

# Динамические формы: Исчерпывающее руководство для новичков

Кирилл Кривошеев<sup>1</sup>

18 Апреля 2019

<sup>1</sup>hard-won with blood and sweat by the happy team ДРУГ



# Оглавление

<b>1 История создания динамических форм</b>	<b>7</b>
Вижу цель не вижу препятствий . . . . .	8
Первый успех и рождение динамических форм . . . . .	9
<b>2 Введение</b>	<b>13</b>
2.1 Что такое программа . . . . .	13
2.2 Сценарные языки программирования . . . . .	14
2.3 Запуск динамических команд . . . . .	15
2.4 Первая программа . . . . .	15
2.5 Чувствительность к регистру . . . . .	16
2.6 Арифметические функции . . . . .	16
2.7 Упражнения . . . . .	17
2.8 Среды выполнения . . . . .	17
2.8.1 FriendFace . . . . .	17
2.8.2 friend-dom-core . . . . .	17
<b>3 Структура программы</b>	<b>19</b>
3.1 Синтаксис . . . . .	19
3.2 Типы данных . . . . .	20
3.3 Структуры данных . . . . .	20
3.4 Функции . . . . .	21
3.4.1 Специальные функции . . . . .	22
3.5 Порядок выполнения . . . . .	22
3.6 Асинхронные функции . . . . .	26
3.6.1 Автоисполнятор . . . . .	28
3.6.2 Обработка ответа от автоисполнителя . . . . .	28

<b>4</b>	<b>Динамическая форма</b>	<b>31</b>
4.1	Что такое динамическая форма? . . . . .	31
4.2	Конструктор динамических форм . . . . .	32
4.3	Режимы открытия динамических форм . . . . .	33
4.3.1	MODE_CREATE_INTERACTION . . . . .	34
4.3.2	MODE_OPEN_INTERACTION . . . . .	34
4.4	Наследование состояния динамической формы . . . . .	35
4.4.1	Кнопка «Копировать» . . . . .	36
4.4.2	Создание связанного обращения . . . . .	37
4.4.3	Передача значений динамических полей в аргументах URL . . . . .	38
4.5	Общее описание динамического поля . . . . .	40
4.5.1	Параметры динамического поля . . . . .	42
4.6	Текстовые поля . . . . .	44
4.6.1	Маски ввода . . . . .	45
4.7	Поля с выбором значения . . . . .	45
4.7.1	Select . . . . .	46
4.7.2	Suggest . . . . .	46
4.7.3	Чекбоксы . . . . .	46
4.7.4	Радиокнопки . . . . .	46
4.7.5	Мультиэлементы . . . . .	47
4.7.6	Особенности элементов с выбором значения . . . . .	47
4.8	Окружение . . . . .	47
4.8.1	Поле информация . . . . .	48
4.8.2	Прикрепление файлов . . . . .	48
4.8.3	Внутренний клиент . . . . .	48
4.8.4	Подписание обращения . . . . .	49
4.8.5	Связанные обращения . . . . .	49
4.8.6	Scoring . . . . .	49
4.8.7	Поточный ввод . . . . .	49
4.8.8	Остальные флаги . . . . .	50
4.9	Общие команды динамической формы . . . . .	50
4.10	Виды групп . . . . .	51
4.11	Копирование групп . . . . .	52
4.12	Удаление групп . . . . .	59

<b>5 Паттерны программирования на динамическом языке</b>	<b>61</b>
5.1 Циклы . . . . .	61
5.2 Вложенные команды . . . . .	63
5.3 Выделение функций . . . . .	64
5.4 Упражнения . . . . .	73
<b>A Описание динамических команд</b>	<b>75</b>
setValue(value, id) . . . . .	75
getUser(attr) . . . . .	78
run(args) . . . . .	79
if(condition, trueExpr, falseExpr) . . . . .	79
isIn(args) . . . . .	80
isNotIn(args) . . . . .	80
isDigit(arg) . . . . .	80
call(expr, id) . . . . .	81
IAm(arg) . . . . .	81
getTexts(map) . . . . .	82
getValues(map) . . . . .	82
find(attr, value, group) . . . . .	83
plus(args)> . . . . .	85
minus(args) -> * . . . . .	86
mul(args) -> * . . . . .	86
div(args) . . . . .	86
indOf(string, substring) . . . . .	87
substr(string, start, end) . . . . .	87
len(string) . . . . .	88
copyGroup(groupId, id, group) . . . . .	88
getDistance(args) . . . . .	88
execute3DTask(TASK, args, map) . . . . .	89
executeTask . . . . .	90
execute3DTask . . . . .	90
getUsersByOrganizationRequest . . . . .	90
setValues(arg0-arg4) . . . . .	91
getValue(id) . . . . .	92
setMandatory(bool, id) . . . . .	92
setVisible(bool, id) . . . . .	93
setEnabled(bool, id) . . . . .	93

setValid(bool, description, id) . . . . .	93
setSelectedValue(val, id) . . . . .	94
setWidth(val, id) . . . . .	95
getText(map, id) . . . . .	95
isMandatory(id) . . . . .	95
isVisible(id) . . . . .	96
isEnabled(id) . . . . .	96
isValid(id) . . . . .	96
isFileMandatory(id) . . . . .	96
isFileEnabled(id) . . . . .	97
clear(id) . . . . .	97
setList(_labels, _texts, id) . . . . .	97
setMask(value, id) . . . . .	98
setFocus(val, id) . . . . .	98
setMax(value, id) . . . . .	98
setMin(value, id) . . . . .	98
string(value) . . . . .	99
setBeforeSave(value, id) . . . . .	99
getApprovalStage(id) . . . . .	99
setCheckSign(bool, id) . . . . .	99
setFileMandatory(bool, id) . . . . .	100
openAlert(type, message) . . . . .	100
setSaveEnabled(bool, id) . . . . .	100
setFileEnabled(bool, id) . . . . .	101
setExternalSystem(extsystem, id) . . . . .	101
setInformationVisible(bool, id) . . . . .	101
setInitiator(value, id) . . . . .	102
setMultiFlow(bool, id) . . . . .	102
setRedirect(value, id, skipConvert) . . . . .	102
setRelationship(objectId, objectType, id) . . . . .	103
setAdditionalTemplate . . . . .	103
setScoringVisible(bool, id) . . . . .	103
getSysDate(value) . . . . .	104
getInitiator(value) . . . . .	104
setMinHours(value, id) . . . . .	104
setMaxHours(value, id) . . . . .	105

setContainCustomObjects(bool, id) . . . . .	105
getSize(param) . . . . .	105
setStep(value, id) . . . . .	106
setStyle(arg, value, id) . . . . .	106
compare(left, operand, rigth) . . . . .	106
setInitiatorVisible(bool, id) . . . . .	107
setDescription(text, id) . . . . .	107
getRelatedId() . . . . .	107
setCopyVisible(bool, id) . . . . .	107
setDeclineVisible(bool, id) . . . . .	108
setCloseVisible(bool, id) . . . . .	108
getClientType() . . . . .	108
isMobileClient() . . . . .	108
isWebClient() . . . . .	108
isIOSClient() . . . . .	109
isAndroidClient() . . . . .	109
isWebSigmaClient() . . . . .	109
isWebAlphaClient() . . . . .	109
isFace20() . . . . .	109
setAction(value, id) . . . . .	109
setMode(value, id) . . . . .	110
dateDiff(start, end, id) . . . . .	110
isCreateMode() . . . . .	110
isOpenMode() . . . . .	110
setSaveVisible(bool, id) . . . . .	110
createDocument . . . . .	111
signAttachment(attachment, mode) . . . . .	112

## B Примеры 3DMap структуры

115



# Глава 1

## История создания динамических форм

Мало кто знает, что история динамического языка началась задолго до появления автоматизированной системы «Лицо Друга». Первые зачатки динамических форм родились в славном городе Ростов, более 10 лет назад.

Кирилл Борисович Кривошеев, создатель динамического языка, в те времена работал начальником отдела внедрения и сопровождения автоматизированных систем Юго-Западного территориального банка ПАО Сбербанк. Работы было много и, несмотря на всю самоотверженность и трудолюбие сотрудников, на отдел постоянно поступали жалобы. И причина тому была воистину «сбербанковская»: с утра и до позднего вечера сотрудники разгребали сотни заявок от пользователей на просьбы разблокироваться в каких-либо системах. На одного сотрудника в день могло назначаться более 200 заявок. Звонили пользователи, просили разблокировать их учётные записи, по звонку регистрировались запросы, которые затем передавались в другие подразделения. Далее искали заявителя в базе и выясняли причину проблемы. А причины могли быть самые нелепые, например, пользователь вместо логина упорно вбивал какую-нибудь хрень.

Вся эта ситуация будоражила сознание Кирилла, а вокруг никто не желал пошевелить и пальцем, чтобы как-то улучшить процесс, зато ежедневных жалоб и нытья было вагон и маленькая тележка. И тут Кирилл начал задумываться о способах автоматизации процессов.

## Вижу цель не вижу препятствий

В 2010 году Кирилл Борисович перешел на должность заместителя директора управления внедрения и сопровождения Юго-Западного территориального банка. После вступления в должность, Кирилл стал отвечать за всю автоматизацию банка, и, помимо отдела внедрения и сопровождения, под его руководством оказалась диспетчерская служба.

Теперь он наблюдал за еще большим количеством сотрудников, страдающих от скучных рутинных процессов.

Понимая, что если не он, то никто, Кирилл принялся за написание программы по оптимизации процесса разблокировки учетных записей. При этом стоит отметить, что на тот момент Кирилл Борисович толком не знал ни одного языка программирования, не считая языка PL+ – помеси SQL и объектно-ориентированного подхода, на котором были написаны многие автоматизированные системы.

Писать на PL+ было не вариант, но Кирилла это не остановило. “Гугл в помощь” – и вот наш герой уже шпарит свою первую программу на свежевыученном языке программирования C#. Всего было 25 программ для 25-ти автоматизированных систем. И эти 25 программ могли экономить для каждого пользователя примерно час жизни, которые ранее уходили на получение доступа, подачу заявки и ожидание в очереди на исполнение. Программа жеправлялась с задачей всего за 30 секунд!

“Зашпибись!” – подумал Кирилл, и пошел продавать идею заместителю председателя банка. И тут барьеры на пути к цели вновь дали о себе знать. Задумка была отклонена, а вместо нее была дана рекомендация нанять побольше людей в контактные центры и научить их нормально работать...

Вера в банковскую утопию, где сотрудники не должны были бы страдать от рутинных бессмысленных процессов, все же не покидала Кирилла. И он решил совершить ход конем. Удачным стечением обстоятельств явилось то, что в том году в банке по всей стране происходило довольно значимое событие: Программа глобального внедрения производственной системы в Сбербанке под руководством самого Германа Грефа. Данная система была вдохновлена лучшим практиками компании Тойота. Именно благодаря ей и по сей день в банке проводится Гемба, – процесс, когда начальник сам становится за станок, чтобы понимать проблемы изнутри, испытывая их на личном опыте.

Вместе с Гембой пришло поручение сверху оптимизировать всё, что

только можно оптимизировать. Например, был процесс связанный с чековой книжкой: после каждой операции, в соответствии с нормативными документами нужно было распечатать чековую книжку, поставить печать, зарегистрировать ее и т.д. На<sup>\*</sup>ера?! Клиент этого не хочет, потребности у него такой точно нет. А сотрудник тратил довольно много времени на эти бессмысленные процессы. И вот, в рамках оптимизации, отменили сначала печати, а потом и вовсе чековые книжки.

Кириллу, как заместителю директора, также пришло поручение оптимизировать процессы внутри отделов. И тут, не растерявшись, Кирилл Борисович решил повторно презентовать свою идею, но только теперь уже напрямую в центральный аппарат. Идея была принята. Так появилась на свет программа B@nk Helper – программа, которая помогала пользователям быстро получить доступ в необходимую систему.

The screenshot shows the application's navigation bar with links: Главная, Разблокировка в АС, Перевод в ВСП, Дополнительные возможности, Информация пользователя, and О программе. Below the navigation bar, there is a section titled "РАЗБЛОКИРОВКА". It contains instructions for requesting an automatic unlock, a dropdown menu for selecting a system ("БИК IB System Object"), a login field ("Ваш Логин в АС: Krivosheev"), a checkbox for "Сбросить пароль?", and a "Отправить запрос" button. To the left of this section is a calendar for July 2011, and to the right is a table showing a list of requests with columns: ЭЦП, Номер, Время, Статус, АС, Ошибка, and Примечания. The table displays the message "Нет запросов на разблокировку".

## Первый успех и рождение динамических форм

Итак, программа B@nk Helper была запущена и представлена всем заведующим филиалов банка в городе Ростов и, чтобы вы понимали, это 80 женщин (!). После презентации к Кириллу подошел директор, и сказал: “Еще никогда в жизни я не видел такого количества одновременно организующих женщин!”.

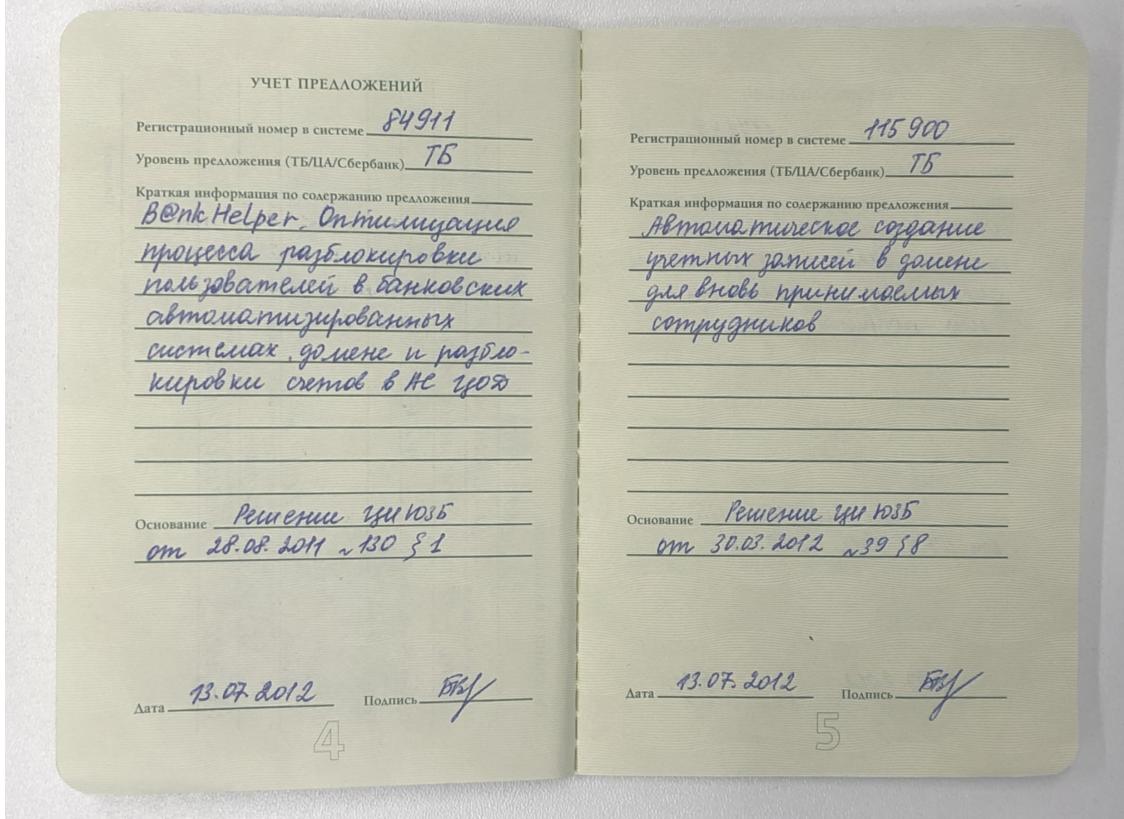
И это было только начало. B@nk Helper сразу полюбился сотрудникам банка, но, как говорится, к хорошему привыкаешь быстро. Вскоре начали поступать запросы на дополнительные изменения в программе, например, добавить галочку или списочек. И Кирилл, конечно же, совершенствовал B@nk Helper. Но изменения в системе давались не так просто, ведь приходилось вносить их в каждой форме для 25 однотипных автоматизированных систем. Так появился еще один процесс, который было необходимо оптимизировать.

Кирилл Борисович начал думать над тем, как унифицировать процессы, как избавить себя от необходимости кодить каждый раз и не тратить на это уйму времени, а иметь возможность через простые настройки настраивать формы как надо. И именно такой способ настройки форм и придумал Кирилл. Способ, при котором не нужно было проводить ПСИ (приемо-сдаточные испытания), отдельно выводить функционал в промышленную эксплуатацию. Формы программы можно было настроить в любой момент так, как необходимо.

Запросы не прекращались, появилась потребность при изменении одного поля ввода, в зависимости от введённых в него данных автоматически подгружать другое поле ввода. Кирилл придумал как справиться и с этой задачей. Эти формы и стали первым прототипом динамических форм, таких, какими мы знаем их сейчас.

Таким образом B@nk Helper стал прекрасным решением как для пользователей, так и для разработчиков. А Кирилл даже получил почетное звание “Инноватор Года”, которое записано в его паспорте участника корпоративной системы работы с инновациями. Этот паспорт по сей день бережно хранится шкафчике в кабинете Кирилла Борисовича.

В тоже время, помимо наград и всеобщего признания Кириллу Борисовичу поступило официальное предложение о переезде в Москву для тиражирования системы B@nk Helper во все территориальные банки.





# Глава 2

## Введение

### 2.1 Что такое программа

Программа – комбинация компьютерных инструкций и данных, позволяющая аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления.

Как правило, компьютерная программа написана программистом на языке программирования. Из программы, написанной на понятном человеку исходном коде, компьютер может сформировать машинный код – набор инструкций, который компьютер может исполнять напрямую. В качестве альтернативы, программа может быть исполнена при помощи интерпретатора.

Несмотря на различия между множеством существующих языков программированния некоторые базовые инструкции присутствуют почти в каждом из них:

**ввод:** получить данные с клавиатуры, из файла или другого устройства

**вывод:** отобразить данные на экране или отправить на другое устройство

**арифметика:** произвести базовые математические операции, такие как сложение или умножение

**ветвление:** проверить определенное состояние и выполнить соответствующий ему код

**повторение:** произвести определенное действие неоднократно, иногда с некоторыми вариациями

В динамическом языке программой принято называть последовательность команд, записанную в формате функций с аргументами. Каждый аргумент указывается в квадратных скобках. Аргументы между собой разделяются запятыми, например:

скажи( [привет] , [как] , [дела] )

## 2.2 Сценарные языки программирования

Сценарный язык (язык сценариев, жарг. скриптовый язык; англ. scripting language) – высокоуровневый язык программирования, автоматизирующий выполнение задач в специализированной среде выполнения. Как правило, сценарные языки являются интерпретируемыми, а не компилируемыми. Сценарный язык можно рассматривать как предметно-ориентированный язык (англ. domain-specific language, DSL — «язык, специфический для предметной области») для определённого окружения.

Язык программирования динамических форм (далее **Динамический язык**) – сценарный предметно-ориентированный язык, предназначенный для автоматизации управления состоянием интерактивных форм регистрации обращений в экосистеме ДРУГа.



## 2.3 Запуск динамических команд

Одним из первых вызовов, который бросают динамические формы начинающему разработчику, является необходимость настройки рабочего окружения HP Service Manager для редактирования шаблонов. Однако, благодаря мощным возможностям динамического языка, сделать первые шаги в разработке можно гораздо проще: достаточно открыть шаблон обращения «Исполнитель» в dev-среде портала «Лицо ДРУГа». Это простая динамическая форма, состоящая всего из двух полей: «Команда» и «Результат».

## 2.4 Первая программа

Talk is cheap. Show me the code

---

*Linus Torvalds*

По давней традиции, первая программа, которую начинающий разработчик пишет на новом языке, является «Hello, world!». На динамическом языке она выглядит так:

```
>>> setValue([Hello, world!], [result])
```

Для выполнения программы введите ее в поле «Команда» и переставьте курсор в поле «Результат». В итоге, в поле «Результат» вы увидите текст:

Hello, World!

Поздравляем! Вы сделали первый шаг на безумно интересном пути к освоению могучих возможностей создания интерактивных форм, который даёт вам Динамический язык. Это пример команды `setValue`, которая устанавливает значение, указанное первым аргументом в квадратных скобках в поле, указанное во втором аргументе.

Здесь и далее, все команды, которые вводятся в поле «Команда» будут предваряться символами

```
>>>
```

## 2.5 Чувствительность к регистру

Динамический язык нечувствителен к регистру. Это означает, что программы:

```
>>> setvalue([Hello, world!], [result])
```

и

```
>>> SETVALUE([Hello, world!], [result])
```

и даже

```
>>> sEtVaLuE([Hello, world!], [result])
```

С точки зрения интерпретатора динамического языка являются идентичными и произведут одинаковый результат. И несмотря на то, что последний пример выглядит, безусловно, наиболее круто, в примерах к книге мы будем придерживаться так называемого CamelCase, так как он наиболее удобочитаем:

```
>>> setValue([Hello, world!], [result])
```

## 2.6 Арифметические функции

Перейдем от «Hello, world!» к арифметике. Как и любой, уважающий себя язык программирования, динамический язык предоставляет возможность работы с основными арифметическими операциями, а именно **plus**, **minus** и **mul** для сложения, вычитания и умножения соответственно, а также **div** для деления:

```
>>> setValue([plus([68],[1])], [result])
69
>>> setValue([minus([69],[1])], [result])
68
>>> setValue([mul([6],[7])], [result])
42
>>> setValue([div([84],[2])], [result])
42
```

На самом деле, указанные функции обладают гораздо большим набором возможностей, но для первого знакомства этого будет достаточно.

## 2.7 Упражнения

Откройте шаблон «Исполнитель» и, при помощи динамического языка, решите следующие задачи:

1. Сколько будет в градусах Цельсия температура в 69 Фаренгейт?
2. Сколько спринтов потребуется для вывода в промышленную эксплуатацию шаблона «Совместная поездка» если каждый sprint исправляется 4 багов, и обнаруживается 6 новых?

## 2.8 Среды выполнения

### 2.8.1 FriendFace

### 2.8.2 friend-dom-core



# Глава 3

## Структура программы

### 3.1 Синтаксис

Инструкции динамического языка представляют собой комбинации из конструкций всего двух типов: вызовов функций и литералов.

Основной конструкцией языка является *функция*. Она состоит из имени и параметров. Сразу после имени функции следует символ '(', за которым могут следовать один или несколько параметров, после этого вся конструкция завершается ')'. Некоторые функции вовсе не принимают параметры, см. листинг 3.1 (1). Также поддерживается несколько параметров которые можно разделять символом ',', пример 3.1 (2, 3), но не все среды выполнения (см. разд. 2.8) поддерживают символ переноса строки, используя вместо него последовательность `\r\n`. Параметры функции можно переносить на следующую строку и добавлять пробелы для лучшей читаемости программы, 3.1 (3)

Скобки после имени функции являются обязательными даже если у функции нет параметров. Каждый параметр функции необходимо обернуть в '[' ']'. Параметром функции может быть литеральное значение или вызов другой функции, вне зависимости от этого параметр необходимо обернуть в '[' ']'.

Литералы представляют собой константы, включаемые непосредственно в текст программы. Литералы не могут быть изменены в тексте программы.

Есть зарезервированный литерал *this* который принимает значение *id* элемента на котором выполняется, подробнее в разд. 3.3.

```
(1) func()  
  
(2) func([arg0], [arg1], [argN])  
  
(3) func(  
    [arg0], [arg1]  
    , [arg2]  
)
```

## 3.2 Типы данных

Динамический язык это языком с динамической типизацией<sup>1</sup>, определение типа происходит непосредственно перед передачей аргумента в функцию.

Правила, по которым происходит приведение типа, следующие:

значение считается **числом**, если его строковое представление содержит только цифры и знак разделителя «,» или «.».

значение считается **логическим**, если строковое представление представляет собой последовательности `true` или `false`, вне зависимости от регистра символов.

В остальных случаях значением считается строкой символов.

**Переменные** Динамический язык не поддерживает определение и присвоение переменных. Однако функции позволяют изменять элементы динамической формы и значения в этих элементах можно считать своего рода переменными.

## 3.3 Структуры данных

Функции помимо литеральных значений могут принимать и возвращать определенные структуры данных Мар и 3DMар. Это необходимо для реализации вызова некоторых функций автоисполнителя (см. 3.6.1)

**Мар** это структура данных **ключ - значение**. Обычно используется для хранения выбранного пункта в элементах типа **select** и **suggest** (см.

---

<sup>1</sup> динамическая типизация – приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной.

4.7.2) как правило ключ является идентификатором пункта, значение – отображаемым пользователю текстом. Но для элементов **suggest** применяются немного другие правила (см. далее).

**3DMap** – структура данных для обмена дин формой между автоисполнителем и клиентской системой. Напрямую данную структуру использовать нельзя: она передается и принимается автоматически при вызове функций название которых начинается с `__` (два нижних подчеркивания). При вызове таких функций клиентский код, помимо передачи параметров функции, также передает информацию обо всех атрибутах дин. формы, а при ответе ожидает структуру, которая описывает изменение атрибутов и значений существующих элементов, а также информацию об элементах которые необходимо добавить. Примеры данных структур можно посмотреть в приложении В

## 3.4 Функции

Функции позволяют выполнять действия на элементами формы, а также возвращать данные из контекста пользователя, шаблона или из другого элемента. Функции бывают нескольких типов: функции стандартной библиотеки, специальные, и асинхронные.

Как было отмечено ранее, функции могут принимать параметры и возвращать значения. Также у функции есть контекст выполнения: `this` – это литерал который в момент выполнения принимает значение `id` элемента, на котором выполняется данная команда.

**Возвращаемые значения** Все функции стандартной библиотеки динамического языка возвращают значение. Например,

`getValue([foo])` – вернет значение элемента, определяемого по идентификатору «`foo`» формы.

Функции, созданные для выполнения побочных эффектов, возвращают значение, как правило – «`id`» элемента формы, над которым производится операция:

`setValue([foo], [bar])` – вернет «`bar`»

Даже функции, для которых в стандартной библиотеке возвращаемое значение не определено – вернут пустой литерал: `[]`

### 3.4.1 Специальные функции

Специальные функции предназначены для позволяют изменить путь выполнения программы с помощью `if`, вызвать программу в контексте другого элемента – `call`. Данные функции встроены в парсер и позволяют обойти ограничение простого вызова функций с параметрами.

Описание специальных функций:

**if** `if([conditon], [then], [else])` – позволяет выполнить динамический язык в зависимости от *condition* *then* *else* не обязательные параметры

**run** `run([cmd0], ..., [cmdN])` – вызывает аргументы функции и просто возвращает *true*. Данная функция удобна когда нужно выполнить несколько команд параллельно, и позволяет сократить вложенность команд.

**call** `call([command], [id])` – позволяет выполнить команду *command* в контексте элементе с *id*, в данном случае *this* внутри команды будет равен *id*. Если результат выполнения команды внутри CALL является динамической функцией создается дерево(см. разд. 3.5) если результат не является динамической функцией - исходная команда внутри CALL. Выполняется над другим элементом динамической формы указанным во втором параметре функции CALL

**string** `string(command)` – экранирует любой текст в рамках ‘‘ ‘’

## 3.5 Порядок выполнения

В основе алгоритма определения порядка выполнения команд динамического языка лежит простая концепция – выполнение функций должно происходить по слоям из самого глубокого вложенной функции к корневому элементу, при этом функция не может быть выполнена, пока не выполнены все ее дочерние элементы.

Итак поэтапно разберем последовательность действий происходящих при парсинге команд:

1. Исходный строковый вид команды передается на вход анализатору, который содержит в себе регулярные выражения для парсинга

2. Происходит валидация кода(в первых версиях реализации валидации не было), в случае ошибки валидации – ошибки в количестве параметров функции, скобочках или невозможных символах - команда не будет выполнена и в лог действий будет выведена ошибка валидации
3. Если команда успешно провалидирована, начнется построение динамического дерева(Динамическое дерево). Важно отметить, что в ходе выполнения команды, дерево может менять свою структуру(команда if), а иногда даже создавать копии самого себя (CALL).
4. Для построения динамического дерева используется стандартный левосторонний обход – каждая следующая функция последовательно добавляется и спускается вглубь к следующему дочернему узлу, обойдя все элементы у первого дочернего узла, переходит ко второму, третьему и т.д.
5. ПРИМЕЧАНИЕ. Особенными для разбора считаются функция IF, CALL и STRING в частности CALL и STRING останавливают дальнейший парсинг дочерних узлов функции и выставляют в результат строковое представление аргументов функции. Условная функция(оператор) IF – разбирает изначально только первый дочерний узел, оставшиеся два так же сохраняет в строковых представлениях
6. После разбора команды в динамическое дерево можно запускать ее на исполнение
7. В изначальной реализации динамического языка - динамическое дерево полностью преобразовывалось в очередь и выполняло все команды последовательно по кругу пока все функции не будут выполнены. В более современной реализации очередь содержит только активные элементы выполняющиеся непосредственно в текущей итерации и последовательно добавляет родительские узлы непосредственно в тот момент, когда имеется возможность их выполнить. Напомним, что родительский узел может быть выполнен только в случае когда все его дочерние команды вернули результат
8. Выполнение начинается с самого глубокого уровня, если представить исходное динамическое дерево, то визуально его необходимо перевернуть и по слоям начать выполнять узлы дерева от листьев к корню, каждый

следующий слой ожидает выполнения дочерних узлов предыдущего, в случае если функция является асинхронной, она не блокирует выполнение остальных команд на уровне

9. ОСОБЕННОСТИ. Функции CALL и IF являются функциями модифицирующими динамическое дерево. Так функция CALL создает локальную копию поддерева из своих аргументов при этому есть два варианта(CALL)
10. Функция IF на основе результата первого дочернего узла добавляет в исходное динамическое дерево либо второй, либо третий аргумент функции(третий может отсутствовать), происходит парсинг необходимой команды и динамическое дерево расширяется узлами дочерней функции внутри IF, параллельно выполняющейся в очереди начиная с уровня на котором размещен узел с IF(подробнее про IF)
11. В случае возникновения ошибок, возможны ситуации, которые прерывают дальнейшее выполнение команды(например: Не найден элемент динамической формы с заданным id).
12. Результат корневого узла дерева обычно не используется и ничего не возвращает, за исключением использование команды CALL.

Для примера рассмотрим варианты динамических команд и представления в виде уровней и выполнения:

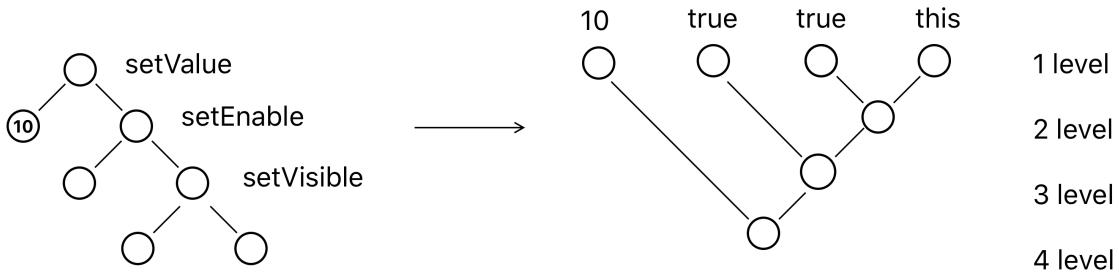
Рассмотрим команду

```
setValue(
    [10],
    [setEnabled([true],
        [setVisible([true], [this])])
    ])
)
```

На изображении выше можно четко увидеть уровни по которым происходит выполнение команд, а теперь перейдем к более сложному варианту с использование функции *if*.

Для примера рассмотри абстрактную команду:

```
run(
```



```

[if(
  [isin(
    [getValue([this])],
    [20],
    [10]
  ]),
  [run(
    [setVisible([true],[this])],
    [setEnable([true],[this])],
  ]),
  [setVisible(
    [false],
    [this]
  )]
),
[setEnable(
  [true],
  [this]
)]
)
  
```

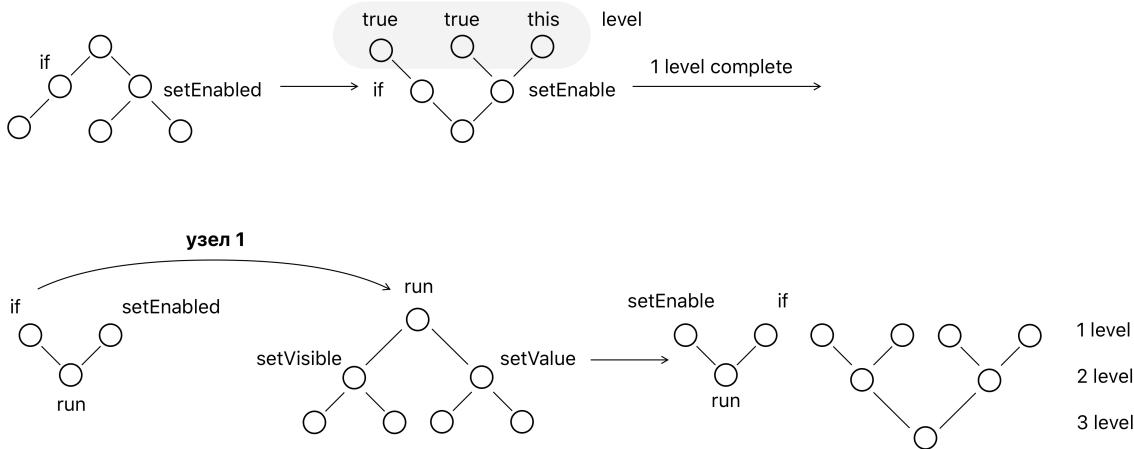
У функции *if* три параметра, предположим, что результат первого выражения в *if* истина, то порядок выполнения будет представлен следующим графиком:

Понимая концепции порядка выполнения команд, а также особенности запуска циклов обработки команд можно автоматизировать сколько угодно сложные наборы операций над динамическими формами.



### 3.6 Асинхронные функции

Помимо функций из стандартной библиотеки которые исполняются синхронно, то есть исполнение кода останавливается пока функция не вернет значение, также в web интерфейсах, выполняется в процессе браузера, что приводит к блокированию пользовательского интерфейса на время выполнения функции. асинхронные же функции не блокируют поток выполнения браузера, но также блокируют выполнение динамического языка. Что примечательно асинхронные функции не блокируют несколько параллельных команд если они запущены внутри функции *run*.



Асинхронные функции необходимы для вызова автоисполнителя.

Асинхронные функции не имеют специального имени – они могут быть названы любым именем, которое не входит в список динамических функций стандартной библиотеки (см. прил. А). Функции без нижнего подчеркивания делают запрос к стандартным скриптам, с нижнем подчеркиванием – к скриптам с обработкой формы.

Стандартный вызов функции выглядит так:

```
func([name], [arg0], ... [argN])
```

*name* – имя метода автоисполнителя, *arg0...argN* аргументы которые передаются в автоисполнитель.

Количество аргументов таких функций не ограничено. Первым аргументом передается название скрипта, а последним – элемент над которым производится выполнение.

Вызов функции с обработкой форму не отличается от обычного вызова функции, но при передаче в автоисполнитель дополнительным параметром передается вся динамическая форма в формате 3DMar. Такие функции всегда возвращают *id* элемента на которым был вызван, а среда выполнение обрабатывает дополнительные параметры которые могут добавить или удалить элемент формы, поменять состояние нескольких элементов и т.д.

### 3.6.1 Автоисполнятор

Автоисполнятор или АИ – представляет собой Task Manager или систему асинхронного выполнения задач. Задачи представляют собой скрипты написанные на языке groovy. Groovy – объектно-ориентированный язык программирования разработанный для платформы Java как альтернатива языку Java с возможностями Python, Ruby и Smalltalk. Каждый скрипт идентифицируется по уникальному имени,

FRIEND\_GET\_LIMIT\_BY\_USER\_TPL\_DOUBLE.

Кроме имени скрипт на вход принимает дополнительные параметры.

### 3.6.2 Обработка ответа от автоисполнителя

В зависимости от типа вызова функции, автоисполнятор может вернуть структуру *Map* и *3DMap*. Структура типа *Map* представляют собой массив значений «ключ-значение».

```
// JSON
[
  { key: "id", value: " " }
]??????
```

Такая структура хорошо подходит для обмена данными между элементами формы типа select.

*3DMap* данная структура данных предназначена для описания изменения в динамической форме. С ее помощью можно описать добавление, удаление, изменение элементов динамической формы.

Данный формат описывает массив команд который нужно выполнить на динамической формой. Команды которые поддерживаются всеми средствами выполнения:

*new* команда позволяет добавить один или несколько новых элементов. *attr* позволяет изменить несколько атрибутов формы *value* устанавливает значение элемента *valueselected* устанавливает выбранное значения для элементов типа select *valuevisible* устанавливает флаг видимости элемента на дин форме *valueenabled* запрещает или разрешает редактирование элемента *command* выполняет команду дин языка *delete* удаляет элемент с формы

*beforesave* устанавливает команду которая будет выполнена перед сохранением обращения



Пример ответа сервера в этом формате можно посмотреть в прил. В.



# Глава 4

## Динамическая форма

Основа пользовательского интерфейса, его разметка, стили, поведение и будущий результат работы пользователя, были воплощены в набор стандартизованных данных, называемый «Динамическая форма». Она свободно интерпретируется специализированными приложениями для web-браузеров (портал «Лицо ДРУГа») и приложениями для операционных систем: iOS и Android. В этой главе мы рассмотрим структуру и правила описания и контроля динамической формы на всех этапах её жизненного цикла.

### 4.1 Что такое динамическая форма?



Динамическая форма состоит из набора динамических полей, представленных в древовидной структуре. Поля представляют собой

стандартные элементы UI: поля ввода текста, чекбоксы, раскрывающиеся списки, кнопки и т. д. Они взаимодействуют между собой, с пользователем и внешними системами, что в конечном счете позволяет автоматизировать любой бизнес процесс.

На низком уровне динамическая форма описана следуя синтаксису XML. Рассмотрим пример простейшего шаблона динамической формы:

```
<form>
    <text id="helloworld"
        label="Простое текстовое поле"
        mandatory="false"
        sbcommand=""
        sbcopyinfo="false"
        sbfield=""
        sbmask=""
        sbmodify="true"
        sbstyle=""
        sbtask=""
        sbtitle=""
        sbtype=""q
        setGroupId=""
        style="text"
        type="2"
        visible="true" width=""></text>
</form>
```

### Примечание

Заметьте, что на изображении присутствуют элементы интерфейса «Добавить файл» и «Отправить». Они не описаны в динамической форме и являются стандартными для всех шаблонов. Но все же поля динамической формы имеют возможность управлять состоянием этих «статических» элементов при помощи функций динамического языка.

## 4.2 Конструктор динамических форм

Во главе концепций динамических форм ставилась скорость и легкость их разработки/сопровождения. Но написание даже самой простой

The screenshot shows a user interface for a dynamic form. At the top, there is a placeholder text 'Простое текстовое поле'. Below it is a large empty rectangular input field. To the left of this field is a dashed square containing a plus sign '+'. To the right of the input field is a blue button labeled 'Добавить файл' (Add file). In the bottom right corner of the form area, there is a large green button with white text that says 'Отправить' (Send).

Рис. 4.1: Простая динамическая форма в пользовательском интерфейсе портала «Лицо ДРУГа»

динамической формы в XML – непростая задача. Необходимо определить множество свойств, полей, событий и отслеживать взаимосвязи между ними.

Для упрощения процесса создания динамической формы был разработан удобный графический конструктор динамических форм на платформе HP Service Manager.

Описание конструктора динамических форм, выходит за рамки данной книги. Его интерфейс интуитивно понятен, поэтому не вызовет трудностей в освоении, даже для людей без опыта в разработке UI.

## 4.3 Режимы открытия динамических форм

Динамические формы поддерживают несколько режимов работы:

1. MODE\_CREATE\_INTERACTION – режим создания нового обращения из шаблона
2. MODE\_OPEN\_INTERACTION – режим чтения обращения
3. MODE\_OPEN INCIDENT – режим работы с инцидентом
4. MODE\_COPY\_TEMPLATE – режим копирования обращения

### Описание динамического поля

Задайте определение динамического поля.  
Должны быть определены фактическое имя поля в XML, подпись поля, отображаемая для пользователя, и тип отображения для поля.



Имя параметра:	<input type="text"/>
Метка на форме:	<input type="text"/>
Всплывающая подсказка:	<input type="text"/>
Маска ввода:	<input type="text"/>
Скопировать значение в поле инцидента:	<input type="checkbox"/> Да
Стиль	<input type="text"/>
Имя группы	<input type="text"/>
Значение	<input type="text"/>
Тип отображения:	<input type="radio"/> Текст <input type="radio"/> Многострочный текст <input type="radio"/> Флажок <input type="radio"/> Список выбора
<input type="checkbox"/> Изменяемое поле	
<input type="button" value="&lt; Назад"/> <input type="button" value="Далее &gt;"/> <input type="button" value="Готово"/> <input type="button" value="Отмена"/>	

Рис. 4.2: Конструктор динамических форм HP Service Manager

### 4.3.1 MODE\_CREATE\_INTERACTION

В режиме создания обращения возможность редактирования полей устанавливается флагом `sbmodify` в элементах шаблона: если он равен `true` – то динамическое поле доступно для редактирования, если такую возможность подразумевает типа данного поля. При открытии шаблона выполняются динамические команды, указанные в свойствах `sbcommand` динамических полей последовательно, в порядке их объявления в шаблоне. Кнопка «Создать обращение» отображается по умолчанию

### 4.3.2 MODE\_OPEN\_INTERACTION

В режиме чтения обращения все поля по умолчанию закрыты для редактирования, флаг `sbmodify` игнорируется. Для «Разблокирования» некоторых полей существует механизм `unreadableFields` – данный атрибут обращения содержит массив идентификаторов динамических полей, которые будут доступны для редактирования в данном режиме.

Команды `sbcommand` выполняются только в разблокированных полях, вместо них отрабатывают команды из атрибутов `sbonload` динамических полей. Кнопка «Сохранить обращение» отображается только в том случае, если массив `unreadEditableFields` непустой.

## 4.4 Наследование состояния динамической формы

Ранее упоминалось, что динамическая форма после исполнения и преобразования пользователем, сохраняет свое состояние в БД. Из этого состояния может быть создана «производная» динамическая форма, которая будет наследовать некоторые свойства исходной. Эта возможность бывает очень полезной, когда мы хотим освободить пользователя от необходимости повторно заполнять одинаковые динамические поля.

Под наследованием состояния динамической формы, понимается последовательный перенос свойств динамических полей из исходной формы в производную. Перенос производится для динамических полей, совпадающих по свойству `id`.

Наследуются следующие свойства:

- `value`
- `visible`
- `enabled`
- `sbmodify`

Некоторые типы динамических элементов, содержат значения в виде набора дочерних элементов, например: чек-боксы, радио-боксы, выпадающие списки. Эти дочерние элементы имеют свои свойства: видимость, доступность для редактирования и значение. Свойства дочерних элементов тоже должны быть унаследованы, и передаются в атрибутах:

- `valueSelected`

- `valueEnabled`
- `valueVisible`

### **Примечание**

У динамических полей, которые наследовали свойства, не будут работать команды динамического языка при старте (`sbcommand`). Обычно команды при старте используются для инициализации значений в динамических полях. Поэтому для избегания конфликтов с наследованием, эти команды отключены.

Существует три способа наследования состояния:

1. копирование динамической формы (кнопка «Копировать»);
2. создание связанного обращения;
3. передача значений динамических полей, через ссылку в аргументах URL.

#### **4.4.1 Кнопка «Копировать»**

Бизнес услуги могут быть востребованы пользователем несколько раз за короткий промежуток времени. Пользователь, заполняет форму (обращения) получения услуги один раз, а далее создает копию этого обращения и лишь незначительно меняет его поля. Для этого, в режиме просмотра обращения (по некоторым услугам), доступна кнопка «Копировать».

Нажатие кнопки «Копировать», переадресовывает пользователя на страницу создания нового обращения по услуге исходного. Свойства динамических полей будут унаследованы по полю `id`, как было указано ранее.

### **Примечание**

Копирование динамических полей, может отрабатывать неочевидным образом, а в некоторых формах вовсе приводит к некорректным заявкам:

- При наследовании значения у полей типа «Дата», будет наследована дата, которую пользователь указал несколько дней назад. При

копировании пользователь может не учесть этого и не скорректировать дату с учетом прошедшего времени.

- В `sbcommand` могут быть использованы команды инициализации значения из внешней системы, например из автоисполнителя. Часто параметры во внешних системах изменяются и замена такой инициализации на наследование устанавливает неактуальные значения.

В таких динамических формах необходимо отключать кнопку «Копировать». Отключить её можно при помощи функции динамического языка:

```
setCopyVisible([bool], [id])
```

#### 4.4.2 Создание связанного обращения

Для исполнения обращения пользователя может потребоваться запуск дополнительных бизнес процессов. Их можно инициировать, создав связанное обращение к исходному. Связанное обращение будет наследовать и хранить информацию об исходном (родительском) обращении.

Для создания связанного обращения используется специальная ссылка, перейдя по которой, пользователь попадает на страницу создания связанного обращения. Например, для портала «Лицо ДРУГа», ссылка имеет следующую структуру:

```
https://friend.sbrf.ru/FriendFace/#createInteraction?FRIEND
&Обеспечение_IT
&SD227598282
&INTERACTION
&PARENT
&Филатов Валентин Васильевич/
```

В параметрах URL передается:

- FRIEND – подсистема, в которую будет направлено обращение;
- Обеспечение\_IT – идентификатор шаблона;

- SD227598282 – идентификатор исходного обращения;
- INTERACTION – тип исходного обращения;
- PARENT – тип связи с исходным обращением;
- Филатов Валентин Васильевич – идентификатор создателя исходного обращения.

Как было описано ранее в этом разделе, связанная (производная) динамическая форма будет наследовать состояние исходной по элементам с совпадающими полями `id`.

Особенностью данного способа наследования является то, что шаблон производной динамической формы не совпадает с шаблоном исходной. Разные шаблоны имеют разные наборы динамических элементов, поэтому одного наследования по `id` в данном способе недостаточно. Динамические элементы, которых нет в производной форме, будут полностью скопированы в отдельную группу элементов и помещены в конец производной формы.

Динамическое поле «counterAlert» – часть исходной динамической формы, а поля «Наименование соглашения», «Действие», «Договор/объемы» – наследованные динамические элементы, которые объединены в группу «Информация обращения SD227598282». Обратите внимание, все элементы наследованной группы недоступны для изменения.

#### 4.4.3 Передача значений динамических полей в аргументах URL

При направлении пользователя из внешней системы на страницу создания обращения (через ссылку), часто возникает необходимость устанавливать начальные значения динамических элементов. Например, если необходимо зарегистрировать обращение с данными из памяти внешней системы. Для этого, ссылка на страницу создания обращения принимает следующий вид (пример для приложения «Лицо ДРУГа»):

```
https://friend.sbrf.ru/FriendFace/#createInteraction?FRIEND  
&Обеспечение_IT&&&&&&&{"elem1":"val1","elem2":"val2"}
```

The screenshot shows a user interface for a dynamic form. At the top, there is a field labeled "counterAlert" containing the value "0". Below it is a section titled "Информация обращения" with the ID "SD227598282". A search bar contains the text "Соглашение для услуги 'Разработка ПО'" with a clear (X) and search (Q) button. Under "Действие", a button labeled "Изменение" is highlighted. In the "Договор/объемы" section, the value "300сп" is entered. At the bottom left is a dashed box with a plus sign and the text "Добавить файл". On the right side, a large green button with the text "Отправить" is visible.

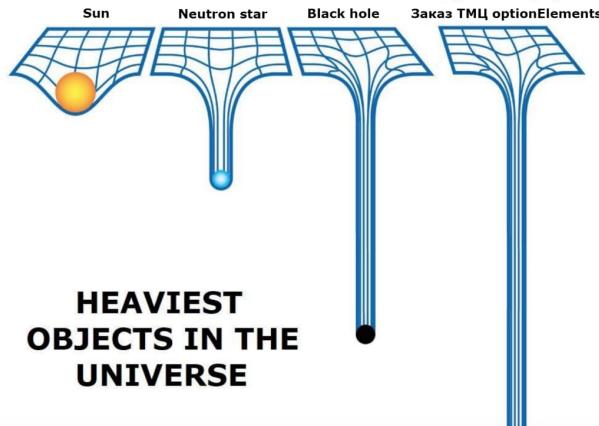
Рис. 4.3: Пример формы наследования связанного обращения

Последним (девятым) параметром URL, может быть передан JSON, в котором задан ассоциативный массив ("ключ": "значение"). При переходе по данной ссылке, на динамической форме будет найден элемент с `id = elem1` и если он есть, ему будет установлено значение `val1`. Для элемента `elem2`, установка значения будет работать аналогично.

### Примечание

Таким образом, можно устанавливать значения скрытых/недоступных для редактирования полей. Это полезно для скрытия/защиты от случайного редактирования передаваемых значений.

## 4.5 Общее описание динамического поля



Каждое динамическое поле представляет собой набор стандартных и дополнительных параметров отвечающих за визуальное отображение и функциональное поведение элемента. Определение элемента задается при формировании шаблонов в НПСМ, разнообразие типов достигается комбинацией значений параметров **sbtype**, **type**, **logicalType**, **object-Type**. В качестве примера приведем простое текстовое поле:

```
{
  "objectType": "text",
  "logicalType": "text",
  "id": "editmarriagenumauto",
  "label": "Свидетельство о браке",
  "mandatory": false,
  "sbcommand": "",
  "sbmask": "",
  "sbmodify": true,
  "sbtask": "",
  "sbtitle": "например \"VI-МЮ-777777\"",
  "sbtype": "",
  "sbstyle": "width:33%",
  "style": "text",
  "visible": false,
  "width": "",
  "sbobject": null,
```

```
"sbdbfield": null,  
"sbaction": null,  
"sbmode": null,  
"sbonload": null,  
"sbcopyinfo": false,  
"childs": null,  
"groupid": "",  
"text": null,  
"options": null,  
"type": "2",  
"multiline": null,  
"button": null,  
"matchTable": null,  
"matchField": null,  
"query": null,  
"hpcGroupByFields": null,  
"sbfield": ""  
}
```

и элемент типа select:

```
{  
    "objectType": "select",  
    "logicalType": "combo",  
    "id": "purpose",  
    "label": "Цель поездки",  
    "mandatory": true,  
    "sbcommand": "setvalue([something], [purpose])",  
    "sbmask": "",  
    "sbmodify": true,  
    "sbtask": "setvalue([something], [purpose])",  
    "sbtitle": "",  
    "sbtype": "",  
    "sbstyle": "",  
    "style": "combo",  
    "visible": true,  
    "width": "",  
    "sbobject": null,
```

```
"sbdbfield": "purpose",
"sbaction": null,
"sbmode": null,
"sbonload": "setvalue([something], [purpose])",
"sbcopyinfo": false,
"children": null,
"groupid": "",
"text": "Встречи с клиентами",
"options": [
  {
    "text": "Выезды на аварии",
    "label": "id02",
    "id": "id02"
  },
  {
    "text": "Встречи с клиентами",
    "label": "Встречи с клиентами что-т там",
    "id": "id03"
  },
  {
    "text": "Выезды в государственные органы",
    "label": "Выезды в государственные органы",
    "id": "id04"
  }
],
"type": "2",
"multiline": null,
"button": null,
"matchTable": null,
"matchField": null,
"query": null,
"hpcGroupByFields": null,
"sbfield": ""
}
```

### 4.5.1 Параметры динамического поля

- **objectType** – параметр отвечающий за определение типа поля
- **logicalType** – параметр отвечающий за определение типа поля
- **sctype** – параметр отвечающий за определение типа поля
- **type** – параметр отвечающий за определение типа поля
- **style** – параметр отвечающий за определение типа поля
- **id** – уникальный идентификатор поля используется в командах динамического языка для поиска элемента
- **text** – значение поля по умолчанию
- **label** – заголовок поля, отображаемое значение для информации и обозначения поля
- **sbttitle** – подсказка для поля, содержит дополнительную информацию о поле, выводиться в виде вопроса
- **groupid** – параметр сортировки и формирования для плоских и legacy групп
- **sbccommand** – содержит команды динамического языка выполняемые при инициализации MODE\_CREATE\_INTERACTION
- **sbttask** – содержит команды динамического языка выполняемые при изменении поля через пользовательский ввода
- **sbonload** – содержит команды динамического языка выполняемые при инициализации MODE\_OPEN\_INTERACTION или MODE\_OPEN\_INCIDENT
- **mandatory** – флаг отвечающий за обязательность поля
- **sbmask** – маска поля ввода(используется для текстовых полей и дат)
- **sbmodify** – флаг определяющий можно ли изменять поле

- **sbstyle** – параметр отвечающий за визуальные стили элемента вызывает команду setStyle над элементом
- **visible** – флаг определяющий видимость поля в пользовательском интерфейсе
- **width** – значение в % ширины элемента относительно блока
- **sbobject** – HP SM property, не изменяется в рамках обработки
- **sbdbfield** – HP SM property, не изменяется в рамках обработки
- **sbaction** – HP SM property, может быть изменено командой setAction
- **sbmode** – HP SM property, может быть изменено командой setMode
- **sbcopyinfo** – флаг отвечающий за копирование значения динамического поля в физическое поле information
- **children** – дочерние элементы используется в элементе типа group
- **options** – варианты значений для поля для полей с выбором значений
- **multiline** – флаг отвечающий за возможность многострочного ввода для текстовых полей
- **button** – HP SM property, не изменяется в рамках обработки
- **matchTable** – HP SM property, не изменяется в рамках обработки
- **matchField** – HP SM property, не изменяется в рамках обработки
- **query** – HP SM property, не изменяется в рамках обработки
- **hpcGroupByFields** – HP SM property, не изменяется в рамках обработки
- **sbfield** – HP SM property, не изменяется в рамках обработки

## 4.6 Текстовые поля

Текстовые поля могут быть четырех разных визуальных типов + все поля для которых не определены правила тоже преобразуются в обычное текстовое поле

1. Обычное текстовое поле
2. Неизменяемое текстовое поле(Label) - текстовое поле с флагом **modifiable = "false"**
3. Многострочное текстовое поле - текстовое поле с флагом **multiline = "true"** и параметром **type = "2"**
4. Числовое поле - текстовое поле с параметром **type = "1"**или **sctype = NUMBER**

Текстовые поля поддерживают ввод по маске(параметр **mask** не пустой)

### 4.6.1 Маски ввода

Для ограничения ввода в текстовые поля используются обобщенные маски. Маски представляют собой собственные элементы по регулярным выражениям:

# - число аналог стандартного [0-9]

9 - число или пробел [0-9 ]

A - символ в верхнем регистре [A-ZА-Я]

а - символ в нижнем регистре [a-za-я]

B - символ в любом регистре [A-ZА-Яa-za-я]

C - символ в любом регистре или число [A-ZА-Яa-za-я ]

Остальные символы не из набора масок соответствует статическим символам, которые не участвуют в валидации маски

## 4.7 Поля с выбором значения

Динамическая форма поддерживает различные комбинации элементов с возможностью выбрать одно из предлагаемых значений, ниже мы рассмотрим подробнее каждый из элементов

### 4.7.1 Select

Есть два варианта определения стандартного поля с выбором значения, которые определяются по одному из параметров **sbtype = COMBOAREA** или **style = COMBO**. Варианты выбора могут быть записаны в поле в параметре **options**, а так же заполняться из команд динамического языка. Вариантами выбора могут быть текстовые значения, ссылки, изображения, а также их комбинация.

### 4.7.2 Suggest

Suggest или поле с подсказкой - расширение стандартного поля Select с возможностью поиска и фильтрации. Элемент определяется параметром **sbtype = SUGGEST**. Частными случаями поля типа Suggest являются поля со значениями **sbtype = SEARCH** и **ADDRESS** предназначенные для поиска(запросы к Автоисполнителю) и получение адреса по вводу города/улицы/дома. Для поля suggest есть дополнительный параметр **longSuggestItems** - отвечает за визуальное отображение элементов с длинными названиями в вариантах.

### 4.7.3 Чекбоксы

Чекбокс - элементы позволяющие сделать выбор из вариантов Да/Нет, Выбрано/Не выбрано. Определяется по параметрам **style = CHECKBOX** или **objectType = CHECKBOX**

### 4.7.4 Радиокнопки

Элемент типа радиокнопка определяется параметром **style = RADIO** и представляет собой выбор одно из предлагаемых значение. Элемент логически не отличается от стандартного Select, но визуальное и

функциональное поведение в рамках выполнения команд динамического языка отличается( подробнее в разд. 4.7.6)

#### 4.7.5 Мультиэлементы

Все типы полей кроме радиокнопок имеют свое представление в виде полей со множественным выбором - `multiSelect`, `multiSuggest`, `multiAddress`, `multiCheckbox`. Все мультиэлементы определяются параметром `sbtType` соответствующим типу поля. Визуально `multiSelect` и `multiSuggest` аналогичны своим единичным типам, но добавляют возможность добавлять несколько значений. При выборе значения сохраняются в поле `text` через символ ";". Так как не все системы поддерживают элементы со множественным выбором - дополнительно придуман метод сохранения таких полей, когда исходное поле визуально скрывается, а после него вставляется текстовое поле с многострочным вводом, каждая строка которого соответствует выбранному значению в исходном поле с индексом значения в начале строки.

#### 4.7.6 Особенности элементов с выбором значения

Все элементы с выбором значения кроме радиокнопок, чекбоксов и мультичекбоксов имеют важную особенность при выполнении команд динамического языка: Если командой динамического языка вызвать установку значения элемента через функции `setValue`, `setValues` или  `setSelectedValue` при этом передать одно единственное значение, то у элемента происходит запуск `sbtTask` таким образом эмулируя действия пользователя. Примечание: функция  `setSelectedValue` всегда вызывает запуск `sbtTask` не зависимо от количества переданных значений для установки.

### 4.8 Окружение

Динамический язык выполняется в определенном окружении. Окружение позволяет получить доступ к данным пользователя, атрибутам шаблона и т.д. Доступ к этим данным осуществляется через команды дин языка. Окружение полезно для автоматического заполнения шаблона

данными о пользователе, изменить поведение созданного обращения, например запросить подпись с помощью механизма ActiveX.

При создании обращения, в зависимости от шаблона могут потребоваться дополнительные данные или определенный порядок создания обращения, для этого существуют специальные флаги которые относятся ко всей форме сразу.

### 4.8.1 Поле информации

У каждого обращение существует поле информации, данное поле необходимо для правильной работы HP Service Manager. Некоторые шаблоны не используют данное поле и его можно скрыть используя флаг *informationVisible* и команду `setInformationVisible([bool], [id])`, также можно установить значение этого поля с помощью:

```
setDescription([string], [id])
```

### 4.8.2 Прикрепление файлов

В некоторых шаблонах необходимо, чтобы пользователь мог отправить файл, который называется вложением и связать его с сущностью обращения. Для этого используют флаги *fileEnable* и *fileMandatory*, для запрета вложений и для обязательности вложения файла соответственно. Для чтения и установки флагов используются команды:

```
isFileEnabled()  
isFileMandatory()  
setFileEnabled([bool], [id])  
setFileMandatory([bool], [id])
```

Флаг *fileEnable* запрещает прикладывать файлы для всей формы, но если в дин форме есть отдельный компонент вложения, то файл все равно можно будет приложить.

### 4.8.3 Внутренний клиент

У каждого обращения есть внутренний клиент (ВК), это сотрудник которому необходимо оказать услугу. В большинстве случаев ВК это тот

пользователь который вошел в приложение. Но бывают случаи когда его необходимо сменить, например если сломался компьютер, и необходимо создать обращение на его пос=чинку, то можно попросить коллегу создать обращение от твоего имени.

Для управления полем ВК используются команды:

`setInitiatorVisible([bool], [id])` – позволяет отключить возможность смены ВК, в интерфейсе

`setInitiator([vkId], [id])` – устанавливает ВК

#### 4.8.4 Подписание обращения

Для включения подписи обращения при сохранении можно использовать команду `setCheckSign`

#### 4.8.5 Связанные обращения

НР SM позволяет связать обращения между собой. Это необходимо для автоматического изменения статусов и для организации бизнес процессов. Связь это два поля: id объекта и тип связи. Из динамического языка устанавливаются командами

```
getRelatedId([id])
setRelationship([relation])
```

#### 4.8.6 Scoring

Некоторые шаблоны поддерживают специальные поля, которые помогают в распределении обращений и их приоритизации. Эти специальные поля – отдельная динамическая форма, которая автоматически заполняется из настроек, но в некоторых случаях пользователь сам может выбрать важность обращения. Для управления видимостью дополнительных полей нужно использовать `setScoringVisible`

#### 4.8.7 Поточный ввод

Иногда есть необходимость а поточном создании обращений, когда необходимо создать несколько обращений по одному шаблону с

незначительными изменениями значений полей, для этого существует флаг *MultiFlow*. В это режиме при отправке обращения не происходит переход на главный экран, а форма остается разрешается изменить поля и создать еще одно обращение. Управление этим флагом осуществляется с помощью функции `setMultiFlow([bool])`.

#### 4.8.8 Остальные флаги

Также существуют дополнительные флаги которыми можно управлять отображением обращения и доп информацией для НР `setMode`

`setDeclineVisible([bool],[id])` используется для отображения кнопки "Отозвать" в обращении `setAdditionalTemplate([templateid])` позволяет задать дополнительный шаблон при создании обращения `setContainCustomObjects([])` зарезервировано для внутреннего использования

### 4.9 Общие команды динамической формы

Динамическая форма позволяет выполнить команды дин языка не только при изменении компонентов, но и при открытии формы, и перед сохранением. Эти команды используются для дополнительной проверки введенных значений, или для скрытия технических полей перед открытием уже созданного обращения.

Для установки команды перед сохранением используется

`setBeforeSave([string([dynlang])])`

Дополнительный вызов `string()` необходим для экранирования символов динамического языка. Без него строка динамического языка была бы интерпретирована как команда и вызвана.

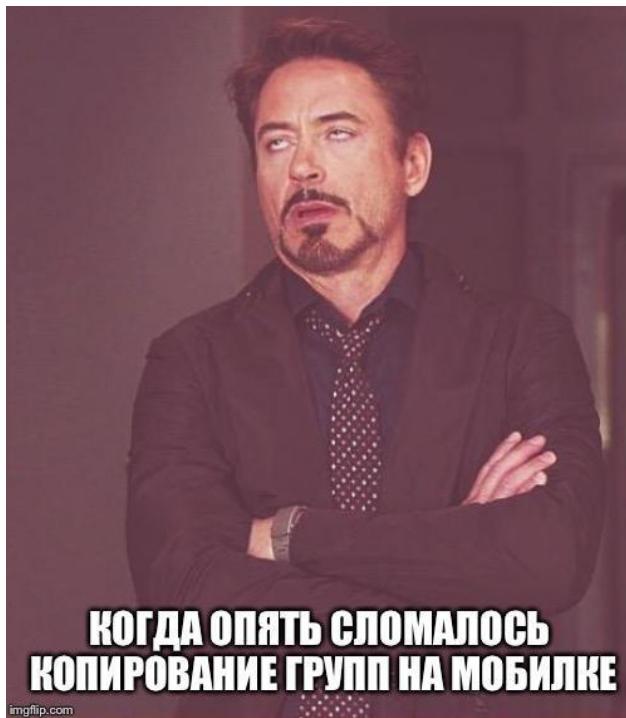
Для установки команды перед открытием необходимо использовать вызов АИ в режиме передачи *3DMap*

Динамические формы позволяют объединять последовательности элементов в единые сущности, называемые группами. Группы можно объединять под единым заголовком, визуально "сворачивать/разворачивать копировать и удалять.

## 4.10 Виды групп

Существуют несколько видов групп: так называемые, «legacy» и «плоские» группы – элементы логически объединенные общим идентификатором группировки – формат SMIT и старых версий ПО ДРУГа и многоуровневые или просто «группы» – современный формат, поддерживаемый ПО ДРУГа и позволяющий описывать сложные древовидные структуры динамических форм. Среда выполнения dom-core приводит legacy-форматы групп к древовидным на время работы с обращением, при этом в некоторых случаях при сохранении может потребоваться обратное преобразование формата для поддержки совместимости со старыми системами.

1. Плоские группы. Плоские группы представляют собой элементы, объединенные по параметру **groupid**. Такие группы не имеют заголовка, и визуально не отделяются от остальных элементов. Основная задача, которую решают данные группы – сортировка и группировка элементов в одном блоке, поддерживающим операции копирования и удаления. При работе через dom-core всегда требуется обратное преобразование в исходный формат.
2. Legacy группы. Основное отличие от плоских групп заключается в наличии заголовка. Есть отличительная особенность поддержки Legacy групп Лице Друга 1.0 и domcore: в первом случае поддерживается наличие более одного заголовка в группе, во втором же принято решение, что заголовок группы может быть один. Все остальные заголовки не обрезаются при обработке динамической формы. При визуализации Legacy Group могут сворачиваться и управляться командой динамического языка **setFocus**. При сохранении через dom-core они могут не разворачиваться и отправляться как нормальные группы, за исключением шаблонов помеченных атрибутом **needunwrap**, требующего развернуть
3. Группы. Целевой механизм групп. В xml-представлении ПО ДРУГа представлен элементом **<group>**, поддерживающим вложенные динамические элементы. Количество и тип вложенных элементов не ограничены: это могут быть и другие группы, что позволяет создавать сложные иерархические структуры динамических элементов. В json-представлении описываются массивом с именем "**childs**".



## 4.11 Копирование групп

Копирование групп – стандартная функциональность, поддерживаемый динамическим языком и позволяющий «тиражировать» элементы динамической формы, объединенные в группу. Копирование группы вызывается командой динамического языка **copyGroup** (разд. А, с. 88). Принято прописывать копирование группы либо в «sbcommand», либо в «sbtask» атрибуте кнопки «addButton». Функциональность копирования групп привязана к исходному (шаблонному) состоянию элементов в динамической форме, а не к их состоянию в момент копирования. Рассмотрим механизм копирования групп на примере представленного ниже json-представления шаблона:

```
{  
    "objectType": "group",  
    "logicalType": "group",  
    "id": "groupaddress",  
    "label": "Пункт отправления/прибытия",  
    "mandatory": false,  
    "style": "group",  
    "visible": true,
```

```
"width": "",  
"sbdbfield": "groupAddressTo",  
"sbonload": "",  
"sbcopyinfo": false,  
"child": [  
    {  
        "objectType": "select",  
        "logicalType": "combo",  
        "id": "addressto",  
        "label": "Маршрут следования",  
        "mandatory": true,  
        "sbcommand": "",  
        "sbmask": "",  
        "sbmodify": true,  
        "sbtask": "if(  
            [run(  
                [setValue(  
                    [getDistance(  
                        [getValues([find([id], [address], [grouptrip])]  
                    )]  
                ),  
                [array]  
            )]  
        )],  
        [if(  
            [isin(  
                [getText([distance], [array])], [])  
            )],  
            [run([clear([array])],  
                [setValue([0,00], [cost])],  
                [setValue(  
                    [plus(  
                        [#0.00],  
                        [getTexts([find([id], [plus([c], [ost])], [])])]  
                    )],  
                    [allcost]  
            )]  
        )]  
    ]  
]
```

```
)],
[setValue([0,000], [distancetext])
)],
[setValue( [0], [timetext])],
[setValue([0], [distancevalue])],
[setValue( [0], [kmdistancevalue] )],
[setValue( [0], [timevalue])],
[setValue(
[plus(
[#0.000],
[div( [#0.000], [plus( [#0.000],
[getTexts(
[find([id],
[plus([distanc], [evalue])], [])])
]),
[1000]
)
]
),
[alldistancetext]
)],
[setValue(
[plus(
[div( [#0], [plus( [#0.00], [getTexts(
[find([id],
[plus([tim], [evalue])], [])])]), [60]
)
]
),
[alltimetext]
)
]
),
[run(
[setValue(
[getText([time_text], [array])],
[timetext]
)],
[setValue(
```



```
[getValue( [date])],
[getUser([template])]
),
[cost]
)
),
[run(
[setValue(
[plus([#0.00],
[getTexts([find([id], [plus([c], [ost])], [])])])
]),
[allcost]
),
[setValue(
[plus( [#0.000], [div([#0.000], [plus( [#0.000],
[getTexts([find([id],
[plus([distanc], [evalue])], [])])], [1000])])
]),
[alldistancetext]
),
[setValue(
[plus([div([#0], [plus([#0.00], [getTexts(
[find([id], [plus([tim], [evalue])], [])])], [60])])
]),
[alltimetext]
)
)
),
[], []
)])]), [], ""),
"sbtitle": "",
"sbtype": "address",
"style": "combo",
"groupid": ""
},
{
"objectType": "text",
"logicalType": "date",
"id": "wait",
```

```
"label": "Время ожидания",
"mandatory": false,
"sbcommand": "run(
[setMinHours(
[0],
[setMaxHours([2], [this])]

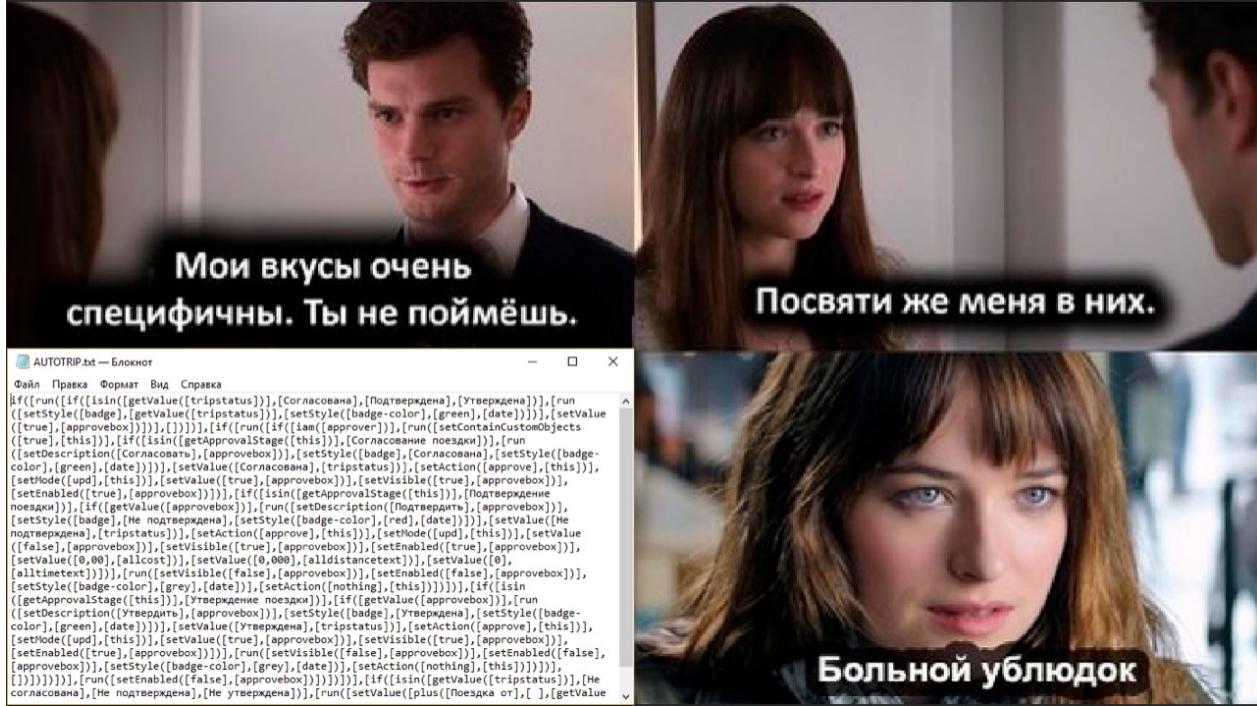
)],
[if(
[isin(
[getValue([cfgtransporttype] )],
[Такси]
)],
[setVisible([true], [this])],
[setVisible([false], [this])]

)
]
),
"sbmask": "HH:mm",
"sbmodify": true,
"sbtask": "",
"style": "text",
"visible": false,
"sbcopyinfo": false,
"groupid": "distanation"
},
{
"objectType": "checkbox",
"logicalType": "buttonAdd",
"id": "addgroupaddress",
"label": "Добавить еще один адрес",
"mandatory": false,
"sbmodify": true,
"sbtask": "copyGroup([groupaddress], [this])",
"stype": "buttonadd",
"style": "checkbox",
"visible": true,
"sbdbfield": "addGroupAddress",
```

```

    "text": "",
    "groupid": "distanation"
}
]
}

```



Пользователь заполняет значения «Маршрут следования» и «Время ожидания», и при нажатии на кнопку «Добавить еще один адрес» выполняется команда `sbtask`, в которой вызывается копирование группы. При этом, все заполненные ранее поля не учитываются: значения полей в скопированных элементах «Маршрут следования» и «Время ожидания» группе будут пустыми. Также не будут копироваться никакие динамические элементы, добавленные динамически. При копировании не учитываются никакие изменения, произведенные с элементами группы и самой группой. В том числе не учитываются изменения видимости, обязательности, возможности изменений и т.д.

Еще одной важной особенностью является визуальное отображения кнопок копирования. Визуально все кнопки для копирования групп скрываются, кроме последней в наборе. Исключительной особенностью обладают группы, элементы которых помещаются в одну строчку. Визуальное отображение элементов отделено от функционального. Если элементы группы для всех

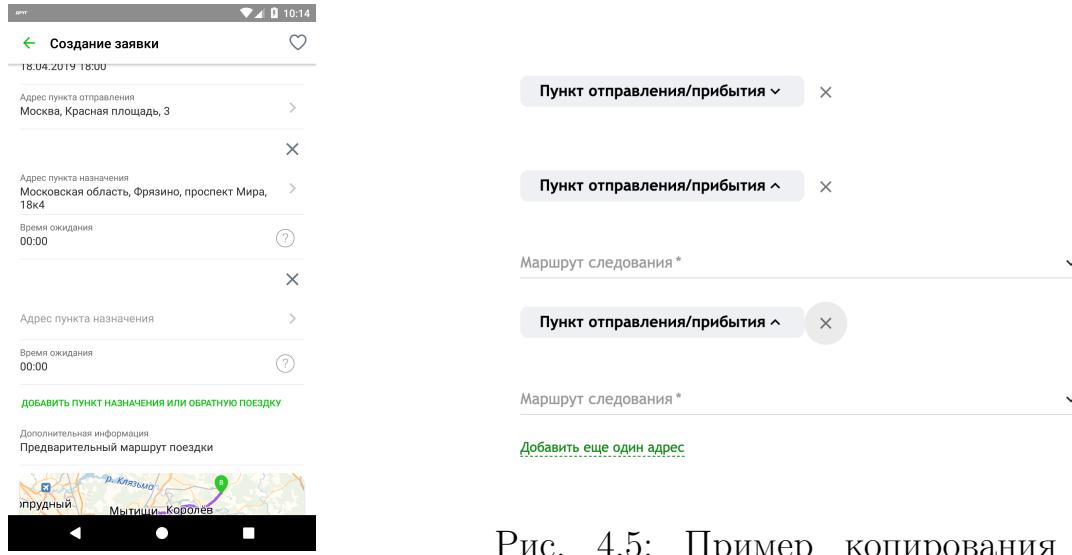


Рис. 4.4: Пример копирования плоской группы в мобильном клиенте

Рис. 4.5: Пример копирования и заголовков группы в интерфейсе Лицо 2.0

скопированных групп из одного множества удовлетворяют условию, что все элементы по ширине могут уместиться в одной строке (кнопка для копирования группы не учитывается при вычислениях), то визуально все группы склеиваются в одну и отображаются как общая группа с одним заголовком, при этом функциональная часть остается неизменной, там все также это представляется как разные группы. Если копирование работает над «Плоской» группой – проверять условия описанные выше не имеет смысла, так как плоские группы не имеют заголовка, поэтому проверка производится только для групп типа «Legacy» и стандартных групп.

## 4.12 Удаление групп

Удаление групп возможно только в том случае, если групп в подмножестве больше, чем одна. При добавлении группы через копирование у всех групп появляется возможность удалить группу через пользовательский интерфейс. Функция удаления – одна из немногих, которые не поддерживаются на уровне команд динамического языка, а разрешаются с использованием вызовов обработчиков через UI.

При нажатии на кнопку закрытия (крестик), происходит удаление группы.

Если удаляемая группа содержит элемент **buttonAdd**, то после выполнения закрытия, выполняется динамическая команда, установленная в параметр **text** этого поля. В рассмотренном ранее примере, данный механизм позволяет пересчитать расстояния и стоимость поездки при удалении адреса из маршрута.

# Глава 5

## Паттерны программирования на динамическом языке

### 5.1 Циклы

Вооружившись ранее изученным набором команд, вам не составит труда реализовать цикл на динамическом языке.

Напишем программу, которая выведет в поле «Результат» 100 точек:

Добавьте на динамическую форму скрытое поле с 'id' : 'counter'. Это поле будет служить счетчиком цикла.

Ниже приведен полный листинг программы:

```
if(
    [isnotin([getValue([counter])], [100])],
    [run(
        [setValue(
            [plus(
                [getValue([result])],
                [.]  

            )],
            [result]
        )],
        [setValue(
```

```

[plus(
    [getValue([counter])],
    [1]
),
[counter]
),
[call(
    [getValue([cmd])],
    [this]
)
]
)
]

```

Запустив программу, можно убедиться, что в поле «Результат» вывелоось ровно 100 точек.

Теперь рассмотрим принцип работы программы подробнее:

```

if(
    [isnotin([getValue([counter])], [100])],

```

В данном фрагменте кода мы проверяем, не выполняется ли условие неравенства нашего счетчика значению 100.

В этом случае выполняем тело цикла, состоящее из 3-х команд внутри блока «run»:

```
setValue([plus([getValue([result])], [.]), [result])]
```

Здесь происходит добавление новой точки в поле с результатом.

```
setValue([plus([getValue([counter])], [1]), [counter])]
```

Увеличиваем значение счетчика на 1.

```
call([getValue([cmd])], [this])
```

И, главная часть, обеспечивающая зацикливание нашей программы - вызываем повторное выполнение скрипта при помощи функции «Call»

## 5.2 Вложенные команды

Рассмотрим классическую задачу, встречающуюся в большинстве шаблонов динамических форм. Зачастую требуется установить ряд свойств динамического элемента, например, "выключить" и "скрыть":

```
run(  
    [setVisible([false],[this])],  
    [setMandatory([false],[this])],  
    [setEnabled([false],[this])]  
)
```

Указанный фрагмент кода прекрасно справляется с поставленной задачей. Однако, скрипт можно заметно облегчить, применив технику «Вложенных команд»:

Обратим внимание, что каждая динамическая команда в данном примере возвращает значение – id динамического элемента. Таким образом, мы можем передавать команды в качестве аргумента с ожидаемым значением id в другие команды. Применив данный подход дважды, сократим код примера до следующей записи:

```
setEnabled([false],  
    [setMandatory([false],  
        [setVisible([false],  
            [this]))])
```

Мы также избавились от необходимости использования динамической команды «run»!

В общем случае, алгоритм рефакторинга по технике «Вложенных команд» выглядит так:

1. В списке аргументов функции «run» убедитесь, что возвращаемые значения всех команд семантически совпадают с входными параметрами других команд. В большинстве случаев - это id элементов на форме.
2. Возьмите последнюю команду из списка и замените подходящий литерал аргумента предшествующей команды ее вызовом.
3. Удалите последнюю команду из списка.

4. Если в списке аргументов функции «run» осталась одна команда – замените блок «run» этой команды, иначе повторяйте шаги 2–3.



```
run(
    [setValue([Не согласовано!],[this])
    [setVisible([true],[this]),
    [setMandatory([true],[this])]
)
```

```
setValue([Не согласовано!],[setVisible([true],[setMandatory([true],[this])]))]
```

Внимание! Избегайте злоупотреблений рассмотренным приемом – избыточная вложенность усложняет читаемость и поддерживаемость кода: удалить или добавить команду в середину цепочки вызова становится затруднительно. Рекомендуемый уровень вложенности – не более 3–4.

### 5.3 Выделение функций

Рассмотрим задачу формирования списка месяцев в зависимости от текущей даты: в элемент типа select требуется вывести название текущего и двух последующих месяцев. Вот как эта задача решается в одном из шаблонов обращений:

```
if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])],
        [01]
    )],
    [setSelectedValue(
        [null],
        [setValue(
            [Март],
            [setValue(
                [Февраль],

```



```
[setValue(
    [Март] ,
    [this]
)]
)
)
)
],
[if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])] ,
        [04]
    )],
    [setSelectedValue(
        [null] ,
        [setValue(
            [Июнь] ,
            [setValue(
                [Май] ,
                [setValue(
                    [Апрель] ,
                    [this]
                )]
            )]
        )]
    )]
),
[if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])] ,
        [05]
    )],
    [setSelectedValue(
        [null] ,
        [setValue(
            [Июль] ,
            [setValue(
                [Июнь] ,
                [ setValue(
                    [Июнь] ,

```

```
[setValue(
    [Май] ,
    [this]
)]
)
)
),
[if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])] ,
        [06]
    )],
    [setSelectedValue(
        [null] ,
        [setValue(
            [Август] ,
            [setValue(
                [Июль] ,
                [setValue(
                    [Июнь] ,
                    [this]
                )]
            )]
        )]
    )]
),
[if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])] ,
        [07]
    )],
    [setSelectedValue(
        [null] ,
        [setValue(
            [Сентябрь] ,
            [setValue(
                [Август] ,

```



```
[setValue(
    [Сентябрь] ,
    [this]
)]
)
)
),
[if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])] ,
        [10]
    )],
    [setSelectedValue(
        [null] ,
        [setValue(
            [Декабрь] ,
            [setValue(
                [Ноябрь] ,
                [setValue(
                    [Октябрь] ,
                    [this]
                )]
            )]
        )]
    )]
),
[if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])] ,
        [11]
    )],
    [setSelectedValue(
        [null] ,
        [setValue(
            [Январь] ,
            [setValue(
                [Декабрь] ,

```

Как видим, данный пример содержит много повторяющихся элементов участков кода – по одному фрагменту на каждый из 12 месяцев. Хорошо, что в нашем календаре их не 42! Представьте объем доработок при изменении требований к задаче: например, заказчику потребуется выводить по 6 названий месяцев

Итак, приступим к рефакторингу.

Первым делом, обратим внимание на повторяющийся фрагмент кода:

```
[substr([getsysdate([+0])],[3],[5])]
```

Это скрипт получения значения номера текущего месяца. Вычислим его один раз и запомним в отдельном скрытом поле формы:

```
<text id="month_num" visible="false"
      sbcommand=
      "setValue([substr([getsysdate([+0])],[3],[5]),[this]])"
/>
```

Далее, нам потребуется структура для выставления соответствия номерам месяцев их названий. С этой задачей отлично справляется элемент «select» :

```
<select id="months" visible="false">

<option label="Январь">1</option>
<option label="Февраль">2</option>
<option label="Март">3</option>
<option label="Апрель">4</option>
<option label="Май">5</option>
<option label="Июнь">6</option>
<option label="Июль">7</option>
<option label="Август">8</option>
<option label="Сентябрь">9</option>
<option label="Октябрь">10</option>
<option label="Ноябрь">11</option>
<option label="Декабрь">12</option>
<option label="Январь">13</option>
<option label="Февраль">14</option>

</select>
```

Обратите внимание, что мы указали 2 «лишних» месяца – это потребуется для корректной работы нашего алгоритма в ноябре и декабре.

Добавим на динамическую форму select для вывода результата:

```
<select id="result" visible="true">
```

Реализуем функцию добавления в наш новый контрол очередного месяца:

```
<text id="add_month">
setValue(
  [getValue(
    [setSelectedValue(
      [plus(
        [getValue([month_num])],
        [getValue([i])]
      )],
      [months]
    )],
    [result]
  )],
  [result]
)
</text>
```

Где «i» - простое скрытое поле для хранения счетчика:

```
<text id="month_num" visible="false">0</text>
```

Для добавления нового месяца в итоговый select достаточно выполнить вызвать нашу функцию «add\_month» через оператор call:

```
call([getValue([add_month])], [this])
```

Теперь, для решения нашей задачи потребуется выполнить скрипт «add\_month» трижды, меняя значение «i»:

```
run(
  [call([getValue([add_month])], [this])],
  [setValue([1], [i])],
  [call([getValue([add_month])], [this])],
  [setValue([2], [i])],
  [call([getValue([add_month])], [this])],
  [setSelectedValue(null), [result]])
)
```

## 5.4 Упражнения

1. Используя код разобранного выше примера, решите задачу: в элемент типа select требуется вывести название текущего и 5 последующих нечетных месяцев. Для решения задачи рекомендуется использовать подход к реализации циклов, рассмотренный в разд. 5.1



# Приложение А

## Описание динамических команд

### setValue(value, id)

Устанавливает значение первого параметра в элемент, указанный во втором параметре.

1. Если элемент не select и значение первого параметра строка, то в text запишется строковое значение
2. Если элемент не select и значение первого параметра Map, то в text запишется первое значение из Map
3. Если элемент не select и значение первого параметра пустая Map-а, то в text запишется пустая строка
4. Если элемент select и первый параметр Map, то options будут записаны значения из Map
5. Если элемент select и первый параметр пустая Map, то будут удалены все options и в text будет установлена пустая строка.
6. Если элемент select и значение первого параметра строка, то будет добавлен один option с label и text равный значение второго параметра.

Возвращает: string - Значение второго параметра (идентификатор элемента)

Параметр	Тип	Описание
value	string   DLMap	Новое значение элемента

Параметр	Тип	Описание
id	string	Идентификатор элемента

## Пример

Элемент:

```
<form>
<text id="1" label="Наименование товара" style="text" type="2">
Ручка шариковая для сотрудника
</text>
</form>
```

1. setValue([Значение 1], [this]) -  
установит в текущий элемент значение test

Результат:

```
<form>
<text id="1"
label="Наименование товара"
style="text" type="2">
Значение 1
</text>
</form>
```

2. setValue([DLMap{"1": "Значение 1", "2": "Значение 2"}], [this])

Результат:

```
<form>
<text id="1" _parentId="0"
label="Наименование товара" style="text" type="2">
Значение 1
</text>
</form>
```

```
3. setValue([DLMap{}], [this])
```

Результат:

```
<form>
<text id="1" label="Наименование товара" style="text" type="2"/>
</form>
```

Элемент:

```
<form>
<select id="f1"> Выбранное значение
  <option id="f1_0"
label="Отсутствует оплата за замещение должностей">
Выбранное значение</option>
</select>
</form>
```

```
4. setValue([DLMap{"1": "Значение 1", "2": "Значение 2"}], [this])
```

Результат:

```
<form>
<select id="f1"> Выбранное значение
<option id="1" label="Значение 1">1</option>
<option id="2" label="Значение 2">2</option>
</select>
</form>
```

```
5. setValue([DLMap{}], [this])
```

Результат:

```
<form>
<select id="f1">
```

```
</select>
</form>

6.setValue([Значение 1], [this])
```

Результат:

```
<form>
<select id="f1"> Выбранное значение
<option id="Значение 1" label="Значение 1">Значение 1</option>
</select>
</form>
```

## getUser(attr)

Функция возвращает информацию о пользователе

Возвращает: string | boolean - Информация о пользователе

---

Параметр	Тип	Описание
attr	string	Значение аргумента

---

Допустимые значения аргумента:

```
getuser(fio)
getuser(firstName)
getuser(secondName)
getuser(middleName)
getuser(tabNum)
getuser(departName)
getuser(position) - Должность
getuser(vsp)
getuser(address)
getuser(phone)
getuser(phoneInner)
getuser(phoneMobile)
getuser(identifier)
```

```
getuser(id)
getuser(kadrCode)
getuser(category)
getuser(terbank)
getuser(findByTabno)
getuser(territoryCode)
```

## run(args)

Выполняет вложенные выражения. Порядок выполнения выражений не определен. Функция доступна только в выражениях динамического языка.

Возвращает: boolean - Возвращает всегда “true”

Параметр	Описание
args	Список выражений динамического языка

### Пример

```
run(
    [setValue([1],[id1])],
    [setValue([2],[id2])],
    [setValue([3],[id3])
])
```

## if(condition, trueExpr, falseExpr)

Вычисляет выражение в первом аргументе, если результат вычислений равно true то выполняется выражение во втором аргументе, иначе третий

Возвращает: boolean - Возвращает значение первого аргумента true/false

Параметр	Описание
condition	Условие
trueExpr	Выражение 1
falseExpr	Выражение 2

## Пример

```
if([idDigit([3])] ,  
    [setValue([Число] , [this])],  
    [setValue([Строка] ,  
[this]))]
```

## isIn(args)

Проверяет первый аргумент равен одному из следующих аргументов

Параметр	Тип	Описание
args	DLMap   *	Список параметров

## Пример

```
isIn([1] , [2] , [56] , [1])
```

## isNotIn(args)

Проверяет первый аргумент не равен одному из следующих аргументов

Параметр	Описание
args	Список параметров

## Пример

```
isNotIn([1] , [2] , [56] , [1])
```

## isDigit(arg)

Проверяет, является ли заданный аргумент числом

---

Параметр	Описание
arg	Аргумент

---

### Пример

```
isDigit([34])
```

## call(expr, id)

Выполняет команду динамического языка, переданную в первом параметре над элементом второго параметра, поддерживает выражения с асинхронными вызовами.

Возвращает: string идентификатор второго аргумента

---

Параметр	Тип	Описание
expr		Выражение на динамическом языке
id	string	Идентификатор элемента

---

### Пример

```
call([setValue([test], [this])], [id])
```

## IAm(arg)

Функция проверяет является ли пользователь приложения заявителем, инициатором или согласующим лицом в текущем обращении

---

Параметр	Описание
arg	Параметр может принимать значения creator, initiator или approver

---

### Пример

```
iAm([creator])
```

## getTexts(map)

Возвращает Map которая в value содержит видимые значения “label” option-элементов.

Возвращает: DLMap Map которая в value содержит видимые значения “label” option-элементов.

Параметр	Тип	Описание
map	DLMap	Мар-а ключи которой содержат id элементов на форме

Наример, если в getTexts передать Map {"f1": "-"} с id элемента "f1" и выполнить над формой:

```
<form>
<select id="f1"> Значение 1
  <option id="f1_0"
  label="Отсутствует оплата за замещение должностей">
  Значение 1
  </option>
</select>
</form>
```

Функция вернет Map структуру вида

```
"f1\_Отсутствует оплата за замещение должностей" :
"Отсутствует оплата за замещение должностей"}
```

## getValues(map)

Возвращает Map которая в value содержит значения “text” option-элементов.

Возвращает: DLMap - Возвращает Map которая в value содержит значения “text” option элементов.

Параметр	Тип	Описание
map	DLMap	Мар-а ключи которой содержат id элементов на форме

## Пример

Если в getValues передать Map {"f1": "-"} с id элемента "f1" и выполнить над формой

```
<form>
  <select id="f1"> Значение 1
    <option id="f1_0"
label="Отсутствует оплата за замещение должностей">Значение 1
      </option>
    </select>
</form>
```

Функция вернет Map структуру

```
{
  "f1_Отсутствует оплата за замещение должностей" :
  "Значение 1"
}
```

## find(attr, value, group)

Универсальная функция, выполняет поиск элементов в зависимости от первого аргумента. Если первый аргумент равен “id” то выполняется поиск элементов, id которых начинается со второго аргумента. Если первый аргумент равен “type”, то выполняется поиск элементов, sbtype - тип которых равен второму аргументу. В третьем параметре можно передать id группы в которой нужно выполнить поиск.

Возвращает: DLMap - Map в которой ключи это id найденных элементов, а value их значения

Param	Type	Description
attr	string	Атрибут по которому будет выполнен поиск, принимает значения “id” или “type”
value	string	Значение для поиска
group	string	Идентификатор группы в которой будет выполнен поиск

## Пример

Пример формы:

```
<form>
    <text id="f1" _parentId="0"
label="Наименование товара" style="text" type="2">
        Ручка шариковая для сотрудника</text>

    <select id="f2"
label="Вопрос" subtype="suggest" style="combo">
        <option id="id0empty" label="" />
        <option id="id00"
label="Отсутствует оплата за работу в ночное время">a1</option>
        <option id="id01"
label="Отсутствует оплата за замещение должностей">a2</option>
    </select>
</form>
```

1. `find([id], [f])`

Результат:

`DLMMap {"f1": "Ручка шариковая для сотрудника", "f2": ""}`

2. `find([type], [suggest])`

Результат:

```
DLMAP {"f2": ""}
```

## plus(args)>

Универсальная функция сложения, которая умеет складывать числа, строки и время в формате НН:мм Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: Результат сложения

Параметр	Описание
args	Список аргументов

### Пример

```
plus([1], [2], [3])
6
plus([], [], [1], [], [2], [3], [], [4], [], [])
10
plus([1], [3], [a])
13a
plus([1], [01:02], [3])
66
plus([a], [01:02], [3])
a01:023
plus([#0.00], [123], [0,0000001])
123.00
plus([1], [DLMAP{a : 1, b: 2, c: 3}], [3])
10
plus([1], [DLMAP{a : 01:00, b: 2, c: 3}], [3])
69
plus([a], [DLMAP{a : 01:00, b: 2, c: 3}], [3])
```

a01:00233

## **minus(args) -> \***

Функция вычитания Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: \* - Результат вычитания из первого аргумента всех последующих аргументов

Параметр	Описание
args	Числа, например [1],[2],[3]

### **Пример**

```
minus([4],[2],[1])  
1  
minus([#0.00],[123,0000002],[0,0000001])  
123.00
```

## **mul(args) -> \***

Функция умножения Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: \* - Результат умножения

Параметр	Описание
args	Числа, например [1],[2],[3]

## **div(args)**

Функция деления Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: \* - Результат деления

---

Параметр	Описание
args	Числа, например [1],[2],[3]

---

## indOf(string, substring)

Возвращает позицию с которой начинается подстрока substring в строке source

Возвращает: number | \* - Позиция в строке

---

Параметр	Тип	Описание
string	DLMap   string	Строка
substring	DLMap   string	Искомая подстрока

---

### Пример

```
indOf([console.log], [ole])  
4
```

## substr(string, start, end)

Возвращает подстроку с позиции start до позиции end

Возвращает: string - Подстрока

---

Параметр	Тип	Описание
string	string   DLMap	Строка в которой производится поиск
start	number   *	Начальная позиция
end	number   *	Конечная позиция (не включая)

---

### Пример

```
substr([console.log], [2], [4])
```

## len(string)

Возвращает длину строки

Возвращает: number | \* - Длина строки

Параметр	Тип	Описание
string	DLMap   string	Строка

### Пример

```
len([console.log])
```

```
11
```

## copyGroup(groupId, id, group)

Копирует группу элементов. Поддерживает два вида групп: группировка по атрибуту setGroupId (ПОДРУГА) или sbgroup (SMIT); группировка по типу элемента - (object type) group. Возвращает второй аргумент.

Возвращает: string - Значение второго аргумента

Param	Type	Description
groupId	string	Идентификатор группы
id	string	Идентификатор
group	string	Идентификатор группы в которой нужно выполнить копирование группы

## getDistance(args)

Возвращает из Яндекса расстояние между координатами

Возвращает: DLMap - Словарь со значениями distance - расстояние в метрах, distance\_text - расстояниях в км., time - время поездки в секундах,

time\_text - время в часах

Параметр	Тип	Описание
args	DLMap	Словарь состоящий из координат “широта:долгота”

### Пример

```
address = { // "longitude:latitude:GUID"
            "a": "37.589283:55.745850:GUID",
            "b": "36.715736:55.745850:GUID",
            "c": "37.515736:55.673816:GUID",
            "d": "37.715736:55.773816:GUID",
        }

getDistance([address])
```

Результат:

```
DLMap {
    distance: '353637.28597939014',
    distance_text: '350 км',
    time: '25415.397077530622',
    time_text: '7 ч 4 мин'
}
```

## execute3DTask(TASK, args, map)

Получение данных из автосполнителя в виде 3DMap  
Возвращает: DLMap - Ответ от автосполнителя

Параметр	Тип	Описание
TASK	DLMap   string	Название задачи в автосполнителе
ARG0	DLMap   string	Произвольный аргумент 1
ARG1	DLMap   string	Произвольный аргумент 2
ARGN	DLMap   string	Произвольный аргумент N
map	DLMap	Произвольный аргумент

## executeTask

Получение данных из автосполнятора

Возвращает: Object | \* - Ответ от автосполнятора

Параметр	Тип	Описание
TASK	DLMap   string	Название задачи в автосполняторе
ARG0	DLMap   string	Произвольный аргумент
ARG1	DLMap   string	Произвольный аргумент
ARGN	DLMap   string	Произвольный аргумент

## execute3DTask

Получение данных из автосполнятора в виде 3DMar объекта с actions

Возвращает: DLMap | \* - Ответ от автосполнятора

Параметр	Тип	Описание
TASK	DLMap   string	Название задачи в автосполняторе
ARG0	DLMap   string	Произвольный аргумент
ARG1	DLMap   string	Произвольный аргумент
ARGN	DLMap   string	Произвольный аргумент
DLMap	DLMap   string	текущее состояние динамической формы

## getUsersByOrganizationRequest

Получение пользователя по подразделению

Параметр	Тип	Описание
realtyObj	DLMap   string	Идентификатор объекта недвижимости
text	DLMap   string	Поисковый запрос

## setValues(arg0-arg4)

Выполняет команды 3DMaps над динамической формой. Поддерживаются команды: attr, value, new, valueselected, command, delete, beforesave

Возвращает: null id - элемент над которым выполняет 3DMap(может отсутствовать, если запрос к АИ) tid - идентификатор запроса который выполняется в рамках обработки callback - функция обработки isDefaultSettings - включает или отключает silentMode

Параметр	Тип	Описание
ARG0	3DMap	map 3DMap структура с командами
ARG1	string	id or tid
ARG2	string   function	tid or callback
ARG3	function   undefined	callback
ARG4	boolean   undefined	isDefaultSettings

### Пример

Пример 3DMap.

```
[  
 {  
   "id": "cost",  
   "actions": [  
     {  
       "action": "value",  
       "values": [  
         {  
           "id": "22.25",  
           "value": "22.25"  
         }  
       ]  
     },  
     {  
       "action": "command",  
       "values": [  
         {  
           "id": "22.25",  
           "value": "22.25"  
         }  
       ]  
     }  
   ]  
 }]
```

```

        "id": "100",
        "value": "setSaveEnabled([true],[cost])"
    }
]
},
{
    "action": "attr",
    "values": [
        {
            "id": "error",
            "value": ""
        },
        {
            "id": "rightValue",
            "value": "true"
        }
    ]
}
]
// map, id, tid, callback, isDefaultSettings

```

## getValue(id)

Возвращает текущее значение элемента

Возвращает: DLMap со значениями элемента или пустую

---

Параметр	Описание
id	id элемента

---

## setMandatory(bool, id)

Функция устанавливает обязательность заполнения контроля

Возвращает: id контрола

Параметр	Тип	Описание
bool		
id	String   DLMap	id контрола

## setVisible(bool, id)

Функция устанавливает признак видимости элемента на форме  
Возвращает: id контрола

Параметр	Тип	Описание
bool		
id	String   DLMap	id контрола

## setEnabled(bool, id)

Функция устанавливает признак возможности редактирования элемента  
Возвращает: id контрола

Параметр	Тип	Описание
bool		
id	String   DLMap	id контрола

## setValid(bool, description, id)

Функция устанавливает контролу признак ошибки

Параметр	Описание
bool	
description	Текст ошибки
id	

## setSelectedValue(val, id)

Функция устанавливает выбранное значение для контроллов типа Select  
Возвращает: id элемента

Параметр	Тип	Описание
val	String   DLMap	Значение option-элемента - элемента списка
id		id элемента

### Пример

```
setselectedvalue([a1], [f2])
```

Пример формы:

```
<form>
<select id="f2" setGroupId="group1" label="Вопрос" >
  <option id="id0empty" label="" />
  <option id="id00"
label="Отсутствует оплата за работу в ночное время">a1
  </option>
  <option id="id01"
label="Отсутствует оплата за замещение должностей">a2
  </option>
</select>
</form>
```

Результат:

```
<form>
<select id="f2" setGroupId="group1" label="Вопрос" >a1
  <option id="id0empty" label="" />
  <option id="id00"
label="Отсутствует оплата за работу в ночное время">a1
  </option>
  <option id="id01"
```

```
label="Отсутствует оплата за замещение должностей">a2
    </option>
</select>
</form>
```

## setWidth(val, id)

Функция устанавливает ширину контрола  
Возвращает: id элемента

Параметр	Описание
val	Значение ширины контрола
id	id элемента

## getText(map, id)

- Если элемент не select, то возвращает map где ключ и значени value элемента
- Если элемент select и передан один аргумент - id элемента, то функция вернет map с text и label выбранного option- элемента
- Если элемент select который не имеет выбранных option-элементов, то вернется map с одним пустым значением
- Если первый аргумент функции map со значениями text - option элементов, то функция вернет map с text и label option- элементов

---

Param

---

map

---

id

---

## isMandatory(id)

Возвращает обязательно ли поле для заполнения

---

Параметр	Описание
id	id элемента

---

## isVisible(id)

Возвращает видимо ли поле на форме

---

Параметр	Описание
id	id элемента

---

## isEnabled(id)

Возвращает доступен ли элемент для редактирования

---

Параметр	Описание
id	id элемента

---

## isValid(id)

Возвращает правильно ли заполнен элемент формы

---

Параметр	Описание
id	id элемента

---

## isFileMandatory(id)

Возвращает обязательность вложений при создании обращений

**Tod:** why func has id of element in params

---

Параметр	Описание
id	id элемента

---

## isFileEnabled(id)

Возвращает признак доступности вложений

---

Параметр	Описание
id	id элемента

---

## clear(id)

Очищает значение элемента

Возвращает: string - id элемента

---

Параметр	Тип	Описание
id	String   DLMap	id элемента

---

## setList(\_labels, \_texts, id)

Устанавливает option-элементы

Возвращает: string - id элемента

---

Параметр	Тип	Описание
_labels	DLMap   string	строка label разделенная pipe-символом
_texts	DLMap   string	строка text разделенная pipe-символом
id		id элемента

---

## Пример

```
setList([label1|label2|label3],[id1|id2|id3],[this])
```

## **setMask(value, id)**

Устанавливает маску ввода

Параметр	Описание
value	Маска ввода
id	id элемента

## **setFocus(val, id)**

Устанавливает фокус на элементе

Параметр	Тип	Описание
val	boolean	Значение
id		id элемента

## **setMax(value, id)**

Устанавливает максимальное значение для элемента

Параметр	Описание
value	Максимальное значение
id	id элемента

## **setMin(value, id)**

Устанавливает минимальное значение для элемента

Параметр	Описание
value	Минимальное значение
id	id элемента

## string(value)

Возвращает строковое значение первого параметра

Возвращает: string - Строковое значение первого параметра

Param

value

## setBeforeSave(value, id)

Устанавливает команду, которая будет выполнена перед сохранением

Возвращает: string - id элемента

Параметр	Описание
value	Комманда
id	id элемента

## getApprovalStage(id)

Возвращает название этапа согласования по обращению

Возвращает: string - Название этапа согласования

**Throws:**

- Error("approvalStage у внешнего объекта не определен")

Параметр	Описание
id	id элемента

## setCheckSign(bool, id)

Устанавливает признак подписания формы при сохранения обращения

**Throws:**

- FunctionNotImplementedError

Параметр	Описание
bool	
id	id элемента

## **setFileMandatory(bool, id)**

Устанавливает признак обязательности вложения файла  
Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

## **openAlert(type, message)**

Открывает окно с сообщением

**Throws:**

- FunctionNotImplementedError

Параметр	Описание
type	Тип сообщения info, warning, error
message	Текст сообщения

## **setSaveEnabled(bool, id)**

Устанавливает признак доступности кнопки “Сохранить”

Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

## setFileEnabled(bool, id)

Устанавливает признак доступности кнопки “Вложить”

Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

## setExternalSystem(extsystem, id)

Устанавливает код внешней системы

Возвращает: string - id элемента

Param	Type	Description
extsystem	SM   FRIEND	Код внешней системы SM или FRIEND
id		id элемента

## setInformationVisible(bool, id)

Устанавливает признак видимости поля “Информация”

Возвращает: string - id элемента

Параметр	Описание
bool	

---

Параметр	Описание
id	элемента

---

## **setInitiator(value, id)**

Устанавливает инициатора в обращении

Возвращает: string - id элемента

---

Параметр	Описание
value	Идентификатор инициатора
id	id элемента

---

## **setMultiFlow(bool, id)**

Устанавливает признак мультипоточного ввода по шаблону

Возвращает: \* - id элемента

---

Параметр	Описание
bool	
id	id элемента

---

## **setRedirect(value, id, skipConvert)**

Устанавливает ссылку на которую надо перейти

Возвращает: \* - id элемента

---

Параметр	Тип	Описание
value	string	Ссылка
id	string	id элемента
skipConvert	Boolean	не преобразовывать ссылку LD1 > LD2

---

## setRelationship(objectId, objectType, id)

Используется только с ПОДРУГА! Функция устанавливает связь созданного обращения с объектом типа objectType с номером objectId

Возвращает: id элемента

Параметр	Описание
objectId	Номер объекта
objectType	Тип объекта ZNO/ZNU
id	id элемента

## setAdditionalTemplate

Устанавливает шаблона по которому нужно дополнительно создать обращение

**Throws:**

- FunctionNotImplementedError

Param	Type	Description
extsystem	SM   FRIEND	Код внешней системы SM или FRIEND
templateId		Идентификатор шаблона обращения
id		id элемента

## setScoringVisible(bool, id)

Устанавливает признак видимости поля «Скоринг» в обращении

Параметр	Описание
bool	

---

Параметр	Описание
id	id элемента

---

## getSysDate(value)

Возвращает смещение относительно текущей даты  
Возвращает: Дата

---

Параметр	Описание
value	Смещение в днях

---

### Пример

getSysDate([+5]) - вернет текущую дату + 5 дней

## getInitiator(value)

Получение информации об инициаторе обращения  
Возвращает: string - Информация по параметру

---

Param	Description
value	Параметр fio, domaincode, ou, tabnum, id, departcode, position, guid, email, vsp, tbcode

---

## setMinHours(value, id)

Устанавливает минимальное значение в часах  
Возвращает: \* - id элемента

---

Параметр	Описание
value	Значение часов в 24 - часовом формате

---

---

Параметр	Описание
----------	----------

---

id	id элемента
----	-------------

---

## setMaxHours(value, id)

Устанавливает максимальное значение в часах

Возвращает: \* - id элемента

---

Параметр	Описание
----------	----------

---

value	Значение часов в 24 - часовом формате
-------	---------------------------------------

id	id элемента
----	-------------

---

## setContainCustomObjects(bool, id)

Устанавливает признак того что созданное обращение будет содержать “хранимые” объекты, например объект закупки, поездки и т.д.

---

Param

---

bool

id

---

## getSize(param)

Возвращает количество option-элементов в элементе если передан идентификатор элемента или количество полей в DLMap если передана DLMap

Возвращает: \* - id элемента

---

Параметр	Тип	Описание
----------	-----	----------

---

param	string   DLMap	id элемента или DLMap
-------	----------------	-----------------------

---

## **setStep(value, id)**

Устанавливает шаг для выбора значения

Возвращает: \* - id элемента

Параметр	Описание
value	Размер шага
id	id элемента

## **setStyle(arg, value, id)**

Устанавливает стиль для атрибута

Поддерживаемые стили:

color - цвет элемента badge - текст badge - элемента badge-color - цвет badge - элемента width - ширина элемента (только web-версии Лица ДРУГА)

Возвращает: string - id элемента

Параметр	Описание
arg	Стиль
value	Значение
id	id элемента

## **compare(left, operand, right)**

Функция сравнения

Доступны операторы сравнения >, >=, ==, <, <=

Параметр	Описание
left	Первый аргумент для сравнения
operand	Оператор сравнения
right	Второй аргумент для сравнения

## **setInitiatorVisible(bool, id)**

Устанавливает признак видимости поля смены ВК

Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

## **setDescription(text, id)**

Устанавливает лейбл у элемента

Возвращает: String - Идентификатор элемента

Параметр	Тип	Описание
text	String	Текст лейбла
id	String	Идентификатор элемента

## **getRelatedId()**

Возвращает идентификатор связанного обращения

Возвращает: String - Идентификатор связанного обращения

## **setCopyVisible(bool, id)**

Устанавливает видимость кнопки копирования шаблона

Возвращает: String - Возвращает второй аргумент

Параметр	Тип
bool	Boolean
id	String

## **setDeclineVisible(bool, id)**

Устанавливает видимость кнопки “Отменить обращение”

Возвращает: String - Возвращает второй аргумент

Параметр	Тип
bool	Boolean
id	String

## **setCloseVisible(bool, id)**

Устанавливает видимость кнопки “Закрыть группу”

Возвращает: String - Возвращает второй аргумент

Параметр	Тип
bool	Boolean
id	String

## **getClientType()**

Возвращает тип клиента. Возможные значение: ios, android, web-sigma, web-alpha

Возвращает: string - Тип клиента

## **isMobileClient()**

Возвращает является ли клиент мобильным приложением

## **isWebClient()**

Возвращает является ли клиент web-приложением

## **isIOSClient()**

Возвращает является ли клиент iOs-приложением

## **isAndroidClient()**

Возвращает является ли клиент Android-приложением

## **isWebSigmaClient()**

Возвращает является ли клиент web sigma приложением

## **isWebAlphaClient()**

Возвращает является ли клиент web alpha приложением

## **isFace20()**

Функция возвращает true если это web приложение face20 Костыль нужен для авто редиректа на /friendface если шаблон не работает на face20. Оставляем пока face20 не стабилизируется.

## **setAction(value, id)**

Устанавливает значение в атрибут sbaction элемента

Возвращает: id

---

Param

---

value

id

---

## **setMode(value, id)**

Устанавливает значение в атрибут sbmode элемента

Возвращает: id

---

Param

---

value

id

---

## **dateDiff(start, end, id)**

Возвращает разницу в днях

---

Param

---

start

end

id

---

## **isCreateMode()**

Форма в режиме создания обращения

## **isOpenMode()**

Форма в режиме созданного обращения

## **setSaveVisible(bool, id)**

Устанавливает видимость кнопки сохранения шаблона

Возвращает: String - Возвращает второй аргумент

---

Параметр Тип

---

bool Boolean

Параметр	Тип
id	String

## createDocument

Заполняет шаблон документа (templateName) данными из обращения и возвращает его в качестве вложения с названием fileName.extension | Param | Type | Description || — | — | — | — | — | fileName | String | Имя возвращаемого вложения | | templateName | String | Имя шаблона документа, куда будут подставлены данные из обращения | | extension | String | Расширение возвращаемого вложения (доступные значения PDF). Не обязательный параметр |

Пример:

```
setValue(
  [createDocument(
    [Заявление] , [ЗаявлениеШаблон] , [PDF])],
  [attachment_el_id]
)
```

Делегирует обращение к серверу и формирование документа-вложения функции doCreateDocument(tid, params, callback) Пример:

```
DOMCore.doCreateDocument = (tid, params, callback) => {
  //params.fileName - имя возвращаемого вложения
  //params.templateName - имя шаблона документа
  //params.extension - расширение возвращаемого вложения
  //params.interaction - объект Interaction
  const response = anyDelegateFunction(params.fileName,
  params.templateName, params.extension, params.interaction);
  callback(tid, response, null);
}
```

Функция доступна из JS шаблона через глобальный вызов doCreateDocument Пример:

```

var fileName = 'Заявление';
var templateName = 'ЗаявлениеШаблон';
var extension = 'PDF';
doCreateDocument(fileName, templateName,
    extension, onSuccess, onError);

var onSuccess = function(attachmentData) {
    console.log(JSON.stringify(attachmentData));
}

var onError = function(err) {
    console.error(err && err.userMessage);
}

```

## signAttachment(attachment,mode)

Запускает процесс подписи вложения (attachment) и возвращает подписанное вложение<

Param	Type	Description
attachment	String	Объект вложения(ий) (AttachmentDTO) сериализованный в JSON
mode	String	Режим подписи (доступные значения CLOUD - облачная подпись, TABLET - подпись через ТМ)

Пример:

```

setValue(
    [signAttachment(
        [getValue([nonsignedattachment])],
        [CLOUD]
    )]
)

```

```
)],  
[attachmentelement]  
)
```

Делегирует подписание функции  
doSignAttachment(tid, signAttachmentParams, callback)  
Пример:

```
DOMCore.doSignAttachment = (tid, params, callback) => {  
    // params.attachment - подписываемое вложения  
    // params.signMethod - метод подписания (CLOUD, TABLET)  
    const response = anyDelegateFunction(params);  
    callback(tid, response, null);  
}
```

Функция доступна из JS шаблона через глобальный вызов

```
doSignAttachment(attachment, method, onSuccess, onError)
```

Пример:

```
var attachment = {asyncId: 12345};  
var method = 'CLOUD';  
doSignAttachment(attachment, method, onSuccess, onError);  
  
var onSuccess = function(attachmentData) {  
    console.log(JSON.stringify(attachmentData));  
}  
  
var onError = function(err) {  
    console.error(err && err.userMessage);  
}
```



## Приложение В

### Примеры 3DMap структур

```
[  
{  
  "id": "ACCESSUSER",  
  "actions": [  
    {  
      "action": "value",  
      "values": [  
        {  
          "id": "Андропов Юрий Владимирович",  
          "value": "Андропов Ю.В. (00543846)"  
        },  
        {  
          "id": "Брежнев Леонид Ильич",  
          "value": "Брежнев Л.И. (00543847)"  
        }  
      ]  
    }  
  ],  
  {  
    "id": "NoBoss",  
    "actions": [
```

```
{  
    "action": "value",  
    "values": [  
        {  
            "id": "не проставлен согласующий",  
            "value": "не проставлен согласующий"  
        }  
    ]  
},  
{  
    "id": "adlinks",  
    "actions": [  
        {  
            "action": "value",  
            "values": [  
                {  
                    "id": "[Установка признаков хранения пароля]",  
                    "value": "[Установка признаков хранения пароля]"  
                }  
            ]  
        }  
    ]  
},  
{  
    "id": "unlocksudirURL",  
    "actions": [  
        {  
            "action": "value",  
            "values": [  
                {  
                    "id": "[Разблокировать УЗ в СУДИР]",  
                    "value": "[Разблокировать УЗ в СУДИР]"  
                }  
            ]  
        }  
    ]  
}
```

```
        }
    ]
},
{
  "id": "otherSudir",
  "actions": [
    {
      "action": "attr",
      "values": [
        {
          "id": "title",
          "value": "Динамические группы"
        }
      ]
    }
  ]
}
```

