq[[code]].206]] = [[contract.199.f200eq[[code]].205]]:
```

```
    result = "ok"
```

```
else:
```

```
    result = "contract error"
```

Код	1			
Дата и время записи	14.07.2022 13:53			
Товары	Редактировать			
Код	Товар	Цена	Количество	Сумма
1	Товар1	100	2	200
Деньги	Редактировать			
Код	Время получения	Сумма		
2	14.07.2022 17:01	100		
1	14.07.2022 17:00	100		
<input type="button" value="Условие чека"/> <input type="button" value="ok"/>				

Условие чека выполнено. Товар попал в техпроцесс Товары в чеке, Деньги попали в тех процесс Деньги по чеку. Условие выполнилось. Контракт можно передать в Историю.

Но здесь пока никак нельзя указать -что деньги то надо оприходовать в кассу, а товар нужно списать со склада.

Теперь стоит вспомнить о том — что существует такая штука как Delay и Fact

Так вот... Delay — это свойство контракта которое сообщает нам что чек — который у нас есть в примере — еще ничего не сделал. В 1С для этой цели сформированный документ нужно «Провести». У них для этой цели нужно написать код, который «двигает» регистры и прочие объекты системы.

Мы можем пойти тем-же путем, но код сопоставления трансляции реквизитов одного объекта в другой — слишком загромождает систему. Поэтому нам понадобится массив реквизитов трансляции (Link Map) из delay в Fact.

На что еще нужно обратить внимание. Контракт так или иначе может находиться в разных стадиях — Обычные стадии — Draft → Delay → Fact → Cancel. Первую и последнюю стадии рассмотрим ниже — когда опишем task. Когда контракт находится в Draft — тогда его можно удалять без опасений что он как-то может повлиять на систему. Начали оформлять Контракт, но его необходимость отпала, значит его можно удалять. Контракт из Draft можно перевести в Delay свободно, и назад вернуть в Draft, но контракт в Delay уже удалять нельзя. Технически возможно, но данные — которые находятся в Delay уже могут использоваться для анализа предварительных данных. Для перевода в Fact понадобится подписать контракт, это гарантирует целостность контракта от изменений. Еще одним дополнительным условием для перевода контракта в Fact — является выполнение условий контракта.

### Проведем исследование с отладкой. (Устарело — оставил как пример для тестов)

Попробуем разобрать ситуацию с чеком. Чек вроде бы контракт Краткий (краткосрочный), но что тут напрягает — товары в чеке — должны списаться со склада сразу - должны быть переведены в Fact, а деньги должны лежать в денежном ящику — до операции инкассации, тут явный Delay. Но вообще говоря — это не так. Деньги в по чеку — и деньги в Кассе — это разные тех процессы.

Тех процесс чека выглядит так

Delay Товар — нужно списать товар со склада (торгового зала)

Delay Деньги — нужно оприходовать деньги (забрать у клиента и положить их в денежный ящик)

Когда выполняется условие контракта — деньги (транзакции) в чеке = сумме товаров в чеке, кассир (или автоматически — с помощью события принтера чека) переводит чек в состояние Fact (пробивает чек в ККМ — аналог подписи). В момент перевода чека из состояния Delay в Fact — смотрим в таблицу сопоставления реквизитов контракта (Link Map) — и пытаемся создать в связанных контрактах записи.

И в итоге Получаем

Fact Товар - Списан товар по чеку — Тригер по складу (торговому залу)

Fact Деньги — Деньги положены в денежный ящик (Тригер по Кассе предприятия).

Теперь вопрос. Что делать — если нет контракта, (он может быть закрыт) тогда нужно создавать новый контракт по параметрам из объекта — который порождает движения. Если контракт закрыт, но с него пытаемся что-то списать — то возвращаем ошибку — и перевести в состояние Fact будет нельзя, но есть и исключения. Касса должна списать товар — который подбирается в чек — и списать товар в «минус». Это должно быть свойством Link Map, также как и алгоритм списания (fifo, lifo, среднее), а также формула преобразования. Бывают ситуации когда нужно создать запись типа -1\*сумму, или курс\*сумму, или итог

массива.

Попробуем реализовать триггер по товару.

```
params = {'19':[[201.code]],'20':[[203]]}
new_obj = create_object(16,1,location='contract',source={'class_id':197,'code':[[code]],'location':'contract'},params)
params_edit = {'227':new_obj.code}
edit_object(197,[[code]],1,location='contract',source={'class_id':197,'code':
[[code]],'location':'contract'},params_edit)
result = 'ok'
```

получили несколько ошибок в первой части.

Также появилась необходимость в программном получении ID класса

```
params = {'17':[[code]],'19':[[201.code]],'20':[[203]]}
source = {'class_id':197,'code':[[code]],'location':'contract'}
new_obj = api_funs.create_object(16,1,'contract',source,**params)
result = str(new_obj)
```

В истории нет данных о создании записи, если пользователя с таким ID не найдено. (такие функции должны быть достаточно жесткими — по отношению к параметру пользователя)

[[user\_id]] – возвращает id пользователя

Результатом функции является какой-то *QuerySet* [<ContractCells: ContractCells object (None)>, <ContractCells: ContractCells object (None)>, <ContractCells: ContractCells object (None)>, <ContractCells: ContractCells object (None)>]

Хотя в этом случае хочется получить созданный объект

Итоговый триггер выглядит так

```
if str([[227]]):
    params = {'17':[[code]],'19':[[201.code]],'20':[[203]]}
    source = {'class_id':197,'code':[[code]],'location':'contract'}
    new_obj = api_funs.create_object(16,8,'contract',source,**params)
    params_edit = {'227':new_obj[0].code}
    source={'class_id':197,'code':[[code]],'location':'contract'}
    api_funs.edit_object(197,[[code]],1,'contract',source,**params_edit)
    result = 'ok'
else:
    result = 'false'
```

из ошибок — такой триггер не работает из-за того что нельзя проверить реквизит на «пустое значение»

if not [[227.code]]:

```
    params = {'17':[[code]],'19':[[201.code]],'20':[[203]]}
    source = {'class_id':197,'code':[[code]],'location':'contract'}
    new_obj = api_funs.create_object(16,[[user_id]],'contract',source,**params)
    params_edit = {'227':new_obj[0].code}
    source={'class_id':197,'code':[[code]],'location':'contract'}
```

```

api_funcs.edit_object(197,[[code]],[[user_id]],'contract',source,**params_edit)
result = 'Create new'

else:
    result = 'Update'

```

и он одноразовый. Мы создаем объект и прописываем в текущем объекте связанный реквизит, чтобы иметь на него ссылку. После создания — мы его не редактируем.

Модифицировать его можно — если научимся проверять на **непустое значение** связанной ссылки.

Но пока сделаем так

*if not [[227.code]]:*

```

params = {'17':[[code]],'19':[[201.code]],'20':[[203]]}
source = {'class_id':197,'code':[[code]],'location':'contract'}
new_obj = api_funcs.create_object(16,[[user_id]],'contract',source,**params)
params_edit = {'227':new_obj[0].code}
source={'class_id':197,'code':[[code]],'location':'contract'}
api_funcs.edit_object(197,[[code]],[[user_id]],'contract',source,**params_edit)
result = 'Create new'

```

*else:*

```

params = {'20':[[203]]}
source={'class_id':197,'code':[[code]],'location':'contract'}
api_funcs.edit_object(16,[[227.code]],[[user_id]],'contract',source,**params)
result = 'Update'

```

Что мы делаем в таком тригере. Создаем запись в контракте — складское хранение — передавая туда ссылку на товар из текущего массива товаров чека, и количество. Конечно параметры не полные — в складское хранение надо передать еще и склад, но этот реквизит не обязательный.

Исправим эту ошибку — потребуем чтобы этот параметр объекта стал обязательным. Система это пропустила. **Контракт обновился — не проверив обязательные реквизиты — а это ошибка.** Попробуем такое же провернуть с созданием объекта. Получили ошибку

Ошибка - *unexpected indent (, line 1)*

ошибка была в коде тригера — код был табулирован.

```

params = {'17':[[code]],'19':[[201.code]],'20':[[203]]}
source = {'class_id':197,'code':[[code]],'location':'contract'}
new_obj = api_funcs.create_object(16,[[user_id]],'contract',source,**params)
params_edit = {'227':new_obj[0].code}
source={'class_id':197,'code':[[code]],'location':'contract'}
api_funcs.edit_object(197,[[code]],[[user_id]],'contract',source,**params_edit)
result = 'Create new'

```

Убрал табуляции

получил ошибку

Ошибка - *'str' object has no attribute 'code'*

Видимо из-за того — что запись нового объекта в контракте не удалась и в строке

```
params_edit = {'227':new_obj[0].code}
```

реквизита *Code* не оказалось. Ошибка при создании объекта с недостающими параметрами не появилась.

*Отладить такое тяжело, если хорошо не знаешь — какие параметры обязательные — а какие нет.*

*Добавим склад в чек — но при получении данных по ссылке Собственник.Склад [[198.229]] — получаю*

*Ошибка. Вероятно ссылка ведет на недопустимый тип данных*

*В Api нужна функция — `get_object`. Причем желательно возвращать и `QuerySet` и `Json`, причем во втором варианте — хорошо бы возвращать `json` со всеми подчиненными массивами.*

*Обход выглядит так:*

```
from app.models import ContractCells  
o = ContractCells.objects.filter(parent_structure_id=195, name_id=229, code=1).values()  
result = str(list(o))  
  
получим  
[{"id": 801, "code": 1, "parent_structure_id": 195, "name_id": 229, "value": "1"}]
```

*Нам из этого нужно значение value*

```
result = str(list(o)[0]['value'])
```

*Итоговый вариант тригера будет выглядеть так*

```
from app.models import ContractCells  
o = ContractCells.objects.filter(parent_structure_id=195, name_id=229, code=[[code]]).values()  
storage = str(list(o)[0]['value'])  
  
if not [[227.code]]:  
    params = {'17':[[code]], '18':storage, '19':[[201.code]], '20':-[[203]]}  
    source = {'class_id':197, 'code':[[code]], 'location':'contract'}  
    new_obj = api_funs.create_object(16, [[user_id]], 'contract', source, **params)  
    params_edit = {'227':new_obj[0].code}  
    source={'class_id':197, 'code':[[code]], 'location':'contract'}  
    api_funs.edit_object(197, [[code]], [[user_id]], 'contract', source, **params_edit)  
    result = 'Create new'  
  
else:  
    params = {'20':-[[203]]}  
    source={'class_id':197, 'code':[[code]], 'location':'contract'}  
    api_funs.edit_object(16, [[227.code]], [[user_id]], 'contract', source, **params)  
    result = 'Update'
```

*Вот такая громоздкая конструкция должна работать когда мы хотим чтобы при подборе товара в чек — этом товар списался со склада, с учетом того — что есть возможность проверить реквизит на непустое значение. Плюс неудобство — что приходится держать лишний реквизит на форме — чтобы сохранить линк на созданный контракт, чтобы обновлять — если что- то изменилось. И так для каждого. В листе разработки есть `link map` (массив преобразований полей одного контракта в другой) и `trigger` (функции контракта, которые собирают пакет данных (может быть типа `draft`, `delay`, `fact` или `draft task` и `delay task`) для отправки в другой контракт).*

**Нужно удалить весь этот кусок — функционал LM, TR — реализован на ± 90%**

Еще одним серьезным вопросом — является работа со значениями получаемых по линку.

Подбираем товар в чек — и например через eval получаем цену на этот товар.

Контракт собираем так

Управление контрактами

ID	230
Название	Тест истории
Тип	Контракт
ID каталога	82

**Новый Сохранить Удалить**

Управление параметрами контракта

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
231	Дата и время записи	string		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
232	Товар	link	Тип: Справочник ID: 431 Название: Товары	<input type="checkbox"/>	<input type="checkbox"/>	
233	Цена	eval	1 result = [[232.789]]	<input type="checkbox"/>	<input type="checkbox"/>	

**Новый Сохранить Удалить**

ладно — соберем тестовый пример из другого контракта. Заменим товар на валюту контракта, а цену на курс.

Управление контрактами

ID	230
Название	Тест истории
Тип	Контракт
ID каталога	82

**Новый Сохранить Удалить**

Управление параметрами контракта

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
231	Дата и время записи	string		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
234	Валюта контракта	link	Тип: Контракт ID: 37 Название: валюты контрактов	<input type="checkbox"/>	<input type="checkbox"/>	
235	Курс при подборе	eval	1 result = [[234.40]]	<input type="checkbox"/>	<input type="checkbox"/>	

**Новый Сохранить Удалить**

Контракт "Тест истории" ID: 230

<input type="button" value="Поиск"/>	<input type="button" value="Найти"/>		
Код	Дата и время записи	Код	1
1	2022-07-20T12:15	Дата и время записи	20.07.2022 12:15
Страница 1 из 10		Валюта контракта	11 2022-03-14T04:55
Курс при подборе 55			
<input type="button" value="Новый"/> <input type="button" value="Сохранить"/> <input type="button" value="Удалить"/>			

Видно что сейчас в контракте — курс при подборе 55. - если я исправлю это реквизит в валюте контракта — **то он и тут обновится по ссылке**. Нам этого ну совсем не надо. Как раз для этой цели основным свойство контракта является «Дата и Время записи». Запрос eval поля должен быть привязан к этому свойству. Для этих целей — в справочнике по формулам есть параметр Version

Другие варианты — это код в TR (eval), который прописывает значение на момент создания очередного контракта.

## Исследование Delay/Fact.

Хочется разобрать эти свойства. Есть несколько примеров из жизни, хочется их привести, а потом постараемся их систематизировать.

Обычная ситуация.

Мы заключили договор на поставку товара. Документы об отгрузке нам послали по электронной почте (системе Электронного Документооборота или по другим каналам связи ), но факт остается фактом — документы к нам пришли быстрее чем товар. Дата отгрузки на этих документах есть, а товара на складе еще нет... Едет в транспортной компании, несет курьер, вообщем где-то, но не у нас. Получается что нам права на товар передали, а самого товара физически нет. Обычно бухгалтер такие документы запишет в свою систему — и будет считать, что раз права есть, то и все хорошо. Менеджер, имеющий информацию только о передаче прав — выписывает бухгалтерские документы об отгрузке, и вроде бы ничего не нарушает. За исключением того — что в складской службе этого товара нет.

Обратная ситуация.

Приходит контейнер на склад из-за рубежа. Документы на этот товар находятся в таможне. Документы на него — не сделать, продавать нельзя, товар не прошел таможенную очистку. Но складские операции на него — можно применять.

Контейнер, или даже товары можно перемещать — конечно в рамках законодательства страны, но как ответственное хранение вполне возможно — к примеру мы занимаемся обеспечением хранения в рамках таможенного оформления. Но получается что Прав на груз нет, Но мы можем с ним что-то делать, прав вроде нет, а перемещать можно.

Теперь разберемся с правами.

Есть лицензия на программный продукт (программное обеспечение). В соглашении явно прописано то-что можно с ним делать. Мы к примеру — продаем это ПО. В соглашении с производителем — явно указаны ограниченные права на перепродажу. Прав на собственность лицензия на ПО обычно не предоставляет. Для упрощения примем — что нам дали только права на перепродажу. В складской службе этого товара нет, максимум есть набор кодов(ключей) на бумажке. Значит перепродажа лицензии — это просто передача Прав. Но если мы продаем конечному покупателю — мы продаем ему права на

использование напрямую от производителя, согласно соглашения о перепродаже.

Такая же штука может происходить и с другими объектами системы. К примеру мы ведем учет акций, облигаций, и других финансовых инструментов. Конечно, такими вещами могут заниматься только специализированные компании, ведущими реестр держателей финансовых инструментов, но опять же — есть учет Прав на объект учета (как бы это не тафтологично звучало).

И обратная ситуация.

Предположим — наш бизнес, это хранение чужих грузов. Складской комплекс, который занимается перевалкой груза. Мы этот товар можем хранить, перемещать по разным локациям, и вообще говоря несем за него какую-то материальную ответственность (страховки, оценка при приемке - в разных бизнесах по разному). Права на объекты в таких бизнесах тоже ограничены. Продавать нельзя (кроме условий по договору хранения). Списания, пересортицы — не допустимы (опять же могут быть указаны в договоре условия форс-мажора).

Что из этих примеров хочется выделить.

**На любой объект учета есть набор прав.**

*Даже если мы наблюдаем и ведем учет космических объектов в небе Земли, тоже есть набор прав. Причем в этом конкретном примере прав на собственность не возникает, если мы не космическая империя, и не объявляли собственность на удаленные звездные системы.*

**Права на учет возникают автоматически при описании объектов учета.**

Права на преобразования, как и права на передачу объекта учета из системы появляются только при праве на собственность.

Однако — в бизнесах, которые явно занимаются преобразованием объектов учета (к примеру ремонтом), появляются ограниченные права на преобразование (разборка, ремонты, комплектация, прошивки и т.д.) и права на передачу объекта (отправка в специализированные ремонтные организации). Соответственно эти права — должны быть указаны в договоре, если они отсутствуют — то и ремонт не возможен (не разберешь и не починишь, производителю не отвезешь...).

Также права на собственность — могут быть ограниченными условиями договора.

Взять например Аренду. Временно передаются права на использование. Использование ограничено условиями договора.

**Также может быть указан перечень доступа лиц к объектам учета.**

**Нужна систематизация прав, для объектов учета, а их перечень нужно сделать свойствами объектов.**

Места хранения.

Присутствие у объектов учета физических характеристик требует организации хранения этих объектов учета в местах хранения. В каких-то вырожденных ситуациях — когда место хранения одно — вроде бы не возникает необходимость как-то называть, обозначать и выделять место хранения.

*Если наш бизнес — торговля с автолавки, то выделять склад вроде бы и не нужно. Все здесь, ~~всяляетя~~ лежит, все и так видно, да и учет как-бы не сильно нужно. С другой стороны — в голове есть четкая модель данных. Есть склад, где хранится товар, есть касса, есть понятие о расходах и доходах. Когда все закуплено, где и что лежит, выручка, продажи. После торговой сессии — все равно происходит инвентаризация, пересчет наличности, подсчет прибылей и убытков. Продвинутые — даже таблички ведут.*

Этот пример я привел, чтобы показать что даже в небольших учетных системах, для хранения которого используются мозг, блокнот с ручкой, или даже планшет — действуют универсальные законы построения учета.

Но вернемся к местам хранения. Когда объект учета имеет физические характеристики — вес, объем то вроде очевидно хочется их рассортировать, разложить в порядке удобства поиска — тогда появляются места хранения. Причем места хранения могут быть совершенно отличными от того, что мы привыкли считать местами хранения. Обычно Товар лежит на Складе, на полке в Торговом Зале, вообщем где-то лежит, причем это место имеет какое-то осмысленное название, чтобы его можно было там быстро найти. Но это не обязательно. Товар может кто-то снять с полки и положить его в карман\рохлю\корзину. Значит местом хранения может быть любой другой объект, не обязательно типизированный как место хранения, это может быть конкретный сотрудник, неопределенный клиент, курьер, транспортная компания и прочее.

При передаче товара контрагенту в качестве демообразца (договорились о выставке нашей продукции у клиента) или в ремонт — мы (частично) теряем информацию о физическом месте хранения объекта учета (особенно если это торговая сеть). А местом хранения становится контрагент — которому мы передали этот товар.

Конечно, нам ничего не мешает добавить в систему места хранения контрагента, особенно если нашу продукцию — можно приобрести, и выстроить систему пополнения этих остатков. Но обычно в таких (агентских) договорах есть всякие отчеты о реализации и прочие навороты. Чаще всего дают товар на хранение без информации о том — где он лежит. Значит место хранения без детализации это Контрагент (с помощью контракта на хранение товара Контрагентом).

Есть еще один пример. Сдаем мы наличные деньги в банк (операция инкассации). Физические вообщем-то объекты (купюры) с конкретным местом хранения мы передаем для нашего расчетного счета в банке. Объект учета (купюры) — при такой операции «разрушаются» и превращаются в объект чистого права и меняет место хранения (сейф-«инкассатор» - расчетный счет). Местом хранения становится договор с банком. Если мы не ведем покупческого учета, то при инкассации мы оперируем суммовым учетом — при котором разрушения объекта учета (в данном случае абстрактные «Деньги») не происходит.

*Эта операция напоминает производство из наличных денег в безналичные, со всеми возможными издержками (банковские комиссии, оплата инкассаторов) — как и в обычном производстве. Любая «трансформация» объекта учета при смене места хранения — порождает какой-то результат (в частности финансовый) в том числе и нулевой.*

Попробуем понять — что же такое место хранения. Гипотетически хочется сказать — что место хранения — это всегда контракт, детализация этого контракта — на зоны, ячейки, коробки, могут порождать субконтракты, но это зависит от системы хранения этих контрактов. **Контрпримеры приветствуются**. Еще хочется отметить что место хранения неявно диктует сам объект. Понятно, что в зависимости от «отношения к работе» договора можно хранить в общей куче бумаг вместе с черновиками, или вообще раскидать по офису, или можно аккуратно и скрупулезно разобрать эту кучу в папочки, файлики, сделать реестр документов, и тогда вроде бы удобно все это находить. Т.е. неявно учет бумажных копий договоров просит «складское» оборудование. И если объектов учета небольшое количество, то вроде бы место хранения не оборудовано, и даже не имеет выделенного места хранения и имени — ну кроме слова - «где-то там поискать надо», то на потоке, когда активно надо рыскать в библиотеке, то хочется место хранения усложнить для организации поиска. Реестр договоров, разбранный по типам этих договоров может лежать в одной папке с закладками, датами, номерами листов и прочих фишках ускоряющих поиск

## **Контракты — универсальный механизм, который позволяет разнотипные места хранения свести в одну структуру.**

Это надо еще раз пояснить, поскольку это звучит не очевидно, плюс закрепить те примеры, которые были описаны выше. Возьмем какой-нибудь физический объект, из тех что можно пощупать руками — возьмем теннисный мячик. Где он может находиться? Да в принципе везде, где он может лежать и сам не укатится. На полу в комнате, в кармане у ребенка, в сейфе (ценный очень, с подписью теннисиста известного), у товарища, который на корт уехал, в упаковке в шкафу, вместе с другими такими-же новыми мячиками. Основное — что тут хочется отметить — то что места хранения тоже — физические объекты, о которых мы помним. В тот момент, когда мы про место хранения забудем, то мы частично можем потерять информацию о самом объекте. Забыли — что мяч отдали товарищу, и мы его больше не найдем, причем мы точно будем помнить что объект такой есть, но информация о его месте хранения будет определяющей для учета. Забыть о месте хранения — значит потерять объект. Объект еще не разрушен, не потерял своих свойств, он существует в этой вселенной, но раз мы не имеем информации о его месте хранения то его вроде бы как и не существует и его надо разрушить. В системах в которых место хранения задано «жестко» — так и происходит, но раз объект существует и «просто» место хранения его не определено, то это неправильно. Как раз из-за того — что место хранения «неопределено», то его не надо уничтожать. Завтра придет товарищ и вернет мячик. И место хранения снова будет восстановлено.

Еще один пример. На складе произвели инвентаризацию и не нашли какой-то товар. Обычно за этим следует какой-нибудь Акт Инвентаризации — и списание товара, которого не обнаружили. Однако при следующей инвентаризации — он загадочным образом снова нашелся. Выяснилось — что он никуда и не пропадал, просто находился в другой локации (или у сотрудника на руках — по логистическим причинам), снова Акт Инвентаризации и Оприходование излишков. Если бы мы могли вести учет товара не только на складе, но и по сотрудникам, контрагентам, транспортным компаниями и вообще любым физическим местам хранения включая самые фантастические — к примеру — объект застрял в телепортации, то из системы учета объект не нужно было бы удалять.

К слову о телепортации. В банковском приложении мы оформляем перевод безналичных денег, пусть для этого примера мы с одного расчетного счета (своего) переводим деньги в другой банк на свой же расчетный счет. Срок перевода пусть составляет какое-то количество времени. Что это за время, и что происходит с местом хранения? С одного расчетного счета деньги ушли, на другой счет еще не пришли. Пользоваться ими нельзя пока объект «в пути». Местом хранения становится «неопределенность».

**Замечание:** Права на деньги вроде и есть, но воспользоваться этими правами нельзя, не очевидно, но получается что объект находящийся в Delay не имеет свойств Прав, кроме теоретического права на чтение.

В таком случае пользуемся термином Delay (Отложить). Объект до окончания периода неопределенности хранится в этом самом Delay. Предположим что деньги все-таки пришли, но не полностью. Банк (условно, для примера) взял комиссию, и перевел нам деньги с вычетом комиссии. Получается что в Delay застряло часть суммы. Чтобы из неопределенности деньги перестать ждать, нужно определить неопределенность и - из Delay перевести в Fact зафиксировав комиссию в издержках.

Еще один пример. Резерв товара на складе. Мы определяем товар на складе в рамках контракта, что на него есть ограниченные права — например запрещаем продавать товар всем, кроме конкретного клиента, или вообще блокируем товар на уход (запрещаем на него любые действия). **Введение на контракты свойства прав и их управление — позволит писать**

**меньше кода.** Для контракта — это свойство массива. Закидываешь в массив объект, и на него уже другие права.

**Замечание:** Система прав никак не влияет на сотрудников склада, которые этот товар переносят из ячейки в ячейку, или вообще могут отправить его в другое подразделение, получается что ограничений там, где не дотягивается система автоматизации мы не можем наложить.

**Замечание:** Технологически при проектировании процессов необходимо выстраивать цепочки — запрос, ответ, в наших терминах (тригер отправитель)->delay(надо сделать)->fact(тригер сделано)->(отправитель). Резерв товара — это fact в рамках контракта по которому идет резерв — delay возникает по контракту склада — в виде задания на терминал оператора склада, (возможно товар зарезервированный нужно переложить в ячейку «резерв»), после окончания задания — оператор склада сообщит контракту с которого поступило задание о успешном, частичном или неуспешном выполнении задания.

Попытаемся обобщить то что мы понимаем под Delay и его теоретические типы/виды.

**Delay Plan (План)**— это когда вам нужно что-то сделать **точно так как вы запланировали**, но что-то другое мешает вам это сделать, и вам нужно подождать, пока это не произойдет. (роботизация события).

Пример: Хотите поставить в резерв товары по контракту, запрашиваете остатки товаров и выясняете что товар в пути и планово он может появится через несколько дней. Вы оформляете резерв товара с планируемой датой на момент прихода товара. Система проверит наличие в указанное время и если все приехало, внесет изменения в fact. Либо, если дата прихода неизвестна, но есть условие проверки на остатки, то система будет ждать с момента создания Delay до появления товара на нужном вам складе с установленной периодичностью.

**Delay Forecast (Прогноз)** — Вы подготавливаете прогноз, и вам нужно следить за его выполнением. Алгоритмически (или с помощью нейросети), или вручную — вы смогли подготовить прогноз для производства, и эти данные вы хотите учесть при закупках к будущему сезону. В контрактах, основанных на прогнозах, вы можете посмотреть на будущее, а также оперативно отслеживать отклонения Fact от прогноза. При заданных параметрах отклонения прогноз может стать невалидным, и система может предложить вам подготовить новый прогноз. Прогноз никогда не переходит в Fact, он предназначен для отображения будущих данных за границей Plan. Нужно это добавить в формулы — что-то такое: [[table|contract.id\_class.id\_object.timestamp.id\_requisite.fact|delay|forecast]]

**Delay Draft (Черновик)**— вы что-то не доделали, и оставили на время, чтобы вернуться к этому позднее. Помещая в Draft вы гарантируете себе, что контракт и объект не будет использован в целях учета, это объект за пределами системы. Также можно использовать Draft при загрузке данных извне, в случае необходимости ручной обработки загруженных данных (при несовместимостях при обмене).

Пример: Оператор создал Платежный документ для банка. Но он хочет чтобы она сегодня не попала в выгрузку, и вообще стала доступна завтра как fact. Записывая платежку — она помещает ее в Draft с Датой и временем исполнения на завтра.

Draft также должен использоваться для сохранения данных перед транзакцией — реквизиты объекта сохраняются сначала в draft, затем применяем транзакцию. Это позволит не потерять данные — если транзакция не завершилась. У Draft есть возможность передать его другому пользователю. Так как он сохраняется и доступен только текущему пользователю — его можно переслать например руководителю, если не хватает прав такое применить.

Таблица хранения данных совпадает с типом объекта сохраненного в draft (table, contract). В

случае изменения Data-driven структуры — тоже должна быть обработана, а также включена в историю.

### Delay PPA (prior period adjustment) (Корректировка прошлого).

Пример: По договоренности с клиентом вы вносите изменения в документах в прошлом. Но так как Fact можно исправить только сейчас — то delay такого класса каскадно вносит изменения в реквизиты delay с прошлой датой.

Object.property	Fact	Delay PPA (delta)	Delay timestamp	Delay Type
Quantity	1	-1	now-1(day)	Корректировка накладной

**Delay Wait for result (в работе)**— вы сделали все что от вас зависит, но остальные еще не закончили — и вы ждете результат. Это общий признак контракта, который каскадно проверяет все реквизиты на наличие delay на всех подчиненных реквизитах, не позволяет контракт удалить, завершить или сдать в архив. В этом признаком можно собирать интегрально какие delay сейчас не закрыты на контракте.

Пример: По контракту вы подготовили все документы, оплата зарегистрирована и товар едет на доставку. Все что зависит о менеджера — выполнено. Контракт вроде может быть передан в архив. Но доставка еще не завершена, следовательно «закрыть» контракт вы не можете, так он заблокирован с помощью delay от логистики.

**Delay InOut**— вы знаете что будет, но это еще не произошло, вы отражаете это как запланированный факт, но откладываете на будущее. Планируемая надстройка системы для хранения оперативной информации по изменениям delay в контракте. Как пример — материальная ведомость. Такое можно сделать и с помощью delay draft — но это создаст новый объект или контракт, а нам это сейчас не нужно. Или с помощью delay plan — но он не позволяет оперативно — как свойство контракта получить расхождение между планом и тем, что в действительности произошло. Попытаемся понять почему такое может понадобится.

Пример: Загружаем курсы валют с расчетами на завтра. Торги уже прошли. Центробанк сказал — что это уже факт. Следовательно сегодня мы получаем fact на завтра. Можно ли это использовать как Fact сегодня? В формулах, в истории эта информация доступна, отчеты можно построить на завтра, но это не Fact - Fact — это оперативная информация, которая имеет актуальную информацию на сейчас, не в прошлом, не в будущем — а только на сейчас и не может быть редактируема в прошлом, а будущее еще не наступило.

Что же такое Delay In/Delay out? Информация полученная о курсах с расчетами на завтра — это Delay In. Это плановое значение, но, что если завтра в результате форс-мажора курс переиграют, или вообще отменят торги? Значит, теоретически в Fact завтра может попасть что-то другое — отличное от плана. Это Delay Out. А информацию об этом нужно знать и учитывать.

Для учета расхождения Планируемых значений и Фактических значений — вводим такой delay.

Он позволяет посчитать расхождение между тем что мы запланировали, что в итоге попало в Fact.

Пример: Мы по датам запланировали оплату поставщикам, но в течении недели фактически оплачивали меньше плана. Затем оплатили больше. Управлеңец отвечающий за финансы — это видит оперативно. Неделю платили меньше, потом заплатили больше, но остались в рамках плана. Или превысили план, по какой-то причине. Эту информацию терять нельзя.

Такой же пример можно дать и по продажам. Везде где есть возможность установить план выполнения — там же мы должны посмотреть как этот план выполняется, и есть ли расхождения.

Delay InOut Выглядит так:

Object.property	Fact	Delay in	Delay out	Delay timestamp	Delay Type
-----------------	------	----------	-----------	-----------------	------------

Так как в таком Delay могут быть расхождения между In и Out значением — то необходимо в таком delay держать существенную информацию. **В текущем релизе такой тип Delay реализовывать не будем** — это **функциональная надстройка над delay** — для оперативного промежуточного хранения материальной ведомости текущей истории изменения delay контракта — которая будет доступна из контракта без перехода в историю.

Давайте практически предложим систему хранения delay и fact. Посмотрим на такую таблицу:

Delay Plan

Object.property	Fact	Delay	Delay timestamp	Delay Type
Name	Товар 1			
Name		Товар 2	Current date+3	Замена ценника 1
Quantity	10			
Quantity		1	Current date-1	Требуется инвентаризация 1
Quantity		1	Current date-2	Требуется инвентаризация 2
Amount	100			
Amount		+10	Current date-1	Требуется инвентаризация 1
Amount		+10	Current date-2	Требуется инвентаризация 2

Что в этой таблице видно?

Наименование у товара поменяется через 3 дня. За это отвечает Delay с именем Замена ценника 1.

У нас есть 10 товаров, на сумму 100 рублей. Но сегодня позвонил наш клиент, который обнаружил, что вчера и позавчера при поставке товара ему товар то ли не положили, то ли найти его он не смог. И получается, что возможно товара у нас больше, и надо бы провести инвентаризацию и если найдем — то отправить клиентский товар, а если не найдем — то возможно нарвемся на конфликт с клиентом.

Что еще тут видно? Видно, когда у нас началась проблема с учетом этого товара, можем это учитывать в товарных отчетах. И пока заявку на инвентаризацию не отработаем, мы можем считать время с начала появления delay в прошлом, что в некоторых бизнесах важно, до его разрешения. Ну к примеру мы выяснили, что месяц товар не пересчитывался, для управляющего склада — это может быть важной информацией. Для менеджера важно знать — что делать дальше, если выяснится что действительно была недоставка товара. А также

свойство `delay` позволяет планировать задачи на будущее. Как в нашем примере — мы создали `delay` — замена ценника 1. Значит что через 3 дня система либо сама должна заменить название, а сотрудник в торговом зале за эти 3 дня должен подготовится к этой замене, либо сотрудник по истечению срока должен подтвердить, что замена ценника произошла, тем самым `delay` превратить в `fact`.

Где такое еще можно применить? Рассмотрим ситуацию. Менеджер увидел что контракт его оплачен, и можно сделать документы для отгрузки. Документы готовы, и тут клиент сообщает, что он передумал что-то из спецификации контракта брать, и просит заменить какой-то товар на другой. Все это может происходить с каким-то времененным лагом.

Подробнее. Менеджер подготовил все документы на отгрузку, и передал их в складскую службу, сборка происходила условно какое-то время, и (условно) через пару дней менеджеру сообщили что все собрано и ждем появления клиента. Клиент приезжает — общается с менеджером и сообщает что он решил заменить один товар на другой. Если прибыль (`profit`), считается по моменту создания отгрузочных документов, то по идеи нужно их исправить и дать команду складу — о замене товара. Но корректировка происходит в текущее время, и значит нужно внести изменение и в прибыль по отгрузке по тем документам что были два дня назад, а команду складу надо дать тоже в текущее время, чтобы довольный клиент получил то, что обсудили.

Это делаем с помощью `delay` в прошлом. `Fact` мы изменять не можем, так как склад уже получил заявку и все сделал, следовательно историю ломать мы не имеем никакого права. Но в альтернативной реальности `delay` — мы убираем один товар, вместо него ставим другой товар, склад получает заявку, что пару дней назад надо было сделать что-то другое, но текущим временем. Прибыль корректируется с помощью `delay` превращаясь в `fact` текущим временем.

Выглядит запутано. Но в документы которые нужно передавать в бухгалтерию — `delay` создает заявку на корректировку задним числом — **заявка это draft** — если мы ждем ответ на запрос. Прибыль в `fact` изменяется сейчас, либо если это `draft` — то после подтверждения заявки.

Посмотрим на таблицу `profit` без расшифровки по товару (по сумме)

	Now - 2	Now - 1	now
fact	100	100	90 (-10 корректир.)
Delay (корректировка)	-10	-10	

Мы пару дней считали что `profit` = 100, а в момент отгрузки выяснили что `profit` = 90

В управлеченческом учете все считается от факта, и изменения в прошлом не допустимы. Но в других системах есть условная возможность исправлять (синхронизировать) документы в «задних числах».

Даже если в управлении нужно будет подготовить данные с учетом исправлений, то пользуясь свойством `delay` — это можно показать пользуясь формулой  $\text{State} = \text{delay} + \text{fact}$ .

Но получается, что не всегда этой формулой можно пользоваться.

Пример.

Object.property	Fact	Delay	Delay timestamp	Delay Type
Quantity	10			
Quantity		1	Current date-1	Требуется

				инвентаризация 1
Quantity		10	Current date-1	Требуется внутристорождское перемещение 1

По факту товара 10 штук.

Менеджер создал delay (запрос на инвентаризацию). Но в это же время Заведующий складом создал delay (задачу на перемещение внутри склада).

Если мы просуммируем все delay – получим в State = -1

Получается что суммировать delay можно только определенного вида, или иметь возможность объединять delay по задачам.

Менеджер может не знать, что запросов на инвентаризацию по этому товару уже несколько, а зав. складом решил переместить его (товар) в ячейку для целей инвентаризации.

Следовательно — запросы на инвентаризацию можно склеивать - но вот суммировать под вопросом.

Попробуем привести другой пример чтобы не опираться на численные реквизиты.

Служба поддержки пользователей приняла 4 заявки о том что пропала сеть на рабочих компьютерах. Причем две заявки отправил один и тот же пользователь, плюс специалисты принимающие заявки были разные.

Итого 4 заявки, 3 пользователя — 3 специалиста службы поддержки. Все заявки передаются в службу связи. Работа началась с первой заявки, и была установлена проблема на коммутаторе. После ремонта по первой заявки были обнаружены и остальные заявки. Их все закрыли фактом ремонта, основываясь на диапазоне времени в которое они появились.

Однако. Пользователь который оставил две заявки — сообщил что у него были проблемы не только со связью, а была еще проблема с запуском приложения и попросил восстановить свою заявку и провести работы на его рабочем компьютере.

Выводы: Delay можно закрывать с помощью Fact - группой, а еще нужна возможность восстанавливать delay ошибочно закрытый ранее. Способ, чтобы не исправлять fact задним числом всего один — это новый delay (возможно — копирование информации из закрытой заявки). Причем тип у такого Delay будет другой — корректировка Fact, чтобы не ломать статистику закрытых заявок, по примеру таблицы с Profit.

Дополнительный вывод. У Delay есть стандартизованные типы. Из наблюдаемых примеров можно выделить:

1. **Delay PPA «Корректировка Fact в прошлом»** имеет обязательное свойство (datatype) и устанавливается на конкретный реквизит объекта или контракта. – позволяет внести и учитывать дельту изменений с момента в прошлом и до текущего момента. Может вызвать каскадный пересчет остальных установленных Delay в прошлом. Корректировка Fact **без datatype** – говорит о том, что Delay не порождается, так как изменение происходит текущим временем, соответственно — такое не допустимо.
2. **Delay Plan «Плановое изменение Fact»** имеет **необязательный** реквизит datatype, **необязательное** поле типа eval, где хранится условие при котором может произойти изменение, а также **необязательный** реквизит «период опроса» в минутах. – Корректировка, которая произойдет в будущем. Может быть как автоматической, когда решение по изменению произойдут по истечению времени указанном в datatype с заданной периодичностью при

наступлении описанных условий в delay, так и требующей пользовательской обработки, если время не указано.

Пример (готова реализация в справочниках, есть вопрос к обновлению формы — приходится обновлять страницу):

The screenshot shows a software interface with a header bar containing 'Версия 25.07.2023 15:41:00 Алексей Титаренко', 'От 25.06.2023 15:44:55' with a calendar icon, 'До 25.07.2023 15:44:55' with a calendar icon, and a button 'Обновить таймлайн'. Below this is a timeline slider with a blue dot at the current position. A 'Код 2' field is present. The main area displays a table with columns 'Наименование', 'Fact', and 'delay'. The first row shows 'Шанцовый инструмент (большая лопата)' in the 'Fact' column and 'delay' in the 'delay' column. To the right of this row is a small box labeled 'delay-date' with the value '25.07.2023 15:48'.

Указываем в конструкторе справочника — что у поля Наименование есть свойство Delay. И в интерфейсе появляется возможность указать — что в будущем, по наступлению времени события (delay-date), нужно заменить «большую лопату» на просто «лопату». Сохраняем элемент и ожидаем время события.

Если нажать на ссылку «delay» то видно какие изменения запланированы.

The screenshot shows the same software interface as above, but with a modal window titled 'Отложенные значения' (Delayed values) open. This window lists a single entry: 'Дата-время выполнения' (Execution date-time) '25.07.2023 15:51', 'Значение' (Value) 'Шанцовый инструмент (лопата)', and 'Подтвержден' (Confirmed) checked. There is a 'Закрыть' (Close) button at the bottom right of the modal.

Результат:

The screenshot shows the software interface after the delay operation. The timeline slider now has a blue dot at the position '25.07.2023 15:53:18'. The 'delay' column in the fact table now contains the value 'дд.мм.гггг --:--' (dd.mm.yyyy --:--) instead of 'delay'.

3. **Delay Forecast (Прогноз)** — Функционально — это Delay Plan, с **обязательным** полем datatime, но без возможности применить это значение в будущем. Позволяет получать аналитику с учетом прогноза и фиксировать расхождения fact от прогноза (Для будущей разработки — можно задать систему прогнозирования изменения реквизита в будущем — алгоритмически или нейросетевым образом, и на основе прогноза строить аналитические контракты для будущих периодов). Отличие Forecast от Plan — в вероятности. Если Plan с высокой вероятностью произойдет, то прогноз — это оценка аналитика на будущие события и его вероятность гораздо ниже. Расхождение влияет на будущие прогнозы и на планы.
4. **Delay Draft «Черновик (Draft)»** - это общая функциональность любого объекта, контракта. Система должна предлагать включать этот режим при попытке записать объект с незаполненными обязательными полями, либо оставить возможность исправить — возможно допущена пользовательская ошибка. А также можно разрешать созданный объект выводить в Draft, если ссылочная целостность основана на link map контракта, и графу

**ссылок (eval)** это позволяют.

Эта функция аналогична при удалении объекта. Хранятся такие объекты в **других структурах данных** (для Справочников в DraftObject, для Контрактов в DraftContract), чтобы не запускать их в систему учета и не давать их подбирать в других объектах системы. Триггеры не должны работать. ID присваивается по порядку Draft. Также включить Draft можно и вручную, но только если объект создан, но не сохранен. После выключения режима Черновика объект удаляется из Draft и помещается с новым ID в справочники или контракты. Структура данных объектов в Draft аналогична структурам данных объектов учета и синхронизируется также. Также в Draft попадает информация при неудачной транзакции, и объект можно перезаполнить из такого Draft, чтобы избежать потери информации при создании объекта. В интерфейсе списков - объекты draft помещены в отдельном меню с информацией о количестве объектов (это кстати можно сделать и с обычными объектами системы- в меню справочников можно видеть количество объектов в справочниках или в контрактах не заходя в него).

Пример:

Справочник "Складское оборудование" ID: 617

Код	Наименование	Тип оборудования	Остаточная стоимость	Ост. стоим. в евро
2	Шанцовый инструмент (лопата)	Расходные инструменты	100.0	0.99
1	Погрузчик	Движущаяся техника	756000.0	7510.71

Страница 1 из 1 [10]

Наименование \*      Факт delay  
Шанцовый инструмент (лопата) черновик delaydate  
Тип оборудования  
Расходные инструменты  
Инвентарный номер  
PS:0000018  
Дата инвентаризации  
dd.mm.yyyy -->--  
Состояние удовлетворительное  
Остаточная стоимость  
100  
Ост. стоим. в евро  
0.99  
Поставщик оборудования  
1  
ООО "АВ Сталь-Мет"  
Акт приемки  
Акт приемки товара.prg  
Обзор Скачать Удалить  
Новый Сохранить В черновик Удалить

Черновики можно использовать для брошенного редактирования. Сохранил в черновик и он будет лежать там, пока к нему не вернешься

Справочники Контракты Задачи (0) История

Конструктор справочников Справочники Черновики (2) -

Черновики справочника "Складское оборудование"

Код	Наименование
2	Шанцовый инструмент (лопата) черновик

5. **Delay InOut** – функциональная надстройка и общая функция контракта для промежуточного хранения истории реквизитов в delay сгруппированных по типу (**для будущей разработки**)

6. **Delay Wait for result (в работе)** — Предопределенная функция контракта, отслеживающий все незавершенные Delay текущего контракта и массивов — возвращает значение типа bool, и json с расшифровкой в каких реквизитах delay не завершен.

Замечание: Delay нужно включать как свойство при создании или редактирования реквизита контракта, и он в зависимости от переданной даты меняет свой тип - для будущих изменений — на Plan, для прошлых на PPA. По умолчанию — выключен. В интерфейсе — при редактировании полей с таким свойством будет происходить запрос дополнительных реквизитов например в модальном окне. Также в интерфейсе нужно иметь возможность видеть значения реквизитов установленные в delay вместе с временем рядом с фактическим значением — пример на рис. в описании Delay Plan.

ID	Название	Тип	Код значения	По умолчанию	O	T	D
618	Наименование	string		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Настройки планирования значений      Ответственный: 0      Автоподтверждение задачи:

В текущей реализации — нет отдельного разрешения на работу с корректировкой прошлого и запланированного значения в будущем. Также не реализована работа с группой ответственных. Он либо есть, либо пустой. Шаблон тоже надо довести до нормального состояния. Блок настроек появляется по галке D.

Есть еще один момент, при работе с delay – значениями.

Рассмотрим пример: Если есть цена на товар, с установленным признаком delay – то в случае корректировки цены в прошлом — есть опасность, что такая корректировка изменит цены в контрактах в прошлом, что может повлиять на целостность условий действующего, или даже закрытого контракта. Понятно, что в контрактах есть контроль условий, и цена может не изменится, и вообще после такого может откатиться весь каскад изменений цены, но как вообще можно поступать если такое изменение необходимо? Объект, участвующий в контракте уже «застывший», и применение изменений извне вроде бы невозможно.

Рассмотрим разные бизнесы.

Изменение цены в магазине «задним числом» - говорит о том, что ценники нужно было поменять условно вчера, и все чеки, которые были пробиты до включения delay — оказываются некорректными, мало того — ценники еще будут какое-то время меняться от текущего момента, до исправления, а продажи все равно пойдут по старой цене при разруливания ситуаций с клиентами. Что может получить менеджер, когда начнет анализировать такую ситуацию? Он получит список чеков и выяснит сумму расхождений за весь период такого delay. Возможно — эту сумму можно выдать в виде штрафа/премии тому сотруднику, который «забыл» поменять цену раньше, но вспомнил и поменял ее задним числом, или зафиксировать потенциальный убыток/прибыль из-за такой ситуации. Но такую же информацию можно получить и без таких экстремальных для деятельности магазина действий.

Вывод: Должна быть возможность для свойства delay в случае PPA — устанавливать возможность ограничивать изменения в прошлом (блокировать такие изменения).

При контрактных продажах в оптовом бизнесе — такое изменение в контрактах возможно при обсуждении с клиентом. Мы закупаем какой-то товар, возможно он даже принят на склад, но потом наш поставщик сообщает о замене документов с понижением цены на какой-то товар. Обычно управляющий такие ситуации готов одобрить, цена то уменьшилась, даже если мы оплатили больше и на взаиморасчетах у нас появится переплата. Хуже ситуация если замена документов происходит с увеличением цены, но управляющий такое тоже может одобрить, но может и отказаться от сделки и сделать возврат. Такие вещи, как правило усугубляются, если мы уже часть товара продали, по факту не зная корректную себестоимость товара.

Замечание: Delay PPA может вызвать каскадный пересчет — который может привести к изменению результата в delay реквизитов контракта, даже закрытого. И соответственно фактические результаты контракта будут расходиться с delay результатами как раз на величины в Delay. **При проектировании контрактов, там где важно пользоваться фактической информацией — как например — подобранная цена не должна изменяться — то в eval полях нужно пользоваться формулой с получением fact, чтобы изменения delay не приводило к пересчету контракта.** Однако бывают ситуации, когда такое может понадобиться.

Вывод: Delay PPA должен предоставлять пользователю исчерпывающую информацию об изменениях, которые он собирается применить и возможно, требовать участия другого пользователя (подпись другого сотрудника). (В интерфейсе — это свойство «Ответственный»)

Пример: Контракт системы с помощью которого мы собираем баланс должен быть всегда основан на Fact, и его исправление задним числом не допустимо. Но его Delay-часть показывает все корректировки, которые были внесены пользователями задним числом. Следовательно управляющий всегда видит уровень ошибок пользователей при работе с балансовыми объектами системы в виде метрики корректировок (сумм в балансе). Суммарный баланс по State = Delay + Fact — покажет информацию с корректировками, что в некоторых ситуациях приближает его к бухгалтерскому балансу в котором корректировки в прошлом допустимы. Расхождения будут только в случае Delay Plan, но Delay Plan это уже инструмент управленческого учета, позволяющий в данном примере построить будущий баланс с учетом планирования и прогнозирования.

Исследования по работе с Тех процессами.

Рассмотрим упрощенный контракт по передаче товара.

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
277	Номер контракта	string		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
275	Дата и время записи	datetime		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
278	Товары в счете	array				
280	Товары переданные по счету	array				

Массивы «Товары в счете» и «Товары переданные по счету» идентичны.

Управление контрактами

ID	278
Название	Товары в счете
Тип	Массив
ID каталога	274

**Новый** **Сохранить** **Удалить**

Управление параметрами контракта

ID	Название	Тип	Код значения	По умолчанию	Обяз.	Видим.
279	Собственник	link	Тип: Контракт ID: 274 Название: Передача товара	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
282	Товар	link	Тип: Справочник ID: 431 Название: Товары	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
284	Количество	float		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Новый** **Сохранить** **Удалить**

И вообще говоря являются техпроцессами контракта. По введенным названиям неявно есть логика о том — где тут вход в тех. процесс — где выход. Но структурно, на уровне описания класса — этого не понять. Также есть очевидные недоработки на уровне интерфейса. Я могу сделать массив, который является частью техпроцесса, но не указать в нем все реквизиты объекта который уже описан в стартовом блоке «Товары в счете».

Нужно понимать что базовый объект в контракте — это элемент массива (связанной подчиненной таблицей). Контракт без массива возможен, если в нем нет техпроцессов. Так скажем — вырожденные контракты, или аналитические.

**Вырожденные контракты** (чтение+запись) — места хранения данных без тех процессов (Стадий ТП) (не обладают переносами, стадиями, не могут быть черновиками... что-то типа таблицы остатков).

**Аналитические контракты** (чтение) — это контракты, которые собирают данные из системы, предназначены для разных отчетных форм.

Но и вырожденные контракты и аналитические могут быть расширены до обычных.

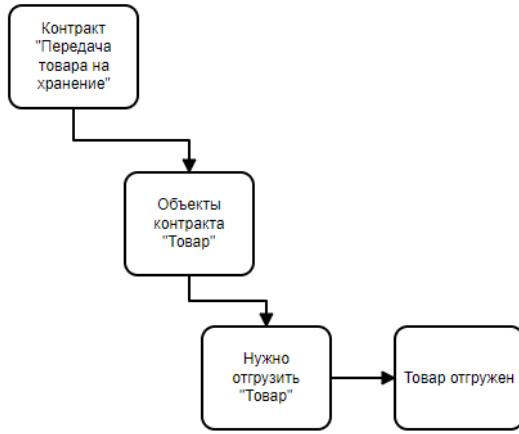
Что еще — в предложенном примере напрягает. При описании класса контракта — мы дублируем реквизиты в массивах — а должны были указать конкретный объект описанный в контракте.

Следовательно в контракте мы должны описать составной объект, который будет перемещаться по тех. процессам сохраняя структуру и состав своих реквизитов.

Действительно — заключая договор мы всегда описываем предмет договора. Это может быть спецификация услуг, товаров, прав которые мы передаем и получаем, а также определяем порядок оплат.

Обычно в при разработке систем учета мы тратим массу времени для описания реквизитов, которые используем в операциях над этими объектами, а также пишем код для трансляции (преобразования) этого объекта при перемещении из одного физического места хранения в базе данных (таблицы) в другое. Но нам достаточно описать объект контракта со всеми реквизитами и допустимые операции в виде поименованных стадий тех. процесса.

Этот сложняк попробуем проиллюстрировать



У нас есть контракт — в котором есть все реквизиты «договора». Дата, номер, стороны и другие дополнительные реквизиты его «шапки».

В массиве подчиненном контракту — мы формируем спецификацию такого «договора» - описываем ссылки, реквизиты, указываем количество — и получаем объект учета контракта — в данном примере «Товар». Отмечу — что этот **товар**, это другой объект, чем тот, который например лежит в справочнике «Товары». У него появляются дополнительные реквизиты и такой объект локализован в данном контракте, за счет набора этих дополнительных реквизитов.

Дальше нам нужно определить доступные операции для этого контракта. В данном примере — мы указываем что стадий у нас две. «Нужно отгрузить» и «Товар отгружен». Мы не случайно определили, что вход в на уровень стадий проходит в строгом порядке. Объекты контракта «Товар» - мы используем в качестве ограничения выборки для того, чтобы нельзя было в стадии тех. процесса произвольно менять спецификацию — только то, что есть в контракте. Ну и соответственно дальше в стадиях мы используем ссылки на «Объекты контракта», и не дублируем реквизиты, что исключает ошибки переноса объектов по стадиям.

Проиллюстрируем реализацию:

Создадим контракт — в примере называем его «свежий».

Название	свежий					
Тип	Контракт					
ID каталога	171					
<input type="button" value="Новый"/> <input type="button" value="Сохранить"/> <input type="button" value="Удалить"/>						
Управление системными параметрами контракта						
ID	Название	Значение				
474	Условие выполнения	1				
475	Link Map	1				
476	Триггер	1				
<input type="button" value="Сохранить"/>						
Управление пользовательскими параметрами						
ID	Название	Тип	Код значения	По умолчанию	О	Т
289	Дата и время записи	datetime		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
556	коммент	string		<input type="checkbox"/>	<input type="checkbox"/>	
557	коэф	float		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
713	Активен	bool		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
6748	количество	float	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
6759	выч	eval	1 [[tp.171.'Создан'.Fact]]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
<input type="button" value="При просмотре"/>						
<input type="button" value="Новый"/> <input type="button" value="Сохранить"/> <input type="button" value="Удалить"/>						

И опишем для «количество» поле с ID 6748 пару ТП.

Первый:

Дерево контрактов	Управление контрактами	
Поиск <input type="text"/> Найти + <input type="checkbox"/> тесты контрактные ID: 4 + <input type="checkbox"/> Контракты с покупателями ID: 52 - <input type="checkbox"/> тесты для истории ID: 171 - <input type="checkbox"/> полный цикл контрактов - тестфайл ID: 485 - <input type="checkbox"/> логирование ID: 570 - <input type="checkbox"/> телепротоколы ID: 597 + <input checked="" type="checkbox"/> особые контракты ID: 28 + <input checked="" type="checkbox"/> новый временный контракт ID: 169 + <input checked="" type="checkbox"/> следующий временный контракт ID: 179 + <input type="checkbox"/> вам то мо ID: 258 + <input checked="" type="checkbox"/> свежий ID: 288 + <input checked="" type="checkbox"/> tp1 ID: 171 + <input checked="" type="checkbox"/> Баланс ID: 176 + <input type="checkbox"/> bbb ID: 6734 + <input checked="" type="checkbox"/> скрипт ID: 6739 + <input type="checkbox"/> ассоциированный ID: 6749 + <input type="checkbox"/> point ID: 6752 - <input type="checkbox"/> ран ID: 883 + <input checked="" type="checkbox"/> скрипт здесь ID: 31 + <input checked="" type="checkbox"/> второй ID: 33 + <input checked="" type="checkbox"/> третий ID: 35 + <input type="checkbox"/> валюты контрактов ID: 37 + <input type="checkbox"/> новый!! ID: 77 + <input type="checkbox"/> за ID: 541 + <input checked="" type="checkbox"/> 14741 ID: 551 + <input checked="" type="checkbox"/> Третье дерево ID: 367	ID 171 Название tp1 Тип Техпроцесс ID каталога 288 <input type="button" value="Новый"/> <input type="button" value="Сохранить"/> <input type="button" value="Удалить"/>	
Управление параметрами техпроцесса		
ID	Название	Значение
172	Условие выполнения	1 [[tp.171.'Создан']] >= 0
173	ЛинкМап	1 [{"name": "Создан", "handler": null, "children": ["В работе"]}, {"name": "В работе", "handler": 8, "children": ["Выполнено", "Отменено"]}, {"name": "Выполнено", "handler": 8, "children": ["Отменено"]}, {"name": "Отменено", "handler": 8, "children": ["Создан"]}]
174	Триггер	1
175	Стадия	1 ['Создан', 'В работе', 'Выполнено', 'Отменено']
<input type="button" value="Сохранить"/>		

Второй:

The screenshot shows a 'Search tree' on the left and several management forms on the right:

- Search tree (Left):**
  - Root node: тесты контрактные ID: 4
  - Child nodes:
    - Контракты с покупателем ID: 82
    - тесты для истории ID: 171
      - полный цикл контрактов - тесты ID: 465
      - логирование ID: 570
      - текущий процесс ID: 597
      - особые контракты ID: 28
      - новый временный контракт ID: 169
      - следующий временный контракт ID: 179
      - вам же ID: 258
      - своих ID: 288
      - тп1 ID: 171
        - валют ID: 176
      - бл0 ID: 6734
      - смит ID: 6739
      - аконtrack ID: 6749
      - ройт ID: 6752
    - Гран ID: 883
      - сюда здесь ID: 31
      - второй ID: 33
      - третий ID: 35
      - валюты контрактов ID: 37
      - новый ID: 77
      - ал1 ID: 541
      - 14741 ID: 551
      - Третье дерево ID: 367
- Management Forms (Right):**
  - Управление контрактами**: Fields: ID (176), Название (Валют), Тип (Техпроцесс), ID каталога (288). Buttons: Новый, Сохранить, Удалить.
  - Управление параметрами техпроцесса**: Table with columns: ID, Название, Значение. Rows:
    - ID 177, Название Условие выполнения, Значение 1
    - ID 178, Название ЛинкМап, Значение JSON array: [{"name": "создан", "handler": 7, "children": ["нал", "безнал"]}, {"name": "нал", "handler": 7, "children": ["доставка нал"]}, {"name": "безнал", "handler": 7, "children": ["доставка безнал"]}, {"name": "доставка нал", "handler": 7, "children": ["доставлено"]}, {"name": "доставка безнал", "handler": 7, "children": ["доставлено"]}, {"name": "доставлено", "handler": 7, "children": []}].
    - ID 179, Название Триггер, Значение 1
    - ID 180, Название Стадии, Значение JSON array: ["создан", "нал", "безнал", "доставка нал", "доставка безнал", "доставлено"]
  - Контрольное поле**: Value 6748.

**Сохранить** button at the bottom.

В ЛинкМап — указываем наименование стадии, ответственного пользователя и допустимые маршруты.

Посмотрим как это выглядит в интерфейсе контракта:

The screenshot shows the 'Contract' interface with LinkMap configuration for the 'Currency' object:

- LinkMap Configuration (Top):**
  - Количество: 9
  - Фильтр: количество = Создан + В работе + Выполнено + Отменен
  - Маршрутизация техпроцесса

тп1	Статус	LinkMap		
		State	Fact	Delay
Создан	5	5		
В работе	4	4		
Выполнено				
Отменен				
- LinkMap Configuration (Middle):**
  - Фильтр: количество = создан + нал + безнал + доставка нал + доставка безнал + доставлено
  - Маршрутизация техпроцесса

Валют	Статус	LinkMap		
		State	Fact	Delay
создан	3	3		
нал				
безнал	6	6		
доставка нал				
доставка безнал				
доставлено				
- Buttons at the bottom:** Выч (5), Новый, Сохранить, Черновик, Удалить.

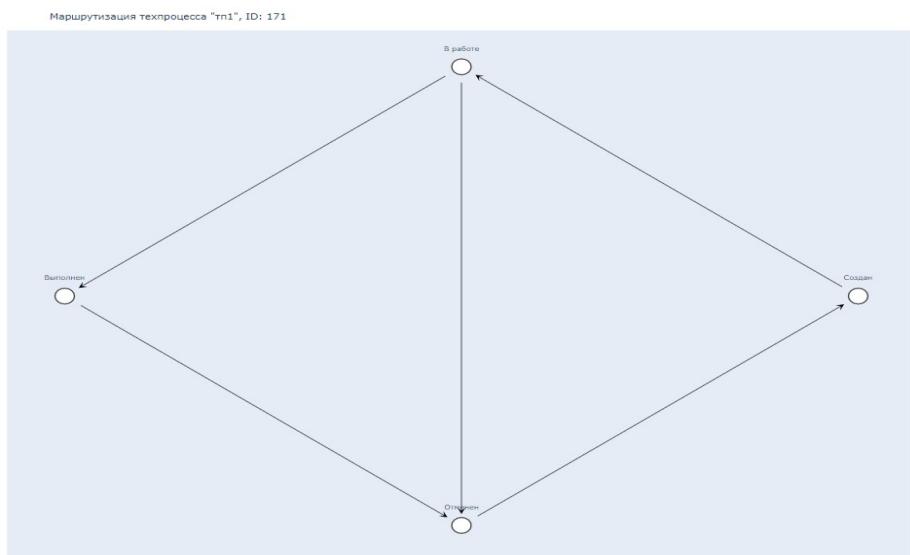
То, как мы это описали — хорошо работает на неделимых объектах. Т.е. таких, реквизиты которых не являются метрикой, или имеют возможность быть дробной (мерной). Например количество. Т.е. если в контракте — в объекте мы указали объект, у которого присутствует уникальное описание и например адрес — и он фактически всегда имеет меру 1 (уникальная штука), как пример — произведение искусства (картина) или право. То его нельзя по стадиям перемещать как 0,2 картины. И с этим мы сейчас прекрасно справимся. Проблемы начинаются когда у нас мера учета отличается от штучного (уникального) объекта учета.

Для этого — при описании объекта учета контракта — мы должны определить реквизит,

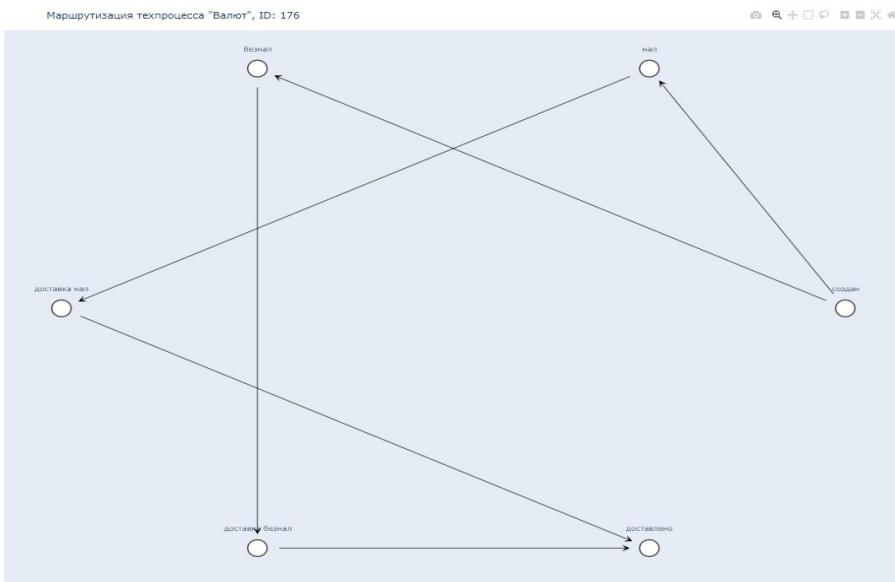
который является «делимым» (размерным), чтобы иметь возможность перемещать по стадиям например 1/10 объекта.

**Замечание и исследование мерных объектов:** Для мерных объектов есть такая штука — которая называется **квантование**. Это когда есть 10 штук в упаковке, или 10 упаковок — это коробка. Квантование в разных контрактах может быть разным. При работе с квантами на складах — неделимый квант — это коробка, в Контрактах для Оптовых поставок это может быть упаковка, в Розничных контрактах — это может быть штука. А когда хотим посмотреть аналитический контракт с остатками на складе — хотим видеть в какой то одной единице, как правило — меньшей неделимой — как в данном примере — в штуках.

Тут же можно посмотреть как будут маршрутизироваться значения — так, как мы указали это в ЛинкМап — нажимаем на ссылку «Маршрутизация техпроцесса»

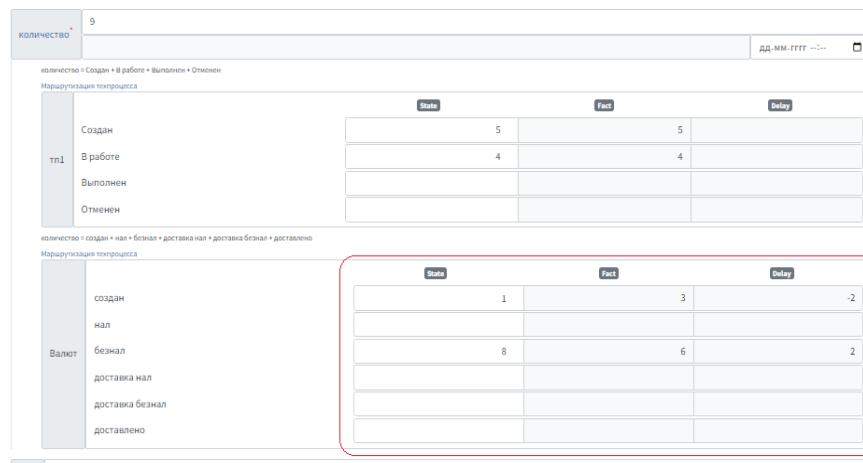


и второй



При редактировании значения в стадии ТП создается Delay Task или User task (Задача), которая в текущей реализации требует ручного подтверждения, если указан handler. Либо, если не указан, то Delay task не создается.

Пример:



ну и соответственно возникает Задача

Мои задачи					
Адресация	От меня	Этап	Не подтвержденные		
Код задачи	Локация	ID класса	Код объекта	Тип	Информация
433	Техпроцесс	176	9	Стадия	Пользователь ID: Алексей Титаренко запрашивает изменить стадию "создан" на величину -2.0. fact = 3.0, state = 1.0, delay = -2.0 Пользователь ID: Алексей Титаренко запрашивает изменить стадию "безнал" на величину 2.0. fact = 6.0, state = 8.0, delay = 2.0
					<input checked="" type="checkbox"/> Выполнить <input checked="" type="checkbox"/> Отклонить

Подтверждаем исполнение задачи. Кнопка - «Выполнить»

количество = Создан + В работе + Выполнено + Отменен

Маршрутизация техпроцесса

тп1		State	Fact	Delay
		Создан	5	5
	В работе	4	4	
	Выполнено			
	Отменен			

количество = создан + нал + безнал + доставка нал + доставка безнал + доставлено

Маршрутизация техпроцесса

Валют		State	Fact	Delay
		создан	1	1
	нал			
	безнал	8	8	
	доставка нал			
	доставка безнал			
	доставлено			

Успешно применилось.

Обычно в системах учета — квантование указывается в свойствах объекта в Справочниках. И для разных контрактов мы начинаем использовать эти данные оттуда. Но правильнее эти реквизиты для квантования либо использовать в режиме по умолчанию (предзаполнение, не влияющие на учет), либо явно указывать в контракте — какие единицы измерения квантуются и с каким коэффициентом.

Пример: В контракте мы описали «товар» в количестве 10 коробок, в которых 100 штук. При попытке передачи этой информации на склад — выясняем, что данный товар поставлялся в разных коробках, и количество упаковок в нем разное в зависимости от поставщиков. На складе при сборке обнаруживают 9 коробок в которых 150 штук. Квантование было разным, и это привело к конфликту. Конечно, такая ситуация — это ошибка оператора, который не отследил появление на складе других коробок с другим квантованием, но только с одной стороны. С другой стороны — если бы мы внесли информацию о другом квантовании в справочник «Товары» в момент появления новых коробок, или даже попытались исправить это в «прошлом», то это может повлиять на контракты — в которых явно указано прошлое квантование. Следовательно — если возникает такая ситуация — то обычно перестают использовать квантование в контрактах (все в штуках), либо переходят к другому виду учета по таким объектам (партионному, серийному, уникальному (маркированному)). Ну и в качестве вывода — использование реквизитов из справочников — предназначенных для разных контрактов допустимо только для физически одинаковых объектов. Если меняются физические характеристики вроде бы одинаковых объектов от поставки к поставке, то это либо другой похожий объект, либо у него другая система учета, у которой физические реквизиты связаны с появлением этих объектов при регистрации в системе.

Контракт самостоятельно отслеживает чтобы спецификация объектов контрактов по размерным реквизитам не превышала то количество — которое указано в спецификации. И если нам понадобится уменьшить спецификацию контракта, то сначала нужно убрать объект

из стадий, в которых он может находиться. Подробнее для этого примера выглядит так:  
Спецификация.Сумма(«Количество») >=НужноОтгрузить.Сумма(«Количество») +  
Отгружено.Сумма(«Количество»).

Отсюда видно — что добавление товара в спецификацию контракта не влияет на тех. процессы, а вот удаление товара из спецификации — невозможно без проверки и возможного отката объектов контракта из тех.процессов.

Замечания и исследования меры оценки. Одним из параметров учета является метрика оценки объекта учета. Это могут быть физические, бизнес — характеристики (оценочные) - метрики, которые позволяют приводить объекты учета из разных размерностей к более универсальным. Цена товара — позволяет понять количество денег (крышечек), или понять выгоден ли бартер.

Пример: Менеджер подготовил спецификацию с товарами и ценами, отправил предложение клиенту. Прошло некоторое время и клиент попросил увеличить количество в контракте. Но цена на товар уже увеличилась, и менеджер не может по этой цене заключить сделку. Хотя прошло всего к примеру несколько минут. Старая цена действует только на то количество товара, которое было в спецификации, так как это fact. Менеджер с клиентом попадает в ситуацию конфликта, решением которой может стать несколько вариантов.

1. Клиент соглашается оставить спецификацию без изменений.
2. Менеджер добавляет недостающий товар по новой цене, как отдельную позицию.
3. Менеджер удаляет товар из спецификации и подбирает заново, как новый с новой ценой.
4. Менеджер создает новый контракт, в который добавляет товар с новой ценой и недостающим количеством, чтобы в одном контракте не было одинаковых товаров с разной ценой.
5. Менеджер «усредняет» цену товара. Товар как объект в спецификации остается уникальным, но цена перерасчитывается «по среднему»
6. В данном примере это исключено, но бывают еще ситуации со скидками и договоренностями с отделом маркетинга, руководителем и т.д.

В зависимости от коммерческой политики в компаниях некоторые ситуации могут быть исключены, а где-то возможны все. Подробнее хочу остановится на варианте 3. Такая ситуация вполне нормальная, так как каждый объект (каждая строка) в спецификации уникальна из-за своих реквизитов, но обычнотогда усложняется контроль за перемещением таких объектов по стадиям контракта. В таких ситуациях в учетных системах обычно допускают ошибку при контроле остатков.

Пример:

На складе есть «товар» с количеством 1.

Менеджер отгружает этот товар, и имеет в спецификации «товар» с количеством 1, по цене 10, и еще один такой же «товар» с количеством 1 но ценой 11. По сочетанию реквизитов — это разные уникальные товары.

Контракт до транзакции проверяет возможность отгрузить такой товар по-объектно. Проверяем первую строчку — на складе есть 1 штука, отгружать можно, проверяем вторую строчку — на складе все еще есть 1 штука (транзакция не зафиксировалась, соответственно такой остаток...), и получаем конфликт со складом, когда в учетной системе списывается остаток в «минус».

Выход из такой ситуации — один. При проверке товаров — мы должны свернуть все товары

по количеству, игнорируя необъектные (контрактные) реквизиты — получив уникальный объект типа «товар» с количеством 2, и передаем эти данные в функцию business rule на проверку. Для данного примера — склад ответит что в наличии только 1, и поэтому откатит попытку отгрузки.

Вывод: Реквизиты предназначены для «усиления» уникальности, локализуемые только в текущем контракте, при передаче данных в другие контракты не учитываются — передаем не массив, а свертку (Итог) объектов по «мерным» реквизитам, тем самым добиваясь уникальности.

Разберем пример в случае варианта 3. В чем засада усреднения позиции. Дело в том — что значение в Fact — при первой инициализации (подборе) — попала информация о цене, актуальная на момент подбора. Далее мы снова подбираем этот же товар, и интерфейсно это выглядит как вариант 3. Два разных Fact значения находятся в разных объектах — это выглядит нормально и структурно правильно. Но нам нужно усреднить два объекта по одному реквизиту «цена». Напрашивается функция сворачивания — которая выполняет эту операцию, но получая среднюю цену — в Fact попадает нечто — которое невозможно подтвердить — откуда это взялось. И происходит потеря информации, на которую мы пойти не можем. Следовательно — при добавлении объекта в спецификацию при совпадении всех уникальных реквизитов — мы будем интерфейсно группировать эти объекты, а их реквизиты, которые являются «оценочными» будем выводить, как **группировки (slice)**.

Замечания по архитектуре. Реквизиты в спецификации объектов контрактов (как и при их движении по тех.процессу) у нас разделились на три группы.

1. Ссылки на объекты из справочников (или контрактов) + статические реквизиты, у которых нет свойства Мера дают **уникальные объекты контракта**, локализованные в текущем контракте и они являются уникальным ключом по которым мы можем проводить группировку.
2. Реквизиты — которые являются статическими (в частности — цены, и они как правило числовые, или составные, но позволяющие проводить операции преобразования для группировки) назовем **Группировкой (slice)** (она же метрика объекта). И при группировке будем их усреднять, но усреднение может зависеть от **связанных мерных реквизитов**. Вполне возможно что алгоритм усреднения может быть описан либо eval формулой, либо — что предпочтительнее — иметь возможность связать меру с оценкой жестко.
3. Реквизиты — которые являются количественным измерением объекта назовем - **Мера объекта**. По ним при группировке будем суммировать. Если нет ни одного мерного реквизита — то используется «скрытый системный» мерный реквизит равный 1. Данный реквизит также предназначен для расчета количества объектов в спецификации (Count).

Формулы — в которых фигурируют реквизиты типа Мера — также суммируются.

Действительно — если мы не хотим ничего усреднять, то и группировать ничего не надо. Если хотим сгруппировать реквизиты — то нам нужно иметь перечень полей — по которым будем группировать, но если есть реквизит, который не уникальный для группировки, но поддается вычислению — то группировать можно — усредняя значение этого реквизита в итоге группировки.

Группировка в интерфейсе включена по умолчанию, и спецификацию можно показать как сгруппированной, так и без групп.

Объект учета — это уникальная объект, и когда мы рассматриваем группу объектов учета — обычно — говорим что в области наблюдения существует группа похожих объектов, не

детализируя. Обычно, когда проектируется система учета, мы говорим что есть некая общность, по которой мы группируем эти объекты. Для нас — это ключ для первого приближения группировки объектов, каждый из которых физически уникален.

т.е Когда мы видим (**еще не реализовано**)

Товары		Редактировать		
Код	Товар	Цена	Количество	Сумма
1	Товар1	100.0	2.0	200.0

на самом деле это группировка такого вида

Товары		Редактировать		
Код	Товар	Цена	Количество	Сумма
1	Товар1	100.0	2.0	200.0
	Товар1	100.0	1.0	100.0
	Товар1	100.0	1.0	100.0

Где количество — это сумма уникальных объектов в группировке, его мера(measure). Цена — его группировка (slice), Сумма — это формула мера\*группировка (цена\*количество).

Это простой пример группировки. Но что делать когда в такой группировке уже хотя-бы больше сотни, тысячи, сотни тысяч уникальных объектов, которых мы не хотим хранить в системе, так как архитектурно не предполагаем индивидуальным управлением каждым объектом.

Тогда мы начинаем вводить квантование при описании объектов при их группировке.

Но сначала хочу обратить внимание на следующее:

Мера объекта — это коэффициент выраженный через другие объекты системы.

Если Мера отсутствует, то это говорит что объект выражается через самого себя — т.е уникальный и единичный.

Примеры:

Объект может измеряться:

сам в себе с коэффициентом 1 штука (название может быть произвольное). Штука = Штука = Сам объект.

в коробках с коэффициентом 10 штук. 1 Коробка = 10 Штук.

в контейнерах с коэффициентом 1000 коробок. 1 Контейнер = 1000 Коробок.

в цене закупки с коэффициентом 100 \$. 1 Цена закупки = 100\$

в розничной цене с коэффициентом 110 \$. 1 Розничная цена = 110\$

Это выглядит довольно просто – и есть определенный коэффициент пересчета, но как насчет такого:

1 Пара тапочек = 2 шт тапочек или 1 шт левая и 1 шт правая.

2 Пары ботинок = 1 пара белых ботинок

1 пара черных ботинок

или

1 телефон = 1 корпус

1 аккумулятор

1 материнская плата

1 экран

10 винтиков

Это тоже мера объекта, которая позволяет посчитать сколько можно телефонов поставить в производство например через delay plan или forecast. Замечу, что на данном примере видно — что объект может считать в других объектах. И меры можно вводить произвольные. Сложность тут в коэффициентах расчета — в оценке. Но об оценках поговорим позже.

Глубина квантования мер позволяет сделать более глубокой группировку объектов, либо наоборот укрупнить учет.

Вводя меру при описании объекта — мы всегда должны определить коэффициент меры и возможное дерево квантований.

Квантование для каждого объекта в контракте уникально, и от стадии к стадии может меняться.

И если нам поставляли молоко в упаковках по 10 штук объемом 800 мл каждая, то при поставке такой же упаковки в 10 штук объемом 750 мл есть расхождение в мере.

Обычно мы начинаем считать что это другой объект, даже если остальные свойства объекта не изменились. Рассмотрим, как это можно оставить в одном контракте. Отмечаем что «поставка» — это реквизит типа оценка с коэффициентом 1000 мл. = 1 л. и 1 уп. = 10 шт. а также 1шт = 750 мл и 1 шт = 800 мл. (Кстати, в зависимости от группировки мы можем переопределить оценку. Например введем что для поставки меньшего объема 1 уп. = 12 шт). Группировка (slice) — это реквизит автоматически или интерфейсно добавляющий меру для группировок.

Объект	Поставка	Мера объем (л)	Мера объем (мл)	Мера кол. (уп)	Мера кол. (шт)
- Молоко	775 мл {1 уп = {10 шт, 12шт} 11 шт; 1 шт = {800 мл, 750 мл} 775 мл; 1000 мл = 1 л}	34 л.	34000 мл.	4 уп.	44 шт.
	- 800 мл {1 уп = 10 шт; 1 шт = 800 мл: 1000 мл = 1 л}	16 л.	16000 мл.	2 уп.	20 шт.
		+8 л.	8000 мл.	1 уп.	10 шт.
		- 8 л.	8000 мл.	1 уп.	5 шт.
			800 мл.		1 шт.
			.....		.... (8 шт.)
			800 мл.		1 шт.
	+ 750 мл {1 уп = 12 шт; 1 шт = 750 мл: 1000 мл = 1 л}	18 л	18000 мл.	2 уп.	24 шт

Отметим — что все что считается через Меру — можно суммировать. Если значения в группировке как в нашем примере выглядят так:

1 уп = {10 шт, 12 шт} 11 шт

То мы должны определится — что мы хотим. Выбрать можно среди Max, Min, Average или Eval. Однако, если в наборе будет что-то такое

1 уп = {10 шт, 12 шт, {13 шт + подарочный сертификат}} — то усреднить мы это не можем, итог по такой группировке мы вычислить не сможем, пока не определим как считать

подарочные сертификаты вложенные в коробки, через коэффициенты меры. [Тут как раз спасает Eval.](#)

Несколько вещей надо пояснить.

В классических системах учета мы привыкли пользоваться вариантами типа:

Объект	Количество (K)	Цена (Ц)	Сумма (C)
Формула обработчика при изменении	$C = K \cdot Ц$	$Ц = C / K$ или $K = C / Ц$	
Итог группировки	СуммаГруппы(K)	СредГруппы(Ц)	СуммаГруппы(C)

Все поля в такой таблице являются полями ввода. А функции пересчета отрабатывают при изменении входных данных.

В нашей системе в простых случаях мы обычно делаем так

Объект	Количество (K)	Цена (Ц)	Сумма (C) Eval
Формула обработчика при изменении			$Ц = [Ц] * [K]$
Итог группировки	[K.all]	[Ц.all] / [K.all]	[Ц.all]

Но при таком подходе мы теряем возможность редактировать Сумму, так как оно не редактируемое - вычисляемое поле.

Как это обойти?

Рассмотрим пример:

В классе мы определяем что поля у нас численные и убираем итоги по колонкам.

Объект	Количество (K)	Цена (Ц)	Сумма (C)
Объект 1	10	10	100

Но нет никакой информации о взаимной связи колонок для пересчетов, формул нет. Вводить можно — только с калькулятором, на что пойти мы не можем. Да еще и итогов нет. Нам нужно где-то добавить информацию о том как считать и подбивать итоги.

Есть два варианта.

Можно сделать реквизит типа eval и в него написать функцию пересчета. Понадобится событие типа `on_edit` и возможность указать источник события, чтобы избежать постоянных пересчетов в несвязанных с `on_edit` реквизитах.

Другой вариант через группировки (slice).

Для Объект мы включаем свойство `slice`. И в реквизите `slice` задаем список преобразований.

1. { сумма = [цена] \* [количество] }
2. { цена = [сумма] / [количество] }
3. { количество = [сумма] / [цена] }

Вводим данные в поле количество. Смотрим в список вычислений, и находим все формулы в которой количество в правой стороне {1,2}. И в порядке обхода применяем. [Кажется, что избыточно пересчитывать группировку по второй формуле.](#)

Замечание: мы не указали как реагировать на разворачивание группировки например так: { [Объект.Количество] = 1 }, следовательно развернуть мы ее не можем.

Таблица стала выглядеть так. Вводить данные можно в каждое поле, и все пересчеты работают.

Объект	Количество (K)	Цена (Ц)	Сумма (C)
+ Объект 1	10	10	100

И при разворачивании получим тоже самое

Объект	Количество (К)	Цена (Ц)	Сумма (С)
- Объект 1	10	10	100

Если добавим в описание группировки параметр квантования  
 $\{ \text{[Объект.количество]} = 1 \}$  то при разворачивании получим:

Объект	Количество (К)	Цена (Ц)	Сумма (С)
- Объект 1	10	10	100
№1	1	10	10
....	....	.....	....
№10	1	10	10

Или например  
 $\{ \text{[Объект.количество]} = 5 \}$

Объект	Количество (К)	Цена (Ц)	Сумма (С)
- Объект 1	10	10	100
№ 1	5	10	50
№ 2	5	10	50

Мы можем купить 4 персика, измеряя их в штуках. Нам их взвешивают, пересчитывая их из штук в килограммы по коэффициенту меры (в данном случае цена за килограмм является коэффициентом пересчета из штук в вес). Коэффициент пересчета может быть функцией например ссылкой на eval.

Зачем нам такие сложности? Рассмотрим пример.

Мы закупили 100 000 одинаковых роликов бумаги для POS принтеров. Нам сообщили что все поместились в грузовую фуру. Исходя из истории поставок мы знаем что ролики могут быть упакованы в пленку по 8, 10, 12 штук в упаковке, В коробке может быть 6, 7 ,8 упаковок, Коробок на грузовом палете может быть 40, 50. И стоит задача — подготовить плановый расчет и передать план разгрузки в разные подразделения (контракты), и запланировать работу вилочного погрузчика, чтобы не перегружать технику с учетом предела максимального веса.

Задаем группировку

Объект.Количество = {

1 фура = {100 000} штук.

1 упаковка = {8, 10, 12} штук

1 коробка = {{5, 6, 7} упаковок; + 0,3 кг } +0,3 вес картонной коробки

1 палетт = {{40, 50} коробок; + 18 кг} // +18 вес палетта добавляем к весу коробок

1 шт = 0,250 кг

}

Такой вариант позволяет нам автоматически расчитать группировку с количеством и весом, а также раздать план для связанных контрактов.

Замечание: Так как мы точно не знаем как нам будет поставлено (какой будет fact) мы оперируем свойством delay plan, и по умолчанию предоставляем расчет исходя из тах значений, которые по fact можно будет переопределить или вручную выбрать из набора значений в группировках.

У нас указано

1 упаковка = {8, 10, 12} штук — система автоматически посчитает что в упаковке 12 шт. Но мы можем это переопределить, выбрав другое значение или сделать slice split (расколоть группировку — режем пиццу на куски). Группировка разрезается так, чтобы при выделении куска все остальное оставалось целым.

грубо округляю до целых в количествах.

Объект	1 фура = 100 000 штук	1 Палетт = 50 коробок	1 коробка = 7 упаково к	1 упаковка = 12 штук	Мера (палетт+1 8кг)	Мера (Коробка+ 0.3кг)	Мера (упаков ка)	Мера (штуки)
- Бумага для POS принтера	1				24;25800.3	1191; 25368.3	8334; 25000	100000
		- № 1			1; 1083	50; 1065	350;1050	4200; 1050
			- № 1			1; 21.3	7;21	84; 21
				- № 1			1;3	12; 3
				+№ 2-12			6;18	72; 18
			+№ 2 -50			49; 1043.7	343;1029	4116; 1029
		+№ 2 - 24			23; 24717.3	1141; 24303.3	7984; 23950	95800; 23950

Добавление группировок позволяет передать данные связанным контрактам. Для данного примера видно - что нам необходимо выгрузить из фуры 24 палетта, вес каждого не превышает 1083 кг в плане. Понадобится task на погрузчик с такой грузоподъемностью.

При разгрузке палетт — нужно будет работать с 50-ю коробками (в каком-то из палет будет меньше, и общее количество тоже видно) — каждая из которых весит 21.3 кг. Эти данные можно передать в task для задачи складской службе или наемной разгрузочной команде, как в количестве коробок, так и по весу.

Данные по упаковкам можно передать в службу логистики, если нужно разобрать коробки и организовать передачу упаковок в службу доставки.

Т.е. вводя slice – для объектов, можно данные о результатах использовать для plan в связанных контрактах еще до fact прихода в систему.

**Замечание:** Группировки сформированы автоматически, но значения в ячейках можно редактировать. Предполагается — что такая конструкция позволяет вводить Fact, соответственно, можно вводить и исправлять Delay Plan с пересчетом как сверху вниз, так и снизу вверх.

Объект может быть уникальным по всем своим характеристикам — и тогда он всегда одиночный и мера его вообще может отсутствовать. К примеру мы ведем учет прав, лицензий или услуг. Мера в таком случае может быть, но существовать в рамках каких-то принятых в данной предметной области правил.

Рассмотрим еще один пример.

Клиенту был подготовлен счет, и в него включили стоимость доставки. Товар отправили в транспортную компанию, затем клиент попросил добавить еще один товар в рамках действующего контракта. Также с товаром добавили еще одну услугу по обеспечению логистики с другой ценой (вес посылки был разный). Группируя Доставку мы увидим 2 доставки и усредненную цену. Но Клиент может попросить чтобы доставка была разбита на две строчки. А остальные товары — чтобы остались усредненными. **Значит в спецификации контракта должно быть системное поле — галка «группа» для всех объектов.** Т.е. свойство группировки - это свойство уникального объекта контракта.

С мерными реквизитами вроде все просто. Числовое поле контролируется контрактом, чтобы нельзя было провести операцию нарушающую целостность контракта и объектов в нем.

Рассмотрим пример.

У нас есть заданная спецификация объектов контракта. Товар, Количество. Эта спецификация условно-статическая (редко меняется). Если при проектировании стадий мы допустим ситуацию, когда последняя Стадия ТП является местом хранения данных для исчерпания другим контрактом, то это может привести к потере данных.

Ну или более подробно. В контракте, пример который мы давали ранее



в результате выполнения тех.процесса у нас массивы «Объекты контракта «Товар» и «Товар отгружен» будут идентичны ну или по другому *Договор.Товар = Отгружен.Товар*, что означает — что данный контракт выполнен. Но если параллельный (несвязанный) контракт начнет исчерпывать данные из «Товар отгружен», удаляя оттуда объекты и пересылая их дальше, то контракт потеряет информацию о том, что мы чего-то там отгружали (история конечно спасает от полного коллапса). Но потеря данных недопустима, следовательно — доступ к массиву данных извне разрешен только на чтение. **Запись и удаление в Стадиях ТП может быть произведена только текущим контрактом в своем контексте для обеспечения целостности.**

Такое требование выглядит избыточно, ведь появляется острое желание — использовать

стадии контракта, как промежуточное место хранения. К примеру — Стадию ТП «Товар отгружен» мы вдруг захотим считать контрактом «товар отгружен со склада и положен в машину» — который передаст объект в структуру Логистики, исчерпывая эту стадию. Но избыточность данных предназначена на повышение устойчивости и сохранения истории. Тем более что логистика не оперирует товарами контракта клиента. Они оперируют упакованными коробками, а это другой объект — по другому контракту.

Тоже самое касается и контрактов по Местам Хранения. Запись из другого контракта допустима в Спецификацию Складского Контракта извне, с учетом правил контракта приемника, но доступ к стадиям складской службы на запись и удаление — заблокирован. Стадия ТП к примеру может быть к примеру — «Заблокировано на инвентаризацию», и тогда при попытке списать со склада такой товар, Контракт ответит — что нельзя нарушать закон сохранения контракта, так как объект находится на инвентаризации в Стадии ТП (Такой вот вид складского резерва).

Еще один вид реквизитов контрактов — это приобретаемые реквизиты, как правило они являются **Трек - реквизитами** (для отслеживания). При движении по Стадиям ТП реквизиты могут быть добавлены к объекту, и локализуются в этой Стадии ТП.

Пример.

В счете есть реквизиты Товар, Цена, Количество (Суммы считаются с помощью eval, поэтому эти реквизиты не описываю). В Стадии ТП «Документы Бухгалтерские» по этому контракту нужен реквизит Код Таможенной Декларации. Причем на один объект этот код должен быть рассчитан согласно какого-то алгоритма (в данном случае fifo). Если в инвойсе Товар в количестве 10 штук, а Коды у нас уникальны на все 10 штук, то в стадии ТП «Документы Бухгалтерские» - должно быть 10 строк. Группировка, конечно работает как и везде в контрактах — но реквизит трека — не мерный и не оценочный, следовательно не сгруппируется.

Для примера

Спецификация контракта Счет

Товар	Цена	Количество
Товар 1	10	10

Стадия ТП «Документы Бухгалтерские»

Id документа (номер)	Дата и время документа	Товар	Цена	Количество	Код Таможенной декларации
1	20.08.21	Товар 1	10	2	01-02-03-04
1	20.08.21	Товар 1	10	5	05-06-07-08
1	20.08.21	Товар 1	10	3	04-06-08-10

Замечание: При переносе объекта по стадиям в которых есть функция с определением обязательных Трек-реквизитов — при получении queryset (набора значений), объект дробится по мерным признакам согласно выбранному алгоритму (например fifo, lifo) либо заставляет пользователя самостоятельно осуществить выбор и дробление.

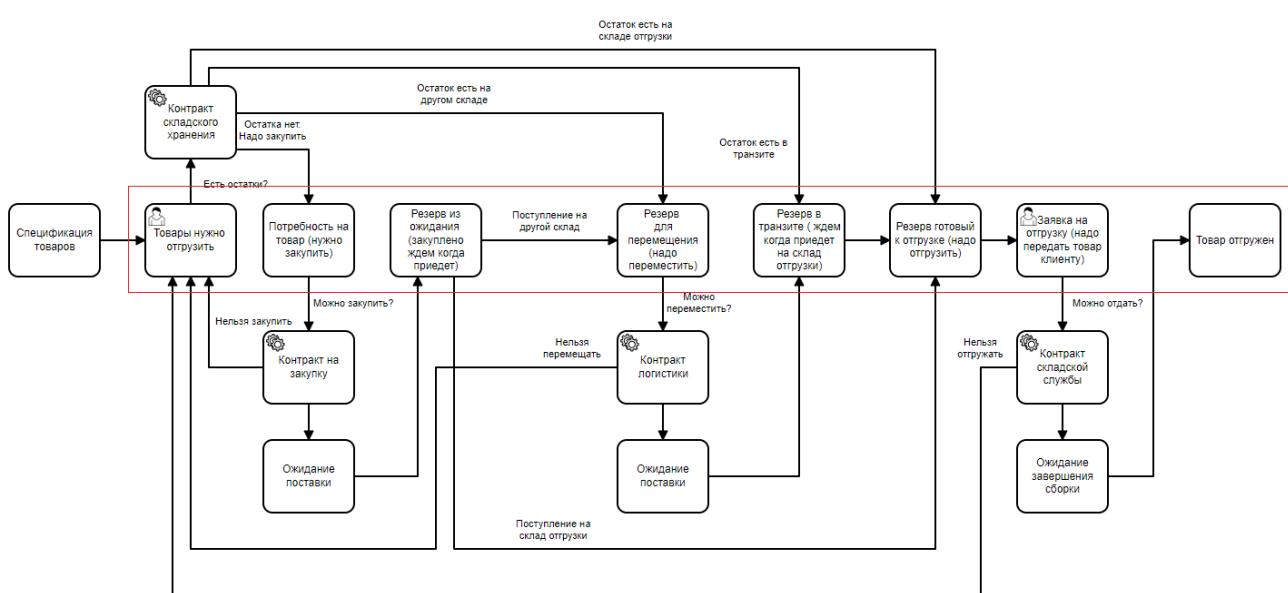
Выставляемый счет не содержит реквизитов, которые нужны в некоторых стадиях — для данного примера — создание бухгалтерского документа , таких как трековые

(идентификатор партии, таможенных кодов, кодов маркировки, серии поставки, срок годности), и они возникают как приобретенные — достигнув определенной Стадии ТП.

Как это все работает. Массив Стадии ТП всегда имеет ссылку на объект спецификации контракта со всеми общими реквизитами, а дополнительные реквизиты, мы указываем в массиве Стадии ТП, как бы «пристегивая» их к уникальным реквизитам расширяя уникальность и **вроде бы** локализуя их в этой стадии без передачи этого реквизита далее. Попробуем поискать контрпример, когда это не так.

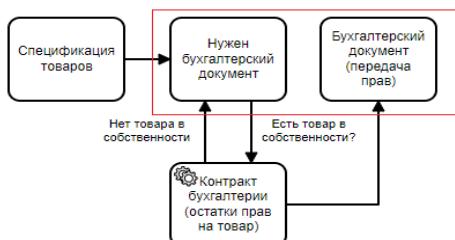
Предположим у нас есть Стадия ТП «Бухгалтерский документ», и нам захотелось отслеживать обмен подписью на таких документах. Добавляем стадии - «Отправлено на Подпись» и «Получено с подписью» (для упрощения без форс-мажорных стадий, типа «отказано», «утеряно», «отправлено повторно»), и в этих стадиях у нас совершенно спокойно доступны реквизиты из Стадия ТП «Бухгалтерский документ». Но, нужно отметить — что к самому общему контракту да и к объектам спецификации эти стадии относятся «слабо». Это подчиненные стадии к Стадии ТП «Бухгалтерский документ» - которая в этом случае является Спецификацией объектов. Что в свою очередь говорит о том, что перед нами независимый контракт — который работает на логистику доставки документов.

Следовательно - для более чистого чтения контрактов (организации доступа и нормализации хранения данных) - не нужно создавать сложные контракты в которых стадии превращаются в деревья с подстадиями и логикой контрактов в стадиях(как спецификации).



Рассмотрим пример сложнее.

Выделенные красной рамкой блоки — это Стадии ТП. Чтобы добавить реальности схеме покажем еще одну диаграмму



Это тоже Стадии ТП по тому же контракту. И этот пример говорит о том, что параллельные Стадии могут быть к одной спецификации. Расчет целостности контракта расчитывается по параллельным стадиям независимо.

Что еще видно на этом примере. Выглядит так, будто другой контракт имеет прямой доступ на запись в стадию. И если на диаграмме с контрактом бухгалтерии — это не совсем очевидно, хотя мы не можем явно перенести объект из стадии «Нужен бухгалтерский документ» в стадию «бухгалтерский документ» без контроля контракта бухгалтерии (прошу прощения за массовую тафтологию), но фактически не контракт пишет данные в крайнюю стадию, а пользователь с помощью автоматизации перевода из стадии в стадию.

Но в сложной диаграмме — это напрашивается - там таких примеров несколько, но возьмем крайний



Что хочется отметить — напрямую из других контрактов записывать данные в Стадии ТП нельзя. Дело в том, что запись осуществляется с помощью контракта, по бизнес правилам (BR) и таблице маршрутизации. Следовательно — Стрелка с «Ожидания завершения сборки» в «Товар отгружен» говорит только о том, что мы передаем данные в контракт в виде ответа для конкретного получателя.

*Вся система построена за логике «запрос» - «ответ». Иногда при проектировке контрактов — пользуемся терминами *delay* и *fact*, но это не свойства реквизитов — это как раз архитектура построенная на ожиданиях и ответах. И построение архитектуры через разные типы *delay* Запрос к *fact* Ответ — и позволяет вводить Стадии ТП — как разные варианты ожидания перед фактическим событием. Причем иногда фактическое событие опережает ожидания.*

Но вернемся к этому короткому примеру на рисунке. Что мы видим. Пользователь хочет отгрузить товар клиенту через заявку на склад. Но для того чтобы он это сделал ему нужно послать запрос в контракт складской службы и дождаться ответа о выполнении — или невозможности этого выполнения. Для того чтобы это сделать — ему нужно перенести объект в «Товар отгружен», но так как есть «связанный» (корреспондирующий) контракт, который осуществляет контроль — он может это сделать только через draft заявку со спецификацией объектов. Draft в виде неподтвержденной заявки попадает в контракт складской службы и одновременно в «товар отгружен». Когда этот Draft станет Fact (исполнится) в получателе — то одновременно исполнится и в Стадии ТП «товар отгружен».

Т.е. список объектов — находящихся в Стадии ТП «Заявка на отгрузку» - нужно передать на исполнение, одновременно поставив себе напоминание — о том что нужно ждать ответа контроля и исполнения. Состояние заявки, напомню, может быть в виде draft – delay – fact – cancel.

В данном примере мы отправляем заявку на склад в виде draft (можем и в delay – если обработка задачи автоматическая, если ручная — то draft). (Link Map позволяет нам транслировать его в правильном виде (совместимом) для складского контракта)

В следующую Стадию ТП «Товар отгружен» эта заявка — в виде списка объектов тоже попадают как draft task. Общее состояние task должно быть видимо в интерфейсе контракта

(это один и тот же объект с одним и тем же ID, но с параметрами корреспондирующих (связанных) между собой контрактов, а также подразделений и возможно конкретных исполнителей )

Когда склад начнет с ней работать — заявка переключится в Delay. И в Стадии ТП – draft станет Delay (в работе - WFR).

Когда Склад закончит — то заявка станет Fact — и в Стадии ТП заявка станет Fact.

Если что-то пойдет не так — на складе, то после стадии delay, или даже перед ней — заявка получит статус Cancel (должна быть причина отмены) (Это тоже draft, но не удаляемый — пока заявка в статусе Draft — ее еще можно удалить со стороны отправителя). Это позволит отслеживать ошибки на складе, и вести по заявкам в статусе Cancel учет, плюс выявлять задержки времени между переключениями стадий. От момента появления draft (заявка), до delay (в работе) и fact (завершено).

Теперь хочется поговорить о заявках (task — у нас это всегда **user task**).

Мы нигде не описали — что же это такое. Во всех наших примерах выше мы рассматривали ситуации простые, когда одно действие — приводит к изменению базы данных условно «мгновенно». Но при построении учетных систем не все можно автоматизировать, и за некоторые узлы тех. процессов отвечают люди, как средство принятия решения, или выполнения задач. Для того чтобы человек приступил к выполнению задачи — он должен получить задание на выполнение. Назовем ее Заявкой (Task). Заявки бывают срочными (с deadline) и бессрочными (datatime = null).

Для функционирования человека в бизнес-логике нам нужно средство позволяющее выстроить систему в режиме «запрос» - «ответ». Т.е. к человеку мы должны обратиться как контракту, передавая ему данные, и он взаимодействуя с интерфейсом — должен ответить системе.

*Далее под **объектом** подразумеваю объект справочников и объект контрактов.*

Однако — запрос человеку означает непрогнозируемое по времени событие, и если на «конвеерных» (нагруженных) системах мы можем пользоваться заявками класса delay plan (событие точно произойдет, и в прогнозируемом будущем или вернется с ошибкой) и не требуется approve на принятие в работу, то на «свободных» (ненагруженных) системах — заявку можно отправить в виде draft.

Немного технических вещей. Draft в системе — это **json с реквизитами и их значениями**, который нужно применить к существующему объекту/объектам, или создать новый объект системы. А также **отправитель и получатель** (может быть группой, списком, или совпадать с отправителем). Значения — которые в объектах указываются как обязательные — в draft можно не указывать и это означает — что когда мы захотим применить изменения — то это поле будет пустым для выбора. В ненагруженных системах учета, draft в интерфейсе показываются как список незавершенных объектов вместе (первыми) в списке объектов — отмеченных другим цветом. В нагруженных — Draft выделяются в меню как черновики объектов (New draft object, New draft contract) и черновики заявок (Draft task).

В примере ниже — на рисунке с меню проекта — черновики выделены для Draft New объектов, когда их количество превышает стандартную пагинацию на странице. (**Рис. прототип**)

json – это надо добавить это как системное поле (New, ID), вместе с отправителем и списком получателей. И соответственно отразить эти реквизиты в интерфейсе (Рис. прототип).

Справочник "Товары" ID: 431

Поиск Найти Export list Export object

Код	Наименование	Валюта учета товара	Отношение к возникновению	Квант перемещения	Артикул
	Черновики				
	Товар тестовый	Доллар США	Товар	1.0	2245
	Объекты				
2	Весы торговые ФорТ-T 586 (15; 2) LCD Азимут	Российский рубль	Товар	13.0	25547
1	Товар1	Доллар США	Товар	1.0	222

Страница 1 из 1 [10]

Версия Код Наименование Комментарий д/а Комментарий д/п

Draft в статусе New – содержит данные таким образом, что из него можно сделать объект — и поэтому он находится в списке объектов справочника первым (у отправителя, и у получателя). Т.е. **Draft New** локализован и доступен только у отправителя и получателя.

**Draft ID** (draft на изменение конкретного объекта) – может быть выбран из списка в объекте с этим ID и применен к текущему контексту, как будто — мы вводили эти изменения руками.

Доступные черновики

Наименование: Казахстанское тенге -> Казахстанский тенге  
Наименование: Казахстанское тенге -> Казахстанский тенге 05.09.22 12:18:33 x  
Курс: 90 -> 100 05.09.22 13:17:20 x

Версия 06.09.2022 13:15:33

Код 21

Наименование казахстанское тенге

Цифровой код 1441

Символьный код kzt

Курс 90

Дата обновления 06.09.2022 12:14

коэффициент 166

Внутренний курс

Тест

Новый Сохранить Удалить

В данном примере в Draft ID были одиночные реквизиты, но их там может быть несколько

(json)

**Draft New** — имеет полную структуру данных объекта, и возможно частично не заполненные реквизиты. Их располагаем вместе с объектами.

**Draft ID** имеет реквизиты, **только те** которые нужно изменить, и их следует поместить в другое место в интерфейсе.

Замечание. **Draft со временем может стать невалидным. Его значения могут стать неактуальными и реквизиты объекта могут измениться из-за внесений изменений в класс объекта.** При создании объекта из любого Draft — невалидные значения указываются в интерфейсе рядом с полем ввода для того чтобы указать их невалидность, а невалидные реквизиты (и их значения) — игнорируются.

Пример. Мы подготовили контракт, составили спецификацию, подобрали цены — и сохранили его для себя — как Draft New, чтобы вернуться к нему завтра. Но на следующий день цены поменялись и создать из этого draft — **новый** контракт уже нельзя из-за несоответствия этих цен. Нужно иметь возможность видеть невалидные реквизиты — и иметь возможность их исправить, так как применить draft можно только для fact (актуальных данных системы). Невалидные реквизиты конечно можно закинуть в delay, но это свойство подключаемое, и не во всех случаях используется. Значит, нужно иметь возможность **редактировать невалидные реквизиты до их сохранения, и не давать сохранять объект, пока есть конфликты с draft.**

Так как draft в себе содержит всю необходимую информацию (результаты link map, business rule, данные о Стадиях ТП) — то на валидность проверяются и эти «кэшированные» расчеты. У Draft времени нет. Его можно применить только текущим временем.

А теперь **Task** (заявки).

Заявка — это объект системы вида draft, основная задача которого обеспечить взаимодействие двух и более контрактов на участках бизнес-логики, требующих контроля пользователя, где автоматический перевод данных из draft в delay и fact без обратной связи не допустим (запрос-ответ). Для таких объектов в интерфейсе есть **отдельное меню и отдельный шаблон страниц, заявки группируются по контрактам получателям и в интерфейсе разделяются на draft, delay и fact.** Однако, в отличии от draft new или draft id он содержит id корреспондирующих контрактов и пользователей — все это заполняется с помощью контракта отправителя (ID отправителя, ID получателя, и возможно, куда отправлять ответ в случае отмены). Фактически, контракты используют task для организации обмена данными.

Пример.

У нас есть пара контрактов которые взаимодействуют между собой. Контракт на отгрузку и контракт на хранение товара на складе (Контракт склада). Для данного примера уточним, что данному складу требуется ручное управление заявками. Т.е. исполнитель — должен получить список товаров и подтвердить, что он принял это на сборку.

Контракт на отгрузку — между Стадиями ТП «Надо отгрузить» и «Передано клиенту» должен отправить заявку в виде запроса в Контракт склада, для получения остатка товара и подтверждения — что все есть в наличии. Контракт отправитель — формирует пакет данных состоящий из спецификации объектов и их количества, добавляет данные о контракте отправителе (и маршрутизации в нем), контракте получателя и куда вернуть ответ (может быть список контрактов — например контракт, который учитывает количество отказов со склада). Также в пакете данных можно указать конкретных пользователей или

группу получателей.

Получатель (пользователь, или группа) видит поступающие заявки и клиента стоящего в зоне выдачи и переводит их (кнопка в интерфейсе, которая называется Применить (approve) , и возможно указывает будущее время выполнения) в состояние delay plan. Во время перевода в delay из draft task в Контракт отправитель — отправляется ответ — и в следующую Стадию ТП «Передано клиенту» записывается delay plan по количеству и спецификации товаров, указанных в заявке. После окончания задания по складу и передачи товара клиенту — кладовщик (оператор, пользователь) отмечает все что он смог выполнить, и что не смог (частично) и закрывает заявку (переводит ее в fact). При переводе в fact заявка отправляет данные отправителю и все, что совпадает по заявке заносит в fact Контракта на отгрузку в Стадию «Передано Клиенту», все что не смог — возвращает delay plan из Стадии ТП «Передано клиенту» - в Стадию «Надо отгрузить».

Замечание: Если Business rule Контракта-отправителя настроен так, что если что-то по заявке не может быть выполнено (частичная отгрузка запрещена), то заявка не должна перейти в fact. Также клиент может отказаться от получения всех товаров, если он не получит все. В случае клиентского отказа заявку можно отменить, переведя ее в «Отменено» (Cancel). В случае невозможности перевода заявки в fact из-за бизнес правила контракта на отгрузку — заявку должен отменить отправитель.

Мы рассмотрели заявки в режиме Draft. Особенность этого типа заявок заключается в том, что архитектурно — эти заявки не влияют на систему до перевода их в delay. Такие заявки нужны например при выдаче банковского чека. Т.е. до тех пор пока чек не будет обналичен, ничего можно и не делать в системе. Конечно можно эту информацию занести в бессрочный delay и ждать списания с банковского счета, но нельзя исключать потери этого чека, или нежелания его обналичить, и в зависимости от ситуации — можно этим управлять.

Заявка может пропустить режим Draft. Значения и спецификация объектов сразу заносится в Delay Plan связанных контрактов и ждет перевода в fact или cancel. Если fact расходится с delay — можно как перейти в cancel, так и перевести в fact. Отличие в том — что ветки ответа для fact и cancel могут быть разными.

Небольшой итог.

Контракты могут общаться между собой и напрямую с помощью триггеров. Но надо понимать, что link map+ BR+ триггер контракта отправителя собирает тот же набор данных (json) что и заявка (task). И отправляет ее в другой контракт, где пакет данных пропускается через связку приемника lm+BR и выполняет внутренний триггер получателя (как будто мы вносим исправления руками). В случае возникновения ошибки — пакет данных отбивается с ошибкой — и отправляется отправителю (получает сообщение о невозможности провести операцию).

Подключение заявок для дополнительного логирования, и интерактивного взаимодействия между контрактами осуществляется на уровне триггера (отмечаем галку — что пакет должен быть отправлен, как заявка — и там определяем тип этой заявки (draft, delay) ).

Замечание. При создании draft task, хоть теоретически он не имеет timestamp, в реквизитах время его создания все-таки надо сохранять. Дело в том — что когда draft был создан - все его данные — валидны, но уже в следующую секунду могут «испортиться», особенно если

контракты высоко нагружены. Следовательно — при переводе task в delay вполне возможно что task сначала применится как delay PPA в прошлом — по времени создания task с переводом в delay plan.

Нагруженный пример: 10 000 менеджеров продают единственный товар (очень ограниченный перечень — ну пусть это будет крупный кирпичный завод) в системе. Допустим что 20 000 человек принимают товар с производства, а еще 20 000 человек выносят этот товар со склада. В одну и ту же секунду могут быть созданы десятки или сотни заявок на отгрузку, который нужно собрать, упаковать и погрузить. Счет времени в таких контрактах идет на милисекунды. А управляет склад - допустим вручную. Draft task непрерывным потоком валится складским командам (как-то распределяясь, чтобы все они не приходили к одному мега - координатору). Для организации очереди, планирования своего ресурса, склад, в этом случае, будет вынужден работать по определенным правилам и приоритетам. Утренняя заявка с большим количеством может быть выполнена не в приоритете, когда как мелкие — более поздние заявки могут выполниться быстрее (или наоборот). Возникает вопрос — как в этом случае предоставлять сведения о корректных остатках менеджерам, если контракты отправляют draft — который не влияет на текущие данные в контрактах. Опять же предположим, что стадий с резервированием в контрактах продаж нет и продажа — это фактически — горный поток.

Попробуем понять — спасет ли нас delay PPA и время создания заявки draft (с милисекундами).

State = (delay PPA + delay plan) + fact – остаток доступный для продажи

Fact – фактические остатки на контракте

Delay plan – происходит отгрузка

Delay PPA – остаток был на момент создания заявки

Draft – нужно отгрузить

Рассмотрим следующую ситуацию. В какой то момент были оформлены 3 заявки, и склад начал их собирать согласно своему нечеткому приоритету. Первую заявку они благополучно не стали брать, но собирают следующие две.

Task ID	Draft	Delay PPA	Delay plan	Fact	State (time line)
1 (draft)	700			800	800
2 (delay)			-200	800	600
3 (delay)			-100	800	500

Первая заявка на этот момент невалидна из-за State и взять ее сейчас в работу нельзя, так как она приведет к нарушению целостности контракта, хотя, если бы ее взяли в работу в порядке очередности — то могли бы исполнить. Получили конфликт — менеджер первым встал в очередь (пусть на милисекунду, но первым), но его заявку обработать сейчас нельзя.

**Можно ли воспользоваться delay PPA?** Попробуем. Предположим что склад завершил одну из заявок. И принял работу по первой заявке так, как будто он ей и занимался сначала. Контракт это позволит, так как это Delay, а fact – не разрушен (не ушел в минус).

Draft нельзя частично перевести в Delay.

Замечание: В зависимости от BR можно вообще запретить переводить невалидные заявки в delay. Отклоняем (cancel) — и проблема решена. Draft отмечается как отмененный — менеджер видит состояние заявок в своем контракте. Нужно снова делать заявку, и отправлять ее — вдруг получится.

В других ситуациях клиент может согласится на дробление отгрузки разными поставками, или соглашается подождать, пока подвезут товар.

И так. На складе 800 кирпичей. Две заявки выполняются, и хотя со склада мы еще ничего не вывезли - факта такого не было, свободных кирпичей осталось всего 500.

Закрываем третью заявку.

И приступаем к первой. Она первая в очереди, но просто команда начала выполнять ее позже, чем остальные. Заявка при переводе в Delay обязана сразу предупредить о наличии конфликта о недостатке ориентируясь на текущий state.

Поэтому сообщаем системе — что создаем его и в прошлом и планируем работу по складу

Task ID	Draft	Delay PPA	Delay plan	Fact	State (time line)
1 (delay)		-200	-500	700	0
2 (delay)			-200	700	-200

Сразу становится понятно — что первая заявка в приоритете, и надо приостановить вторую, так как она делает что-то не по порядку. С другой стороны — первую заявку взяли к выполнению позднее — поэтому вторую никто не будет приостанавливать. Получили конфликт внутри складского контракта.

В нашем случае, скорее всего, на складе примут решение откатить вторую заявку чтобы соблюсти очередь. Мало того, task по умолчанию - обнаруживая delay PPA – блокирует следующие заявки от перевода в fact, чтобы не допускать «влезание мимо очереди».

Но предположим худший вариант. Команды сборки разные, и никто общей картины не видит (упустили эту возможность) и контроль task на наличие ранних PPA отключен.

Замечание: *Delay PPA показывает что заявку невозможно выполнить полностью, хотя она и была создана ранее, но 200 штук правдивы из-за времени создания и порядка очереди, а 500 штук есть в наличии и они запланированы к отгрузке.*

*Менеджер получит обратную связь в контракте, о том что по его заявке могут быть проблемы. Есть неопределенность при ее выполнении и он видит что его заявка взята в работу, но выполнение ее полностью невозможно. (delay по его контракту выглядит как 200/500)*

Далее: Команда работающая по второй заявке уже успела погрузить наш товар в машину и хочет счастливо закрыть вторую заявку. И система должна это допустить — так как факт отгрузки есть.

Task ID	Draft	Delay PPA	Delay plan	Fact	State
1		-200	-500	500	-200

Заявка так и осталась не обработанной. На складе осталось 500 штук, но дальнейшие продажи невозможны — другие менеджеры видят что в остатках минуса (чего быть не должно) и начинают массово связываться со складом выясняя чего произошло. Получается

что Delay PPA нам мешает больше, чем помогает. Единственная польза от него — это помочь исполнителю понять, кто главнее в конфликтных ситуациях.

С другой стороны — склад «видит» что у них есть 500 штук. Менеджер видит, что его заявку поместили в «прошлое» - не могут они ее обработать, и когда они это сделают, тоже пока не знают (в теории — скоро — машина с кирпичом уже разгружается).

Если BR разрешает частичные переводы заявки в Fact то с заявкой они могут поступить так. Сначала устранием PPA. А остальное оставляем в бессрочном (или с указанием ориентировочного времени поставки) Delay plan.

Частичная отгрузка (если договорились с менеджером)

Task ID	Draft	Delay PPA	Delay plan	Fact	State
1			-200	0	-200

В контракте менеджера 500 попадут в fact, а delay на 200 будет ждать новой партии кирпича.

Если Delay PPA мы использовать не хотим, и очередность task нам не важна, то это все может сделать обычный delay plan через свойство позиции даты и времени. Тогда все происходит текущим временем и будет выглядеть так:

Task ID	Draft	Delay plan	Fact	State
1		-500 (Now + 30min)	500	0
1		-200 (ASAP)	0	-200

## Глоссарий

**Заголовок (header)** - Реквизит любого класса. Если представить класс в виде таблицы, то заголовками таблицы будут реквизиты класса, строки — объектами, а ячейки — реквизитами объектов. ID реквизита класса будет называться соответственно ID заголовка

## Примеры готовых конфигураций

В репозитории есть файл базы данных db\_coach.sqlite3. Подключите его в файле settings.py, и Вам будет доступен пример конфигурации “Тренер”.

**Назначение:** Учет работы спортзала. Предназначен для одного пользователя — тренера или администратора спортзала. Не предполагает личных кабинетов для клиентов спортзала.

По умолчанию доступен один пользователь admin (password: admin) – с правами суперпользователя, т. е. с возможностью конфигурировать систему.

Клиенты в системе организованы как справочники, а не как пользователи. Это единственный справочник в данной конфигурации.

первый в списке контракт — Счета (ID: 8). Объект счета имеет два реквизита — ссылку на клиента и количество средств на балансе.

ID	Название	Тип	Код значения	По умолчанию	O	T	D
13	Системные данные	string		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
14	Клиент	link	Тип: Справочник ID: 1 Название: Клиенты		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
15	Сумма на балансе	float		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Сумма на балансе не может быть отрицательной, что указано в бизнес-правиле счета:  $[[15]] \geq 0$ . У данного контракта условие завершения = False. Это означает, что этот контракт нельзя удалить.

### Управление системными параметрами контракта

ID	Название	Значение
9	Бизнес-правило	1 $[[15]] \geq 0$
10	Link Map	1
11	Триггер	1
12	Условие завершения	1 False

Для каждого клиента необходимо самостоятельно создать счет. Если счет не создан, то провести урок по типу оплаты «абонемент» не получится. Внесение средств на счет проводится тренером вручную при оплате клиентом абонемента на месяц или на больший период.

Следующий контракт — Уроки (ID: 17). Здесь каждая запись контракта — это информация об отдельном уроке. Ученики — подчиненный массив (ID: 27) контракта «Уроки». У каждого ученика есть реквизит «Тип оплаты» со значениями: «абонемент / разовое». Свойство влияет на стоимость урока для данного ученика (по абонементу имеется скидка) и на способ оплаты. Если выбрана оплата абонементом, то снимаются средства со счета ученика, если разово, то оплата не снимается ни с какого счета. Считается, что разовое занятие клиент оплачивает при входе.

Итак, порядок проведения занятия следующий:

- На странице «Уроки» [http://127.0.0.1:8000/contract?class\\_id=17](http://127.0.0.1:8000/contract?class_id=17) необходимо нажать кнопку «Новый» для создания нового урока. Укажите дату и время занятия в соответствующем поле формы управления объектами.

The screenshot shows the 'New Lesson' form. At the top, there is a header with 'Версия' (Version) set to '11.01.2024 15:37:14'. Below it are date and time fields: 'От' (From) set to '11.12.2023 15:37:14' and 'До' (To) set to '11.01.2024 15:37:14'. A button 'Обновить таймлайн' (Update timeline) is next to the 'До' field. A blue circular progress bar is shown below the dates. Underneath, there is a checkbox labeled 'Бизнес-правило' (Business rule) which is checked. Below that is a section for 'Дата и время создания' (Creation date and time) with the value '11.01.2024 15:37:14'. There is also a checkbox for 'Условие выполнения' (Execution condition). The 'Код' (Code) field contains '19'. The 'Ученики' (Students) section has a 'Редактировать' (Edit) button and a table with columns: № (№), Ученик (Student), and тип оплаты (Payment type). The first row shows '11.01.2024 09:00:00' under 'дата и время занятия' (Lesson date and time) and an empty checkbox under 'Урок проведен' (Lesson conducted). The bottom part of the form is partially visible.

Затем сохраните урок, обратите внимание, чтобы чекбокс «Урок проведен» не был отмечен. При отсутствии учеников невозможно провести урок, поэтому без них объект контракта «Урок» не создастся.

2. После создания урока в форме управления объектами найдите п. Ученики. Нажмите кнопку «Редактировать» рядом со ссылкой «Ученики». Откроется модальное окно с объектами учеников. Создайте необходимое количество учеников (тех, кто посещает данное занятие), указав коды ученика и код урока. Для каждого ученика укажите тип оплаты урока — абонемент или разовая оплата. После создания всех учеников на данном уроке нажмите кнопку «Закрыть» в модальном окне «Ученики». После нажатия кнопки «Закрыть» обновится окно с данными уроков. У нашего урока появится таблица с учениками. Поставьте галочку «Урок проведен» и сохраните урок. Для всех учеников с типом оплаты «Абонемент» снимутся средства по тарифу абонемента. Если на счету ученика недостаточно средств для посещения урока — то система не разрешит провести урок.

3. Кроме сохранения урока, изменения состояния счетов всех учеников, посетивших урок по абонементу система также создаст объекты класса «Чеки уроков» (ID: 32). В чеках отображается ссылка на урок, ссылка на ученика и стоимость урока.

4. Если на странице урока снять галку «Урок проведен» и сохранить урок (т. н. операция распределения или отмены урока), то удалятся чеки об оплате уроков. Также на счета

учеников, прошедших этот урок по абонементу, вернутся средства за отмененное занятие. Распроведение занятия может понадобиться, если необходимо внести изменения, чувствительные к деньгам.

Все данные, вносимые в систему сохраняются. Они доступны на странице [«История»](#) а также на таймлайнах каждого объекта. Например, если мы зайдем на страницу уроков и посмотрим историю уроков, то увидим все события: добавление или удаление учеников, проведение или отмена занятия, изменение времени занятия и т. д. Также на странице счетов будет видно все изменения баланса, в т. ч. при распроведении уроков.

На странице [«Отчеты»](#) расположены два отчета: Доход в текущем и предыдущем месяцах.