
GlobalServerAppLessons

Выпуск 0.0.65

"Бизнес Технологии" ООО

нояб. 21, 2025

Содержание

1	Предисловие	4
1.1	Структура курса	4
1.2	Учебные материалы	4
1.3	Требование к оборудованию	5
1.4	Рабочее место	5
2	Урок 1. Архитектура, средства и языки разработки	7
2.1	Архитектура системы	8
2.2	База данных	9
2.3	Классы	10
2.4	Выборки	11
2.5	Модуль	11
2.6	Dbeaver	11
2.7	SQL	11
2.8	PL/pgSQL	12
2.9	IntelliJ Idea	12
2.10	Конфигуратор	12
2.11	Язык scala	12
2.12	Язык jexl	13
2.13	Настройка рабочего места	13
2.14	Запуск сервера приложения в режиме отладки	15
2.15	Запуск конфигуратора	16
2.16	Создание модуля	17
2.17	Учебная схема для работы с sql	19
2.18	Практика работы с sql	21
2.19	Упражнения по языку sql	22
2.20	Практика работы с scala	26
2.21	Упражнения по scala	31
2.22	Практика работа с jexl	33
2.23	Упражнения по jexl	34
2.24	Самостоятельное изучение	34
3	Урок 2. Сессии, Модель данных	34
3.1	Цель курса	34
3.2	Сессия	35
3.3	Бизнес объект	36

3.4	Общие сведения о классах	36
3.5	Общие сведения о выборках	39
3.6	Взаимодействие с базой данных	39
3.7	Практика	43
3.8	Создание классов для модуля библиотека	43
3.9	Создание выборки основного меню приложения	44
4	Урок 3. Интерактивная бизнес логика.	51
4.1	Отображение	51
4.2	Операции	51
4.3	Атрибуты	52
4.4	Типы фреймов	52
4.5	Средства отладки	52
4.6	Практика	52
5	Урок 4. Практика разработки. Часть 1	54
5.1	Миксины	54
5.2	Документы	54
5.3	Коллекции	54
5.4	Автономумерация	54
5.5	Практические задания	54
6	Урок 5. Практика разработки. Часть 2	60
6.1	Типы объектов	60
6.2	Состояния	60
6.3	Практические задания	60
7	Урок 6. Практика разработки. Часть 3	63
7.1	Диалог с пользователем	63
7.2	Мультиселект	64
7.3	Группировка объектов	64
7.4	Объектные характеристики	64
7.5	Практические задания	64
8	Урок 7. Практика разработки. Часть 4	68
8.1	Сервис универсальных коллекций (v-collection)	68
8.2	Денормализация классов-деревьев	68
8.3	Режимы редактирования выборки	68
8.4	Передача параметров в выборку	69
8.5	Клонируемые операции	69
8.6	Фильтры	69
8.7	Практика	70
9	Урок 8. Дополнительные возможности	71
9.1	Поиск по шаблону	72
9.2	Универсальный фильтр	72
9.3	Работа с файлами	72
9.4	Шаблоны Thymeleaf	72
9.5	Точки расширений	72
9.6	Проектное переопределение кода	72
9.7	Логирование	73
9.8	Мониторинг производительности	73
9.9	SSH-сервис сервера приложений	74
9.10	Локализация приложений	75
9.11	Миграция данных	75

9.12 Практическое задание	75
10 Урок 9. Настройка печатных форм	77
10.1 1. Печатные формы	77
10.2 2. Шаблоны отчетов	83
10.3 3. Jaspersoft Studio	85
10.4 Практики	90
11 Урок 10. Администрирование, аудит	92
11.1 Аудит	92
11.2 Пользователь	92
11.3 Администрируемый объект	92
11.4 Объектные привилегии	92
11.5 Элементарная привилегия	93
11.6 Роль	93
11.7 Профиль пользователя	93
11.8 Практика	93
12 Контрольное задание	99
12.1 Вариант 1	99
12.2 Вариант 2	103
13 Приложение 1. Руководство разработчика	107
13.1 Введение	107
13.2 Классы	153
13.3 Выборки	228
13.4 Инструменты	318
13.5 Отчеты	357
13.6 Организация разработки	408
13.7 Руководство Jexl	490
13.8 Дополнительно	531
13.9 Приложение	635
14 Приложение 2. Руководство администратора	777
14.1 Предисловие	777
14.2 Введение	777
14.3 Приложение администратор	784
14.4 Дискретный доступ	796
14.5 Связь выборок с администрируемыми объектами	812
14.6 Телеметрия	820
14.7 Политика паролей	828
14.8 Администрирование Jexl	833
14.9 Журнал доступа к объектам системы	835
14.10 Администрирование Rest-сервисов	835
14.11 Администрирование SOAP-вызовов	836
15 Приложение 3. Ответы на задания	836
16 Приложение 4. GlobalFrameworkStarterKit	836
16.1 Описание	836
16.2 Установка	837
16.3 Шаблоны IntelliJ IDEA (GigaIDE)	837
16.4 External Tools	839

1 Предисловие

Курс предназначен для изучения средств быстрой разработки Global FrameWork for PostgreSQL и получения практических навыков по созданию на нем полноценных приложений.

В курсе рассматриваются основные понятия, устройство и принципы работы Global3-FrameWork, а также технологии (архитектура, алгоритмы, структуры данных), лежащие в его основе.

При прохождении курса обучающийся ознакомится с:

- Инструментами и принципами разработки
- Перечнем документации по используемым технологиям

По прохождению курса обучающийся сможет самостоятельно:

- Осуществлять разработку приложений, построенных при помощи платформы Global
- Настраивать логику работы и создавать пользовательские интерфейсы
- Создавать печатные формы
- Администрировать систему с точки зрения доступа
- Использовать на практике многочисленные сервисные возможности среды Global FrameWork

1.1 Структура курса

Курс разбит на 10 уроков, каждый урок ориентировочно занимает от 4 до 8 часов в зависимости от объема выполнения самостоятельной практики. Каждый урок содержит:

- Перечень тем и их краткое описание
- Ссылки на подробную документацию по темам
- Практические занятия под руководством специалиста(опционально)
Для подобных упражнений будут описаны пошаговые инструкции и примеры кода
- Примеры кода
Примеры кода располагаются в модуле **demo** для учебного проекта
- Самостоятельные практические задания

1.2 Учебные материалы

Курс распространяется в формате:

- html
Удобно для чтения
- pdf
Удобно для печати

Видеоуроки курса:

№	Название	Видео
1	Архитектура, средства и языки разработки. Настройка рабочего места	YT VK
2	Сессии и модель данных	YT VK
3	Интерактивная бизнес логика	YT VK
4	Практика разработки. Часть 1	YT VK
5	Практика разработки. Часть 2	YT VK
6	Практика разработки. Часть 3	YT VK
7	Практика разработки. Часть 4	YT VK
8	Дополнительные возможности	YT VK
9	Настройка печатных форм	YT VK
10	Администрирование, аудит	YT VK

Опционально к курсу могут прилагаться:

- Образы *virtual vm*
- Видео материалы уроков
- Учебный проект

1.3 Требование к оборудованию

Характеристика сервера	Рекомендуемые параметры
ОС	Windows 10 и выше
CPU	8 ядер 3.5 GHz или более
Оперативная память	16 – 32 Gb или более
Память	SSD 20 Gb свободного места
Монитор	разрешение 1920×1080

1.4 Рабочее место

Virtual vm

В случае прохождения курса с опцией `virtual vm`, вам буду доступны образы виртуальной машины:

- **base**
Базовый образ с предустановленной jdk а так же дистрибутивами для установки. Дистрибутивы для установки располагаются в каталоге `c:\distr`
- **start**
Образ с настроенным рабочим местом. При этом разделы с настройкой программного обеспечения можно пропустить

Запустить виртуальный образ можно практически на любой виртуальной машине.

Совет

Используйте **VMware workstation player** как наиболее простую в использовании и достаточно высокопроизводительную виртуальную машину.

Виртуальные машины на которых проводилось тестирование в порядке убывания производительности:

- Hyper-v
- VMWare
- Virtual box

VMWare workstation player

Установка:

1. Скачайте [VMWare workstation player](#)
2. Запустите установку и следуйте рекомендациям
3. Создайте виртуальную машину
4. Переключите диск на требуемый образ

Для этого:

1. Зайдите в настройки виртуальной машины
2. Удалите текущий **hard disk**
3. Создайте новый **hard disk**
При этом укажите требуемый файл с образом

Совет

Если виртуальная машина советует отключить миграцию диска для повышения производительности, добавьте строку `ulm.disableMitigations="TRUE"` в файл конфигурации виртуальной машины (`.vmx`)

Virtual box

Установка:

1. Скачайте дистрибутив [virtual box](#)
2. Запустите установку и следуйте рекомендациям
3. Подключите виртуальную машину
Подробнее в [Руководство пользователя virtual box](#) # создание виртуальной машины

Совет

Если у вас не работает буфер обмена между гостевой ос и основной ос, переустановите гостевое дополнение

2 Урок 1. Архитектура, средства и языки разработки

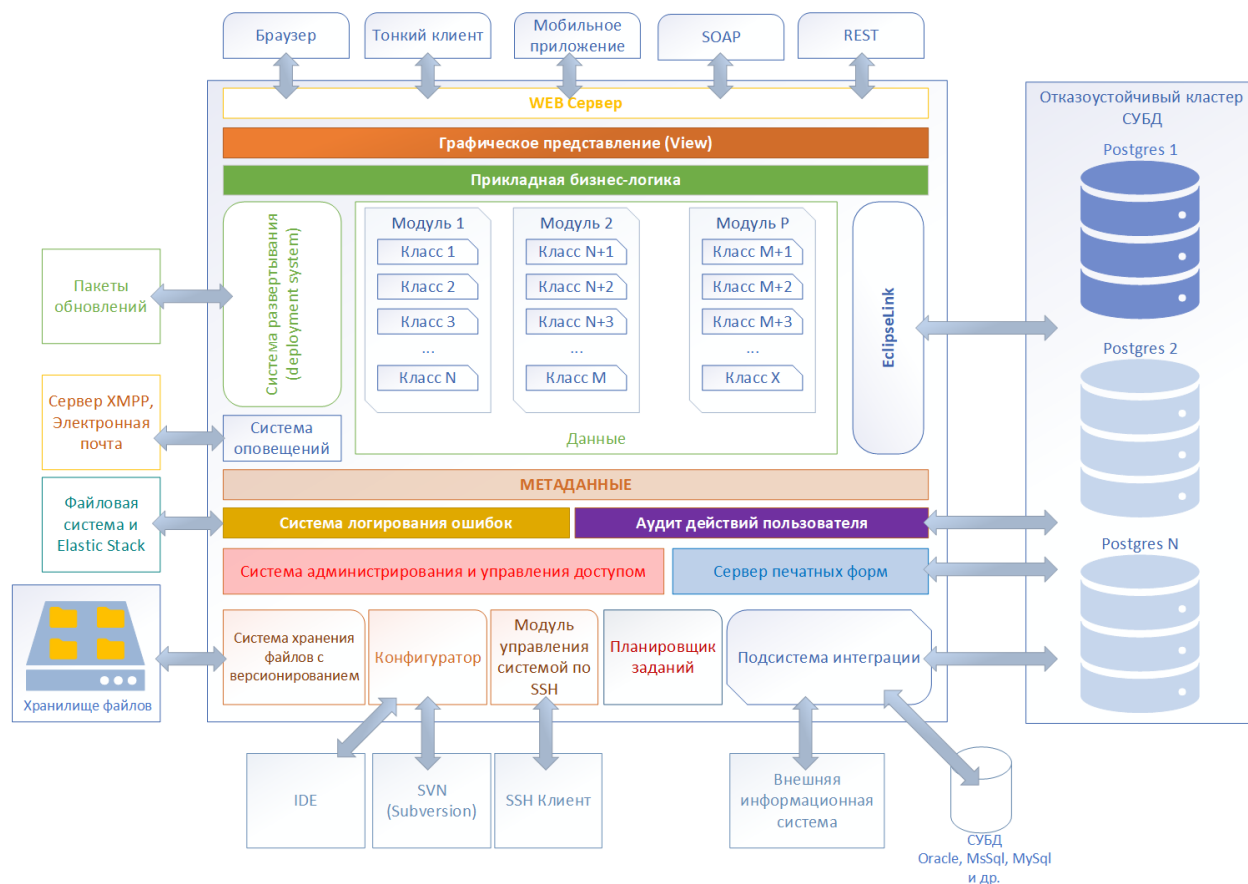
Цель данного урока:

- Познакомить слушателей с архитектурой фреймворка и принципами разработки в нем
- Познакомить слушателей с базой данных `Postgresql`.
- Дать понять какие знания и навыки по работе с языком `sql` потребуются для дальнейшей работы
- Ознакомить со структурой документации `Postgresql`
- Дать понять какие знания и навыки по работе с языками `scala` и `jexl` потребуются для дальнейшей работы
- Ознакомить со структурой документации `scala` и `jexl`

Данный урок предоставляет:

- Обзор архитектуры
- Основные понятия фреймворка
- Основные понятия по работе с базой данных
- Упражнения по языку `sql`
- Упражнения по `scala`
- Упражнения по `jexl`
- Литературу для самостоятельного изучения

2.1 Архитектура системы



Для изучения смотри:

- *Руководство разработчика: Введение*

Общий принцип работы

Пользователь подключается с помощью браузера к серверу приложения в режиме терминального доступа. Сервер приложения возвращает web страницу с рабочим столом на котором пользователь может открывать экранные формы и просматривать\загружать отчеты.

Протокол терминального доступа позволяет работать через тонкий канал за счет того что пользователю передается только видимая часть данных.

Для предоставления пользователю требуемой бизнес логики, сервер приложения запускает решение, которое состоит из набора модулей. Где модуль это неделимая часть решения. Каждый модуль содержит автономную(классы) и интерактивную(выборки) бизнес логику.

2.2 База данных

О базе данных

Для работы система Global использует базу данных PostgreSQL. PostgreSQL - это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), с наиболее богатым sql из открытых СУБД и являющаяся реальной альтернативой коммерческим базам данных. PostgreSQL произносится как *post-gress-Q-L* (сокр. *Postgres*).

Postgres — это реляционная система управления базами данных (РСУБД). Где данные хранятся в виде строк в таблицах, с возможностью ссылаться на другие таблицы.

Все строки таблицы имеют одинаковый набор именованных столбцов, при этом каждому столбцу назначается определённый тип данных. Хотя порядок столбцов во всех строках фиксирован, важно помнить, что SQL не гарантирует какой-либо порядок строк в таблице (хотя их можно явно отсортировать при выводе).

Все изменения данных в *Postgres* проходит по средствам транзакций (логически операция, которая может быть совершена только полностью).

Надежность

Надежность PostgreSQL обеспечивается следующими возможностями:

- **ACID** - Atomicity, Consistency, Isolation, Durability
 - *Atomicity* - Атомарность
Транзакция рассматривается как единая логическая единица, все ее изменения или сохраняются целиком, или полностью откатываются
 - *Consistency* - Согласованность
Транзакция переводит базу данных из одного непротиворечивого состояния (на момент старта транзакции) в другое непротиворечивое состояние (на момент завершения транзакции). Непротиворечивым считается состояние базы, когда выполняются все ограничения физической и логической целостности базы данных, при этом допускается нарушение ограничений целостности в течение транзакции, но на момент завершения все ограничения целостности, как физические, так и логические, должны быть соблюдены.
 - *Isolation* - Изоляция
Изменения данных при конкурентных транзакциях изолированы друг от друга на основе системы версионности
 - *Durability* - Устойчивость
PostgreSQL заботится о том, что результаты успешных транзакций гарантировано сохраняются на жесткий диск вне зависимости от сбоев аппаратуры.
- **MVCC** - Multiversion Concurrency Control
Один из механизмов СУБД для обеспечения параллельного доступа к базам данных, заключающийся в предоставлении каждому пользователю так называемого «снимка» базы, обладающего тем свойством, что вносимые пользователем изменения невидимы другим пользователям до момента фиксации транзакции. Этот способ управления позволяет добиться того, что пишущие транзакции не блокируют читающих, и читающие транзакции не блокируют пишущих.
- **WAL** - Write Ahead Logging
Общепринятый механизм протоколирования всех транзакций, что позволяет восстановить систему после возможных сбоев. Основная идея WAL состоит в том, что все изменения должны записываться в файлы на диск только после того, как эти записи журнала, описывающие эти

изменения будут и гарантировано записаны на диск. Это позволяет не сбрасывать страницы данных на диск после фиксации каждой транзакции, так как мы знаем и уверены, что сможем всегда восстановить базу данных используя журнал транзакций.

- **PITR - Point in Time Recovery**
Предоставляет возможность восстановления базы данных (используя WAL) на любой момент в прошлом, что позволяет осуществлять непрерывное резервное копирование кластера PostgreSQL. Репликация также повышает надежность PostgreSQL.
- **Integrity - Целостность**
PostgreSQL поддерживает целостность данных на уровне схемы - это внешние ключи (foreign keys), ограничения (constraints).

Производительность

Postgres обеспечивает высокую высокую масштабируемость за счет следующих возможностей:

- **Поддержка индексов**
Индексы — это традиционное средство увеличения производительности БД. Используя индекс, сервер баз данных может находить и извлекать нужные строки гораздо быстрее, чем без него. Однако с индексами связана дополнительная нагрузка на СУБД в целом, поэтому применять их следует обдуманно. Postgres поддерживает:
 - Стандартные индексы
 - Частичные индексы
 - Функциональные индексы
- **Планировщик запросов**
Планирует выполнение запроса основывается на стоимости различных планов, учитывая множество факторов таким образом чтобы минимизировать используемые ресурсы сервера.
- **Система блокировок**
Блокировка поддерживается на уровне таблиц и записей. Блокировка оптимизирована под конкретную ОС и архитектуру.
- **Кэширование**
Управление буферами и кэширование используют развитые алгоритмы для поддержания эффективности использования выделенных ресурсов.
- **Табличные пространства**
Tablespaces позволяют гибкое использование дискового пространства для хранения объектов системы.

2.3 Классы

Определяет правила хранения и обработки таблицы базы данных.

Для изучения смотри:

- *Руководство разработчика:Класс*

2.4 Выборки

Определяет правило получения, отображение данных и обеспечивает взаимодействие с пользователем
Для изучения смотри:

- *Руководство разработчика:Выборка*

2.5 Модуль

Модуль это неделимая группа файлов которая может включаться в произвольный прикладной проект.

Бизнес логика разрабатываемая с помощью фреймворка компонуется между прикладными проектами с помощью модулей.

Где:

- **Прикладной проект**
Набор модулей собираемых в конечное решение. Прикладной проект компонует бизнес логику модулей и связывает их между собой посредством инъекций зависимости(плагины, проектные переопределения).
- **Решение**
Набор исполняемых файлов и ресурсов разворачиваемый на сервере приложения в рамках одной схемы базы данных.

Для изучения смотри:

- *Руководство разработчика: Введение # Прикладной проект*

2.6 Dbeaver

Dbeaver - свободный кроссплатформенный инструмент разработки базы данных.

Используется для написания и отладки комплексных sql запросов.

2.7 SQL

Декларативный язык запросов, применяемый для создания, модификации и управления данными. В отличие от процедурных языков, в которых есть условия, циклы и функции, в sql таких конструкций почти нет. Декларативные выражения представляют собой описание того, что необходимо получить. По данному описанию планировщик Postgres строит алгоритмы для получения результата.

Для изучения языка смотрите:

- *Учебное пособие*
- *Русскоязычная документация от PostgresPro*

2.8 PL/pgSQL

PL/pgSQL это процедурное расширение, наследует все пользовательские типы, функции и операторы SQL.

Язык позволяет:

- создавать функции и триггеры
- использовать управляющие структуры
- выполнять сложные вычисления

Функции PL/pgSQL могут использоваться везде, где допустимы встроенные функции. Например, можно создать функции со сложными вычислениями и условной логикой, а затем использовать их при определении операторов или в индексных выражениях.

Для изучения языка смотрите:

- Учебное пособие
- Официальная документация

2.9 IntelliJ Idea

Интегрированная среда разработки для написания исходного кода модулей.

Для изучения смотри:

- Руководство по IntelliJ Idea
- *Руководство разработчика: Работа в IntelliJ IDEA*

2.10 Конфигуратор

Системное приложение Конфигуратор, развернутое в сервере приложения, предназначено для автоматизации и ускорению выполнения рутинных задач по разработке проекта.

Для изучения смотри:

- *Руководство разработчика: Конфигуратор*

2.11 Язык scala

Язык разработки используемый для программирования бизнес логики сервера приложения.

Для изучения смотри:

- Основы языка
- Структуры управления
- Гайд по коллекциям
- Показатели производительности коллекций
- *Руководство разработчика: Языки разработки # Scala*
- Библиотека юнит тестирования в скала

2.12 Язык jexl

Java Expression Language - Язык выражений java. Используется для выполнения динамической бизнес логики. Скрипты на jexl не требуют перекомпиляции прикладного проекта, поэтому могут исполняться в сервере приложений в любое время.

Данный язык широко используется в скриптах миграции, а так же в событиях рабочих процессов настраиваемых на проекте.

Для изучения смотри:

- Обзор
- Синтаксис
- *Руководство разработчика: Языки разработки # Jexl скрипты*

2.13 Настройка рабочего места

Подготовка БД для урока

1. Установите и настройте *postgres*
2. Установите и настройте *Dbeaver*

Примечание

Можно пропустить этот шаг если вы используете курс с опцией **virtual-vm** и образ *start*

Установка postgres

Для установки postgres:

1. Запустите установщик postgres

Примечание

В случае, если вы используете курс с опцией **virtual-vm** дистрибутив находится в каталоги **c:\distr**

2. Следуйте указаниям мастера
При установке достаточно выбирать опции по умолчанию.

Внимание

Имя служебного пользователя должно быть **postgres**

Для загрузки схемы для обучения:

1. Положите файл с учебной схемой в **c:\distr**

Примечание

В случае, если вы используете курс с опцией `virtual-vm` данный шаг можно пропустить

2. Откройте командный процессор `cmd`
3. Перейдите в каталог исполняемых файлов postgres
В случае, установки по умолчанию 14 версии:

```
cd "C:\Program Files\PostgreSQL\14\bin"
```

4. Запустите скрипт установки

```
psql -U postgres -f c:\distr\clubdata.sql -d postgres -x -q
```

Данный скрипт создаст базу данных `exercises`

Установка и настройка Dbeaver

Для установки:

1. Запустите установщик `Dbeaver`

Совет

В случае, если вы используете курс с опцией `virtual-vm` дистрибутив находится в каталоги `c:\distr`

2. Следуйте указаниям мастера
При установке достаточно выбирать опции по умолчанию.

Если нет подключения к интернету, настройте драйвер базы данных для автономной работы:

1. Откройте `Dbeaver`
2. Откройте **Управление драйверами**
Пункт меню `Database > Driver Manager`
3. Найдите драйвер `PostgreSQL`
4. Откройте карточку редактирования
5. Перейдите на закладку `Libraries`
6. Удалите все ссылки на библиотеки для загрузки
7. Добавьте `jar` файл с драйвером

Примечание

В случае, если вы используете курс с опцией `virtual-vm` файл находится в каталоги `c:\distr`

Для создания нового подключения к основной базе:

1. Создайте новое подключение
Пункт меню `Database > New connection`

2. Выберите драйвер PostgreSQL
Если вы устанавливали `postgres` с настройками по умолчанию, созданное соединение будет работать
3. Протестируйте соединение
Нажмите кнопку `Test connection` в мастере соединения

Для создания нового подключения к базе `exercises`:

1. Создайте новое подключение
Пункт меню `Database > New connection`
2. Выберите драйвер PostgreSQL
Если вы устанавливали `postgres` с настройками по умолчанию, созданное соединение будет работать
3. Задайте базу подключения `exercises`
4. Протестируйте соединение
Нажмите кнопку `Test connection` в мастере соединения

Стандартная настройка

Для стандартной настройки рабочего места воспользуйтесь инструкцией *Руководство разработчика: Начало работы с фреймворком # Настройка рабочего места*

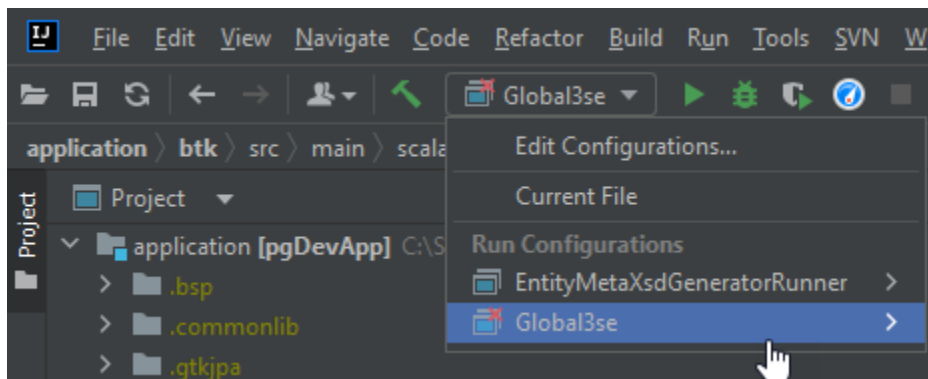
Совет

С инструкцией по установке *Global3FrameworkStarterKit* можно ознакомиться в приложении данного урока

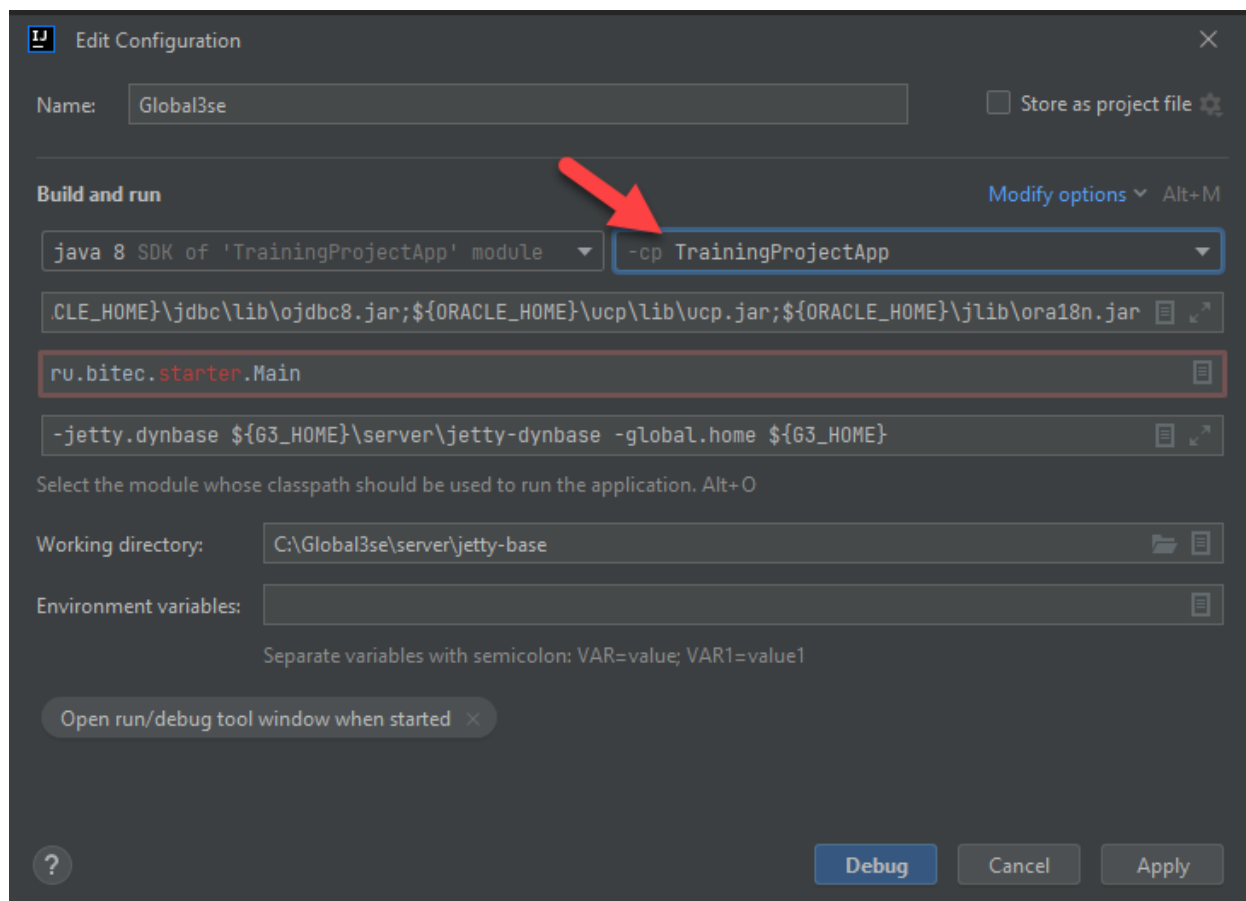
2.14 Запуск сервера приложения в режиме отладки

После настройки рабочего места и сборки проекта можно запустить локальный сервер приложений в режиме отладки.

Выберите в списке стартовых конфигураций `Global3se` и нажмите на кнопку `Debug` (`Shift+F9`).



При первом запуске IDEA попросит уточнить настройки отладочной конфигурации. В поле `-cp` (ClassPath) необходимо указать модуль с наименьшим количеством зависимостей, обычно выбирают модуль проекта с окончанием `App`.



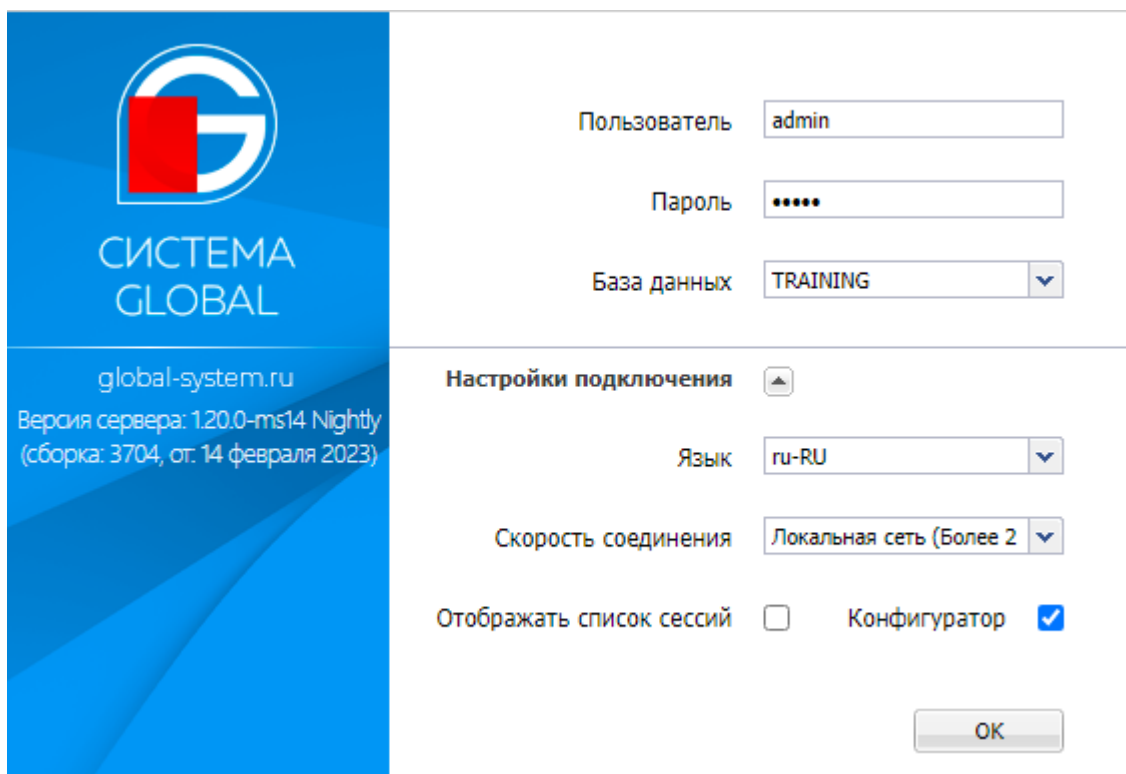
Подробнее об отладке можно узнать по [ссылке](#).

2.15 Запуск конфигуратора

Внимание

Раздел находится в разработке.

1. Откройте в браузере <http://localhost:8080/>
2. Раскройте Настройки подключения и установите признак Конфигуратор
3. Для входа используйте логин admin, пароль admin



СИСТЕМА GLOBAL

global-system.ru
Версия сервера: 1.20.0-ms14 Nightly
(сборка: 3704, от 14 февраля 2023)

Пользователь:

Пароль:

База данных:

Настройки подключения

Язык:

Скорость соединения:

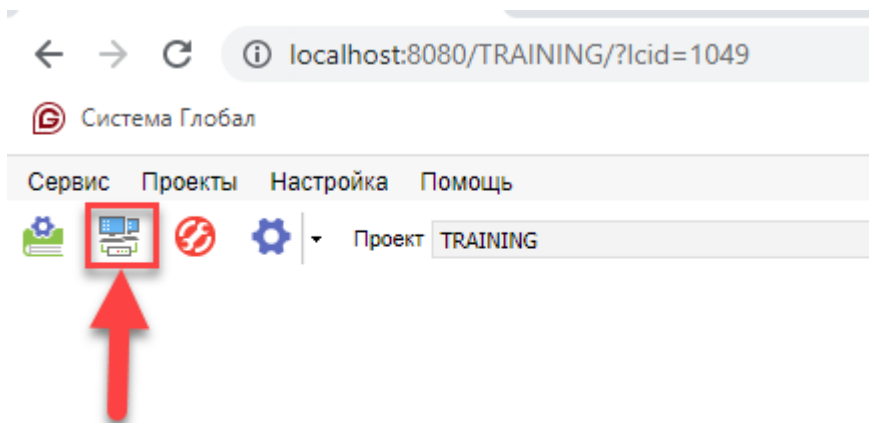
Отображать список сессий: ☐ Конфигуратор: ☒

2.16 Создание модуля

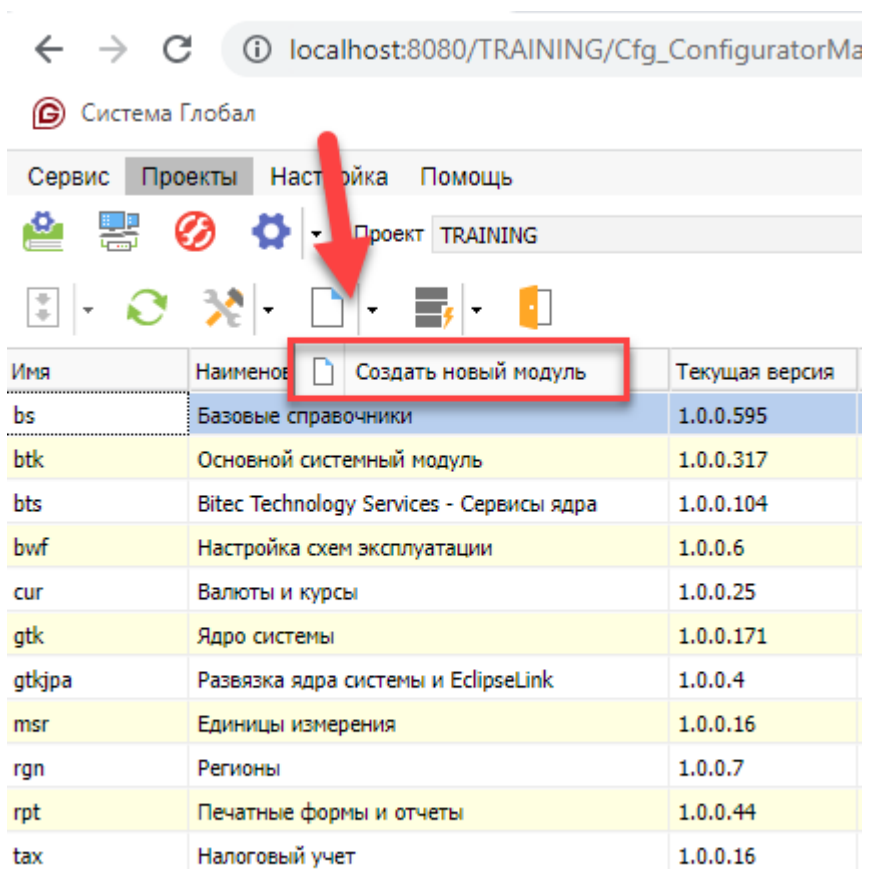
Внимание

Раздел находится в разработке.

1. Откройте конфигуратор
2. Откройте список модулей, выполнив операцию на главной панели приложения.

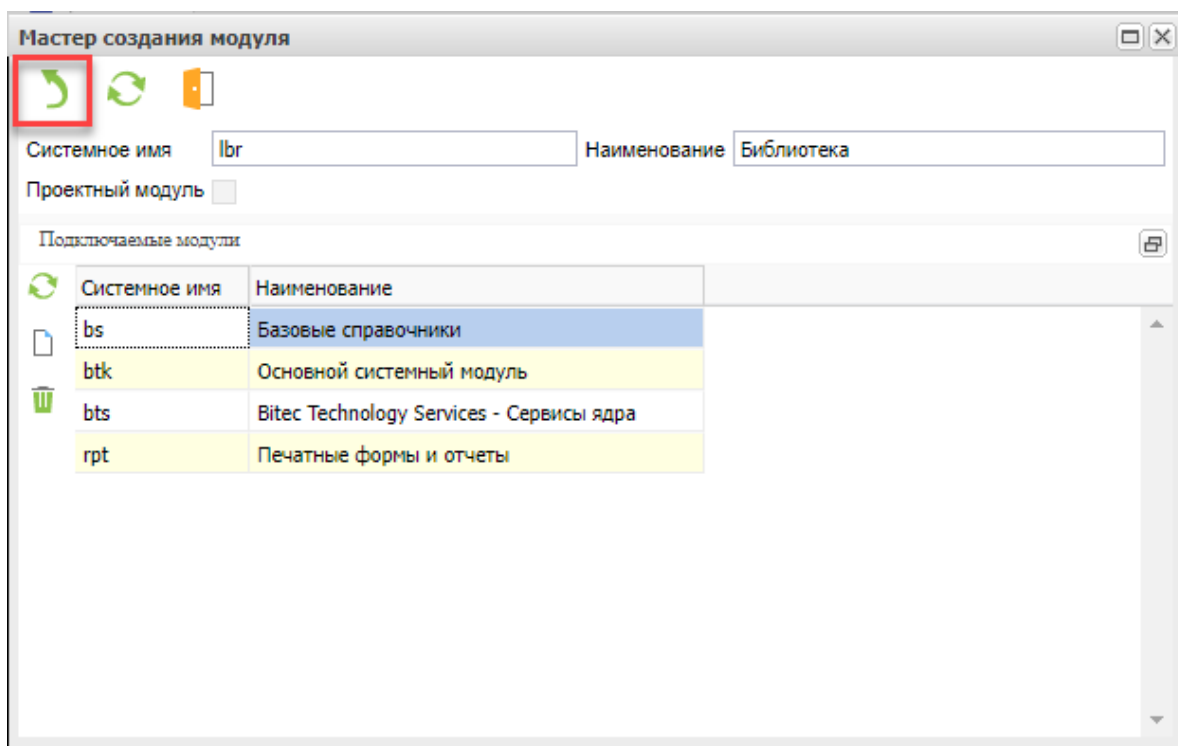


3. В списке модулей выполните операцию Создать новый модуль



4. В мастере укажите системное имя модуля lbr, наименование Библиотека

5. Подключите модули: btk, bs, bts, rpt и завершите работу мастера, выполнив операцию Выбор



2.17 Учебная схема для работы с sql

Схема для практики и упражнений состоит из набора данных для недавно созданного загородного клуба с перечнем участников. Данные содержат:

- Участников клуба
- Теннисные корты(объекты)
- Историю бронирования этих объектов

Внимание

Не воспринимайте набор как пример хорошего дизайна. Этот набор данных разработан исключительно для поддержки упражнений. Схема базы данных имеет ряд недостатков.

Схема:

```
-- Участники
CREATE TABLE cd.members
(
  --Идентификатор
  --memid = 0 служебная запись, для всех гостей
  memid integer NOT NULL,
  --Фамилия
  surname character varying(200) NOT NULL,
  --Имя
  firstname character varying(200) NOT NULL,
```

(продолжается на следующей странице)

```

--Адрес
address character varying(300) NOT NULL,
--Почтовый код
zipcode integer NOT NULL,
--Телефон
telephone character varying(20) NOT NULL,
--memid участника который рекомендовал текущего участника
recommendedby integer,
--Дата вступления
joindate timestamp NOT NULL,
CONSTRAINT members_pk PRIMARY KEY (memid),
CONSTRAINT fk_members_recommendedby FOREIGN KEY (recommendedby)
REFERENCES cd.members(memid) ON DELETE SET NULL
);
-- Объекты
CREATE TABLE cd.facilities
(
--Идентификатор
facid integer NOT NULL,
--Наименование
name character varying(100) NOT NULL,
--Стоимость слота(bookings.slots) для участника
membercost numeric NOT NULL,
--Стоимость слота(bookings.slots) для гостя( memid = 0 )
guestcost numeric NOT NULL,
--Первоначальные затраты
initialoutlay numeric NOT NULL,
--Затраты на ежемесячное обслуживание
monthlymaintenance numeric NOT NULL,
CONSTRAINT facilities_pk PRIMARY KEY (facid)
);
-- История заказов
CREATE TABLE cd.bookings
(
--Идентификатор
bookid integer NOT NULL,
--Ссылка на объекты
facid integer NOT NULL,
--Ссылка на участников
memid integer NOT NULL,
--Время начала использования объекта
starttime timestamp NOT NULL,
--Интервал использования объекта, выражается слотами, 1 слот - полчаса
slots integer NOT NULL,
CONSTRAINT bookings_pk PRIMARY KEY (bookid),
CONSTRAINT fk_bookings_facid FOREIGN KEY (facid) REFERENCES cd.facilities(facid),
CONSTRAINT fk_bookings_memid FOREIGN KEY (memid) REFERENCES cd.members(memid)
);

```


2.18 Практика работы с sql

Просмотр учебной схемы

В навигаторе базы данных(см. пункт меню Windows > Database Navigator):

1. Перейдите к учебной схеме `exercises \ Databases \ exercises \ Schemas \ cd`
2. В контекстном меню выберите `View schema`
3. В открывшемся окне перейдите на закладку `ER diagram`

Выполнение запроса

1. Создайте новый запрос
Для этого выберите пункт меню `Sql Editor > New Sql Script`. При этом откроется окно для ввода sql
2. Установите активное соединение `exercises`
Для этого выберите пункт меню `Sql Editor > Set Active Connection`
3. В окне ввода sql введите

```
select * from cd.facilities;
```
4. Выполните sql
Для этого выберите пункт меню `Sql Editor > Execute Sql Statement`
5. Посмотрите план запроса
Для этого выберите пункт меню `Sql Editor > Explain Execution Plan`

Совет

Понимания плана запроса позволяет принять решение о необходимости дальнейшей оптимизации. Для более подробной информации смотрите [оптимизация производительности](#) в документации PostgreSQL

Результат:

facid	name	membercost	guestcost	initialoutlay	monthlymaintenance
0	Tennis Court 1	5	25	10,000	200
1	Tennis Court 2	5	25	8,000	200
2	Badminton Court	0	15.5	4,000	50
3	Table Tennis	0	5	320	10
4	Massage Room 1	35	80	4,000	3,000
5	Massage Room 2	35	80	4,000	3,000
6	Squash Court	3.5	17.5	5,000	80
7	Snooker Table	0	5	450	15
8	Pool Table	0	5	400	15

2.19 Упражнения по языку sql

Простое соединение таблиц

Напишите запрос получения списка времени начала использования объекта для участников с именем и фамилией „David Farrell“

Совет

Смотрите [соединённые таблицы](#) в документации PostgreSQL

Результат:

starttime

2012-09-18 09:00:00.000
2012-09-18 13:30:00.000
2012-09-18 17:30:00.000
2012-09-18 20:00:00.000
2012-09-19 09:30:00.000
2012-09-19 12:00:00.000
2012-09-19 15:00:00.000
2012-09-20 11:30:00.000
2012-09-20 14:00:00.000
2012-09-20 15:30:00.000
2012-09-21 10:30:00.000
2012-09-21 14:00:00.000
2012-09-22 08:30:00.000
2012-09-22 17:00:00.000
2012-09-23 08:30:00.000
2012-09-23 17:30:00.000
2012-09-23 19:00:00.000
2012-09-24 08:00:00.000
2012-09-24 12:30:00.000
2012-09-24 16:30:00.000
2012-09-25 15:30:00.000
2012-09-25 17:00:00.000
2012-09-26 13:00:00.000
2012-09-26 17:00:00.000
2012-09-27 08:00:00.000
2012-09-28 09:30:00.000
2012-09-28 11:30:00.000
2012-09-28 13:00:00.000
2012-09-29 10:30:00.000
2012-09-29 13:30:00.000
2012-09-29 14:30:00.000
2012-09-29 16:00:00.000
2012-09-29 17:30:00.000
2012-09-30 14:30:00.000

Подзапросы

Создайте запрос, возвращающий список заказов на день 14.09.2012 которые будет стоить участнику (или гостю) более 30 у.е. Добавьте в вывод:

- Имя объекта
- Имя, фамилия участника
Информация должна быть выведена одной колонкой
- Стоимость

Отсортируйте вывод по убыванию. Используйте подзапрос для выбора данных и запрос для сортировки, это позволит уменьшить текстовое дублирование.

Примечание

Гость имеет отличную от участника стоимость (Стоимость учитывается на 1 слот, который равняется полчаса). Идентификатор гостевого пользователя всегда 0.

Совет

Смотрите:

- [Подзапросы](#)
- [Функции](#) в документации PostgreSQL.

Для расчета стоимости используйте атрибуты:

- `bookings.slots`
- `facilities.guestcost`
- `facilities.membercost`

Результат:

member	facility	cost
-----	-----	----
GUEST GUEST	Massage Room 2	320
GUEST GUEST	Massage Room 1	160
GUEST GUEST	Massage Room 1	160
GUEST GUEST	Massage Room 1	160
GUEST GUEST	Tennis Court 2	150
Jemima Farrell	Massage Room 1	140
GUEST GUEST	Tennis Court 1	75
GUEST GUEST	Tennis Court 2	75
GUEST GUEST	Tennis Court 1	75
Matthew Genting	Massage Room 1	70
Florence Bader	Massage Room 2	70
GUEST GUEST	Squash Court	70
Jemima Farrell	Massage Room 1	70
Ponder Stibbons	Massage Room 1	70
Burton Tracy	Massage Room 1	70
Jack Smith	Massage Room 1	70

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

GUEST GUEST	Squash Court	35	
GUEST GUEST	Squash Court	35	

Агрегация

Создайте запрос возвращающий список количества слотов заказанных каждый месяц в 2012. Колонки списка:

- Идентификатор объекта
- Номер месяца
- Количество слотов

Отсортируйте результат по идентификатору и номеру месяца.

Совет

Смотрите:

- Группировку строк
- Функции для работы с датами в документации PostgreSQL.

Результат:

facid	month	Total Slots	
-----	-----	-----	
0	7	270	
0	8	459	
0	9	591	
1	7	207	
1	8	483	
1	9	588	
2	7	180	
2	8	459	
2	9	570	
3	7	104	
3	8	304	
3	9	422	
4	7	264	
4	8	492	
4	9	648	
5	7	24	
5	8	82	
5	9	122	
6	7	164	
6	8	400	
6	9	540	
7	7	156	
7	8	326	
7	9	426	
8	7	117	

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

8	8	322	
8	9	471	

Оконные функции

Найдите три лучших объекта, приносящих доход

Совет

Смотрите [оконные функции](#) в документации PostgreSQL

Результат:

name	rank
-----	----
Message Room 1	1
Message Room 2	2
Tennis Court 2	3

Рекурсия

Найдите восходящую цепочку рекомендаций для участника(A) с идентификатором 27 : то есть участника(B), который рекомендовал участника A и участника(C) который рекомендовал участника (B) и так далее.

Верните идентификатор участника, имя и фамилию. Порядок по убыванию идентификатора участника.

Совет

Смотрите [рекурсивные запросы](#) в документации PostgreSQL

Результат:

recommender	firstname	surname
-----	-----	-----
20	Matthew	Genting
5	Gerald	Butters
1	Darren	Smith

Json

Напишите запрос возвращающий одну строку (используя агрегацию) содержащую json массив для первых 5 записей по членам клуба. Записи в массиве должны быть отсортированы по идентификатору члена клуба. Указывайте формат даты явным образом (`to_char(joindate, 'DD.MM.YYYY HH24:MI:SS')`), чтобы избежать зависимостей от настроек базы данных.

Совет

Смотрите:

- [Функции по работе с json](#) в документации PostgreSQL.

```
{
  "members": [
    {
      "memid": 0,
      "surname": "GUEST",
      "firstname": "GUEST",
      "address": "GUEST",
      "zipcode": 0,
      "telephone": "(000) 000-0000",
      "recommendedby": null,
      "joindate": "01.07.2012 00:00:00"
    },
    {
      "memid": 1,
      "surname": "Smith",
      "firstname": "Darren",
      "address": "8 Bloomsbury Close, Boston",
      "zipcode": 4321,
      "telephone": "555-555-5555",
      "recommendedby": null,
      "joindate": "02.07.2012 12:02:05"
    },
    {
      "memid": 2,
      "surname": "Smith",
      "firstname": "Tracy",
      "address": "8 Bloomsbury Close, New York",
      "zipcode": 4321,
      "telephone": "555-555-5555",
      "recommendedby": null,
      "joindate": "02.07.2012 12:08:23"
    },
    {
      "memid": 3,
      "surname": "Rownam",
      "firstname": "Tim",
      "address": "23 Highway Way, Boston",
      "zipcode": 23423,
      "telephone": "(844) 693-0723",
      "recommendedby": null,
      "joindate": "03.07.2012 09:32:15"
    },
    {
      "memid": 4,
      "surname": "Joplette",
      "firstname": "Janice",
      "address": "20 Crossing Road, New York",
      "zipcode": 234,
      "telephone": "(833) 942-4710",
      "recommendedby": 1,
      "joindate": "03.07.2012 10:25:05"
    }
  ]
}
```

2.20 Практика работы с scala

Совет

Прочитайте:

- [Главу основы scala](#)
- [Главу структуры управления](#)

Создание класса с тестами

1. Перейдите в окно проекта
2. Выберите целевой модуль
3. Перейдите в папку с исходными кодами
`[module_name]/src/test/scala`

Совет

Создать недостающую папку можно из контекстного меню в idea **New > Directory**

4. Создайте пакет `ru.bitec.app.[module_name]`
5. Создайте тестовый класс

```
class Lesson1Test extends LangFunSuite{
  test("HelloWorld"){
    println("hello world")
  }
}
```

Совет

Запустить тест можно из контекстного меню, для этого:

1. Переведите курсор на декларацию функции
2. В контекстном меню выполните операцию „Debug“ Подробнее смотрите [выполнение тестов в руководстве idea](#).

Для дополнительной информации смотрите также библиотеку unit тестирования: `scalatest`

Примечание

Существуют два специализированных базовых класса для тестовых случаев:

- **LangFunSuite**
Используется для тестов которые не нуждаются в контроллерах бизнес логики.
Данные тесты не могут использовать `Api`, `Pkg`, и не имеют подключения к базе данных по умолчанию.
- **ApiTest**
В данном тексте доступен контекст автономной бизнес логики, тестовые случаи могут использовать `Api`, `Pkg` объекты. При этом запуск теста становится медленней из за необходимости инициализировать контекст.

Объявление null типов

Смотри: *Руководство разработчика: языки разработки # scala типы*

Пример объявления null типов показан в следующей тестовой функции:

```
test("NullTypes"){
  val nvNumber = "1.3".nn
  val nvNullNumber = None.nn
  val nvDate = "21.11.2003 00:00:00".nd
  val nvNullString = None.nn
  val nvEmptyString = "".ns
  assert(nvNullString != nvEmptyString)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
assert((nvNumber + 1.nn) === "2.3".nn)
assert((nvNullNumber + 1.nn).isNull)
assert((nvDate+1.day).asNDate() === "22.11.2003".nd)
}
```

Null типы являются доменным аналогом scala типа опций(`Option`), они используются для более компактной записи выражений.

Пример использования null типов и option:

```
//Null типы
val a1 = 1.nn
val b1 = None.nn
assert(
  (a1+b1).isNull
)
//Option
val a2 = Some(1)
val b2: Option[Int] = None
assert(
  (for(a<-a2;b<-b2) yield a+b).isEmpty
)
```

Работа с коллекциями

Совет

Для более подробной информации по работе с коллекциями смотрите:

- Гайд по коллекциям
- Руководство scala: Типы коллекций
- Выбор последовательности

ArrayBuffer

`ArrayBuffer` используется тогда, когда нужна изменяемая индексированная последовательность общего назначения. Поскольку `ArrayBuffer` индексирован, произвольный доступ к элементам выполняется быстро.

```
val nums = ArrayBuffer(1, 2, 3) // ArrayBuffer(1, 2, 3)
nums += 4                       // ArrayBuffer(1, 2, 3, 4)
nums += List(5, 6)
```

Совет

Если есть возможность(добавление идет только в конец) используйте `ArrayBuffer` вместо `ListBuffer`, так как это экономит оперативную память и процессорное время.

Массив

Массив содержит фиксированный набор элементов, массивы удобно применять в константных последовательностях так как они потребляют минимальное количество оперативной памяти, и обладаю высоким быстродействием при переборе значений.

```
val numbers = Array(1, 2, 3, 4)
```

Совет

Если вы неуверены в том какую коллекцию выбрать используйте массив.

Набор

Наборы не содержат одинаковых элементов.

```
//Эквивалентно Set(1, 2)
val set = Set(1, 1, 2)
```

Кортеж

Кортеж объединяет простые логические элементы коллекции без использования классов.

```
//Создание кортежа
val ingredient = ("Sugar", 25)
//Распаковка кортежа
val (name, quantity) = ingredient

val ingredients = Array(ingredient)
//Распаковка в анонимной функции
ingredients.foreach{case (name, quantity) =>
    println(s"$name:$quantity")
}
//Распаковка из опции
ingredients.headOption match {
    case Some((name3, quantity3))=>
    case _ =>
}
```

Кортежи часто используются при работе с картами для комбинации (key,value)

Совет

Более подробно смотрите [кортежи](#)

Карты

Карты используются для организации структуры позволяющей быстро получить значение по ключу.

```
val map = Map("foo" -> "bar")
```

Опция

Опция представляет собой контейнер, который хранит какое-то значение или не хранит ничего совсем.

Опция часто используется как результат работы метода который может ничего не вернуть к примеру:

```
val numbers = Map("one" -> 1, "two" -> 2)
numbers.get("two") match {
  case Some(v) =>
  case _ => AppException("Ошибка в коде".ns).raise()
}
```

Комбинаторы

Комбинаторы позволяют выполнять операции преобразования и обхода коллекций. Пример работы с коллекциями:

```
//Объявление массива номеров
val numbers = Array(1.nn, 2.nn, 3.nn, 4.nn)
//Преобразования массива номеров в массив строк
val strings = numbers.map(_.toString())
// Объявление карты
val map = Map("steve" -> 100.nn, "bob" -> 101.nn, "joe" -> 201.nn)
// Печать элементов массива
strings.foreach{v =>
  println(v)
}
//Печать элементов карты
//Выражение case ( k,v ) преобразовывает входящий кортеж
//в переменные k,v
map.foreach{case ( k,v )=>
  println(s"$k:$v")
}
//Фильтрация номеров
val someNumbers = numbers.filter(_ >= 3.nn)
//Группировка номеров по 2 элемента в группу
val groupedNumbers = numbers.grouped(2).toArray
//Поиск номера
val num = numbers.find(_ == 2.nn).getOrElse(None.nn)
//Сортировка номеров по убыванию
val sortedNumbers = numbers.sortBy(_.desc)
//Суммирование номеров с лева на право
//Первый аргумент задает начальное значение
val sum = numbers.foldLeft(0.nn)(_+_)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
//Преобразование коллекции в строку через запятую и вывод на экран
println(sortedNumbers.mkString(", "))
```

Json

Загрузка json в модель:

```
"""{
  "values":[
    {"id":1},
    {"id":2},
    {"id":1},
    {"id":3}
  ]
}""" .ns.jsonToValue(classOf[Data])
```

Модель `Data` должна быть объявлена внешними классами:

```
/*
 *Класс, типизирующий какую либо строку(из базы данных, или массива json) с полем id.
 *Может быть несколько полей с разными типами.
 */
class Rows{
  var id: NNumber = _
}
/*
 *Класс предоставляющий модель данных.
 */
class Data{
  val values: Array[Rows] = Array.empty
}
```

Примечание

Модель `Data` необходимо так как коллекции в java не хранят информации о вложенных объектах, соответственно выражение `classOf[Array[Member]]` не может предоставить достаточной информации для типизированной десериализации.

2.21 Упражнения по scala

Создание функции

Создайте функцию прибавляющую 1 час к дате.

Создание класса

Создайте scala класс для хранения данных по членам клуба, назовите атрибуты класса в соответствии с примером на json:

```
{
  "memid":0,
  "surname":"GUEST",
  "firstname":"GUEST",
  "address":"GUEST",
  "zipcode":0,
  "telephone":"(000) 000-0000",
  "recommendedby":null,
  "joindate":"01.07.2012 00:00:00"
}
```

Загрузка классов из json

Загрузите json в модель скала классов:

```
{
  "members": [
    {
      "memid":0,"surname":"GUEST","firstname":"GUEST","address":"GUEST","zipcode":0,
      "telephone":"(000) 000-0000","recommendedby":null,"joindate":"01.07.2012 00:00:00"},
    {
      "memid":1,"surname":"Smith","firstname":"Darren","address":"8 Bloomsbury Close, Boston",
      "zipcode":4321,"telephone":"555-555-5555","recommendedby":null,"joindate":"02.07.2012 12:02:05"},
    {
      "memid":2,"surname":"Smith","firstname":"Tracy","address":"8 Bloomsbury Close, New York",
      "zipcode":4321,"telephone":"555-555-5555","recommendedby":null,"joindate":"02.07.2012 12:08:23"},
    {
      "memid":3,"surname":"Rownam","firstname":"Tim","address":"23 Highway Way, Boston",
      "zipcode":23423,"telephone":"(844) 693-0723","recommendedby":null,"joindate":"03.07.2012 09:32:15"},
    {
      "memid":4,"surname":"Joplette","firstname":"Janice","address":"20 Crossing Road, New York",
      "zipcode":234,"telephone":"(833) 942-4710","recommendedby":1,"joindate":"03.07.2012 10:25:05"}
  ]
}
```

Описание модели данных:

- data
Используется для типизации модели данных
 - members
Типизированный массив участников
 - * Member
Участник

Совет

Для участника вы можете использовать класс созданные в предыдущем задании.

Поиск последнего участника

Напишите выражение возвращающее члена клуба который присоединился к клубу последним.

В качестве входных данных используйте результат полученный в задании *Загрузка классов из json*

Для определение последнего присоединившегося необходимо использовать поле `joindate`

Совет

Смотрите [методы](#) доступные для работы с коллекциями

Сортировка при помощи операции

Напишите scala функцию сравнивающую двух членов клуба по выражению `member1.memid < member2.memid`. Отсортируйте членов клуба с помощью данной функции.

Совет

Смотрите [методы](#) доступные для работы с коллекциями

2.22 Практика работа с jexl

Выполнение произвольного jexl.

```
val jexl = new JexlBuilder().permissions(JexlPermissions.UNRESTRICTED).create()
/**
 * Выполнение jexl выражения
 * @param jexlString строка с jexl выражением
 * @param jexlContext контекст jexl,
 *           где контекст это набор переменных и функций, доступных в выражении.
 * @return результат выполнения jexl
 */
def evaluateJexl(jexlString: String, jexlContext: MapContext): AnyRef = {
  val e = jexl.createExpression(jexlString)
  val o = e.evaluate(jexlContext)
  o
}
test("eval_jexl") {
  //Контекст выполнения содержит переменные доступные jexl выражению
  //в момент выполнения
  val jc = new MapContext()
  //Задаёт переменную a в контексте выполнения
  jc.set("a", 1)
  //Задаёт переменную b в контексте выполнения
  jc.set("b", 2)
  assert(evaluateJexl("a + b", jc) == 3)
}
```

Совет

Для дополнительной информации по синтаксису jexl смотрите [справку по синтаксису jexl](#).

Для понимания типов используемых в выражении смотрите [Арифметику jexl](#)

2.23 Упражнения по jexl

Функция сравнения на jexl

Напишите scala функцию которая используя выражения jexl сравнит двух участников по выражению `member1.memid < member2.memid`

Примечание

Для чтения свойства скала класса в jexl необходимо к имени свойства добавить скобки.

```
m1.memid()
```

Сортировка через jexl

Напишите сортировку списка участников через jexl выражение `member1.memid > member2.memid`

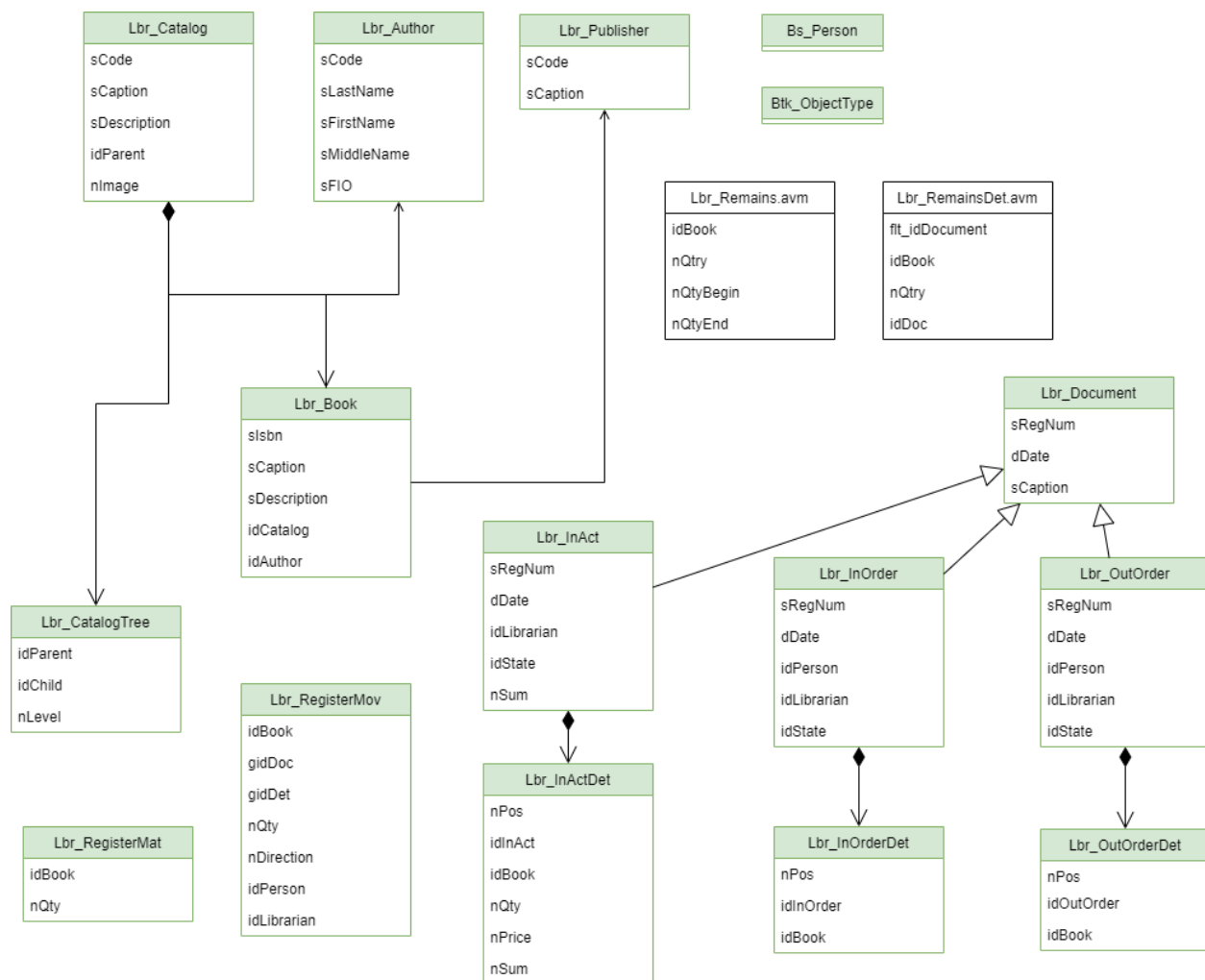
2.24 Самостоятельное изучение

- [Официальная документация postgresQL на русском](#)
- [Перечень полезной документации](#)
- *Руководстве прикладного разработчика : Введение*
- *Руководстве прикладного разработчика : Языки разработки*
- `scala`
- `jexl`

3 Урок 2. Сессии, Модель данных

3.1 Цель курса

Целью курса является познакомить слушателей с возможностями Global Framework. В рамках этого курса будет создано приложение по управлению библиотекой.



3.2 Сессия

Сессия приложения создается на поток прикладной бизнес логики и предоставляет доступ к сессии базы данных, EclipseLink кэшу, серверу приложения. А так же содержит необходимые инъекции зависимости для работы прикладной бизнес логики.

Сессия приложения создается:

- на каждую mdi форму открытую в приложении
- на rest запрос к серверу приложения

Примечание

Для ускорение rest запросов возможна настройка пула сессий

Для изучения смотри: *Руководство разработчика: Сессия-приложения*

3.3 Бизнес объект

Бизнес-объект (БО) - объединение нескольких классов и их коллекций в группу для более удобного манипулирования ими при работе с кэшем и конфигурировании вспомогательных сервисов.

Бизнес объект позволяет:

- Массово загружать данные в транзакционный кэш
Для бизнес объекта можно указать стратегию загрузки данных существенно уменьшающую количество запросов в базу данных. Так как запросы пойдут не по каждому объекту а по каждому классу бизнес объекта.
- Настраивать права доступа
По бизнес объекту создается административный объект на котором можно массово выдать привилегии для всех классов бизнес объекта
- Управлять электронной подписью
Можно настроить правила подписи всего бизнес объекта включая не только шапку но и все вложенные коллекции.
- Настраивать интеграцию и репликацию

Для изучения смотри: *Руководство разработчика: Бизнес-объект*

3.4 Общие сведения о классах

Определяет правила хранения и обработки таблицы базы данных.

Класс позволяет существенно ускорить разработку бизнес логики ориентированную на работу с данными. Программисту достаточно объявить перечень атрибутов класса чтобы за счет кодо-генерации получить набор готовых сервисов.

Перечень генерируемых элементов:

- *Доменная автономная бизнес логика(Dpi)*
Содержит код для автономной бизнес логики
- *Каркас прикладной автономной логики(Api)*
`scala` класс с окончанием `Api`, в котором пишется автономная бизнес логика для работы с классом. Наследуется от `Dpi`
- *Доменная интерактивная бизнес логика(Dvi)*
- *Каркас прикладной интерактивной логики(Avi)*
`scala` класс с окончанием `Avi`, в котором пишется интерактивная бизнес логика. Наследуется от `Dvi`
- *Доменная разметка выборки(dvm.xml)*
Содержит сгенерированную по умолчанию разметку выборки.
- *Каркас прикладной декларации пользовательского интерфейса(Avm)*
`xml` файл с расширением `avm.xml`, в котором пишется разметка выборки
- Интеграция с `Orm`
 - `Rojo` объект для хранения данных в кэше
 - `Argo` объект интеграции `rojo` в фреймворк

Типы данных

В система имеет специализированный набор простых и объектных типов для удобной обработки данных. Объектные типы при необходимости интегрированы в контекст сессии что позволяет обеспечить высокую производительность системы за счет минимизации операций сериализации\десириализации.

Основные типы данных, используемые для атрибутов класса:

- Целое число
- Дробное число
- Строка
- Дата
- Ссылка (на объект заданного класса)
- Ссылка на класс
- Переменная ссылка (на объект произвольного класса)
- Глобальный идентификатор gid
- Json контейнер

Простые типы

Простыми типами являются: Число, строка, дата

Тип	postgresql	odm	Рекомендация по использованию
Строка фиксированной длины	varchar	Varchar	Текст до 4000 символов
Строка переменной длины	text	Text	Текст до 15 мегабайт
Число	number	Number	Целое или дробное число
Дата	date	Date	Дата, дата и время

Ссылочные типы

Данные классов могут быть связаны между собой. Для организации связей между классами используются специальные типы.

Ссылка на класс

Хранит ссылку на класс. Обычно используется совместно с типом **переменная ссылка** для хранения класса, объект которого содержится в переменной ссылке.

Глобальный идентификатор gid

Gid является уникальным идентификатором в рамках системы. Переменная ссылка на gid является альтернативой системе переменной ссылочности из двух атрибутов (ссылка на класс + переменная ссылка на объект). Для организации переменной ссылки через gid используется один атрибут.

Json контейнер

Json контейнер – это расширение объекта класса NoSQL нотацией в реляционной СУБД. Контейнер не имеет жесткой, заранее определенной схемы и основан на множестве пар «ключ-значение». Это позволяет использовать его как динамическое расширение объекта. Для добавления новых данных в контейнер не требуется перекомпиляция кода или изменение структуры СУБД.

Для создания нового класса в модуле необходимо:

1. Создать файл спецификации класса odm. Данный файл удобнее всего создать по шаблону в IntelliJ IDEA (как добавить такой шаблон в ide показано в 1вом уроке).

Внимание

Шаблон по умолчанию содержит коллекцию без имени и включенную группировку. Если в группировке нет необходимости её нужно удалить. Так же необходимо определить или удалить шаблоны для коллекций, в противном случае при генерации кода могут возникнуть ошибки.

2. Определить атрибуты будущего класса и подключить коллекции если они имеются.
3. Сгенерировать код по файлу спецификации
4. Собрать проект
5. Сгенерировать таблицы
6. Добавить orm класса в файл src/main/resources/orm/all.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><!--suppress
↪ JpaConfigDomFacetInspection --><persistence xmlns="http://java.sun.com/xml/ns/
↪ persistence" version="2.0">
<persistence-unit name="all">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <!--<mapping-file>ru/bitec/app/lbr/Lbr_Name.orm.xml</mapping-file>-->
  ...
  <mapping-file>ru/bitec/app/lbr/Lbr_ClassName.orm.xml</mapping-file>
  ...
</persistence-unit>
</persistence>
```

Работа с провайдерами строк

Провайдер строки - Rop используется для работы со строкой данных (Aro), загруженных в рабочее пространство, обеспечивая гарантию того, что при доступе к строке данная строка будет находиться в рабочем пространстве.

Метод получения rop:

```
thisApi().load(идентификатор.asNLong)
```

Примеры сеттеров в файле выборки:

```
val rop = thisRop  
thisApi().setidContras(rop, getVar("super$id").asNLong)
```

Работа с rop в API:

```
for (ropGrade <- new OQuery(entityAta.Type){  
    where (t.idGdsGrade === idpGdsGrade)  
}) {  
    setidGdsGrade(ropGrade, None.nl)  
}
```

Для изучения смотри: *Руководство разработчика: Класс*

3.5 Общие сведения о выборках

Выборка определяет правило получения, отображение данных и обеспечивает взаимодействие с пользователем. Выборки содержат основную часть интерактивной бизнес логики.

Выборка определяет:

- Способ получения данных
- Способ отображения данных пользователю
- Бизнес логику обработки пользовательских действий

Выборка может создаваться от класса с использованием кодо-генерации или вручную.

Пользовательский интерфейс приложения является совокупностью экземпляров отображений выборок.

Для изучения смотри: *Руководство разработчика: Выборка*

3.6 Взаимодействие с базой данных

Объектные запросы

Кроссплатформенные запросы, которые выполняются на уровне объектов класса.

При выполнении запроса идет обращение к базе данных, за исключением случаев кеширования.

Пример запроса

```
new OQuery(Bs_GoodsAta.Type) {  
    where(t.sSystemName === spMnemoCode)  
}
```

Методы

- **where**
Условие запроса.
- **orderBy**
Выражение для сортировки результата
- **batchAll**
Массовая загрузка объектов. Возвращает строки с прогруженными записями всех коллекций этого класса.
- **forUpdate**
Выполнение запроса с блокированием вернувшихся записей
- **forUpdateNowait**
Выполнение запроса с блокированием вернувшихся записей, без ожидания разблокирования, если записи уже заблокированы другой сессией.
- **tryCacheQueryResults**
Попытаться закешировать результат запроса. Смотри пункт «Кеширование»
- **unique**
Говорит, что запрос возвращает одну уникальную запись. Позволяет использовать cache-index'ы, указанные в orm класса

Кеширование объектных запросов

Кеширование запросов работает только для классов с разделяемым режимом кеширования(**Shared**).

Кеширование по полю

Кеширование через cache-index'ы указанные в orm класса. Такой запрос должен возвращать одну строку и дополняется командой **unique()**. Например для атрибутов мнемокода класса в orm формируется запись:

```
<cache-index>  
  <column-name>SSYSTEMNAME</column-name>  
</cache-index>
```

Запрос выглядит следующим образом:

```
new OQuery(entityAta.Type) {  
  unique()  
  where(t.sSystemName*=== spMnemoCode)  
}
```

Кэширование объектных запросов

Кэширование объектных запросов возможно по требованию в случае если класс настроен для сохранения в разделяемом кэше.

Чтобы включить кэширования запроса:

1. Добавьте в запрос опцию `tryCacheQueryResults()`.
Результат такого запроса будет кэширован, если транзакция не находится в режиме редактирования разделяемых объектов.

Пример запроса:

```
new OQuery(entityAta.Type) {  
    tryCacheQueryResults ()  
    where(t.sSystemName === spMnemonicCode)  
}
```

Транзакционный индекс

Позволяет получить перечень строк по значению индексируемого атрибута. Индекс подгружает данные из базы данных по мере обращения к ключам индекса, а так же отслеживает транзакционные изменения для получения согласованного набора строк. Это позволяет получить согласованный доступ к множеству строк по ключу, даже если индексируемое значение строки меняется в рамках этой транзакции.

Пример объявления:

```
lazy val idxidParent = TxIndex(Btk_GroupAta.Type)(_idParentGroup)
```

Методы

- **byKey**
Посетитель по ключу индекса.
- **refreshByKey**
Посетитель по ключу индекса с обновлением из базы данных.
- **queryKeys**
Кэширование ключей индекса.
- **forPartition**
Открывает секцию для массового обновления индекса. Используется для прозрачного массового обновления после очистки транзакционного кэша. Секции могут быть вложенными друг в друга, в таком случае ключи суммируются.

byParent

Метод есть у классов коллекций, возвращает обходчик записей отфильтрованных по идентификатору предка

Реляционные запросы

Для обработки реляционных запросов в основном используется методы на базе `anorm`.

Для более удобного использования в контекст бизнес логики добавлены дополнительные функции.

ASQL

Выполнение запроса на чтение Создаёт объект `AnormSql` для выполнения запроса к базе без модификации данных. Если в текущей сессии начата транзакция, SQL-выражение выполняется в ней. Если транзакция в сессии не начата, запрос выполнится в автономной sql-транзакции.

Пример:

```
val idTrigger = 1
ASQL"SELECT t.sCaption as sHeadline FROM Btk_JobSchedule t WHERE t.id = $idTrigger".
  ↪as(nStr("sHeadline").*).headOption
```

ATSQL

Выполнение запроса с изменением данных или блокировками Создаёт объект `AnormSql` с возможностью модификации данных. Если в текущей сессии начата транзакция, SQL-выражение выполняется в ней. Если в текущей сессии транзакция не начата, она будет начата.

Пример:

```
ATSQL("alter table bs_barcode add id int8;").execute()
```

ASelect

Выполнение запросов на чтение\запись с большим кол-вом колонок.

Пример:

```
for (rv <- new ASelect {
  val nParentLevel = asInt("nParentLevel")
  val gidParent = asString("gidParent")
  val gidChild = asString("gidChild")
  val idParent = asLong("idParent")
  SQL"""
  select nParentLevel,gidParent,gidChild,idParent,idChild from table
  """) {
  println(rv.nParentLevel())
  //запрос поля без его предварительного объявления
  println(rv.get("idChild").asNLong())
}
```

3.7 Практика

3.8 Создание классов для модуля библиотека

Задача: Создать классы:

- Справочник: Lbr_Publisher
- Справочник: Lbr_Author
- Справочник: Lbr_Book

Справочник Lbr_Publisher - Издатель

Системное имя	Наименование	Тип данных атрибута	Тип атрибута	Примечание
gid	gid	Varchar	Basic	
sSystemName	Системное имя	Varchar	Basic	Мнемокод
sCaption	Наименование	Varchar	Basic	Заголовок
sDescription	Описание	Varchar	Basic	

Совет

Создайте класс по адресу `lbr/src/main/resources/ru/bitec/app/lbr/Lbr_Publisher.odm.xml` где `lbr` наименование модуля

Справочник Lbr_Author - Автор

В классе включить группировку

Системное имя	Наименование	Тип данных атрибута	Тип атрибута	Обязательный	Примечание
gid	gid	Varchar	Basic		
sCode	Код	Varchar	Basic	V	Мнемокод
sLastName	Фамилия	Varchar	Basic	V	
sFirstName	Имя	Varchar	Basic	V	
sMiddleName	Отчество	Varchar	Basic	V	
sFIO	ФИО	Varchar	Basic		Заголовок

Совет

Создайте класс по адресу `lbr/src/main/resources/ru/bitec/app/lbr/Lbr_Author.odm.xml` где `lbr` наименование модуля

Справочник Lbr_Book - Книга

Системное имя	Наименование	Тип данных атрибута	Тип атрибута	Примечание
gid	gid	Varchar	Basic	
sISBN	ISBN	Varchar	Basic	Мнемокод
sCaption	Наименование	Varchar	Basic	Заголовок
idPublisher	Издатель	Long	refObject	Ссылка на класс Lbr_Publisher
idAuthor	Автор	Long	refObject	Ссылка на класс Lbr_Author
nYear	Год издания	Number	basic	
nColPage	Кол-во страниц	Number	basic	
sDesc	Описание	Varchar	basic	

Совет

Создайте класс по адресу `lbr/src/main/resources/ru/bitec/app/lbr/Lbr_Book.odm.xml` где `lbr` наименование модуля

3.9 Создание выборки основного меню приложения

Создайте основную выборку приложения Библиотека

1. Создайте файл `src/main/resources/ru/bitec/app/lbr/Lbr_MainMenu.avm.xml`

```
<?xml version="1.0"?>
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.global-system.ru/xsd/global3-view-1.0.xsd"
      name="Lbr_MainMenu">

  <acObject caption="Управление библиотекой"/>

  <representation name="Default" caption="Управление библиотекой" editMode=
    ↪"edit" >

    <layout>
      <simpleComposer>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        <frame caption="Управление библиотекой">
            <grid/>
        </frame>
    </simpleComposer>
</layout>

<attributes>
    <attr name="ID" caption="Идентификатор" isVisible="false"/>
    <attr name="IDACCESSGROUP#" caption="Группа доступа" isVisible="false
→"/>
    <attr name="dGlobalCurrentDate" caption="Дата" editorType="datePick"
→order="40"/>
    <attr name="idGlobalDepOwner" caption="Организация" isVisible="false"
→ order="40"/>
    <attr name="idGlobalDepOwnerHL" caption="Организация" order="40.2">
        <editor>
            <lookup lookupQuery="gtk-ru.bitec.app.bs.Bs_DepOwnerAvi#Lookup"
                changeableAttr="idGlobalDepOwner" isLookupLazyLoad=
→"true"
                lookupKeyAttr="id" lookupListAttr="sHeadLine"
→isResetButtonVisible="true"/>
        </editor>
        <ref class="Bs_DepOwner"/>
    </attr>
    <attr name="dGlobalBeginDate" caption="Период с" editorType="datePick
→" order="50" isLastInLine="true"/>
    <attr name="dGlobalEndDate" caption="по" editorType="datePick" order=
→"60" isLastInLine="true"/>
    <attr name="nChoosePeriodBtn" caption="Выбрать период" editorType=
→"button" order="65"/>
</attributes>

<operations>

</operations>
</representation>
</view>
```

2. Создайте файл `src/main/scala/ru/bitec/app/lbr/Lbr_MainMenuAvi.scala`

```
package ru.bitec.app.lbr

import ru.bitec.app.bs._
import ru.bitec.app.btk._
import ru.bitec.app.gtk.gl.avi.Visibilities
import ru.bitec.gtk.core.AvmFile

@AvmFile(name = "Lbr_MainMenu.avm.xml")
object Lbr_MainMenuAvi extends Bs_ApplicationAvi {
    override def default(): Default = {
        new Default {
```

(продолжается на следующей странице)

```

    override def meta = this
  }
}

trait Default extends super.Default {
  override def onLoadMeta(): Unit = {
    super.onLoadMeta()
  }

  //===== Справочники =====

  @Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
  ↳ Visibilities.Invisible, imageCollection="Btk_Application", imageIndex=6,
    visibleOnMainMenu = Visibilities.Visible, caption = "Справочники", order = 10)
  def mm_ReferenceRoot(): Unit = {}

  //-----Подразделения-----

  @Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
  ↳ Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Подразделения", order = 10,
  ↳ headOperation = "mm_ReferenceRoot")
  def mm_DepartmentRoot(): Unit = {}

  @Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
  ↳ Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Подразделения", order = 5,
  ↳ headOperation = "mm_DepartmentRoot")
  def mm_Department(): Unit = {
    Bs_DepartmentAvi.tree().newForm().open()
  }

  @Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
  ↳ Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Организации", order = 10,
  ↳ headOperation = "mm_DepartmentRoot")
  def mm_Bs_DepOwner(): Unit = {
    Bs_DepOwnerAvi.list().newForm().open()
  }

  @Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
  ↳ Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Виды подразделений", order
  ↳ = 20, headOperation = "mm_DepartmentRoot")
  def mm_Bs_DepartmentSort(): Unit = {
    Bs_DepartmentSortAvi.list().newForm().open()
  }

  @Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
  ↳ Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "-", order = 20,
  ↳ headOperation = "mm_ReferenceRoot")

```

```

def mm_Delimiter1(): Unit = {}

// Контрагенты
@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Контрагенты", order = 30,
↪headOperation = "mm_ReferenceRoot")
def mm_Bs_ContrasRoot(): Unit = {}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Контрагенты", order = 10,
↪headOperation = "mm_Bs_ContrasRoot")
def mm_Bs_Contras(): Unit = {
    Bs_ContrasAvi.mainList().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Типы контрагентов", order
↪= 20, headOperation = "mm_Bs_ContrasRoot")
def mm_Bs_ContrasType(): Unit = {
    Btk_ObjectTypeAvi.list().newForm().params(Map(
        "FLT_SREFCLASSNAME" -> "Bs_Contras",
        "FLT_SREFCLASSNAME.readOnly" -> true
    )).open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "-", order = 30,
↪headOperation = "mm_ReferenceRoot")
def mm_Delimiter11(): Unit = {}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Физические лица", order =
↪50, headOperation = "mm_ReferenceRoot")
def mm_Bs_Person(): Unit = {
    Bs_PersonAvi.list().newForm().open()
}

//-----
@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "-", order = 60,
↪headOperation = "mm_ReferenceRoot")
def mm_Delimiter12(): Unit = {}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Авторы", order = 80,

```

```

↪headOperation = "mm_ReferenceRoot")
def mm_Lbr_Autor(): Unit = {
    Lbr_AuthorAvi.list().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Издательства", order = 90,
↪headOperation = "mm_ReferenceRoot")
def mm_Lbr_Publisher(): Unit = {
    Lbr_PublisherAvi.list().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Книги", order = 110,
↪headOperation = "mm_ReferenceRoot")
def mm_Lbr_Book(): Unit = {
    Lbr_BookAvi.list().newForm().open()
}

//===== Документы =====

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible, imageCollection="Btk_Application", imageIndex=7,
    visibleOnMainMenu = Visibilities.Visible, caption = "Документы", order = 20)
def mm_DocumentRoot(): Unit = {}

//===== Отчеты =====

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible, imageCollection="Btk_Application", imageIndex=8,
    visibleOnMainMenu = Visibilities.Visible, caption = "Отчеты", order = 50)
def mm_LbrReportRoot(): Unit = {}

//===== Настройки =====

```

```

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible, imageCollection="Btk_Application", imageIndex=9,
    visibleOnMainMenu = Visibilities.Visible, caption = "Настройки", order = 60)
def mm_SettingRoot(): Unit = {}

//===== Настройки документации =====
@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Настройки документации",
↪order = 10, headOperation = "mm_SettingRoot")
def mm_DocSetting(): Unit = {}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Типы объектов", order = 10,
↪ headOperation = "mm_DocSetting")
def mm_Btk_ObjectType(): Unit = {
    Btk_ObjectTypeAvi.list().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Настройка детализации
↪типов объектов", order = 15, headOperation = "mm_DocSetting")
def mm_Btk_ObjectTypeDet(): Unit = {
    Btk_ObjectTypeDetAvi.list().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Жизненные циклы", order =
↪40, headOperation = "mm_DocSetting")
def mm_Btk_LifeCycle(): Unit = {
    Btk_LifeCycleAvi.list().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Закладки", order = 60,
↪headOperation = "mm_DocSetting")
def mm_Btk_Tab(): Unit = {
    Btk_TabAvi.list().newForm().open()
}

//===== Печатные формы =====
@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Печатные формы", order =
↪90, headOperation = "mm_SettingRoot")
def mm_Rpt_Report(): Unit = {
    val vRpt_ReportAvi = Btk_Lib().getAviBySimpleName("Rpt_ReportAvi")

```

```

    if (vRpt_ReportAvi != null)
        vRpt_ReportAvi.list().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "-", order = 120,
↪headOperation = "mm_SettingRoot")
def mm_Delimiter60(): Unit = {}

//===== Дополнительно =====
@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Дополнительно", order =
↪130, headOperation = "mm_SettingRoot")
def mm_AdditionalSettings(): Unit = {}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Группировка объектов",
↪order = 130, headOperation = "mm_AdditionalSettings")
def mm_Btk_Group(): Unit = {
    Btk_GroupAvi.list().newForm().open()
}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Состояния классов", order
↪= 150, headOperation = "mm_AdditionalSettings")
def mm_Btk_ClassList(): Unit = {
    Btk_ClassListAvi.list().newForm().open()
}

//===== Менеджер заданий =====
@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "-", order = 140,
↪headOperation = "mm_SettingRoot")
def mm_Delimiter140(): Unit = {}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Менеджер заданий", order =
↪150, headOperation = "mm_SettingRoot")
def mm_Btk_Job(): Unit = {
    Btk_JobGroupAvi.tree().newForm().open()
    //Btk_JobAvi.tree().newForm().open()
}

//===== Тестирование и отладка =====
@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,

```

(продолжение с предыдущей страницы)

```
visibleOnMainMenu = Visibilities.Visible, caption = "-", order = 9998,
↪headOperation = "mm_SettingRoot")
def mm_Delimiter9998(): Unit = {}

@Oper(visibleOnContextMenu = Visibilities.Invisible, visibleOnToolbar =
↪Visibilities.Invisible,
    visibleOnMainMenu = Visibilities.Visible, caption = "Тесты и отладка", order =
↪9999, headOperation = "mm_SettingRoot")
def mm_Testing(): Unit = {}
    } //-----
}
```

3. Создайте файл src/main/resources/META-INF/applications.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<apps>
    <!--<app mainSelection="Lbr_MainMenuAvi" caption="Наименование приложения"
↪imageIndex="1"/>-->
    <app mainSelection="Lbr_MainMenuAvi" caption="Управление библиотекой"
↪imageIndex="6"/>
</apps>
```

4 Урок 3. Интерактивная бизнес логика.

Данный урок знакомит слушателя с принципами построения интерактивной бизнес логики.

4.1 Отображение

Группирует бизнес логику выборки

Для изучения смотри:

- *Руководство разработчика: Отображение*

4.2 Операции

Обработка события, вызванного пользователем или каким-то процессом

Для изучения смотри:

- *Руководство разработчика: Операции*

4.3 Атрибуты

Перечень колонок, попавших в набор данных для отображения в результате операции onRefresh (а также onRefreshExt в случае объектного запроса). Видимость, порядковый номер, отображаемое наименование, тип редактора и т.д. для атрибута настраивается в файле с разметкой (файл с расширением .avm.xml)

4.4 Типы фреймов

Фреймы — это средство представления информации в выборке.

Полный перечень фреймов описан:

- *Руководство разработчика: Основные типы фреймов*

4.5 Средства отладки

Наиболее популярными средствами отладки являются:

- Клиентское окно отладчика.

Для изучения смотри:

Руководство разработчика: Клиентское окно отладчика

- Отладка в среде IDE.

Для изучения смотри:

Руководство разработчика: Отладка сервера в среде IDE

- Отладка в базе данных.

Отладка производится через обращения к БД с получением актуальных значений атрибутов.

Внимание

В БД хранятся только закоммиченные значения, т.е. пока пользователь не сохранил изменения, они не отобразятся в результате запроса

4.6 Практика

Создание древовидного справочника

- Создайте справочник Lbr_Catalog - Каталог с представлением объектов класса в виде дерева и указанными ниже атрибутами:

name	attribute-type	caption	Дополнительно
idParent	Long	Предок	Ссылается на класс Lbr_Catalog. Невидимый атрибут
sCode	Varchar	Код	Обязательный для заполнения, мнемокод
sCaption	Varchar	Наименование	Обязательный для заполнения, хедлайн

Чтобы генератор кода автоматически создал нужные для отображения дерева методы, необходимо в метаданных указать `viewOptions.classType="tree"` и `viewOptions.classParentAttribute="idParent"`

- Добавьте в класс `Lbr_Book` хранимую ссылку на каталог.
- Добавьте в `Lbr_MainMenuAvi` открытие `Lbr_CatalogAvi.defList`.

Добавление вычисляемого поля

Добавьте поле с количеством книг в каталоге.

1. Переопределите `selectStatement` и `onRefreshExt` и добавьте поле с количеством книг в каталоге, используя подзапрос.
2. Добавьте в разметку описание нового атрибута, чтобы он был видимым и выводился после описания.

Добавление закладки

Добавьте нередактируемую закладку к каталогу отображающую список книг без возможности создания и удаления.

1. В `Lbr_BookApi` объявите *транзакционный индекс* по полю `idCatalog` и метод получения книг по каталогу (`refreshByCatalog`), используя метод `refreshByKey` объявленного ранее транзакционного индекса.
2. Для `Lbr_BookAvi` добавьте новое отображение, которое будет выводить список книг для каталога, используя метод из предыдущего пункта.
3. В `avm` добавьте разметку для нового отображения.
4. Для этого отображения скройте операции `insert`, `delete` и `uniFilter`, используя аннотацию `@Oper` в `Avi` или блок `<operations/>` в `avm`. Так же скройте вывод панели фильтрации.
5. Для `Lbr_Catalog` в разметке отображений `Tree` и `Card` поменяйте настройки `layout` с `simpleComposer` на `tabComposer` и добавьте блок `tabItems` с конкретным `tabItem`, который будет выводить новое отображение в качестве закладки.

Источники для самостоятельного изучения

Изучите следующие разделы в Руководстве прикладного разработчика:

- *Выборка*
- *Взаимодействие с базой данных*
- *Отладка приложений*

5 Урок 4. Практика разработки. Часть 1

В данном уроке мы рассмотрим:

1. Миксины
2. Документы
3. Коллекции
4. Автономумерацию на атрибутах класса

5.1 Миксины

Особый вид классов, которые служат для хранения данных из разных классов. Используются для построения общих списочных форм различных диалогов подбора в пользовательских интерфейсах, а также для удобства обработки данных в прикладной бизнес-логике. Миксин позволяет объединить несколько разных таблиц вместе что дает возможность использовать внешние ключи и индексы на данное объединение. Пример: миксин *Bs_Settler* - Сторона расчета, которая объединяет в себе классы *Bs_Contras* - Контрагент и *Bs_Person* - Физические лица. *Руководство разработчика: Класс # Миксин*

5.2 Документы

Документ – это прикладной объект, который хранит данные о событиях или операциях на предприятии. Пример: Заявка на отгрузку; Приходная накладная; Акт сверки. *Руководство разработчика: Класс # Супертипы классов*

5.3 Коллекции

Коллекции представляют собой классы, объекты которых не имеют права на самостоятельное существование и могут быть созданы только для объектов других классов. Коллекции применяются в качестве табличных частей документов или логических развязок между классами. Пример: Позиции накладной; Позиции складского ордера *Руководство разработчика: Класс # Коллекции*

5.4 Автономумерация

Процесс автоматической выдачи значения для атрибута класса. В основе лежит выдача порядкового номера (числа), которое может быть преобразовано в значение по маске. *Руководство разработчика: Сервисы класса # Автономумерация*

5.5 Практические задания

Внимание

Соблюдайте общие принципы наименования. *Руководство разработчика: Языки разработки # Общие принципы наименования*

Добавление автонумерации без разреза

- В классе `Lbr_Author` сделайте автонумерующимся поле `sCode`, без разреза и маской с дополнением до 4 нулей слева (пример 0001, 0002... 0010). Также уберите обязательность для ввода, чтобы не было ошибки при сохранении.

Создание классов

Миксин для документов

- Создайте директорию `document` и в нем класс `Lbr_Document` - Документ с типом `mixin` и атрибутами:

name	attribute-type	caption	Дополнительно
gidRef	Varchar	Идентификатор	Не видимый (<code>isVisible="false"</code>)
sNumDoc	Varchar	№	
dDoc	Date	Дата документа	Редактор даты (<code>editorType="datePick"</code>)
idDepOwner	Long	Организация	Ссылочный на объект, ссылается на класс <code>Bs_DepOwner</code>

- Удалите поля `id`, `idClass`, `gid`, так как в миксине они не нужны (если есть)
- Запустите кодогенерацию по данному классу и запустите генератор таблиц для него
- Добавьте в `Lbr_MainMenuAvi` пункт **Миксин для документов** под **Тесты и отладка**, в котором пропишите открытие умол. списка для `Lbr_DocumentAvi`

Документ выдача книг

- Создайте директорию `outorder` и в нем класс `Lbr_OutOrder` - Выдача книг с типом `document` и атрибутами:

name	attribut type	caption	Дополнительно
sNumI	Varcha	№	Автонумерующийся в разрезе организации и года (от даты), хедлайн (isHeadLine="true"), не копируется при выполнении копирования (isCopyInCopyObject="false")
dDoc	Date	Дата	Хедлайн, значение атрибута по умолчанию - sysdate (defaultValue="sysdate"), не копируется при выполнении копирования, редактор даты
idLibra	Long	Библиотечка	Ссылочный на объект, Ссылается на класс Bs_Person
idPersc	Long	Читатель	Ссылочный на объект, Ссылается на класс Bs_Person
idDepC	Long	Организация	Ссылочный на объект, Ссылается на класс Bs_DepOwner

- Подключите к нему миксин **Lbr_Document**
- В этой же директории создайте класс **Lbr_OutOrderDet** - Позиции с типом **collection** и атрибутами:

name	attribut type	caption	Дополнительно
idDo	Long	Выдача книг	Ссылочный на объект, Ссылается на класс Lbr_OutOrder, не видимый, формирования отображения выборки для вывода данных в качестве коллекции (genListCollectionRep="true")
nRov	Numbe	№ п/п	Не копируется при выполнении копирования
idBo	Long	Книга	Ссылается на класс Lbr_Book

- Укажите данный класс как коллекцию в классе **Lbr_OutOrder**
- Запустите кодогенерацию для **Lbr_OutOrder** и **Lbr_OutOrderDet**, запустите генератор таблиц для обоих классов
- В avm документа для карточки и списка подключите закладку с позициями, в списке по умолчанию закладка не видима
- В avi документа для карточки на операцию сохранения пропишите обновление формы
- Добавьте в **Lbr_MainMenuAvi** пункт **Выдача книг** под **Документы**, в котором пропишите открытие умол. списка для **Lbr_OutOrderAvi**

Примечание

При выборе данных из справочника физ. лиц в фильтре установите флаг **Не является сотрудником**, для того, чтобы отображались все строки.

Документ возврат книг

- Создайте директорию `inorder` и в нем класс `Lbr_InOrder` - Возврат книг с типом `document` и атрибутами:

name	attribute-type	caption	Дополнительно
sNumDo	Varchar	№	Автонумерующийся в разрезе организации и года (от даты), хедлайн, не копируется при выполнении копирования
dDoc	Date	Дата	Хедлайн, значение атрибута по умолчанию - sysdate, не копируется при выполнении копирования, редактор даты
idLibrari	Long	Библио-текарь	Ссылочный на объект, Ссылается на класс Bs_Person
idPerson	Long	Чита-тель	Ссылочный на объект, Ссылается на класс Bs_Person
idDepOw	Long	Органи-зация	Ссылочный на объект, Ссылается на класс Bs_DepOwner

- Подключите к нему миксин `Lbr_Document`
- В этой же директории создайте класс `Lbr_InOrderDet` - Позиции с типом `collection` и атрибутами:

name	attribute-type	caption	Дополнительно
idDo	Long	Воз-врат книг	Ссылочный на объект, Ссылается на класс Lbr_InOrder, не видимый, формирования отображения выборки для вывода данных в качестве коллекции
nRov	Number	№ п/п	Не копируется при выполнении копирования
idBo	Long	Книга	Ссылается на класс Lbr_Book

- Укажите данный класс как коллекцию в классе `Lbr_InOrder`
- Запустите кодогенерацию для `Lbr_InOrder` и `Lbr_InOrderDet`, запустите генератор таблиц для обоих классов
- В `avm` документа для карточки и списка подключите закладку с позициями, в списке по умолчанию закладка не видима
- В `avi` документа для карточки на операцию сохранения пропишите обновление формы
- Добавьте в `Lbr_MainMenuAvi` пункт **Возврат книг** под **Документы**, в котором пропишите открытие умол. списка для `Lbr_InOrderAvi`

Документ приходная накладная

- Создайте директорию `inact` и в нем класс `Lbr_InAct` - Приходная накладная с типом `document` и атрибутами:

name	attribute-type	caption	Дополнительно
sNumDo	Varchar	№	Автономерующийся в разрезе организации и года (от даты), хедлайн, не копируется при выполнении копирования
dDoc	Date	Дата	Хедлайн, значение атрибута по умолчанию - sysdate, не копируется при выполнении копирования, редактор даты
idLibrari	Long	Библиотекарь	Ссылочный на объект, Ссылается на класс Bs_Person
nSum	Number	Сумма	Денежный редактор (<code>editorType="currency"</code>)
idDepOw	Long	Организация	Ссылочный на объект, Ссылается на класс Bs_DepOwner

- Подключите к нему миксин `Lbr_Document`
- В этой же директории создайте класс `Lbr_InActDet` - Состав накладной с типом `collection` и атрибутами:

name	attribute-type	caption	Дополнительно
idInA	Long	Приходная накладная	Ссылается на класс <code>Lbr_InAct</code> , не видимый, формирования отображения выборки для вывода данных в качестве коллекции
nRow	Number	№ п/п	Не копируется при выполнении копирования
idBoc	Long	Книга	Ссылается на класс <code>Lbr_Book</code>
nQty	Number	Кол-во	
nPric	Number	Цена	Денежный редактор
nSum	Number	Сумма	Денежный редактор

- Укажите данный класс как коллекцию в классе `Lbr_InAct`
- Запустите кодогенерацию для `Lbr_InAct` и `Lbr_InActDet`, запустите генератор таблиц для обоих классов
- В `avi` документа для карточки и списка подключите закладку с позициями, в списке по умолчанию закладки сделайте не видимыми
- В `avi` документа для карточки на операцию сохранения пропишите обновление формы
- Добавьте в `Lbr_MainMenuAvi` пункт Приходные накладные под Документы, в котором пропишите открытие умол. списка для `Lbr_InActAvi`

Разработка бизнес логики

Вычисление заголовка

- В Api классов-документов переопределите метод вычисление заголовка `calcHeadLine`, заголовок должен быть в формате `{HL класса} № {sNumDoc} от {dDoc в формате дд.мм.гггг}`.
- Для созданных ранее объектов пересчитайте заголовки, для этого в карточках классов, под молотком и ключом запустите операцию **Пересчитать заголовки и мнемкоды объектов класса**. Для проверки можно посмотреть данные столбца `sheadline_dz` в таблицах.

Установка порядкового номера для позиций

- В Api позиций документов переопределите метод `insertByParent` и напишите установку значения в поле `nRow`, как максимальный текущий номер в позициях + 1. Если позиций еще нет, то первая позиция должна начинаться с 1. Можно воспользоваться `scala` функцией `reduceOption(_ max _).getOrElse(0.nn)`
- В Api позиций документов для отображений `List_idDoc` или `List_idInAct` переопределите метод `onRefresh` и допишите сортировку по полю `nRow`, используя `sortBy`.

Пересчитываемые поля

Добавьте пересчеты для приходной накладной `Lbr_InActDetApi`:

- Переопределите методы установки цены и количества (сеттеры `setnPrice` и `setnQty`), добавьте логику вычисления суммы в позиции с округление до 2х знаков
- Переопределите сеттер суммы, добавьте вычисление цены в позиции с округление до 2х знаков
- Добавьте пересчет общей суммы документа при изменении суммы в позициях или удалении самой позиции
- На сеттеры цены, количества и суммы добавьте проверку на отрицательный ввод, вызов ошибки осуществляется методом `throw AppException("Текст ошибки")`

`Lbr_InActDetApi`:

- Переопределите сеттеры цены, количества и суммы, добавьте в конце обновление текущей записи (`selection.refreshItem()`) и обновление карточки мастера (`selection.master.refreshItem()`)
- Переопределите метод удаления (`delete`) и пропишите в конце обновление карточки мастера

`Lbr_InAct.avm.xml`:

- Сумму документа сделайте не редактируемой в карточке

6 Урок 5. Практика разработки. Часть 2

В данном уроке мы рассмотрим:

1. Типы объектов
2. Состояния классов
3. Динамическое управление редактируемостью полей
4. Действия на перевод состояния

6.1 Типы объектов

Сервис предоставляет набор стандартных возможностей, доступных для всех типизируемых классов. Основным назначением типа объекта является хранение настроек по документу, которые могут быть переопределены на проекте. *Руководство разработчика: Тип объекта*

6.2 Состояния

Состояния класса позволяют управлять бизнес-логикой объекта. *Руководство разработчика: Тип объекта # Переходы состояний*

6.3 Практические задания

Добавление атрибутов типа объекта и состояния

- В классе Lbr_OutOrder, Lbr_InOrder и Lbr_InAct добавьте атрибуты:

name	attribute-type	caption	Дополнительно
idState	Long	Состояние	Ссылочный на объект, Ссылается на класс Btk_ClassState, не копируется при выполнении копирования
idStateM	Number	Номер состояния	Не видимый, не копируется при выполнении копирования, является порядковым номером состояния (isStateMC="true")
idObject'	Long	Тип	Ссылочный на объект, Ссылается на класс Btk_ObjectType, хедлайн

- Запустите кодогенерацию по этим классам и запустите генератор таблиц для них

Добавление состояния для классов

- Откройте карточку класса Lbr_InAct (Настройка системы \ Сущности > Классы) и добавьте состояния:

Системное имя	Наименование	Порядковый номер	Стартовое состояния
Create	Оформляется	100	1
Executed	Выполнен	300	0

- Для классов `Lbr_InOrder` и `Lbr_OutOrder` в `api` сделайте метод `regState` для регистрации аналогичных состояний и пропишите его в `odm` в тэге `dbData`. *Руководство разработчика: Тип объекта # Регистрация состояний*
- Запустите генерацию таблиц для данных классов, чтобы отработал скрипт регистрации состояний. При перезапуске скрипта необходимо увеличивать его версию.

Создание закладки для класса через справочник закладок

- Создайте закладку в справочнике закладок для позиций `Lbr_InActDet`.
 - Системное имя - `Lbr_InActDetAvi.List_idInAct`
 - Наименование - Позиции
 - Имя выборки - `Lbr_InActDetAvi`
 - Имя отображения - `List_idInAct`

В детализации укажите класс `Lbr_InAct`. *Руководство разработчика: Тип объекта # Закладки*

- Для классов `Lbr_InOrder` и `Lbr_OutOrder` в `api` сделайте метод `regTab` для регистрации закладок со своими позициями и пропишите его в `odm`. *Руководство разработчика: Тип объекта # Регистрация закладок*
- Запустите генерацию таблиц для данных классов, чтобы отработал скрипт регистрации.

Данные закладки будут нужны для дальнейшей настройки на типе объекта.

Настройка типов объектов

- Создайте тип объекта с кодом, наименованием и кр. наименованием `Lbr_InAct` - Приходная накладная через справочник типов объекта и установите на нем флаг по умолчанию, настройте для него закладку с позициями и переходы состояний. Доступные закладки для класса будут отображаться, если установить в фильтре флаг **Отображать неактивные**
- Для того, чтобы при создании объекта подставлялся тип по умолчанию, в `Lbr_InActApi` переопределите метод `insert` и пропишите в конце вызов установки типа объекта по умолчанию `setIdObjectType(rop, Btk_ObjectTypeApi().getDefaultObjType(idClass))`
- В `Lbr_InAct.avm` отображении `Card` подключите закладки от типа объекта. *Руководство разработчика: Тип объекта # Детализация виде закладок*
- Там же у атрибута `idObjectTypeHL` измените тип редактора на выпадающий список с `lookupQuery="gtk-Btk_ObjectTypeAvi#MainLookup"`
- В `Lbr_InActApi` создайте метод `regObjectType` для регистрации еще одного типа `Lbr_InAct_Add - Дополнительная приходная накладная`, с параметром `bpIsDefault = 0`. Там же пропишите регистрацию закладок для типа и переходов состояния, пропишите метод в `odm`. Запустите генерацию таблиц по данному классу. *Руководство разработчика: Тип объекта # Регистрация подкласса, типа объекта, закладок для типа и переходов состояний*
- Для классов `Lbr_InOrder` и `Lbr_OutOrder` сделайте аналогичную настройку в `avm` и в `Api()`. `insert`, также в `Api` сделайте метод для регистрации типа по умолчанию с кодом и наименованием из класса. Там же пропишите регистрацию закладок для типа и переходов состояния, пропишите метод в `odm`, с зависимостями от предыдущих скриптов `regTab` и `regState`. Запустите генерацию таблиц по данным классам.

Так как тип объекта и его коллекции является классами с разделяемым режимом кеширования (`Shared`), то необходимо после любых изменений на них, сбрасывать `Shared` кэш

Разработка бизнес логики

Внимание

Не используйте `соруАgo` для получения значений атрибутивного состава `гор`, так как это увеличивает расход оперативной памяти, для получения значений атрибута следует применять `гор.get(_.атрибут)`

Вычисление заголовка с учетом типа объекта

- В `Api` классов-документов измените вычисление заголовка `calcHeadLine`, заголовок должен быть в формате `{(Краткое наименование из типа объекта).nvl(HL класса)} № {sNumDoc}` от `{dDoc}` в формате `дд.мм.гггг`.
- Для созданных ранее объектов пересчитайте заголовки, для этого в карточках классов, под молотком и ключом запустите операцию **Пересчитать заголовки и мнемокоды объектов класса**. Для проверки можно посмотреть данные столбца `sheadline_dz` в таблицах.

Динамическое управление редактируемостью полей

- В `Avi` документов для карточек переопределите сеттер состояния `setidState`, в конце пропишите методы для корректного срабатывания `checkWorkability` мастера и деталей, для того чтобы последующие задания работали корректно при переводе состояния:

```
//не обязательные действия
//завершение транзакции
session.commit()
//снятие блокировки
Btk_FormSessionApi().closeLockUnit()

//необходимые действия
//обновление карточки
selection.refreshItem()
//вызов checkWorkability в карточке
selection.checkWorkability()
//вызов checkWorkability в деталях
selection.cwaDetails()
```

- Для реализации динамического управления редактируемостью в `Avi` позиций документов, для отображений `List_idDoc` и `List_idInAct` переопределите метод `checkWorkability`, напишите блокировку добавления, удаления, редактирования позиций, если номер состояния документа отличен от 100. Можно воспользоваться данными методами:

```
//переменная для управления редактируемостью
val bvR0 = getVar("super$idStateMC").asNNumber.isDistinct(100.nn)
//для одного атрибута
selection.attrs("Имя атрибута").isReadOnly = bvR0
//для списка атрибутов
selection.attrs().foreach(_.isReadOnly = bvR0)
//для операций
selection.opers("Имя операции").isEnabled = !bvR0
```

- В `Api` документов для карточек переопределите `checkWorkability`, напишите блокировку на редактирование всех полей, кроме `idState`, `idStateHL` и `nSum`, если номер состояния документа отличен от 100

Действия на перевод состояния

В `Lbr_InActApi`:

- Создайте метод проверки документа (например `validateDoc`), на вход будет `gor` документа.
- В нем пропишите вызов ошибки, если не будут выполнены какие либо условия (указаны ниже). Ошибку можно сделать одну общую с не выполненными условиями или отдельную по каждому условию. Вызов ошибки осуществляется методом `throw AppException("Текст ошибки")`. Перечень условий:
 1. Заполнены атрибуты документа (дата, организация, библиотекарь)
 2. Наличие хотя бы одной записи в позициях
 3. Заполнены атрибуты в позициях (книга, количество)
- Переопределите сеттер состояния (`setidState`), в начале метода пропишите вызов метода проверки, если номер нового состояния будет ≥ 300 и номер старого состояния < 300 .
- В `Lbr_InOrderApi` и `Lbr_OutOrderApi` напишите аналогичную логику, только без проверки количества в позиции.

Проверка при удалении

в `Api` документов переопределите метод `delete` и напишите в начале проверку, чтобы нельзя было удалить документ, если он находится не в состоянии `Оформляется`

7 Урок 6. Практика разработки. Часть 3

В данном уроке мы рассмотрим:

1. Диалог с пользователем
2. Мультиселект (множественный выбор)
3. Группировка объектов
4. Объектные характеристики

7.1 Диалог с пользователем

Диалоги позволяют задавать вопросы пользователю и получать от него ответы на эти вопросы.

Руководство разработчика: Выборки # Диалоги

7.2 Мультиселект

Мультиселект - возможность выбрать несколько строк в списках или деревьях.

Руководство разработчика: Выборки # Мультиселект

Руководство разработчика: Выборки # Диалоги # Форма в режиме выбора значения

7.3 Группировка объектов

Группировка используется для систематизации хранения объектов и удобства восприятия пользователем, так же группировка позволяет массово управлять характеристиками и настройками объектов класса.

Руководство разработчика: Сервисные возможности для классов # Группировка

7.4 Объектные характеристики

Характеристика – это качественное или количественное свойство объекта. Характеристиками в системе являются специальные атрибуты, которые содержат дополнительную пользовательскую информацию.

Руководство разработчика: Сервисные возможности для классов # Объектные характеристики

7.5 Практические задания

Диалог с пользователем

1. В классе `Lbr_Publisher` добавьте диалог, в котором у пользователя запрашивается подтверждение на удаление объекта.
2. Для заказов `Lbr_InOrder` и `Lbr_OutOrder` на закладках с детализацией реализуйте следующую логику для операции «Создать»:
 - при нажатии на иконку операции открывается список книг в режиме выбора
 - если пользователь выбрал книгу и подтвердил выбор, то создается новая запись в детализации заказа.

Мультиселект

1. Для заказов на закладке «Состав» измените работу операции «Создать» на мультиселект:
 - В `Api` классов `Lbr_OutOrderDet` и `Lbr_InOrderDet` реализуйте метод регистрации записи. Метод принимает `id` заказа и `id` книги, осуществляет поиск записи в с такими идентификаторами заказа и книги в классе, если запись есть, то возвращает ее `id`, если записи нет, то создает ее и возвращает ее `id`.
 - В операции создания открывать форму выбора книг в режиме мультиселекта, для каждой выбранной книги вызывать метод регистрации, таким образом книги в позициях заказа не будут дублироваться, если пользователь выбрал книгу, уже подобранную в заказе.
2. В списке книг (`Lbr_Book#List`) реализовать операцию подбора книг в заказы:
 - создать на тулбаре новую операцию

- при нажатии на операцию создается диалог с произвольными кнопками, текст диалога В какой тип заказа добавить выбранные книги?, список кнопок:
 - Возврат
 - Выдача
 - Отмена
- Если пользователь нажал «Отмена», то выполнение операции завершается без дальнейших действий.
- Если пользователь выбрал «Возврат» или «Выдача», то открывается список заказов в режиме выбора с мультиселектом (Возврат - список `Lbr_InOrder`, Выдача - список `Lbr_OutOrder`).
- Для каждого выбранного заказа выполняется следующая логика:
 - обходим все выделенные книги и регистрируем их для заказа методом регистрации созданном в пункте 1.

Группировка объектов

1. Включите группировку для класса `Lbr_Publisher`
 - в `odm` укажите тип группировки «Множественная»
 - переформируйте исходный код и таблицы
 - соберите проект.
2. Настройте дерево групп:
 - Под корневой группой создайте 2 подгруппы:
 - Зарубежные издательства
 - Отечественные издательства
3. Выведите закладку с группами объекта в карточке (отображение `Card`).
4. Активируйте в выборке `Lbr_Publisher#List` операцию Дополнительно (имя метода `extraOperations`).
5. Под операцией Дополнительно создайте новую операцию Статистика текущей группы, выводящую статистику по группе, о количестве объектов в группе и количестве подгрупп.
 - Создайте новую операцию на тулбаре, укажите ей операцию-предка
 - Для определения текущей группы используйте `getVar("idGroup#")`
 - Используя диалоги, вывести на экран информацию вида:

```
Количество объектов в группе: 42
Количество объектов без учета подгрупп: 1
Количество подгрупп: 10
```

- Запросить количество объектов в группе можно через таблицу `Btk_FlatObjectGroup`
- Запросить количество объектов в группе без учета подгрупп можно через таблицу `Btk_ObjectGroup`
- Запросить количество подгрупп можно через таблицу `Btk_GroupTree` по полю `idParent` и исключить запись, в которой `idParent` равняется `idChild`

6. Под операцией Дополнительно реализуйте операцию Зарегистрировать объект в корневой группе, которая добавляет объект в корневую группу, для этого:
 - Создайте новую операцию на тулбаре, укажите ей операцию-предка
 - Операция получает `top` текущего объекта
 - Регистрирует объект в корневой группе методом `Btk_ObjectGroupApi#register`
 - параметр `bpMultiGroup` передавать в значении 1, т.к. на классе включена множественная группировка.
 - параметр `bpMainGroup` передавать в значении 1, так для объекта эта группа станет «Основной»
 - для получение `id` корневой группы используйте код `Btk_ClassApi().load(Lbr_PublisherApi().idClass).get(_idGroupRoot)`

Объектные характеристики

1. Добавьте в `odm` класса `Lbr_Publisher` логический атрибут `bIsForeign` - Зарубежное издательство, укажите ему свойство, что он является объектной характеристикой.
2. Настройте карточку объектных характеристик для класса `Lbr_Publisher`, зависящую от групп объекта.
 - укажите нужный тип карточки в `odm.xml`

`objectAttrCardType="group"`
 - выведите закладку в карточке объекта (используйте выборку `Btk_ObjectGroupApi#List_Tab`).
3. Добавьте объектные характеристики, хранящиеся в `json`-контейнере, в карточке класса.
 - зайдите в карточку класса `Lbr_Publisher` и откройте закладку Атрибуты
 - выполните операцию создать
 - заполните поля для нового атрибута:
 - системное имя = `dSanctionBegin`
 - наименование = Дата начала санкций
 - тип данных = `Date`
 - Тип атрибута = `Basic`
 - сохраните результат.
 - создайте второй атрибут:
 - системное имя = `idJsonAuthor`
 - наименование = Любимый автор издательства
 - тип данных = `Long`
 - Тип атрибута = `RefObject`
 - Ссылается на класс = выберите `Lbr_Author`
4. Настройте вывод объектных характеристик на группах объекта.

- Откройте интерфейс редактирования групп и закладку **Атрибуты** группы и включите редактирование в списке
 - Для корневой группы добавьте атрибут **bIsForeign**
 - Для группы «Зарубежные издательства»
 - добавьте атрибут **dSanctionBegin**
 - переопределите атрибут **bIsForeign**, установите значение по умолчанию 1. В предложенном диалоге согласитесь установить значение по умолчанию для всех объектов с перезаписью имеющихся значений.
 - Для группы «Отечественные издательства»
 - добавьте атрибут **idJsonAuthor**
 - переопределите атрибут **bIsForeign**, установите значение по умолчанию 0. В предложенном диалоге согласитесь установить значение по умолчанию для всех объектов с перезаписью имеющихся значений.
 - После настройки на карточке об. характеристик будут доступны атрибуты:
 - **bIsForeign** - для всех объектов, при включении в группу «Зарубежные издательства» будет устанавливаться значение 1, при включении в «Отечественные издательства» будет устанавливаться значение 0
 - **dSanctionBegin** - для объектов группы «Зарубежные издательства», снимите признак **Использовать для основных групп**
 - **idJsonAuthor** - для объектов группы «Отечественные издательства», снимите признак **Использовать для основных групп**
5. Под операцией **Дополнительно** сделайте операцию **Обновить характеристики**, которая будет обновлять значения характеристик объекта.
- Создайте новую операцию на тулбаре, укажите ей операцию-предка
 - Проверяем какое это издательство, получая значение характеристики **bIsForeign**
 - Если зарубежное:
 - * проверяем текущее значение характеристики **dSanctionBegin**, если оно заполнено, то выдаем сообщение **Для издательства <Наименование> уже введены санкции, начало санкций: <значение характеристики dSanctionBegin>**
 - * если текущее значение **dSanctionBegin** не заполнено, то заполняем текущей датой, и выдаем сообщение **Для издательства <Наименование> успешно введены санкции**
 - Если отечественное:
 - * проверяем текущее значение характеристики **idJsonAuthor**, если оно заполнено, то выдаем сообщение **Для издательства <Наименование> уже выбран любимый автор: <headline для значение характеристики idJsonAuthor>**
 - * если текущее значение **idJsonAuthor** не заполнено, то открываем диалог выбора автора, и устанавливаем выбранное значение в характеристику

Примечание

- Для получения значения характеристики, хранящейся в таблице класса используется стандартный способ, как и для других атрибутов класса.

- Для получения значения json-характеристики используется метод `ru.bitec.app.btk.class_.attr.AttrApi#getAttribute`. Пример вызова:

```
val dDate = Lbr_PublisherApi().getAttribute(rop, "dSanctionBegin").asNDate
```

- Для установки значения json-характеристики используется метод `ru.bitec.app.btk.class_.attr.AttrApi#setAttribute`. Пример вызова:

```
Lbr_PublisherApi().setAttribute(rop, "dSanctionBegin", NDate.now())
```

8 Урок 7. Практика разработки. Часть 4

Цель данного урока познакомить слушателя с различными возможностями классов и выборов

8.1 Сервис универсальных коллекций (v-collection)

Универсальные коллекции - это коллекции, для которых в качестве мастера могут выступать объекты разных классов.

Для изучения смотри:

- *Руководство разработчика: Коллекции переменной ссылочности*

8.2 Денормализация классов-деревьев

В нашем случае под денормализацией понимается вспомогательный класс, который обеспечивает более быстрый доступ к данным по иерархичным связям, чем рекурсивные запросы.

Для изучения смотри:

- *Руководство разработчика: Денормализация классов-деревьев*

8.3 Режимы редактирования выборки

Для отображения имеется 3 варианта редактируемости, которые указываются в свойстве отображения `editMode` в разметке (файле `avm`):

- `edit` - Разрешено редактирование данных. Можно запретить редактирование
- `notEdit` - Запрещено редактирование данных. Можно разрешить редактирование
- `readOnly` - Запрещено редактирование данных.

Для редактируемых списков доступна операция `allowEdit`, которая меняет режим редактирования.

8.4 Передача параметров в выборку

Передача параметров в выборку - способ передачи данных между выборками, которые не зависят друг от друга. По умолчанию такая передача используется при открытии карточки, в которую передаются id объекта и режим редактирования.

В передаче дополнительных параметров при создании объекта в карточке используется метод `insert_Params()`, который можно переопределить для расширения списка параметров без переопределения всего метода создания.

Для изучения смотри:

- *Руководство разработчика: Передача параметров в выборку*

8.5 Клонировемые операции

Операции, которые размножаются по результату запроса. Запрос для размножения операции может быть указан в аннотации `@Oper` в свойстве `cloneQuery` или в разметке в блоке `oper` в свойстве `cloneQuery`

Результат запроса должен содержать поля:

- `sName` - системное имя операции
- `nImageIndex` - номер изображения для операции
- `sCaption` - отображаемое имя операции

Получение значений атрибутов запроса происходит как обычное получение атрибутов через `getVar`. Для получения значений атрибутов выборки, откуда была вызвана операция, необходимо добавить `super$` к имени атрибута

8.6 Фильтры

Фильтрация набора данных производится с помощью макроса фильтра, генерируемого на основе условий с признаком `isExpression="true"` или непосредственного добавления условия в запрос получения данных. Имя макроса берется из разметки фильтра и подставляется с префиксом `&`. Например, для `<macros name="DefFltReferenceMacro">` в запрос необходимо подставить `&DefFltReferenceMacro`, в котором будет сгенерирован макрос фильтра.

По умолчанию для справочников и классов с группировкой в `Dvi` генерируется подстановка макроса фильтра в запрос с помощью `prepareSelectStatement`

Разметка для фильтров аналогична разметке атрибутов. Атрибуты фильтров выводятся на панели фильтрации и имеют свои собственные сеттеры. По умолчанию для справочников генерируется фильтр «Отображать неиспользуемые». В свойствах фрейма можно отключить отображение панели фильтрации с помощью свойства `frame` «`filter.isVisible`» (по умолчанию отображение отключено для карточки).

Пример условия для частичного совпадения по тексту

```
expression="upper(t.sCode) like upper('%'||:flt_sCode||'%')"
```

8.7 Практика

Создание универсальной коллекции

- Создайте v-collection **Lbr_Link** - «Связанные документы» с указанными ниже атрибутами

name	attribute-type	caption	Дополнительно
gidDoc	Varchar	Документ	Ссылается на класс Lbr_Document . Невидимый атрибут, ссылающийся на мастера
gidLin	Varchar	Связанный документ	Ссылается на класс Lbr_Document

- Подключите универсальную коллекцию к классам:
 - **Lbr_InAct** - Приходная накладная
 - **Lbr_InOrder** - Возврат книг
 - **Lbr_OutOrder** - Выдача книг
- Выведите отображение связанных документов в закладки документов, настроенные на тип объекта.
- В **Lbr_linkApi** реализуйте метод регистрации записи. Метод принимает **gid** документа и **gid** документа для связывания, осуществляет поиск записи с такими идентификаторами в классе, если запись есть, то возвращает ее, если записи нет, то создает и возвращает ее.
- В отображении связанных документов в операции создания открывать форму выбора из списка документов **Lbr_Document** в режиме мультиселекта, для каждой выбранной записи вызывать метод регистрации, таким образом связанные документы не будут дублироваться, если пользователь выбрал документ, уже подобранный в связях.

Создание денормализации

- Создайте журнал **Lbr_CatalogTree** - «Денормализованное дерево каталога»
- Подключите **Lbr_CatalogTree** в качестве денормализации для **Lbr_Catalog**
- Заполните денормализацию для существующей структуры каталогов, используя метод **updateDenormFull** из **TreeDenormApi**
- Для поддержания в актуальном состоянии структуры денормализованного дерева вызовите в серверных процедурах создания(**api#insert**), удаления(**api#delete**) и установки предка(**api#setidPatent**) класса **Lbr_CatalogTree** процедуры **updateDenormAfterFlush(idpObject: NLong)** и **deleteDenormAfterFlush(idpObject: NLong)** из апи денормализованного дерева.
- Добавьте в **Lbr_CatalogAvi** нехранимый атрибут, который с помощью денормализации будет отображать количество книг в каталоге и всех, входящих в него.

Режимы редактирования выборки

- Для закладки книг из *Урока 3* выведите операцию редактирования в списке `allowEdit`

Передача параметров в выборку

- Для закладки книг из *Урока 3* выведите операции:
 - Удалить.
 - Создать.

При создании записи в нее сразу должен подставиться каталог, для которого отображается перечень книг. Подстановка должна работать как при создании в списке для редактируемого списка, так и при создании в карточке для нередатируемого списка

Добавление фильтров

- Добавьте в `Lbr_BookAvi#List` фильтры:
 - ISBN
 - Наименование
 - Издатель - фильтр по трем точкам. Выбор издателя из умолчательного списка

Для добавления фильтров необходимо:

1. В разметке в блоке `filter` объявить или дописать условия в `macros`. Для ссылочного фильтра необходимо так же объявить `HL` атрибут для вывода пользователю.
2. Для `HL` атрибута ссылочного фильтра в `Avi` объявите сеттер, используя вызов метода `Lbr_PublisherAvi.processIdHLEvent()` с постановкой результата в ссылочный атрибут фильтра.
3. Добавьте макрос фильтра к запросу данных, используя `prepareSelectStatement`, если его там нет.

9 Урок 8. Дополнительные возможности

В данном уроке рассматриваются:

- Поиск по шаблону
- Универсальный фильтр
- Работа с файлами
- Точки расширений
- Проектное переопределение кода
- Шаблоны `thymeleaf`
- Логирование
- Мониторинг производительности
- SSH-сервис сервера приложений
- Локализация приложений

- Миграция данных

9.1 Поиск по шаблону

Руководство разработчика: Сервисные возможности для классов # Поиск по шаблону

9.2 Универсальный фильтр

Универсальный фильтр - сервис позволяющий осуществлять фильтрацию в списках. Позволяет фильтровать записи по коллекциям и значениям атрибутов объектов, на которые ссылаются объекты текущего класса.

Руководство по универсальному фильтру

9.3 Работа с файлами

Сервис прикрепленных файлов позволяет прикреплять к объектам класса произвольные файлы, которые сохраняются в специальном хранилище на сервере и сопоставляются с конечным объектом-владельцем.

Руководство разработчика: Сервисы класса # Сервис прикрепленных файлов

9.4 Шаблоны Thymeleaf

Шаблонизатор **Thymeleaf** используется для реализации наследования разметки выборки.

Руководство разработчика: Выборки # Использование шаблонов в разметке

Кроме использования шаблонизации в разметке, есть возможность использования шаблонов и в прикладном коде, например, формировать текст для html-фреймов.

9.5 Точки расширений

Функционал для исполнения кода в модулях, от которых нет зависимости.

Руководство разработчика: Дополнительные возможности # Точка расширения

9.6 Проектное переопределение кода

С помощью проектного переопределения возможно заменить оригинальные выборки/отображения или исполняемые методы проектными.

Руководство разработчика: Дополнительные возможности # Проектное перекрытие кода Api, Api, Lib, Pkg

9.7 Логирование

Для логирования существует сразу несколько инструментов:

- Логирование через `Btk_Log`

Использует специальную коллекцию переменной ссылаемости `Btk_Log` и пишет данные в базу. Для логирования коллекция подключается как обычно, после чего доступны для использования методы вставки записей через обычные транзакции или автономные

- Логирование сервера приложений

Использует `Logger` сервера для вывода информации в консоль. Для логирования есть методы `trace`, `debug`, `info`, `warn` и `error`, которые сохраняют в консоли сервера переданные сообщения с соответствующим уровнем.

Пример использования -

```
_logger.info("Пересчет регистра прав ролей")
```

- Логирование в инфо-панель

Использует `InfoLog` приложения для записи информации в лог сообщений. Для вывода сообщений создано отображение `Btk_InfoLogAvi#Default`, которое принято выводить на отдельной панели для основной выборки приложения в нижней части. Для логирования есть методы `info`, `warn`, `error` и `confirm`, которые сохраняют в логере переданные сообщения с соответствующим уровнем.

Пример использования -

```
application.infoLog.error("Отмена загрузки файла")
```

Пример открытия инфо-панели -

```
val vMaster = application.mainSelection
if (vMaster.form.findSelection(Btk_InfoLogAvi.default()) == null) {
    vMaster.createPanelBuilder(Btk_InfoLogAvi.default()).align(Align.bottom).toggle()
}
```

Вручную инфо-панель можно открыть/закрыть из меню Сервис -> Открыть информационную панель

- Логирование действий пользователя

Используется для отслеживания процесса выполнения запросов и подстановки параметров, последовательности вызова операций и открытия форм.

Руководство разработчика: Отладка приложений # Логирование на сторону клиента

9.8 Мониторинг производительности

Руководство разработчика: Отладка приложений # Мониторинг производительности

Наиболее часто используется `VisualVM` для поиска узких мест в части обращений к базе данных. Пример результата мониторинга производительности с использованием `VisualVM`:

Profiling results			
Results: Collected data: Snapshot			
SQL Query	Total Time	Invocations	
SELECT count(*) FROM pg_stat_activity where lower(datname) = lower('trainingDb')	9,39 ms (39,5%)	17	^
select set_config('cursor_tuple_fraction',0.001::text,false);	5,34 ms (22,5%)	15	
[DELETE FROM BTK_OBJECT WHERE (GIDREF = '36351/23'), DELETE FROM BTK_OBJECT WHERE (GIDREF = '36351/23')]	2,0 ms (8,4%)	1	
UPDATE BTK_OBJECT SET GIDREF = 'NULL' WHERE (GIDREF = 'NULL')	1,61 ms (6,8%)	7	
SELECT ID, BNOTACTIVE, DCREATEDATE_DZ, DEXPIRYDATE, DMODIFYDATE_DZ, GID, GIDROOT_DZ, IDC	0,463 ms (1,9%)	1	
SELECT ID, DCREATEDATE_DZ, DMODIFYDATE_DZ, GID, GIDREF, GIDROOT_DZ, IDCLASS, IDOBJECTTYPE,	0,453 ms (1,9%)	1	
SELECT ID, DCREATEDATE_DZ, DMODIFYDATE_DZ, GID, GIDREF, GIDROOT_DZ, IDCLASS, IDOBJECTTYPE,	0,425 ms (1,8%)	1	
SELECT ID, BNOTACTIVE, DCREATEDATE_DZ, DEXPIRYDATE, DMODIFYDATE_DZ, GID, GIDROOT_DZ, IDC	0,345 ms (1,5%)	1	
SELECT ID, DCREATEDATE_DZ, DMODIFYDATE_DZ, GID, GIDREF, GIDROOT_DZ, IDCLASS, IDOBJECTTYPE,	0,287 ms (1,2%)	1	
savepoint gtk_flush	0,269 ms (1,1%)	1	
SELECT ID, BNOTACTIVE, DCREATEDATE_DZ, DEXPIRYDATE, DMODIFYDATE_DZ, GID, GIDROOT_DZ, IDC	0,256 ms (1,1%)	1	
SELECT ID, BNOTACTIVE, DCREATEDATE_DZ, DEXPIRYDATE, DMODIFYDATE_DZ, GID, GIDROOT_DZ, IDC	0,252 ms (1,1%)	1	
SELECT ID, BNOTACTIVE, DCREATEDATE_DZ, DEXPIRYDATE, DMODIFYDATE_DZ, GID, GIDROOT_DZ, IDC	0,252 ms (1,1%)	1	
SELECT ID, BNOTACTIVE, DCREATEDATE_DZ, DEXPIRYDATE, DMODIFYDATE_DZ, GID, GIDROOT_DZ, IDC	0,237 ms (1%)	1	
SELECT ID, DCREATEDATE_DZ, DMODIFYDATE_DZ, GID, GIDREF, GIDROOT_DZ, IDCLASS, IDOBJECTTYPE,	0,216 ms (0,9%)	1	
DELETE FROM DEMO_LESSON8 WHERE (ID = 23)	0,211 ms (0,9%)	1	
SELECT ID, DCREATEDATE_DZ, DMODIFYDATE_DZ, GID, GIDREF, GIDROOT_DZ, IDCLASS, IDOBJECTTYPE,	0,205 ms (0,9%)	1	
DELETE FROM DEMO_LESSON8 WHERE (ID = 26)	0,203 ms (0,9%)	1	
DELETE FROM DEMO_LESSON8 WHERE (ID = 24)	0,176 ms (0,7%)	1	
DELETE FROM DEMO_LESSON8 WHERE (ID = 29)	0,176 ms (0,7%)	1	
SELECT ID, DCREATEDATE_DZ, DMODIFYDATE_DZ, GID, GIDREF, GIDROOT_DZ, IDCLASS, IDOBJECTTYPE,	0,171 ms (0,7%)	1	
SELECT ID, DCREATEDATE_DZ, DMODIFYDATE_DZ, GID, GIDREF, GIDROOT_DZ, IDCLASS, IDOBJECTTYPE,	0,170 ms (0,7%)	1	
DELETE FROM DEMO_LESSON8 WHERE (ID = 27)	0,170 ms (0,7%)	1	
DELETE FROM DEMO_LESSON8 WHERE (ID = 28)	0,166 ms (0,7%)	1	
DELETE FROM DEMO_LESSON8 WHERE (ID = 25)	0,160 ms (0,7%)	1	
SELECT ID, BNOTACTIVE, DCREATEDATE_DZ, DEXPIRYDATE, DMODIFYDATE_DZ, GID, GIDROOT_DZ, IDC	0,148 ms (0,6%)	1	v

Queries:
 Commands ▾ Tables ▾ Statements ▾

По данным можно определить наличие одинаковых повторяющихся обращений к базе данных или запросы, которые выполняются слишком долго, для дальнейшей оптимизации работы бизнес-логики

9.9 SSH-сервис сервера приложений

Руководство разработчика: Сервисы сервера приложений # SSH консоль сервера

9.10 Локализация приложений

Руководство разработчика: Сервисы сервера приложений # SSH консоль сервера

9.11 Миграция данных

Руководство разработчика: Релиз прикладного решения

9.12 Практическое задание

Поиск по шаблону

1. Для книг добавьте поле «Описание» в перечень полей, по которым будет работать поиск.
Проверьте, что при вводе части описания книги осуществляется подбор книги в ссылочном поле.
2. Для книг переопределите метод поиска по шаблону, сделайте возможность поиска по автору и году публикации книги.

Например, при вводе значения **Иванов 2002** ищались книги, чей автор **Иванов** и год публикации **2002**

Для этого:

- Переопределите метод поиска по шаблону в `Api` класса **Книга**
- Если текст не содержит пробелов, то вызывайте стандартный метод поиска по шаблону.
- Входящий текст бейте на 2 части, определяя автора и год выпуска.
- Если вторая часть текста - число, то:
 - осуществите поиск авторов по первой части введенного текста (вызовите метод поиска по шаблону)
 - если авторы найдены, то запросите книги по этим авторам, чей год публикации подходит под условие поиска.
 - если записей не найдено, то вызывайте стандартный метод поиска по шаблону

Проверьте, что при вводе автора и года выпуска в ссылочное на книгу поле осуществляется новая логика поиска (в позициях документа).

Примечание

Для проверки, что строка является числом, используйте метод `org.apache.commons.lang3.math.NumberUtils#isCreatable`

Универсальный фильтр

1. Отключите доступность фильтрации по полю «Описание» для списка книг.
2. Для книг подключите новую коллекцию универсального фильтра, позволяющую фильтроваться по позициям заказов `Lbr_InOrderDet`.
3. Из списка авторов реализуйте операцию открытия книг текущего автора, для этого передайте в открываемый список книг условие универсального фильтра, наложенное по полю `idAuthor`

Работа с файлами

1. Создайте на диске `C` папку `storage` и настройте в системе файловые хранилища, указав данный путь `c:\storage` и тип хранилища Локальное *Руководство разработчика: Сервисы класса # Файловое хранилище*
2. В odm класса `Lbr_InAct` настройте сервис прикрепленных файлов в простом режиме.
3. Запустите кодогенерацию, скомпилитесь, обновите код, библиотеки, odm.
4. В справочнике закладок найдите закладку с системным именем `Btk_AttachItemAvi.List_SimpleAttach` и добавьте в детализацию класс `Lbr_InAct`
5. В справочнике типов объекта отфильтруйтесь по классу `Lbr_InAct` и для найденных типов настройте новую закладку **Прикрепленные файлы**
6. После сброса всех кэшей, переоткройте карточку накладной и на новой закладке попробуйте добавить файлы (например `readme.txt`)

Точки расширений

1. Добавьте scala class для расширений `Lbr_BtkExt` (шаблон для создания Global Ext)
2. Добавьте расширением метода `deleteObjectTypeBefore`, в котором будет проверка, что для удаляемого типа объекта нет документов, в которых он был бы указан.

Проектное переопределение кода

1. Переопределите в модуле `lbr` класс `Bs_PersonAvi`
2. Создайте `avm`-файл, и унаследуйте его от базовой `avm` используя язык шаблонов.
3. Добавьте в переопределенную выборку фильтр-галку «С невыполненными приходными накладными». При включенном фильтре должны выводиться только те записи, на которые есть ссылки в документах `Lbr_InAct` с состоянием `< 300`
4. Убедитесь, что при открытии списка физических лиц выводится новый фильтр (список можно открыть из карточки Приходной накладной - атрибута «Библиотекарь»)

Логирование

1. В позициях возврата книг **Lbr_InOrderDet** на создание записей по мультиселекту добавьте запись в инфо-панель с перечнем книг, добавленных по операции.
2. При создании позиции **Lbr_InOrderDet** должна открываться инфо-панель, если она закрыта.

Мониторинг производительности

1. Подключите **VisualVM** к запущенному серверу
2. Запустите запись отслеживания соединений с БД
3. Добавьте книги в позиции возврата книг **Lbr_InOrderDet** и посмотрите результат мониторинга

SSH-сервис сервера приложений

1. Используя **Putty** подключитесь к SSH-серверу
2. Подключитесь к базе данных для выполнения сервисных операций
3. Выполните создание/обновление объектов схемы БД

Миграция данных

1. Создайте миграционный пакет для модуля **lbr**
 1. Добавьте в него **upTask**, в котором для всех объектов класса **Lbr_Publisher**, которые не входят не в одну из групп, будет регистрироваться запись в **Btk_ObjectGroup** и устанавливаться корневая группа класса.
 2. Запустите миграцию по модулю **lbr** через **external tools - Run Migrations**
2. Создайте миграционные скрипты модуля **lbr**
 1. Напишите скрипт, который обновит для таблицы **lbr_book** колонку **sDesc**. Для всех записей, у которых не заполнена это колонка, установите в него значение колонки **sCaption**
 2. Запустите миграцию по модулю **lbr** через **external tools - Run Migrations**

10 Урок 9. Настройка печатных форм

Цель данного урока, ознакомить слушателя с принципами создания отчетов.

10.1 1.Печатные формы

Печатная форма определяет тип шаблона отчёта, формат и правила его построения. При выполнении печатной формы формируется отчет в виде файла с заданным форматом.

Печатная форма может быть подключена к документу с *типа объекта* без внесения изменений в программный код.

1.1.Версии печатных форм

Версия печатной формы позволяет:

- Безопасно вернуться к предыдущей реализации;
- Запускать реализацию в зависимости от периода;

Версия печатной формы содержит:

- Тип шаблона печатной формы;
- Бинарный файл для строителя отчетов;
- Дату версии;
- Описание;

1.2.Типы шаблонов печатных форм

Определяют формат в котором задаются шаблоны печатных форм.

- **jasper**
Java библиотека для построения печатных отчетов. Позволяет строить:
 - Готовые к печати PDF-файлы в интерактивном динамическом HTML с навигацией внутри или за пределами отчета
 - Высококачественные документы PowerPoint, RTF, Word
 - Электронные таблицы или необработанные CSV, JSON или XML

Для изучения подробнее смотри:

- *Руководство разработчика: Jasper Reports*

Jaspersoft Studio

Шаблоны для построения отчетов пишутся в **Jaspersoft Studio** **Jaspersoft Studio** - это программное обеспечение для редактирования **JasperReports**. Студия помогает:

- Разрабатывать и запускать шаблоны отчетов;
- Создавать запросы к отчетам;
- Писать сложные выражения;
- Компоновать визуальные компоненты.

Шаблон представляет из себя zip архив содержащий набор xml файлов сформированных в *jasper studio*. Jasper studio это специализированное средство построения печатных форм.

- **xlxs**
Шаблоном xlxs является документ в формате **xlxs**, размеченный согласно общим правилам разметки шаблонов Для изучения подробнее смотри:
 - *Руководство разработчика: Шаблоны отчетов*
- **docx**
Шаблоном docx является документ в формате **docx**, размеченный согласно общим правилам разметки шаблонов Для изучения подробнее смотри:
 - *Руководство разработчика: Шаблоны отчетов*

1.3. Выходные форматы шаблонов печатных форм

Если тип шаблона печатной формы поддерживает конвертацию в другие форматы, можно указать перечень доступных форматов.

Если указано несколько доступных форматов, при печати печатной формы пользователю будет задан вопрос в каком формате построить отчет.

Доступные форматы шаблонов печатных форм

- jasper
 - pdf
 - png
 - txt
 - html
 - xls
 - prt
- docx
 - docx
 - pdf

Внимание

При установке выходного формата pdf шаблона docx на месте тегов в документе будут подставлены изображения. Детально данная функциональность будет описана ниже.

1.4. Загрузка файлов

Чтобы загрузить файл шаблона печатной формы в Global необходимо:

- Создать новую печатную форму в Global.
Чтобы настроить новую печатную форму в Global:
 1. Откройте приложение **Настройка системы**.
 2. Откройте перечень печатных форм **Отчеты > Печатные формы**.
 3. Используя операцию **Создать** добавьте новый отчет.
 4. Заполните обязательные атрибуты и нажмите кнопку **Редактировать**, чтобы перейти в карточку.
- Привязать новую печатную форму к типу объекта.
Чтобы настроить новую версию печатной формы необходимо:
 1. На вкладке **Версии печатной формы** создайте новую версию.
 2. Для версии выберите **Тип шаблона печатной формы**.
 3. Нажмите на кнопку **Загрузить файл в систему**.
 4. Выберите файл шаблона печатной формы:

- jasper
Для шаблона типа «jasper» файлом шаблона будет являться архив, содержащий шаблон с наименованием «main.jrml».

Внимание

1. Шаблоны суботчетов могут иметь любое название
2. В архиве может быть только один основной отчёт с наименованием main

- docx
Файл формата docx
- xlsx
Файл формата docx

1.5. Параметры отчета

Параметры отчета передаются в запрос, формирующий данные для отчета.

Пользовательские параметры

Настраиваются на печатной форме, и значения для таких параметров пользователь может внести перед выполнением отчета.

Служебные параметры

Формируются автоматически

- SIGNDATA_DZ - Блок подписи
- IDUSER - Пользователь
- IDSRCOBJECT - Объект источник
Идентификатор объекта от которого выполняется печатная форма.
- IDSRCCLASS - Класс источник
Идентификатор класса объекта от которого выполняется печатная форма

Програмные параметры

Параметры переданные в Map программного метода при построении отчёта.

1.6. Вызов печатных форм

Печатные формы могут вызываться:

- из операции выборки;
- из запроса к сервису печати;
- из rest сервиса;
- От произвольного объекта системы;
- От интерфейса свободные отчеты.

Вызов печатной формы из операции выборки

Для вызова отчета из операции выборки используйте функцию `ru.bitec.app.gtk.gl.Reports#createReportEx`:

```
/**
 * Выполняет построение отчёта по системному имени отчёта.
 *
 * @param reportName      Имя отчёта
 * @param reportVersionDate Дата
 * @param postBuildAction Действие, которое необходимо произвести после заполнения
↳ отчёта
 * @param propertyMap      Карта входящих параметров
 */
@throws[ApplicationException]
def createReportEx(reportName: String, reportVersionDate: Date, postBuildAction:
↳ PostBuildAction, propertyMap: Map[String, Any]): Unit
```

Данная функция может быть вызвана только в контексте интерактивной бизнес логики(интерфейса пользователя).

Пример:

```
reports.createReportEx("Mct_OrderSheetMaterials2", null, PostBuildAction.print,
    Map[String, Any]("IDSRCOBJECT" -> getVar("id").asNLong,
        "GIDSRCOBJECT" -> getVar("gid").asNString,
        "SDESIGNATION" -> getVar("sCode").asNString)
    )
```

Формирования файла с отчетом

Для формирования файла с отчетом используйте функцию `ru.bitec.app.rpt.Rpt_Pkg#getReportStreamEx`:

```
/**
 * Выполняет построение отчёта по системному имени отчёта.
 * Если для версии отчета указано несколько доступных для печати форматов,
 * то будет выдана ошибка построения.
 * Требуется явно указать формат построения, указав параметр [[Rpt_Pkg.
↳ ParamFormatType]]
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
*  
* @param reportName      Имя отчёта  
* @param reportVersionDate Дата  
* @param propertyMap      Карта входящих параметров  
* @return InputStream, содержащий результат построения отчёта. ByteArrayInputStream  
↪ не требует закрытия.  
*/  
def getReportStreamEx(reportName: String, reportVersionDate: Date, propertyMap: ↪  
↪ Map[String, Any]): Option[InputStream]
```

Данная функция может быть вызвана в контексте автономной логики(rest сервиса)

Внимание

В случае, если получен поток, его необходимо обязательно закрыть.

Вызов печатной формы от произвольного объекта

При открытии карточки любого объекта, выводятся стандартные операции печати по которым можно выполнять вызов печатных форм. Для добавления печатной формы к списку печати для типа объекта:

1. Откройте приложение **Настройка системы**.
2. Откройте типы объектов.
Выполните пункт меню **Сущности > Типы объектов > Типы объектов**.
3. Перейдите на вкладку **Печатные формы**.
4. Добавьте необходимые печатные формы.

Вызов печатной формы в свободных отчетах

Свободные отчеты позволяют настроить для пользователя и приложения перечень отчетов которые можно построить без привязки к каким либо типом объектов.

Для вызова интерфейса построения свободных отчетов:

1. Откройте приложение.
2. Откройте свободные отчеты.
Выполните пункт меню **Отчеты > Свободные отчеты**.
3. Выберите нужный отчет.
4. Заполните параметры.
5. Напечатайте отчет.
Для этого выполните операцию **Печать**.

Для того чтобы ПФ могла быть вызвана из свободных отчетов:

1. Откройте приложение **Настройки системы**.
2. Откройте печатные формы.
Пункт меню **Отчеты > Печатные формы**.

3. Выберите требуемую печатную форму.
4. Включите признак **Свободный отчет**.
5. Настройте параметры отчета.
6. Укажите приложение для печати.
7. Укажите требуемые роли.

Примечание

Если роли не указаны, печатная форма будет доступна всем пользователям

10.2 2. Шаблоны отчетов

Шаблоны документов имеют одинаковую разметку, что позволяет, научившись размечать шаблоны одного типа, создавать отчёты других типов.

Рассмотрим основные положения.

Основным элементом разметки шаблона является

- Тэг

Текст окружённый спецсимволами, управляющий формированием отчёта.

Все тэги имеют следующий вид: `[#{значение}#]`

Простейшим случаем является Тэг, возвращающий значение атрибута строки: `[#{имя атрибута}#]`.

С помощью таких тэгов обозначаются места, которые должны быть заполнены данными. Если в данных не будет найден атрибут или параметр с именем `{имя атрибута}`, текст будет заменён на пустоту.

Для печати в отчете набора данных предназначены переменные части, позволяющие выполнять запросы и выводить в отчёт данные с изменяющимся количеством строк.

Переменной частью называется область шаблона, ограниченная открывающим и закрывающим тэгами. Переменная часть будет выведена в отчёт столько раз, сколько записей находится в выборке переменной части.

Форматы тегов для печати данных из запроса:

- открывающий тэг

```
[#{произвольное системное имя}={SQL запрос}#]
```

- закрывающий тэг

```
[#{/произвольное системное имя}#]
```

Внимание

Важно понимать, что при построении шаблона отчёта все строчки внутри открывающего и закрывающего тэга будут удалены из документа. Поэтому для корректного построения отчёта они не должны содержать ничего лишнего.

Переменная часть обязательно должна заканчиваться закрывающим тэгом, системное имя которого совпадает с системным именем открывающего тэга. Все системные имена открывающих тэгов начинаются с символа &, а все закрывающие тэги начинаются с символа /.

При выводе данных в отчет из запроса открывающий тэг переменной части состоит из произвольного уникального в рамках шаблона имени и SQL запроса. В этом случае для данной переменной части также будет создан набор данных с запросом указанным в тэге.

Переменные части могут быть вложенными друг в друга, образуя связь мастер-деталь.

Настройки вставки изображений в печатную форму типа docx

Данная настройка производится в коллекции к печатной форме.

Для изучения подробнее смотри: *Настройки вставки изображений в печатную форму типа docx*

Для вставки изображений в печатную форму типа docx должны быть выполнены следующие условия:

1. Печатная форма должна содержать теги вида [SomeTag]

Внимание

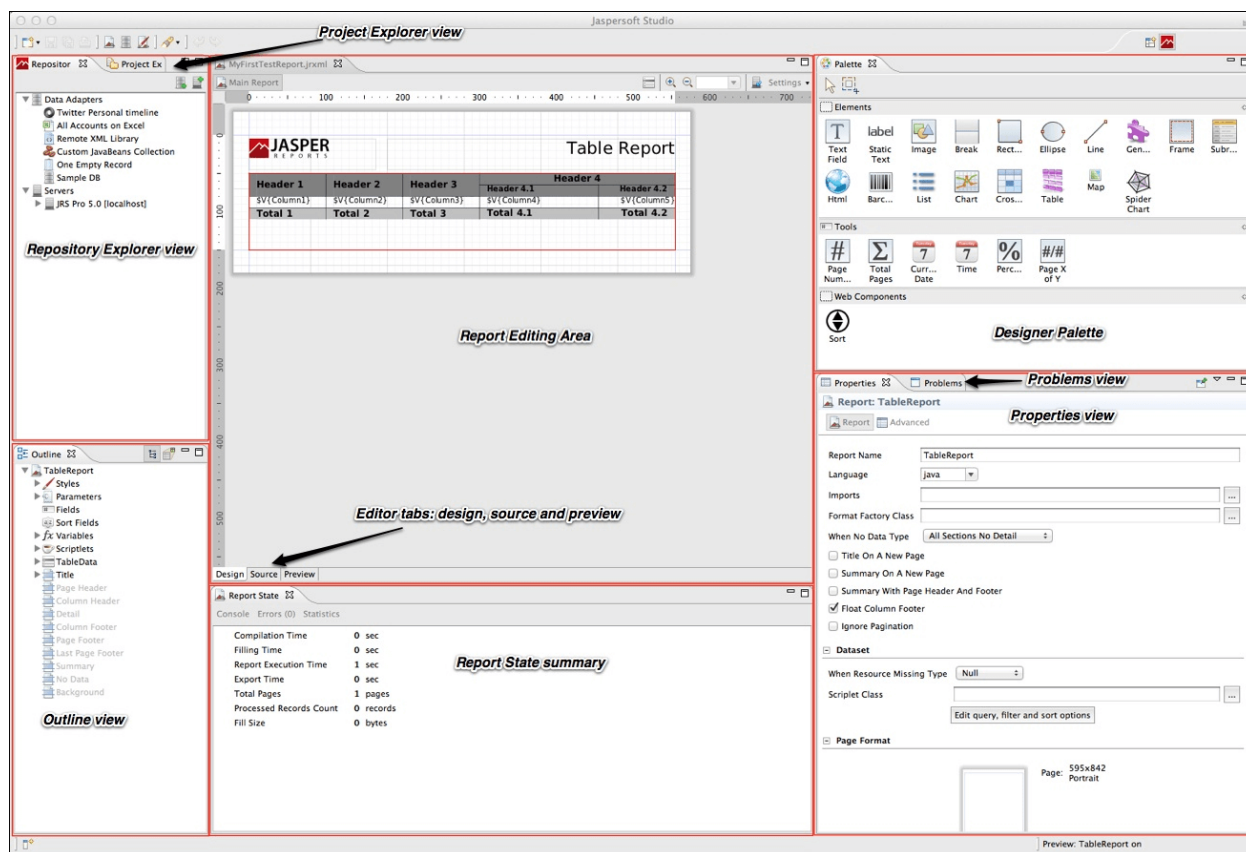
Не путать с тегами разметки шаблона!

2. Тип шаблона печатной формы - docx
3. Формат печатной формы - pdf
4. В коллекции к печатной форме(Настройки вставки изображений в печатную форму) настроены необходимые изображения для вставки

Если все требования выполнены корректно, то при печати такой печатной формы будет получен pdf файл с изображениями на месте тегов.

10.3 3. Jaspersoft Studio

3.1. Основные визуальные элементы



Repository

Содержит перечень подключений к разным базам данных. Подключения к базе данных необходимо настраивать для того чтобы была возможность тестировать отчеты из jasper studio.

Outline

Отображает структуру открытого отчета.

Properties

Отображает свойства элемента выбранного в outline. Набор закладок, отображаемых в данной области, зависит от выбранного элемента.

Editor

Отображает открытый отчет. Содержит следующие закладки:

- Design - используется для редактирования шаблона с использованием визуальных компонентов Jaspersoft Studio.
- Source - на закладке содержится xml-разметка шаблона, которую при необходимости можно отредактировать.
- Preview - используется для предпросмотра отчёта, сформированного на основе шаблона.

Project Explorer

Обозреватель проектов используется для добавления, удаления новых проектов, папок, отчетов, а также их выбор для модификации.

3.2. Проект

Все шаблоны печатных форм в Jaspersoft Studio создаются внутри проекта, который содержит все необходимые для этого библиотеки. При установке в Jaspersoft Studio создается проект по умолчанию «MyReports». Однако при необходимости можно создать новый проект.

Совет

Печатные формы проще создавать и редактировать в одном проекте если они не содержат скриплетов.

Для создания проекта отчёта:

1. Выполните **File > New > Project ...**
2. Выберите **Jaspersoft Studio / JasperReports Project**
3. Задайте имя отчёта в следующем диалоге.

По завершению на закладке **Project Explorer** появится новый проект, пока ещё не содержащий шаблонов отчёта.

3.3. Параметры

Для изучения подробнее смотри: *Параметризация шаблонов*

Параметры отчета находятся в окне **Outline** по адресу **{Report} \ parameters** Параметры отчёта используются для следующих целей:

- В SQL-запросах
Для ограничения данных по переданным из вне параметрам
- Связи между частями шаблонов
- Для связи между движком отчёта и внешним окружением.

Параметры могут быть любого java-типа.

Например:

- REPORT_CONNECTION тип `java.sql.Connection`
- REPORT_PARAMETERS_MAP тип `Map<String,Object>`

Шаблон отчёта содержит множество служебных параметров. В дереве элементов шаблона они отображаются серыми. Их невозможно удалить, или изменить значение из дизайнера. Их значения могут быть переданы извне или изменены в коде.

Внимание

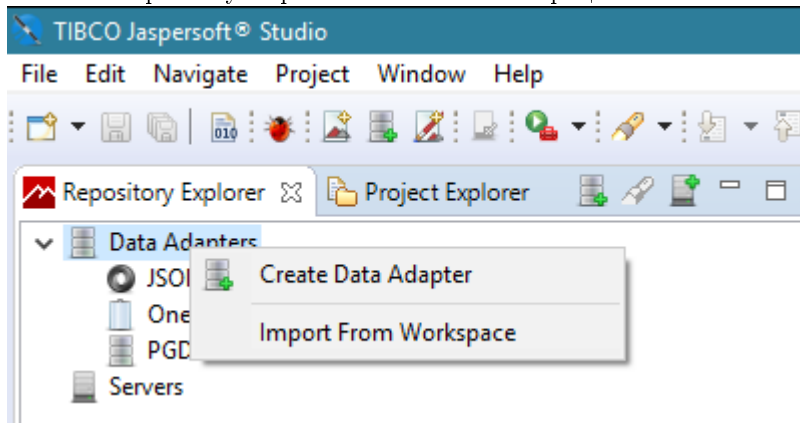
Создавайте параметры с именами в верхнем регистре. Это связано с тем, что имена параметров, переданных извне, приводятся к верхнему регистру автоматом.

Использовать параметры можно в полях отчёта в запросе данных даже в других параметрах для того чтобы использовать параметр наименование этого параметра нужно обернуть в тэг `$P{PARAMETER_NAME}`

3.4. Адаптеры данных

Для создания адаптера

1. В окне «Repository Explorer» выполните операцию «Create Data Adapter»



2. В открывшемся диалоге выберите «Data base JDBC connectivity»
3. Укажите реквизиты подключения к базе Postgres
 - name={имя подключения}
 - JDBC driver=`org.postgresql.Driver`
 - JDBC url=`jdbc:postgresql://{host}/{db}`
 - Username =`{username}`

Примечание

Подключение к БД необходимо для того, чтобы тестировать отчет из студии.

3.5. Запрос данных

3.6. Структура страницы шаблона

Созданный шаблон разделен на следующие области:

- Title – заголовок шаблона;
- Page Header – верхний колонтитул шаблона;
- Column Header – область для названия столбцов отчета;
- Detail N – область детализации отчета. В отчет могут быть добавлены несколько областей такого типа (Detail 1, Detail 2 и т.п.) в случае, если есть необходимость разделить детализацию отчета. Например, эта возможность используется, когда необходимо добавить два вложенных отчета;
- Column Footer – нижний колонтитул табличной части шаблона;
- Page Footer – нижний колонтитул шаблона;
- Summary - содержит итоги по отчету. Данные этой области выводятся в конце отчета, построенного по текущему шаблону;
- Background – часть шаблона, на которой размещается фон для печатной формы. Так же в отчет можно добавить дополнительные области;
- Last Page Footer – нижний колонтитул, данные которого выводятся в отчете на последней странице отчета, построенного по текущему шаблону;
- No Data – область, которая выводится при построении печатной формы, если отчет не содержит данных (т.е. является пустым).

3.7. Скриплеты

Для изучения подробнее смотри: *Скриплеты*

Скриптлет - это Java-класс, позволяющий кастомизировать обработку событий построения отчета а так же добавить вспомогательные функции

Дополнительная информация:

- [all-you-want-know-about-scriptlets](#)
- [hello-world-sample-example](#)
- [sample reference](#)

Скриптлет наследуется от одного из классов:

- `net.sf.jasperreports.engine.JRAbstractScriptlet`
- `net.sf.jasperreports.engine.JRDefaultScriptlet`

Экземпляр данного класса создаётся при заполнении отчёта, и его методы вызываются на различных этапах в качестве обработчиков. Так же скриптлет может содержать пользовательские методы, которые можно вызвать через выражения «Expression» полей, переменных и т.п.

3.8. Публикация отчётов в БД

Для обеспечения доступа сервера Global 3 к шаблону отчёта, zip-архив с шаблоном необходимо загрузить в базу данных через приложение «Настройка системы», пункт меню **Отчёты > Печатные формы > Версии печатной формы > Загрузить в базу**.

Шаблон JasperReports-отчёта может состоять из нескольких файлов (мастер-деталь, несколько страниц) и содержать различные ресурсы (картинки, скриплеты). Поэтому, для сохранения в базу, шаблон необходимо заархивировать в zip-архив. Файл **main.jrxml** должен находиться в корне архива, иначе он не будет найден. Остальные файлы могут быть во вложенных каталогах, главное, что бы в шаблоне были указаны верные относительные ссылки.

Архив не должен содержать бинарных *.jasper файлов.

При построении, zip-архив будет разархивирован во временный каталог.

Ручная загрузка отчетов

1. В **Project Explorer** выберите папку содержащую отчеты печатной формы.
2. В контекстном меню выполните **Export**.
3. Выберите тип **Archive file**.
4. Выберите файлы для экспорта.

Внимание

Допустимы только следующие расширения:

- jrxml
- class
- jar

5. Укажите опцию **save in zip archive**.
6. Укажите файл куда сохранить шаблон.
7. Откройте приложение **Настройки системы**.
8. Откройте печатные формы.
Пункт меню **Отчеты > Печатные формы**.
9. Создайте версию печатной формы.
10. Загрузите шаблон печатной формы.

Внимание

Убедитесь что файлы экспортируются в корень архива. Для этого при экспорте должна быть установлена опция **Create only selected directories**

Загрузка отчета на dataInstall

Загрузку отчета в базу можно добавить в dataInstall получив blob отчета с помощью метода `Rpt_ReportVersionApi().compress`.

Пример:

```
val blobData = Rpt_ReportVersionApi().compress(  
    List(  
        "subreport1.jrxml",  
        getClass.getResourceAsStream("/reports/Mes_PlanReport/subreport1.jrxml")  
    ),(  
        "subreport2.jrxml",  
        getClass.getResourceAsStream("/reports/Mes_PlanReport/subreport2.jrxml")  
    ),(  
        "order.jrxml",  
        getClass.getResourceAsStream("/reports/Mes_PlanReport/order.jrxml")  
    ),(  
        "main.jrxml",  
        getClass.getResourceAsStream("/reports/Mes_PlanReport/main.jrxml"))  
    )  
)
```

Полностью пример можно посмотреть в `ru.bitec.app.bs.Bs_DataInstallPkg#reportInstall`.

10.4 Практики

Создание jasper-отчета

Для документа Приходная накладная создать отчёт с деталью(суботчётом) Поля отчёта(берём из класса Приходная накладная):

- Документ - выводим sNumDoc;
- Владелец - выводим idDepOwner;
- От(Дата) - выводим dDoc;
- Состояние - выводим idState;
- Сумма - выводим nSum;
- Дата - выводим Дата печати отчёта;
- Акт # - выводим Мнемокод документа.

		Ваша компания Адрес _____ Город _____ Телефон _____		Дата new java.util. Акт # \$F{id}	
Документ: \$F{snumdoc}		от \$F{ddoc}		Состояние \$F{idstatehl}	
Владелец: \$F{iddepownerhl}		Сумма		\$F{nsum}	
Позиции документа					
Table for positions					
Last Page Footer					
THANKYOU FOR YOUR BUSINESS					

Поля для детали отчёта (берём из класса Состав накладной):

- Номер - выводим sNumber;
- Книга - выводим idBook;
- Цена - выводим nPrice;
- Кол-во - выводим nQty.

Номер	Книга	Цена	Кол-во
\$F	\$F{sbookhl}	\$F{nprice}	\$F{nqty}

Создание docx шаблона

Отчёт из предыдущего задания передаётся как шаблон docx.

Формирование печатных форм из приложения

Загрузить 2 отчёта с помощью метода `datainstall` класса `Приходная накладная`. Настроить данные отчёта на типе объекта для этого класса

11 Урок 10. Администрирование, аудит

Цель данного урока познакомить слушателя с принципами администрирования и аудита.

11.1 Аудит

Аудит предназначен для фиксации различных событий при работе пользователей в системе: вставка, изменение, удаление, выполнение операций, и т.д.

Аудит является сервисной возможностью и может быть включен для любого класса в системе.

Для изучения смотри:

- *Руководство разработчика: Сервисные возможности для классов # Аудит*

11.2 Пользователь

С точки зрения системы пользователем является учетная запись, на которую выдается перечень прав дающий возможность юридическому или физическому лицу выполнять действия в системе.

Для изучения смотри:

- *Администрирование приложений: Введение # Пользователь*

11.3 Администрируемый объект

Администрируемый объект - это логически связанный для массовой раздачи прав набор привилегий. Административный объект генерируется по объектам системы и позволяет единым образом выдавать права на них.

Для изучения смотри:

- *Администрирование приложений: Введение # Администрируемый объект*

11.4 Объектные привилегии

Объектные привилегии позволяют раздавать права на бизнес логику.

Для изучения смотри:

- *Администрирование приложений: Введение # Объектные привилегии*

11.5 Элементарная привилегия

Элементарные привилегии используются для разграничения действий пользователя в пользовательском интерфейсе.

Для изучения смотри:

- *Администрирование приложений: Введение # Элементарная привилегия*

11.6 Роль

Роль - административная единица для выдачи прав. В роле выдаются гранты на привилегии.

Для изучения смотри:

- *Администрирование приложений: Введение # Роль*

11.7 Профиль пользователя

Профили пользователя используются для группировки ролей и более удобной раздачи их пользователю.

Для изучения смотри:

- *Администрирование приложений: Введение # Профиль пользователя*

11.8 Практика

Аудит

1. Включите аудит на документе `Lbr_InAct` и его позициях `Lbr_InActDet`
2. Создайте структуру хранения аудита для класса `Lbr_InAct`
3. Создайте новый документ
4. Посмотрите аудит по новому документу.

Произвольная запись в аудит.

1. Включите аудит на справочнике `Lbr_Publisher`
2. Доработайте операцию **Обновить характеристики**: если успешно введены санкции, то в аудит пишется запись «Санкции успешно введены». Для создания произвольной записи используйте метод `ru.bitec.app.btk.class_.service.audit.Btk_AuditPkg#insertRow`

Синхронизация исходного кода и данных администрирования

1. Обновить администрируемые объекты модуля lbr.

Руководство администратора: Приложение администратор # Групповая настройка привилегий ролей

Создание роли доступа к приложению

1. Откройте список ролей.
2. В дереве групп добавьте новую группу **Роли библиотеки**
3. В новой созданной группе создайте роль **Доступ к библиотеке**
 - на закладке **Приложения**, доступные для роли выдайте права на приложение **Управление библиотекой**

Создание профиля

1. Откройте список профилей.
2. В дереве групп добавьте новую группу **Профили библиотеки**
3. В новой созданной группе создайте профиль **Библиотека, только чтение**
4. Подключите к профилю роль **Доступ к библиотеке**

Создание шаблона пользователя

1. Откройте список пользователей.
2. В дереве групп добавьте новую группу **Пользователи библиотеки**
3. В новой созданной группе создайте запись и укажите значение полей:
 - Учетная запись = `LbrUserTemplate`
 - Тип = **Внутренний пользователь**
 - Является шаблоном = `true`
 - Наименование шаблона = **Пользователь библиотеки**
 - Выводить под операцией создания = `true`
4. Выдайте права на профиль **Библиотека, только чтение**

Создание пользователей

1. Переоткройте список пользователей, чтобы обновилась клонируемая операция создания по шаблону.
2. Под операцией **Создать** теперь доступна операция **Пользователь библиотеки**
3. Используя новую операцию создайте пользователей:
 - `userR0`
 - `userFull`

4. Изначально пароль идентичен логину. Что бы оставить его, необходимо убрать галочку **Пользователь должен сменить пароль при входе** (рекомендуется сделать простой пароль или оставить его равным логину, для последующего быстрого тестирования).
5. Пользователи обладают всеми настройками шаблона, и имеют права на профиль **Библиотека, только чтение**
6. Выполните для обоих пользователей операцию **Пересчитать индексацию привилегий**

Проверка работоспособности пользователей

1. Авторизуйтесь под обоими пользователями, и проверьте, что доступно одно приложение (когда у пользователя доступно одно приложение, оно сразу запускается без выбора.)

Разграничение прав на заказы

1. Создайте роль **Просмотр заказов**
2. На закладке **Права роли** найдите адм. объекты **Lbr_InOrder** и **Lbr_OutOrder**
3. Снимите с них признак **Не распространяются настройки администрирования**
4. И выдайте права на **Чтение** для всего адм. объекта.
5. Профилю **Библиотека, только чтение** подключите роль **Просмотр заказов**
6. В карточке профиля выполните операцию **Пересчитать индексацию привилегий для всех пользователей профиля**
7. Зайдите под пользователями и убедитесь, что теперь они не могут редактировать заказы.
8. Создайте роль **Редактирование заказов**
9. Для адм. объектов **Lbr_InOrder** и **Lbr_OutOrder** выдайте ей права **Полный доступ**
10. Создайте профиль **Библиотека, полный доступ**, подключите к нему роль **Редактирование заказов**
11. Пользователю **userFull** выдайте профиль **Библиотека, полный доступ**
12. Синхронизируйте привилегии пользователя **userFull**
13. Прелогируйтесь под пользователем **userFull** и убедитесь, что у него есть полные права на заказы.
14. Прелогируйтесь под пользователем **userR0** и убедитесь, что у него остались права только на чтение заказов

Разграничение прав на пункты меню

1. Каждое приложение - отдельный адм. объект. Снимите с адм. объекта **Lbr_MainMenuAvi** признак **Не распространяются настройки администрирования**
2. Перейдите в карточку роли **Доступ к библиотеке**
3. Перейдите на закладку **Приложения, доступные для роли**
4. Откройте детализацию, выдайте доступ ко всем **элементарным привилегиям**, кроме **Настройки**
5. Выполните операцию **Пересчитать индексацию привилегий для всех пользователей роли**
6. Перелогиньтесь под пользователями, убедитесь, что для них не доступен пункт меню «Настройки»

7. Создайте роль **Настройка библиотеки**
8. Выдайте ей права на приложение библиотеки, и на пункт меню **Настройки**
9. Выдайте эту роль пользователю **userFull**
10. Проверьте, что у пользователя появились права на пункт меню

Управление типом привилегии операции.

1. В **Lbr_PublisherAvi** для операций **Дополнительно** и **Статистика** текущей группы смените тип привилегии на чтение, используя аннотацию **AcPrivilegeType**
2. Соберите проект, обновите адм. объект **Lbr_Publisher**
3. Создайте роль **Просмотр издателей**
 - выдайте ей права только чтение на адм. объект **Lbr_Publisher**
 - выдайте **userFull** и **userR0** права на эту роль
4. Пересчитайте индексацию для пользователей роли (операция в карточке или списке ролей), убедитесь что у пользователей **userFull** и **userR0** появилась операции **Дополнительно** и **Статистика** текущей группы а издательства не редактируются

Запрет элементарной привилегии

1. Создайте роль **Запрет просмотра статистики по группе издательств**
2. Найдите адм. объект **Lbr_Publisher** и для адм. элемента **Lbr_PublisherAvi#Default**, в детализации установите признак **Запрещено** на операции просмотра статистики по группе.
3. Создайте новый профиль, подключите к нему эту роль, выдайте профиль пользователю **userR0**
4. Пересчитайте индексацию для пользователей, убедитесь что у пользователя **userR0** нет доступа к операции просмотра статистики

Администрирование переходов состояний

1. Для адм. объекта **Lbr_OutOrder** снимите признак **Не требуется настройка прав доступа на состояния**.
2. Для класса **Lbr_OutOrder** добавьте новое состояние **Согласуется** и настройте переходы состояний.

Примечание

Переход состояний должен быть: Оформляется -> Согласуется -> Выполнен

Руководство разработчика: Тип объекта # Переходы состояний

3. Создайте роль **Права на все переходы состояний**, дайте ей доступ на все возможные переходы адм. объекта **Lbr_OutOrder**
4. Создайте роль **Права на перевод в Согласуется**, дайте ей доступ на перевод только в **Согласуется** для адм. объекта **Lbr_OutOrder**

5. Выдайте этим ролям права на редактирование атрибута `Состояние` элемента `Lbr_OutOrderAvi#Default`
6. Роль `Права на перевод в Согласуется` выдайте профилю `Только чтение`, пересчитайте индексацию пользователей профиля.
7. Роль `Права на все переходы состояний` выдайте профилю `Полный доступ`, пересчитайте индексацию пользователей профиля.
8. Зайдите под пользователем `userR0` в систему, он может переводить состояния заказов `Lbr_OutOrder` только в `Согласуется`
9. Зайдите под пользователем `userFull` в систему, он может переводить заказы `Lbr_OutOrder` в любое состояние

Объектные привилегии

1. Объявите в классе `Lbr_Publisher` новую объектную привилегию `Ввод санкций` *Руководство администратора: Введение # Объектные привилегии*
2. Соберите проект, обновите адм. объект `Lbr_Publisher`
3. В `Lbr_PublisherApi` переопределите метод установки объектных характеристик `setAttrValue` (см. урок 6), в котором:
 - проверяйте, если устанавливается значение атрибута с именем `dSanctionBegin`, проверяйте наличие объектной привилегии «Ввод санкций» методом `ru.bitec.app.btk.Btk_AdminPkg#hasObjPriv`
 - если привилегии у пользователя нет, то выдавайте ошибку `Не удалось ввести санкции в отношении <Headline издательства>`, т.к. не достаточно прав
4. Создайте роль `Ввод санкций` на издательства
5. Выдайте ей права на объектную привилегию `Ввод санкций`
6. Выдайте роль пользователю `userFull` через профиль `Полный доступ`
7. Для роли `Просмотр издателей` выдайте права на операцию `Обновить характеристики`.
8. Синхронизируйте пользователей, проверьте что у обоих юзеров есть права на операцию, но `userR0` при попытке ввести санкции получает ошибку о недостаточности прав, а `userFull` успешно их вводит.

Управление принадлежностью выборки к адм. объектам

В этом примере будет разобран пример, когда стандартное определение адм. объекта не будет работать.

1. Создайте отображение для отображения связей документов в отдельной форме
 - в `Lbr_linkAvi` создайте новый `trait List_gidDocForChoose` наследник от `List_gidDoc`
 - переопределите в нем метод `getGidMaster`, он должен получать значение переменной `gidDoc_dz` в текущей выборке (`getVar`)
 - создайте метод создания отображения
 - объявите в `avm` новое отображение
2. В `Lbr_InOrderAvi` создайте новую операцию:
 - тип привилегии - `Чтение`

- операция открывает выборку `Lbr_linkAvi#List_gidDocForChoose` с передачей в нее параметра с именем `gidDoc_dz` и значением равным `gid`-у текущей записи
3. Соберите проект, и под пользователем `admin` попробуйте выполнить эту операцию, вы должны увидеть все связи текущего заказа.
 4. Теперь попробуйте сделать это под одним из ограниченных пользователей. Перед этим нужно обновить адм. объект `Lbr_InOrder` и пересчитать индексацию пользователей, т.к. появилась новая операция. Пользователи при открытии списка связей в отдельной форме не будут иметь прав на выборку, хотя эта же выборка хорошо работает как закладка, и права на нее выданы.

Это происходит из-за того, что стандартный способ определения адм. объекта не работает, когда мы открыли коллекцию переменной ссылочности вне формы мастера.

В этом случае необходимо самому определять алгоритм определения адм. объекта.
 5. Т.к. мы отображаем записи одного документа, то адм. объект будем определять от переданного `gid-a`, для этого в `Lbr_linkAvi#List_gidDocForChoose`:
 - переопределите метод `acObject`, который определяет класс из переданного `gid-a`, и возвращает имя этого класса (т.к. имя адм. объекта равно имени класса)
 - переопределите метод `acObjectItem`, чтобы он возвращал строку вида: `<Имя класса>\Lbr_linkAvi` - так же как эта коллекция отображается в администраторе в детализации к адм. объекту, но без суффикса `#Default`

Создание отдельной группы элементарных привилегий

Отображение как отдельный элемент

1. Для отображения `Lbr_linkAvi#List_gidDocForChoose` укажите в `avm` свойство `acItemRep="List_gidDocForChoose"`
2. Обновите адм. объект `Lbr_InOrder`
3. В списке элементов появилась новая запись `Lbr_InOrder\Lbr_linkAvi#List_gidDocForChoose`. Теперь отображение `Lbr_linkAvi#List_gidDocForChoose` администрируется как отдельный элемент, и позволяет выдавать на него права отличные от всех остальных отображений.

Элементарные привилегии как отдельный элемент

1. В `Lbr_InOrderAvi` для операции отображения связей документа и удаления укажите аннотацию `@AcItemRep(name="SeparatePrivs")`
2. Обновите адм. объект `Lbr_InOrder`
3. В списке элементов появилась новая запись `Lbr_InOrderAvi#SeparatePrivs`. Теперь на эти операции можно выдавать права через отдельный элемент.

Методы работы с правами пользователя

В главном меню приложения Управление библиотекой добавьте операцию Информация о пользователе под пунктом меню Помощь.

Операция должна выводить сообщение следующего вида:

```
Идентификатор пользователя: 101
Имя пользователя: userFull
Супер-пользователь: нет
Количество ролей пользователя: 42
Есть роль "Права на все переходы состояний": да
```

Примечание

- Получить ИД пользователя `Btk_UserApi().getCurrentUserID()`
- Получить имя пользователя `Btk_UserApi().getCurrentUserName`
- Признак супер-пользователя `session.user.isSuperUser`
- Получить роли пользователя `Btk_UserApi().getRoles`
- Проверка наличия роли `Btk_AdminPkg().hasRole`

Замещение прав

1. В замещении прав укажите, что `userR0` замещает `userFull`
2. Синхронизируйте пользователей, убедитесь что `userR0` обладает всеми правами пользователя `userFull`

12 Контрольное задание

12.1 Вариант 1

Регистр движения книг

Создание класса-журнала

- Создайте директорию `regmov` и в нем класс `Lbr_RegisterMov` - Регистр движения книг с типом `journal` и атрибутами:

name	attribute-type	caption	Дополнительно
idBook	Long	Книга	Ссылается на класс Lbr_Book
gidSrcDoc	Varchar	Документ источник	Атрибут переменной ссылаемости (type="refAnyObject"), ссылается на класс Lbr_Document
gidDet	Varchar	Позиция источник	Атрибут переменной ссылаемости
nDirection	Number	Направление	
idLibrarian	Long	Библиотекарь	Ссылочный на объект, ссылается на класс Bs_Person
idPerson	Long	Читатель	Ссылочный на объект, Ссылается на класс Bs_Person
nQty	Number	Кол-во	
dReg	Date	Дата операции	Редактор даты
idDepOwner	Long	Организация	Ссылочный на объект, Ссылается на класс Bs_DepOwner

- Запустите кодогенерацию по данному классу
- Добавьте `orm` класса в `all.xml` и запустите генератор таблиц для него
- Добавьте в `Lbr_MainMenuAvi` пункт **Регистр движения книг** под **Тесты и отладка**, в котором пропишите открытие умол. списка для `Lbr_RegisterMovApi`

Отражение в регистр и удаление из него

В `Lbr_RegisterMovApi` напишите логику для отражения и удаления из регистра:

- Для универсальности и изоляции логики регистра объявите `case class` (для удобства можно объявить перед `class Lbr_RegisterMovApi`, например назвать `objReg`) с полями из таблицы ниже
- Создайте метод отражения в регистр по списку объектов данного `case class` (например `wrtByObj(apObj: List[objReg])`). В нем напишите обход (например через `foreach`) по списку объектов `apObj` и создание записей в регистре по данным из них, при установке количества, можно записывать сразу с учетом `nDirection`, чтобы в дальнейшем при вычислении остатков не писать `nQty*nDirection`.
- Напишите метод удаления из регистра по полю `gidSrcDoc`, на вход будет `gid` документа (например `deleteBySrcDoc(gidSrcDoc: NGid)`). Вместо запросов можно использовать транзакционный индекс. *Руководство разработчика: Взаимодействие с базой данных # Транзакционный индекс.*

В `Api` документов:

- Напишите метод отражения в регистр по позициям документа, на вход будет `rop` документа (например `wrtToReg(rop: ApiRop)`). Для этого создайте переменную `val avObj = ArrayBuffer[objReg]()` и при обходе по позициям документа (через `refreshByParent`) заполните `avObj` объектами из `case class` с данными по таблице ниже. После обхода позиций вызовите метод отражения в регистр по данным заполненного списка `Lbr_RegisterMovApi().wrtByObj(avObj.toList)`
- На переход состояния в ≥ 300 и из < 300 , после вызова метода с проверками, пропишите вызов метода отражения в регистр, на обратный переход пропишите вызов метода удаления из регистра.

Таблица с полями для `case class` и откуда надо брать для них значения:

Имя поля в регистре	Тип	Значение
idBook	NLong	idBook из позиции
gidSrcDoc	NGid	gid документа
gidDet	NGid	gid позиции
nDirection	NNumber	1 для приходной накладной и возврата книг, -1 для выдачи книг
idLibrarian	NLong	idLibrarian из документа
idPerson	NLong	idPerson из документа, если есть такое поле
nQty	NNumber	Количество в позиции, если атрибута с количеством нет, то берем 1
dReg	NDate	Дата документа
idDepOwner	NLong	idDepOwner из документа

Метод проверки на отрицательные остатки

В `Lbr_RegisterMovApi` напишите метод проверки на наличие отрицательных остатков для регистра, в разрезе `idBook` и `idDepOwner`. Для этого:

- Добавьте еще один `case class` с атрибутами разреза (например `case class objRegVal(idBook: NLong, idDepOwner: NLong)`) уже внутри `Lbr_RegisterMovApi`
- Добавьте новый метод, на вход которого будет список из данного `case class` (например `validateReg(apObjVal: List[objRegVal])`)
- Для упрощения написания проверки, можно использовать `for (rv <- new ASelect` с запросом по регистру, с группировкой по разрезам и условием на отрицательное количество. Для передачи в запрос данных из параметра `apObjVal`, создайте массивы с типом `LongPgArray` для `idBook` и `idDepOwner` (например `val idavBook = LongPgArray(apObjVal.map(_.idBook))`) и в запрос пропишите в конструкции `with` трансформацию массивов в таблицу со столбцами. Далее уже можно обращаться к `data` как к обычной таблице.

```
with data as(
  select unnest(${idavBook}) idBook
         ,unnest(${idavDepOwner}) idDepOwner
)
```

- Внутри `for` соберите полученные данные запроса с отрицательными остатками в формате книга `${HL книги} - остаток ${получившееся количество} (организация ${HL организации})"` в переменную (например `val avError = ArrayBuffer[NString]()`) объявленную до `for` и уже в конце если она не пустая, то пропишите вызов ошибки в формате Действие невозможно, так как в регистре образовались отрицательные остатки: `${собранные в avError данные}`.
- Если проверка будет осуществляться через запрос, то обязательно перед `for` пропишите `session.flush()`, чтобы в БД были актуальные данные.
- В методе `wrtByObj` и `deleteBySrcDoc` допишите сбор данных разрезов в переменную со списком `case class` (например `val avObjRegVal = ArrayBuffer[objRegVal]()`) и в конце допишите вызов метода проверки `validateReg(avObjRegVal.toList)`

Отчет по остаткам

Шапка

Создайте отдельную `avi` для отчета по остаткам с полями:

- ISBN книги
- Наименование книги
- Остаток на начало периода
- Возврат/поступление за период
- Выдача за период
- Изменение за период
- Остаток на конец периода

Особенности реализации:

1. Данные для отчета берем по регистру `Lbr_RegisterMov`
2. Берем данные до конца периода и фильтруем их по организации.
3. Для расчета количество на начало периода суммируем только те данные, дата которых меньше начала периода.
4. Для расчета изменений в текущем периоде суммируем только те данные, дата которых входит в наш период.

Примечание

Для того чтобы исключить данные из суммирования можно воспользоваться следующей конструкцией: `sum(case when r.dRegDate <= :flt_dBegin then <расчет количества> end)`

Фильтр с подстановкой значений по умолчанию из главной выборки приложения:

- Период с
редактор даты.
- по
редактор даты
- Организация
выпадающий список

Расположите фильтры в одну строчку (принцип размещения атрибутов на панели фильтров аналогичен размещению в карточке. Если будете делать через разметку `hBox`, `vBox`, то используйте тег `layout` внутри тега `filter`)

Примечание

В этом задании нельзя использовать макрос фильтра в тексте запроса, т.к. нам нужны данные до начала периода, чтобы посчитать количество на начало. Следует обращаться к переменным фильтра напрямую в запросе, например, `:flt_dBegin`

Перечень атрибутов выборки главного приложения можно посмотреть через **debugger**. Для получения значения используйте метод **getVar**, передав в него имя нужного атрибута с префиксом **super\$**

Детализация

Создайте и подключите детализацию к мастер-выборке с расшифровкой по документам:

- HL документа (для удобства заголовков можно взять из **Lbr_Document**)
- Дата
- Количество с учетом направления

Примечание

1. Данные отчета берем по регистру **Lbr_RegisterMov**
2. Фильтруем их по периоду, книге и организации
3. Группируем по документу, считаем количество

Добавьте операцию открытия документа в карточке с помощью метода открытия по **gid** **Btk_CommonOperLib().cardEdit**

12.2 Вариант 2

Регистр движения книг

Создание класса-журнала

- Создайте директорию **regmov** и в нем класс **Lbr_RegisterMov** - Регистр движения книг с типом **journal** и атрибутами:

name	attribute-type	caption	Дополнительно
idBook	Long	Книга	Ссылается на класс Lbr_Book
gidSrcDoc	Varchar	Документ источник	Атрибут переменной ссылочности (type="refAnyObject"), ссылается на класс Lbr_Document
gidDet	Varchar	Позиция источник	Атрибут переменной ссылочности
nDirection	Number	Направление	
idLibrarian	Long	Библиотекарь	Ссылочный на объект, ссылается на класс Bs_Person
idPerson	Long	Читатель	Ссылочный на объект, Ссылается на класс Bs_Person
nQty	Number	Кол-во	
dReg	Date	Дата операции	Редактор даты
idDepOwner	Long	Организация	Ссылочный на объект, Ссылается на класс Bs_DepOwner

- Запустите кодогенерацию по данному классу
- Добавьте `orm` класса в `all.xml` и запустите генератор таблиц для него
- Добавьте в `Lbr_MainMenuAvi` пункт **Регистр движения книг** под **Тесты и отладка**, в котором пропишите открытие умол. списка для `Lbr_RegisterMovAvi`

Отражение в регистр и удаление из него

В `Lbr_RegisterMovApi` напишите логику для отражения и удаления из регистра:

- Для универсальности и изоляции логики регистра объявите `case class` (для удобства можно объявить перед `class Lbr_RegisterMovApi`, например назвать `objReg`) с полями из таблицы ниже
- Создайте метод отражения в регистр по списку объектов данного `case class` (например `wrtByObj(apObj: List[objReg])`). В нем напишите обход (например через `foreach`) по списку объектов `apObj` и создание записей в регистре по данным из них, при установке количества, можно записывать сразу с учетом `nDirection`, чтобы в дальнейшем при вычислении остатков не писать `nQty*nDirection`.
- Напишите метод удаления из регистра по полю `gidSrcDoc`, на вход будет `gid` документа (например `deleteBySrcDoc(gipSrcDoc: NGid)`). Вместо запросов можно использовать транзакционный индекс. *Руководство разработчика: Взаимодействие с базой данных # Транзакционный индекс.*

В `Api` документов:

- Напишите метод отражения в регистр по позициям документа, на вход будет `rop` документа (например `wrtToReg(rop: ApiRop)`). Для этого создайте переменную `val avObj = ArrayBuffer[objReg]()` и при обходе по позициям документа (через `refreshByParent`) заполните `avObj` объектами из `case class` с данными по таблице ниже. После обхода позиций вызовите метод отражения в регистр по данным заполненного списка `Lbr_RegisterMovApi().wrtByObj(avObj.toList)`
- На переход состояния в ≥ 300 и из < 300 , после вызова метода с проверками, пропишите вызов метода отражения в регистр, на обратный переход пропишите вызов метода удаления из регистра.

Таблица с полями для `case class` и откуда надо брать для них значения:

Имя поля в регистре	Тип	Значение
<code>idBook</code>	<code>NLong</code>	<code>idBook</code> из позиции
<code>gidSrcDoc</code>	<code>NGid</code>	<code>gid</code> документа
<code>gidDet</code>	<code>NGid</code>	<code>gid</code> позиции
<code>nDirection</code>	<code>NNumber</code>	1 для приходной накладной и возврата книг, -1 для выдачи книг
<code>idLibrarian</code>	<code>NLong</code>	<code>idLibrarian</code> из документа
<code>idPerson</code>	<code>NLong</code>	<code>idPerson</code> из документа, если есть такое поле
<code>nQty</code>	<code>NNumber</code>	Количество в позиции, если атрибута с количеством нет, то берем 1
<code>dReg</code>	<code>NDate</code>	Дата документа
<code>idDepOwner</code>	<code>NLong</code>	<code>idDepOwner</code> из документа

Метод проверки на отрицательные остатки

В `Lbr_RegisterMovApi` напишите метод проверки на наличие отрицательных остатков для регистра, в разрезе `idBook` и `idDepOwner`. Для этого:

- Добавьте еще один `case class` с атрибутами разреза (например `case class objRegVal(idBook: NLong, idDepOwner: NLong)`) уже внутри `Lbr_RegisterMovApi`
- Добавьте новый метод, на вход которого будет список из данного `case class` (например `validateReg(apObjVal: List[objRegVal])`)
- Для упрощения написания проверки, можно использовать `for (rv <- new ASelect` с запросом по регистру, с группировкой по разрезам и условием на отрицательное количество. Для передачи в запрос данных из параметра `apObjVal`, создайте массивы с типом `LongPgArray` для `idBook` и `idDepOwner` (например `val idavBook = LongPgArray(apObjVal.map(_.idBook))`) и в запрос пропишите в конструкции `with` трансформацию массивов в таблицу со столбцами. Далее уже можно обращаться к `data` как к обычной таблице.

```
with data as(
  select unnest(${idavBook}) idBook
        ,unnest(${idavDepOwner}) idDepOwner
)
```

- Внутри `for` соберите полученные данные запроса с отрицательными остатками в формате книги `${HL книги} - остаток ${получившееся количество} (организация ${HL организации})"` в переменную (например `val avError = ArrayBuffer[NString]()`) объявленную до `for` и уже в конце если она не пустая, то пропишите вызов ошибки в формате Действие невозможно, так как в регистре образовались отрицательные остатки: `${собранные в avError данные}`.
- Если проверка будет осуществляться через запрос, то обязательно перед `for` пропишите `session.flush()`, чтобы в БД были актуальные данные.
- В методе `wrtByObj` и `deleteBySrcDoc` допишите сбор данных разрезов в переменную со списком `case class` (например `val avObjRegVal = ArrayBuffer[objRegVal]()`) и в конце допишите вызов метода проверки `validateReg(avObjRegVal.toList)`

Отчет по остаткам

Главная форма

Создайте отдельную `avi` для отчета по остаткам с полями:

- ISBN книги
- Наименование книги
- HL документа
- Дата документа
- Номер документа
- Количество с учетом направления

Особенности реализации:

1. Данные для отчета берем по регистру `Lbr_RegisterMov`
2. Поля, которые относятся к документу можно взять из миксина `Lbr_Document`

3. Фильтруем их по периоду, организации и документу
4. Рабочий период определяется от даты в фильтре +- 30 дн.
5. Группируем по документу, считаем количество

Фильтры:

- **Дата**
Текущая дата.
- **Организация**
выпадающий список. Значение берется из главной выборки приложения
- **Документ** Фильтр с тремя точками. Позволяет открыть список объектов миксина `Lbr_Document`.

Фильтры **дата** и **Организация** расположите в одной строке. Фильтр **Документ** в отдельной строке (принцип размещения атрибутов на панели фильтров аналогичен размещению в карточке. Если будете делать через разметку `hBox`, `vBox`, то используйте тег `layout` внутри тега `filter`)

Примечание

Перечень атрибутов выборки главного приложения можно посмотреть через `debugger`. Для получения значения используйте метод `getVar`, передав в него имя нужного атрибута с префиксом `super$`

Текущую дату можно получить через `NDate.now()`

Добавьте операцию открытия документа в карточке с помощью метода открытия по `gid` `Btk_CommonOperLib().cardEdit`

Форма с остатками

В главной форме добавьте операцию на тулбар, которая будет по выделенной строке регистра выводить информацию об остатках по книгам, по которым было движение в указанном периоде.

Для этого необходимо:

1. Создать еще одно отображение в вышей выборке, которая будет отображать информацию с остатками
2. Из операции главной формы необходимо открывать форму с остатками, передав туда параметры о текущем выбранном периоде, организации, и документе в выделенной строке.
3. В форме-расшифровке реализуйте вывод следующих полей:
 - ISBN книги
 - Наименование книги
 - Остаток на начало периода
 - Возврат/поступление за период
 - Выдача за период
 - Изменение за период
 - Остаток на конец периода

Особенности реализации:

1. Данные для отчета берем по регистру `Lbr_RegisterMov`
2. Берем данные до конца периода и фильтруем их по организации (они переданы через параметры в форму).
3. Берем только книги, по которым было движение в переданном документе.
4. Рабочий период определяется от переданной даты ± 30 дн.
5. Для расчета количество на начало периода суммируем только те данные, дата которых меньше начала периода.
6. Для расчета изменений в текущем периоде суммируем только те данные, дата которых входит в наш период.

Примечание

Следует обращаться к переданным параметрам напрямую в запросе, например, `:dDate_dz`

Для того чтобы исключить данные из суммирования можно воспользоваться следующей конструкцией: `sum(case when r.dRegDate <= :dDate_dz - interval '30 days' then <расчет количества> end)`

13 Приложение 1. Руководство разработчика

13.1 Введение

Предисловие

Добро пожаловать в Руководство разработчика `GlobalFramework for Postgresql`.

Цель руководства - предоставить информацию о разработке в `GlobalFramework`. Руководство так или иначе касается всех аспектов работы с данной технологией.

На кого ориентировано руководство

Руководство ориентировано на инженеров-разработчиков, а также выпускников (студентов старших курсов) классических и технических университетов и других вузов, имеющих подготовку по программированию и продолжающих специализироваться в разработке корпоративных информационных систем на базе `Global3 Framework`.

Программные средства для работы

Обязательные инструменты:

- `IntelliJ IDEA` - Среда разработки
- `Liberica JDK 8`
- `DBeaver` - Среда написания `sql` запросов
- `Global Server` - Сервер приложения (поставляется отдельно)
- `GlobalFramework Starter Kit` - Набор шаблонов и файлов для упрощенной настройки рабочего места (поставляется отдельно)

- git

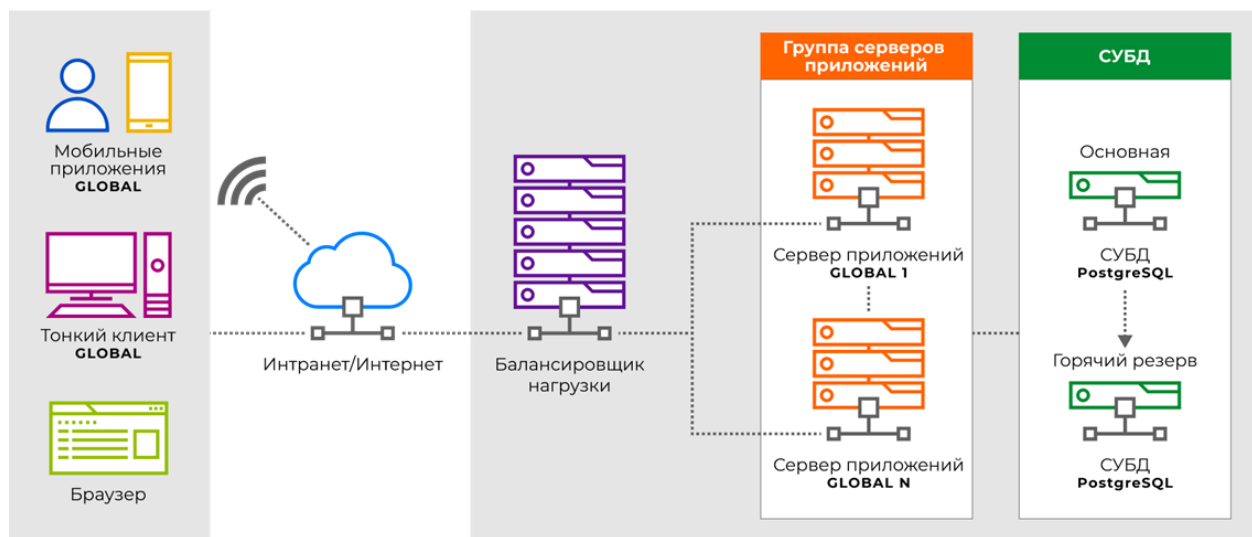
Рекомендуемые инструменты:

- Notepad++ - Удобный быстрый текстовый редактор
- VS Code - Среда разработки для работы с контейнерами и написания документации
- Chrome
- Adobe Acrobat Reader DC
- PuTTY - Доступ к контейнеру через ssh в виде командной строки
- Winscp - Доступ к контейнерам через ssh в виде файлового менеджера
- TortoiseSVN - Удобное средство работы с svn(система контроля версий)

Обзор

Global3-FrameWork является мощным и удобным инструментом разработки кроссплатформенных решений.

Архитектура системы



Тонкий клиент

Тонкий клиент подключается к серверу приложений в терминальном режиме, что позволяет работать на компьютерах и планшетах через канал с низкой пропускной способностью. Это возможно, так как в терминальном режиме пользователь получает только минимально необходимый для работы набор данных.

Например, если пользователь открыл список содержащий 10 000 записей, к нему на экран будет отправлена только видимая часть и кэш (примерно 100-200 записей, зависит от размеров экрана).

Тонкий клиент может работать как веб приложение в любом браузере, поддерживающим стандарт html 5.0, или поставляться в виде отдельного приложения на базе chromium.

В браузер может быть установлено дополнительное расширение, позволяющее тонкому клиенту напрямую работать с файлами, оборудованием и крипто провайдерами. При наличии такого расширения, работа в браузере неотличима от работы в нативном приложении.

Мобильный клиент

Существует линейка мобильных приложений, позволяющая использовать полный перечень возможностей мобильного устройства.

Для реализации специализированных приложений, GlobalFramework предоставляет возможность написания web сервисов, посредством которых возможно взаимодействие с нативным мобильным приложением, а так же набор библиотек, облегчающих написание мобильного приложения.

Балансировщик нагрузки

Балансировщик нагрузки распределяет пользователей между серверами приложений в кластере.

Кластер серверов приложений

Нагрузка горизонтально масштабируется между серверами приложений.

Сервера приложений объединяются в кластер для возможности централизованного управления и мониторинга.

Решение

Набор скомпилированных библиотек, устанавливаемых в сервер приложения. Решение предоставляет весь перечень прикладной бизнес логики необходимый для работы сервера приложения.

Кластер баз данных

Кластеризация базы данных поддерживается штатными средствами `postgreSQL`. В случае синхронной репликации горячей копии, возможно перенаправление на нее запросов на чтение.

Уровни разработки

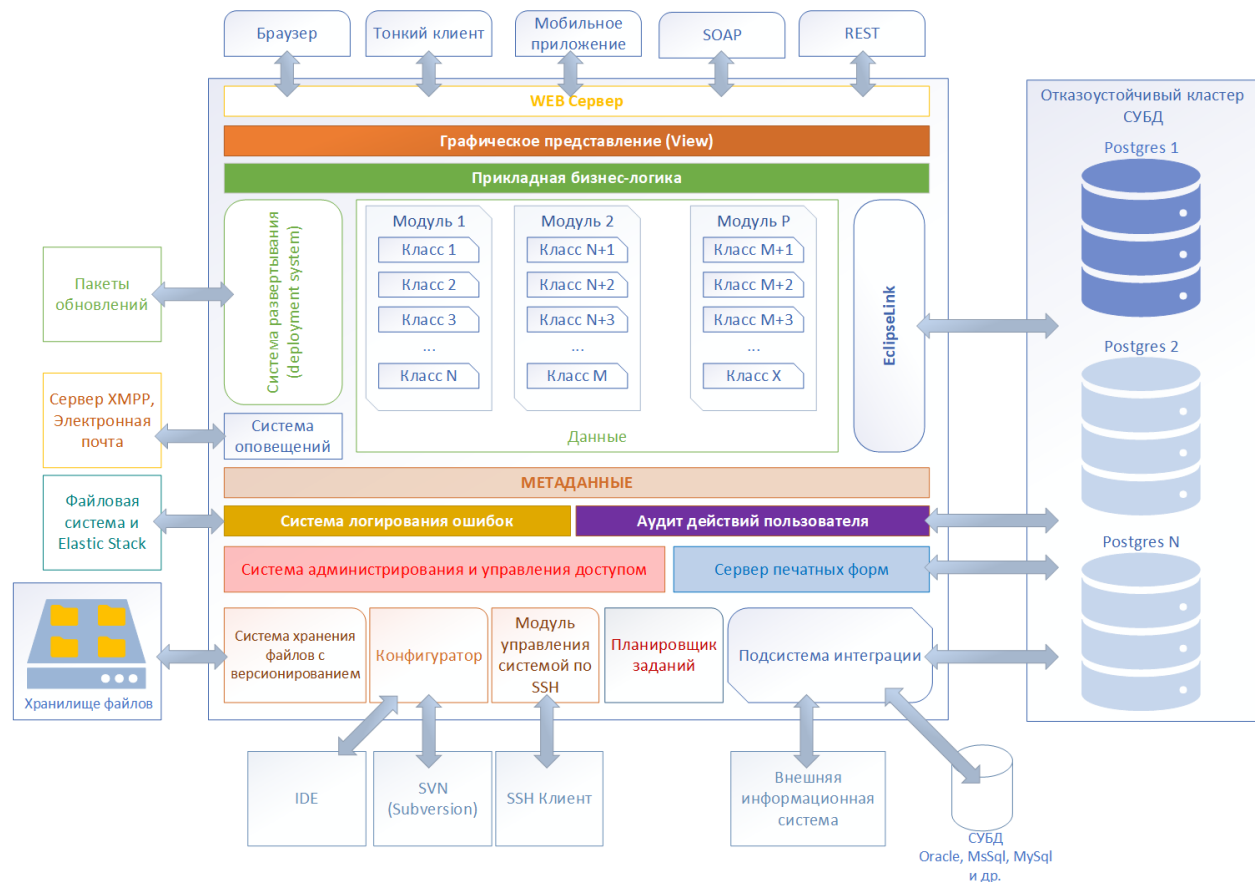
Разделение бизнес логики по уровням позволяет существенно повысить повторное использование кода между пользовательскими интерфейсами и фоновыми сервисами(интеграция, выполнение задач по расписанию и т.д.).

Система Global состоит из следующих уровней:

- Интерактивная бизнес логика - Отвечает за ввод/вывод информации и обработку действий пользователя.
- Автономная бизнес логика - Осуществляет основную обработку данных.
- Хранение данных - Реализован СУБД.

Фреймворк предоставляет удобные инструменты для взаимодействия между уровнями.

Принципиальная схема



где:

- **Графическое представление**
Вид интерфейса определяется по средством декларации пользовательского интерфейса. Реакция на пользовательские действия программируется на уровне интерактивной бизнес логики.
- **Прикладная бизнес логика**
Располагается по большей части в слое автономной бизнес логики. Прикладная бизнес логика группируется по модулям, классам и пакетам.
- **Метаданные**
Объекты автономной и интерактивной бизнес логики регистрируются в базе данных, в слое мета данных, что позволяет выполнять администрирование и кэширование элементов бизнес логики при необходимости
- **Система администрирования**
Определяет права доступа пользователя к интерактивной и автономной бизнес логике
- **Система развёртывания**
Отвечает за создание и обновление схемы базы данных

Основные понятия

Класс

Определяет правила хранения и обработки таблицы базы данных.

Класс позволяет существенно ускорить разработку бизнес логики, ориентированную на работу с данными. Программисту достаточно объявить перечень атрибутов класса чтобы за счет кода генерации получить набор готовых сервисов:

- Интеграцию с `Orm`
 - Роjo объект для хранения данных в кэше
 - Аго объект интеграции роjo в фреймворк
- Каркас автономной логики(`Api`) - `scala` класс с постфиксом `Api`
- Каркас интерактивной логики(`Avi`) - `scala` класс с постфиксом `Avi`
- Каркас декларации пользовательского интерфейса(`Avm`) - `xml` файл с расширением `avm.xml`
- Прикладные сервисы

Подробнее смотри:

- *Классы*
- *Сервисы класса*
- *Тип объекта*

Бизнес объект

Бизнес-объект (БО) - объединение нескольких классов и их коллекций в группу для более удобного манипулирования ими при работе с кэшем и конфигурировании вспомогательных сервисов.

Бизнес объект позволяет:

- Массово загружать данные в транзакционный кэш
Для бизнес объекта можно указать стратегию загрузки данных существенно уменьшающую количество запросов в базу данных. Так как запросы пойдут не по каждому объекту а по каждому классу бизнес объекта.
- Настраивать права доступа
По бизнес объекту создается административный объект на котором можно массово выдать привилегии для всех классов бизнес объекта
- Управлять электронной подписью
Можно настроить правила подписи всего бизнес объекта включая не только шапку но и все вложенные коллекции.
- Настраивать интеграцию и репликацию

Пакет

Позволяют сгруппировать общую бизнес логику в контексте сессии приложения. Пакет является `scala` классом с постфиксом `Pkg`.

Выборка

Выборка определяет правило получения, отображение данных и обеспечивает взаимодействие с пользователем. Выборки содержат основную часть интерактивной бизнес логики.

Выборка определяет:

- способ получения данных
- способ отображения данных пользователю
- обработки пользовательских действий

Подробнее смотри:

- *Выборка*

Модуль

Неделимая часть решения. Набор модулей определяет доступную функциональность развернутую в сервере приложения.

Модуль может зависеть от других модулей. В таком случае, такие зависимости так же должны быть установлены в решения для обеспечения корректной работы.

Приложение

Приложение является коренной выборкой рабочего сеанса пользователя и задает:

- Главное меню
Содержит перечень форм, отчетов и операций
- Панель быстрого доступа
Содержит операции быстрого доступа, а так же поля задающие контекст работы рабочего сеанса, например рабочий период

Пользователь при входе в систему выбирает требуемое ему для работы приложение.

Перечень приложений доступных для пользователя задается администратором системы.

Сессия приложения

Сессия приложения создается на поток прикладной бизнес логики. Сессия приложения предоставляет доступ к сессии базы данных, EclipseLink кэшу, серверу приложения. А так же содержит необходимые инъекции зависимости для работы прикладной бизнес логики.

Сессия приложения создается:

- на каждую `mdt` форму открытую в приложении

- на rest запрос к серверу приложения

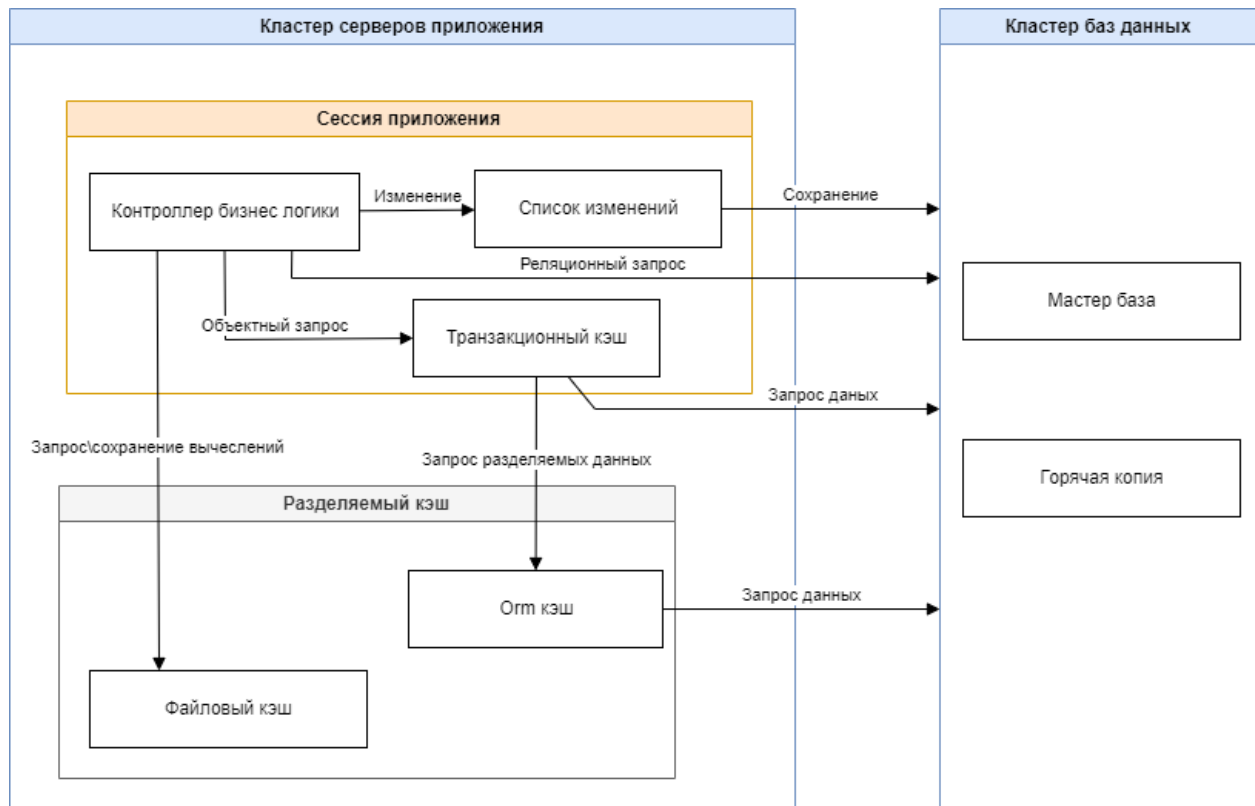
Примечание

Для ускорение rest запросов возможна настройка пула сессий

Подробнее смотри:

- *Сессия приложения*

Взаимодействие с данными



где:

- **Объектный запрос**
Запрос данных через JQPL, данные запрошенные на данном языке попадают в транзакционный кэш, что позволяет их редактировать.
Применяется к примеру в карточке документа.
- **Реляционный запрос**
Запрос в базу данных на языке sql, позволяет использовать аналитические функции и сложную фильтрацию.
Применяется к примеру в списках.
- **Транзакционный кэш**
Перечень запрошенных объектов в текущей сессии.

- *Отм кэш*

Кэш разделяемых объектов. На классе можно включить режим кэширования в разделяемом кэше.

Применяется на классах типизации, для снижения нагрузки на базу.

- *Файловый кэш*

Повторяемые вычисления можно кэшировать в файловом кэше.

Объектное взаимодействие

Объектное взаимодействие реализовано на основе библиотеки EclipseLink, которая отвечает за низко-уровневое взаимодействие между сервером приложений и базой данных через механизмы ORM.

Объектное взаимодействие позволяет повысить быстродействие системы за счет:

- Кэширования данных
- Применения ленивых коллекций
- Пакетных (Bulk) запросов, которые обеспечивают:
 - Сбор данных по бизнес-объекту
 - Обновление\вставку\удаление

За счет работы с объектами через объектный кэш возможна реализация прикладных сервисов, таких как аудит.

Реляционные запросы

При необходимости возможно взаимодействие с базой данных напрямую по средством язык `sql`.

Начало работы с фреймворком

Требования к аппаратному обеспечению рабочего места разработчика

Характеристика сервера	Рекомендуемые параметры
ОС	Linux (Debian, Astra, Alt), Windows 10 и выше
CPU	8 ядер 3.5 GHz или более
Оперативная память	16 – 32 Gb или более
Память	SSD 20 Gb свободного места
Монитор	разрешение 1920×1080

Использование виртуальной машины может заметно снизить производительность при сборке и отладки проекта.

Создание новой БД для проекта

Для создания базы данных используется команда `CREATE DATABASE`, после которой указывается название базы данных.

Для выполнения запросов будем использовать графический клиент `pgAdmin`, хотя также можно использовать консольный клиент `psql`.

Создадим для проекта нового пользователя бд и базу данных, например:

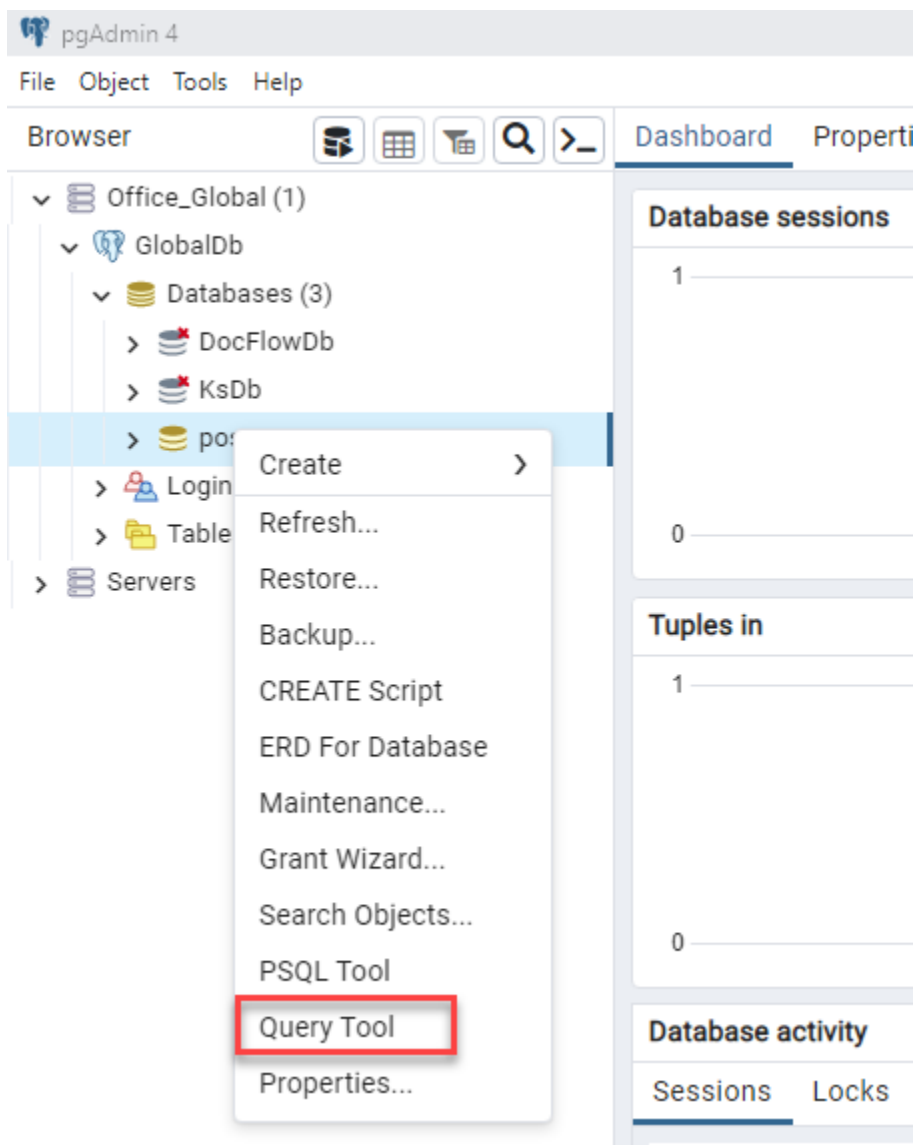
Логин: `sampleproject`

Пароль: `sampleproject`

БД: `sampleproject`

Чтобы создать нового пользователя и новую БД:

1. Откройте `pgAdmin`.
2. Подключитесь под суперпользователем к СУБД.
3. В левой части программы выберите базу данных `postgres`
4. Выделите ее и правой кнопкой мыши запустите инструмент `Query Tool`



Создание пользователя

В центральной части программы откроется поле для ввода кода SQL.

1. В окно запроса введите следующий код:

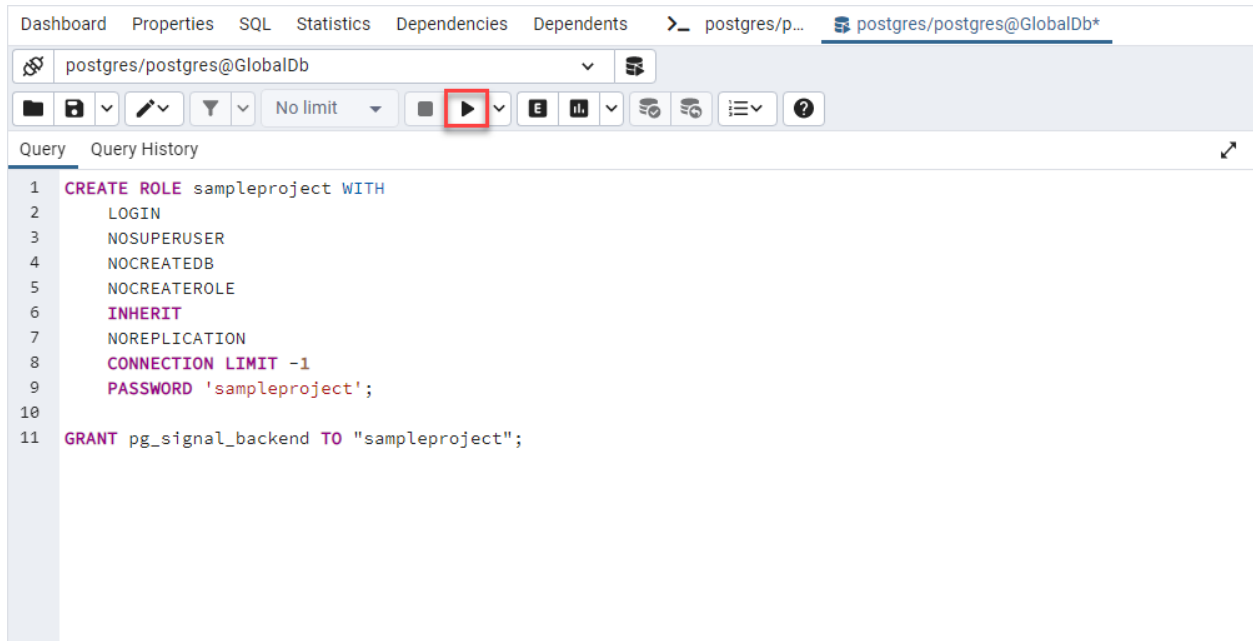
```
CREATE ROLE sampleproject WITH  
  LOGIN  
  NOSUPERUSER  
  NOCREATEDB  
  NOCREATEROLE  
  INHERIT  
  NOREPLICATION  
  CONNECTION LIMIT -1  
  PASSWORD 'sampleproject';
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
GRANT pg_signal_backend TO "sampleproject";
```

2. Для создания пользователя выполним операцию **Execute/refresh**.



Создание БД

1. В окно запроса введите следующий код:

```
CREATE DATABASE "sampleproject"
WITH
  OWNER = "sampleproject"
  ENCODING = 'UTF8'
  CONNECTION LIMIT = -1;
```

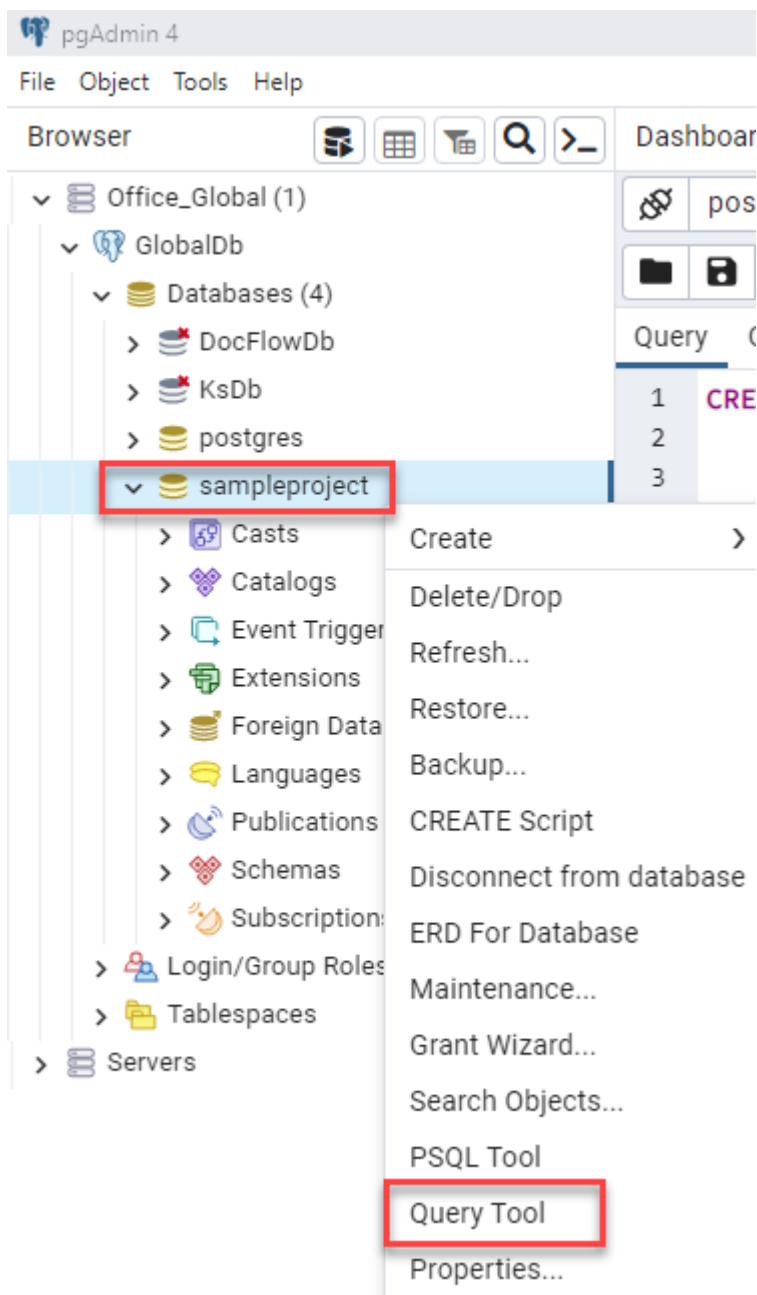
2. Для создания выполните операцию **Execute/refresh**.

Подключение расширений

К созданной БД необходимо подключить расширения, которые использует система Global.

Для этого:

1. Обновите список баз
2. В левой части программы выберите базу данных **sampleproject**
3. Выделите ее и правой кнопкой мыши запустите инструмент **Query Tool**



4. В окно запроса введите следующий код:

```
CREATE EXTENSION if not exists plpgsql;  
CREATE EXTENSION if not exists fuzzystmatch;  
CREATE EXTENSION if not exists pg_trgm;  
CREATE EXTENSION if not exists pg_stat_statements;  
CREATE EXTENSION if not exists "uuid-oss";  
CREATE EXTENSION if not exists dict_xsyn;  
CREATE EXTENSION if not exists ltree;
```

5. Для подключения расширений выполните операцию Execute/refresh.

Настройка рабочего места

Развёртывание рабочего места под Windows

Развёртывание рабочего места под Astra Linux

Развёртывание рабочего места под Alt Linux

Прикладной проект

Прикладной проект собирает конечное решение из набора прикладных модулей. Собранный проект разворачивается на сервере приложения для работы с пользователями.

Структура проекта

- Проект(Application)
 - build.sbt
Настройка сборки проекта
 - project
Дополнительные настройки проекта
 - модуль
 - * src.main.java
 - * src.main.scala
 - ru.bitec.app.[модуль]
Файлы модуля компилируемые языком scala
 - * src.main.resources
 - META-INF\module-info.xml
Мета данные о модуле
 - ru.bitec.app.[модуль]
Файлы модуля не требующие компиляции
 - * build.sbt
Файл сборки модуля
 - модуль_N
 - модуль_N+1

Внимание

`ru.bitec.app.[moduleName]` – это **отдельные** каталоги!

Точка в именах директорий запрещена! IDE IntelliJ Idea в обозревателе проекта группирует каталоги через точку для удобства просмотра

Модуль

Каждый модуль представлен как отдельная директория со строго заданной системой подкаталогов.

Для включения модуля в сборку необходимо объявить его и включить в агрегацию в корневом build.sbt проекта application.

Модуль представляет собой некоторую неделимую совокупность функциональности системы, которая может быть включена в тот или иной прикладной проект.

Каждый объект, зарегистрированный в системе, принадлежит какому-либо модулю.

Основными составляющими модуля являются классы, выборки и серверные методы, группированные в пакеты.

Структура каталогов модуля

[moduleName]

Директория модуля разбита на строгую иерархию:

- [moduleName]/src/main/java/ru/bitec/app/[moduleName]
Содержит java сущности модуля: рою классов, перечисления и т.д. Чаще всего сущности создаются автоматически генератором кода.
- [moduleName]/src/main/resources/META-INF
Содержит метаданные модуля: Описание, версию, зависимости, перечень приложений.
- [moduleName]/src/main/resources/orm
Содержит индексные файлы ORM.
- [moduleName]/src/main/resources/ru/bitec/app/[moduleName]
Содержит файлы моделей Odm, Orm, Dvm, Avm
- [moduleName]/src/main/scala/ru/bitec/app/[moduleName]
Содержит файлы прикладной бизнес-логики и контроллера view, пакетов: Dvi, Dpi, Api, Avi, Pkg
- [moduleName]/src/test/scala/ru/bitec/app/[moduleName]
Содержит юнит-тесты прикладных методов

Зависимости

Модуль может использовать или требовать наличия некоторого набора других модулей. Это требование выражается в том, что объекты или функциональность, заложенная в модуль, не будут работать в случае отсутствия в системе другого, требуемого модуля.

Зависимости позволяют увеличить повторное использование кода, за счет выделения часто используемого функционала в низкоуровневые модули.

Базовые модули:

- gtk
Обеспечивает интеграцию с сервером приложения
- btk
Основная функциональность фреймворка, доступная всем более высокоуровневым модулям.

Эти модули не требуют для своей работы наличия других модулей, являясь платформой системы.

Зависимость модулей описывается в файле настроек сборки build.sbt для модуля.

Подключение внешних библиотек

Допускается подключение к прикладным модулям внешних библиотек (*.jar). В файле build.sbt прикладного проекта необходимо прописать зависимость модуля от этих библиотек.

```
lazy val bts = project.  
  dependsOn(btk).  
  settings(CommonSetting.setting: _*).  
  settings(  
    libraryDependencies += Seq(  
      "org.apache.httpcomponents" % "httpmime" % "4.5.2",  
      "org.apache.httpcomponents" % "httpcore" % "4.4.5",  
      "org.apache.httpcomponents" % "httpclient" % "4.5.2" )  
  )
```

Если подключаемая библиотека опубликована в нестандартном репозитории, необходимо объявить путь к этому репозиторию.

```
lazy val btk = project.  
  dependsOn(gtk).  
  settings(commonSettings: _*).  
  settings(  
    resolvers += "nuiton-maven" at  
      "https://nexus.nuiton.org/nexus/content/groups/releases/",  
    libraryDependencies += "nl.knaw.dans.common" % "dans-dbf-lib" % "1.0.0-beta-10"  
  )
```

Если объявить зависимость с двумя символами «%», в качестве разделителя:

```
"nl.knaw.dans.common" %% "dans-dbf-lib" % "1.0.0-beta-10"
```

к имени библиотеки будет добавляться суффикс с версией scala.

Все библиотеки зависимостей из списка «libraryDependencies» копируются в каталог .commonlib.

За формирование каталога .commonlib отвечают sbt-задания publishLibDependencies и publishDevDependencies, выполняемые в зависимости от режима сборки проекта автоматически или вручную.

Внимание

При выполнении sbt задачи publishDevDependencies в IntelliJ Idea может выдаваться ошибка вида:

```
[error] download error: Caught javax.net.ssl.SSLHandshakeException:  
sun.security.validator.ValidatorException: PKIX path building failed:  
sun.security.provider.certpath.SunCertPathBuilderException: unable to  
find valid certification path to requested target  
(sun.security.validator.ValidatorException: PKIX path building failed:  
sun.security.provider.certpath.SunCertPathBuilderException: unable to  
find valid certification path to requested target) while downloading  
https://ivy.global-system.ru/
```

Ошибка проверки возникает из-за того, что в хранилище доверенных корневых сертификатов Java нет СА-сертификата промежуточного сервера для let's encrypt.

Исправить ошибку можно следующим образом:

1. Добавить сертификат в стандартное хранилище доверенных CA-сертификатов
Для Java приложений сгенерировать новый CA-сертификат R3 для Let's Encrypt можно с помощью утилиты keytool.

Java Keytool — это инструмент командной строки, который может генерировать пары открытый ключ / закрытый ключ и сохранять их в хранилище ключей. Исполняемый файл утилиты распространяется вместе с Java SDK (или JRE)

пример использования:

```
keytool -importcert -file mycertfile.pem -keystore cacerts -alias "Alias"
```

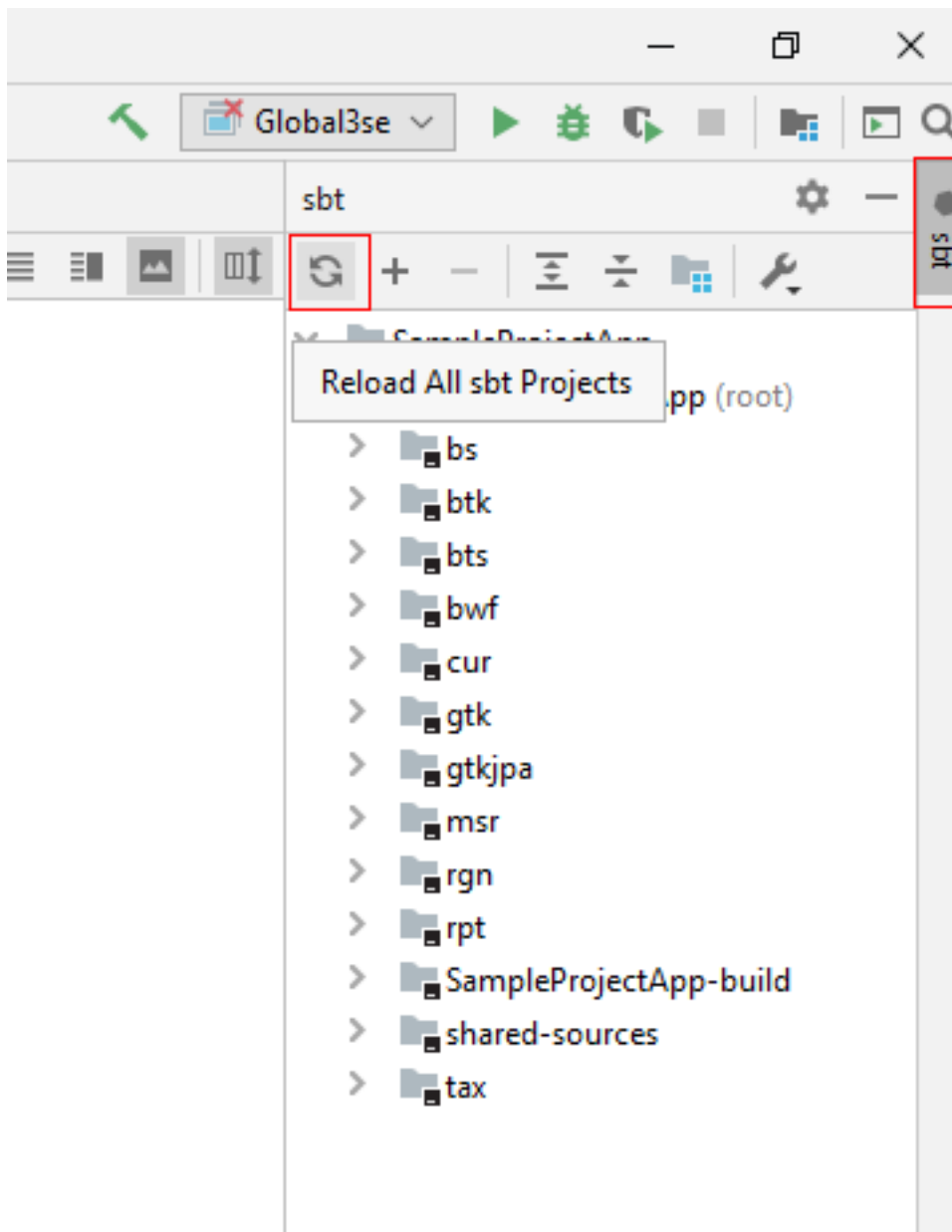
Стандартный пароль хранилища: changeit

2. Заменяем хранилище
C:\Program Files\Java\jdk[версия]\jre\lib\security\cacerts файлом из архива Global3FrameworkStarterKit.zip

Первая сборка проекта

Когда проект открывается впервые, IDE IntelliJ Idea проиндексирует его состав и выполнит обновление sbt проекта. После этого необходимо выполнить агрегацию библиотек проекта и собрать проект.

1. Дождитесь окончания инициализации проекта и индексации исходного кода
2. Обновите sbt проект



3. Выполните агрегацию библиотек проекта

Для этого запустите на панели sbt задачу `publishDevDependencies`

При этом sbt задача просканирует проект на наличие зависимостей и сформирует в корне проекта каталог `.commonlib` со всеми необходимыми библиотеками.

Внимание

При выполнении sbt задачи `publishDevDependencies` в IntelliJ Idea может выдаваться ошибка вида:

```
[error] download error: Caught javax.net.ssl.SSLHandshakeException:  
sun.security.validator.ValidatorException: PKIX path building failed:  
sun.security.provider.certpath.SunCertPathBuilderException: unable to  
find valid certification path to requested target  
(sun.security.validator.ValidatorException: PKIX path building failed:
```

```
sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target) while downloading
https://ivy.global-system.ru/
```

Ошибка проверки возникает из-за того, что в хранилище доверенных корневых сертификатов Java нет CA-сертификата промежуточного сервера для let's encrypt.

Для исправления ошибки загрузите хранилище C:\Program Files\Java\jdk\[версия]\jre\lib\security\cacerts файлом из архива Global3FrameworkStarterKit.zip

4. Выполнить сборку проекта: Build | Build Project

Развертывание первоначальных данных

Для генерации начальных данных в БД проекта необходимо провести успешную сборку проекта и запустить утилиту создания таблиц.

1. Если требуется сборка проекта, выполните команду **Build project**
Пункт меню IntelliJ Idea: **Build > Build Project**
2. После сборки проекта запустите генерацию таблиц
В обозревателе проекта IntelliJ Idea вызовите контекстное меню на корневой директории проекта **application** пункт **External Tools > Generate tables**
При этом происходит создание таблиц схемы, установка начальных данных и создается пользователь **admin** с полными правами(пароль **admin**)

После этого конфигурация готова для разработки.

Запуск приложения

Для запуска и отладки приложения вам необходимо ознакомиться с основными принципами [запуска и отладки в idea](#)

Для отладки проекта:

1. Выберите в выпадающем списке конфигураций запуска **Global 3 SE**
Данная конфигурация создается автоматически дополнительным плагином. Конфигурация появляется при следующем запуске idea. В случае если плагин не установлен, конфигурацию для запуска необходимо создать вручную.
2. Выполните **Debug Global 3 SE**
3. Подключитесь к запущенному серверу
Для этого откройте браузер по адресу **http://localhost:8080/**

При этом в консоли будет выводиться лог работы сервера. При запуске не должно быть сообщений с типом **[ERROR]** или стеков ошибок.

Регистрация

При первом запуске система Global Postgres потребует зарегистрировать БД. Для регистрации базы данных.

1. Сформируйте файл запроса лицензии
Для формирования файла запроса лицензии необходимо нажать на кнопку «Сформировать и скачать файл запроса лицензии».
2. Запросите лицензионный файл
Сформированный файл запроса лицензии вместе с заявкой на регистрацию отправьте по электронной почте контактному лицу ООО «Бизнес-Технологии». В заявке рекомендуется указать имена модулей, которые будут созданы.
3. Зарегистрируйте лицензию
Для этого в диалоге требования регистрации нажмите на кнопку **Установить лицензионный файл**, и выберите полученный файл в открывшемся диалоге.

Отладка приложений

Для отладки прикладного кода:

1. Откройте файл с кодом для отладки
2. Установите точку остановки на требуемой строке
3. Откройте приложение в браузере и выполните отлаживаемое действие
Выполнение программы будет остановлено на отмеченной строке.

Отладка бизнес-логики

Для отладки и тестирования кода Dpi/Api-классов можно воспользоваться Unit-тестами. Для этого, в каталоге с тестами соответствующего модуля создайте класс, унаследованный от `ru.bitec.app.gtk.eclipse.api.ApiTest`.

Смотрите пример: `ru.bitec.app.cur.Cur_CurrencyTest`

Перекомпиляция кода без перезапуска сервера

При отладке приложений, в большинстве случаев, нет необходимости перезапуска веб-сервера при внесении изменений в код любого модуля проекта Application. Для применения изменений необходимо:

1. Выключите кэш
Выполните пункт меню **Файл \ Использовать кэш** в отлаживаемом приложении
2. Скомпилируйте изменённый модуль
Compile module в контекстном меню модуля или **Build project** на панели инструментов. В отдельных случаях можно компилировать отдельный класс.
3. Обновите приложение
В главном меню отлаживаемого приложения выполнить **Файл \ SBT \ Обновить и переоткрыть текущую форму**
4. Повторите отлаживаемое действие.

Отладка выборки

Для отладки пользовательских интерфейсов предусмотрен специальный инструмент `Selection debug window`. Вызвать его можно в любом месте системы с помощью сочетания клавиш `ctrl+alt+shift+W`.

Языки разработки

Основными языками разработки являются

- `scala`
Для программирования автономной и интерактивной бизнес логики.
- `jexl`
Для программирования динамических скриптов.
- `PL/pgSQL`
Для обработки данных в БД

Scala

Язык разработки, используемый для программирования бизнес логики сервера приложения.

- Основы языка
- Гайд по коллекциям
Может быть полезен, так как данные библиотеки очень удобны при манипуляциях со строками, к примеру операции сортировки и фильтрации
- Руководство `scala`: Типы коллекций
- Выбор последовательности

Прикладные разработчики в основном используют процедурную парадигму, что позволяет избежать сложного порога вхождения со стороны разработчиков `java`

При этом `scala` увеличивает производительность программистов за счет:

- Наличие удобной библиотеки коллекций
- Инлайн классов
Позволяет использовать специализированную арифметику при работе с `null` типами, при этом не перегружая сборщик мусора
- Интерполяции строк
Облегчает работу с `sql` запросами
- Трейтов
Позволяют реализовать наследование отображений
- Прозрачного механизма инъекций зависимости

```
//При этом автоматически пробрасывается контекст сессии базы данных.  
//Использование фабрик позволяет прозрачно перекрывать контроллеры бизнес логики  
//в итоговом решении при необходимости  
SomeApi().load(id)
```

- Фабрик контроллеров
За счет того что обращения к любому контроллеру идет не статично, а через фабрику(`SomeApi()`),

возможно динамическое аспектирование(к примеру запись макросов) и проектное расширение кода (подмена базового контроллера его проектным наследником)

Обработка исключений

При обработке исключений нельзя перехватывать системные сообщения. Системные исключения всегда должны быть сброшены наверх. Это в том числе позволяет при необходимости прерывать сессию.

Для упрощения корректной обработки исключений введены вспомогательные функции и классы.

Объявление исключений

Прикладные исключения объявляются только как наследники от класса `AppException` или от потомков класса `AppException`.

При объявлении класса исключения нужно создать класс и фабрику.

```
class ApiException(params:AnyRef) extends AppException(params)
object ApiException extends ExceptionFactory(new ApiException(_))
```

Выбрасывания исключения

Для выбрасывания исключения используется конструкция

```
throw ApiException(args)
```

Внимание

Все прикладные исключения должны создаваться только от фабрики.

Перехват исключений

Перехватываться могут только прикладные исключения, если необходима принудительная очистка ресурсов в любом случае используйте секцию `finally`

Пример:

```
try App{
  println("start")
  throw ApiException("Ups")
}catch {
  case e:AppException =>
    e.raise("Ups2")
}finally{
  println("end")
}
```

Внимание

- Используйте `try app`
Это позволит гарантировано получать `AppException`
- При необходимости продолжить выбрасывания сообщения используйте конструкцию `e.raise`
- При необходимости выбросить новый тип сообщения используйте конструкцию `e.raise(ApiException, "Ups2")`

Данная практика позволит избегать потери стека.

Конвертация java исключений в прикладные исключения

Все прикладные исключения являются наследниками от `AppException`.

При необходимости типизированной обработки java исключений в прикладном коде, необходимо выполнять конвертацию java исключений в прикладные исключения, что делается в функции:

```
ru.bitec.app.gtk.exception.App#apply
```

Jexl скрипты

Для выполнения динамического кода, существует `jexl` расширение.

Возможно выполнить `jexl` скрипт из:

- Ssh консоли

```
login user/password@db
jexl
doSomething()
/
```

- Api

```
JexlScript().eval("doSomething")
```

- Бизнес приложения, если у пользователя есть привелегии администратора
Используйте пункт меню
Сервис > Инструменты > Выполнить `jexl` скрипт

Обработка исключений в jexl

Jexl в ядре не поддерживает исключения, механизм исключений реализован через аннотации

пример:

```
@begin{
  logInfo("ok")
}@exception function(e){
  rollback()
} end
```

Для бросания исключения можно воспользоваться командой `raise`:

```
raise("Текст исключения");
```

Добавленные методы

- `lpad`
Дополнение строки символами слева
- `rpadd`
Дополнение строки символами справа
- `flush()`
Выполняет `session.flush`
- `commit()`
Выполняет `session.commit`
- `rollback()`
Выполняет `session.rollback`
- `raise`
Бросает исключение `AppException` с указанным текстом
- `nvl`
Возвращает первый параметр, если он не null, иначе второй параметр. Поддерживает nullable-типы
- `isNull`
Проверка на null указанного параметра
- `isNotNull`
Проверка, что параметр не null
- `pgArrayToNLongList`
Преобразует массив Long в список NLong

Примечание

Для просмотра списка поддерживаемых методов смотрите: \

```
ru.bitec.app.gtk.jexl.session.JexlScriptContextExtension#ScopeBuilder
```

Работа с датами

Методы работы с датами:

- `sysDate()`
Текущая дата
- `truncDate`
Дата на начало дня от переданной
- `truncYear`
Дата на начало года от переданной

- **toDate**
Переводит строку в дату по указанному формату. Если формат не указан, то используется стандартный. Стандартный формат `dd.MM.yyyy` и `dd.MM.yyyy HH:mm:ss`
- **toString**
Переводит дату в строку по указанному формату. Если формат не указан, то используется стандартный. Стандартный формат `dd.MM.yyyy` и `dd.MM.yyyy HH:mm:ss`
- **+, -**
Используются для изменения даты, операторы добавляют или отнимают от указанной даты количество дней. Например `sysDate() + 1` вернет дату на один день больше текущей

Работа с sql

Для работы с sql используется команды

- **sql(sqlText:String)**
Для запросов на чтение
- **tsql(sqlText:String)**
Для запросов на запись

Основные функции:

- **execute**
Выполняет sql команду
- **asList**
Выполняет запрос, и возвращает список строк
- **asSingle**
Возвращает запись
- **foreach**
Принимает на вход функцию для потоковой обработки запроса

Пример

```
var l= sql("select 1 d").asList()
for(r:l){
    logInfo(r.d)
}
```

Работа в контексте выборки

Контекст выборки (JexlSelScript) – расширяет контекст бизнес-логики возможностью работы с выборками и пользовательским интерфейсом. Используется в тех случаях, когда необходимо получить данные из полей для пользовательского ввода. Пример методов, доступных из jexl-скрипта контекста выборки:

- **varExists**
Проверка наличия переменной в выборке(если не найдено, пойдет поиск в мастер-выборках)
- **selfVarExists**
Проверка наличия переменной только в текущей выборке
- **setVar**
Установить значение переменной(если не найдено, пойдет поиск в мастер-выборках)

- **setSelfVar**
Установить значение переменной только в текущей выборке
- **getVar**
Получить значение переменной текущей выборки(если не найдено, пойдет поиск в мастер-выборках)
- **getSelfVar**
Получить значение переменной только из текущей выборки
- **addVar**
Добавить переменную в выборку с установкой значения
- **newForm**
Создание форм

Пример:

```
//Получаем значение из атрибута "id" выборки, приводим его к типу NSString и пишем в
↪ переменную idTree
var idTree = getVar("id").asNSString()

//Получаем атрибут DGLOBALENDDATE из мастер-выборки
var dEndDate = getVar("super$DGLOBALENDDATE").asNSDate()

//Вызываем метод из пакета с передачей параметров
//Обратите внимание, для обращения к Api или пакетам используются их короткие
↪ имена(без скобок)
var fltCond = Act_UniversalReportPkg.getUniFilterCondByIdTree(idTree, dEndDate)

//Вызывается ещё один метод, возвращающий объект scala-класса immutable.Map[NSString,
↪ Any]
var filters = Act_UniversalReportPkg.getFilterValues(idTree)

/*Определение Map-ы внутри jexl-скрипта. Отметим, что Map внутри jexl и объект scala-
↪ класса
Map (неважно mutable или immutable) - это разные объекты. Наиболее важным
фактом является то, что передать scala-объект Map, полученный в предыдущей строке,
в пакетный метод, принимающий scala-объект Map, в jexl-скрипте напрямую нельзя,
именно поэтому приходится получать значения из переменной filters, перезаписывать
их в jexl-Map param и потом передавать param в scala-метод.
var param = {"flt_idDepOwner" : filters['flt_idDepOwner'],
            "flt_idAcc" : filters['flt_idAcc'],
            "flt_dFrom" : filters['flt_dFrom'],
            "flt_dTo" : filters['flt_dTo'],
            "flt_idAdjustMethod" : filters['flt_idAdjustMethod'],
            "flt_idAccCor" : filters['flt_idAccCor'],
            "uniFilterCondition_dz": fltCond}

*/
//Открываем новую форму с переданными параметрами
Act_TransAvi.defList().newForm().params(param).open()
```

Контекстная справка

В выборке выполнения jexl-скрипта есть возможность открытия выборки контекстной помощи, при нажатии на операцию «Помощь». В этой выборке в левой части поле ввода команд помощи, а права отображает справку.

Если методы справки вызывать из ssh-консоли, то результат будет напечатан в консоль.

Команды

- **help**
Справка по глобальным методам;
- **listObj**
Список доступных Api и Pkg
- **describe**
Описание Api и Pkg

Написание библиотек

Можно получить библиотеку с помощью `lib('название либы')`

Примечание

Посмотреть на существующие или написать свою можно в «Настройка системы» -> «Настройки и сервисы» -> «Сервисы JEXL» -> «Функции JEXL» Класс `Btk_JexlFunctionLib`

Пример либы «test»:

```
// Магическое число 10
let value = () => 10;
/*
 * Складывает числа
 * @param x: Первое число
 * @param y: Второе число
 * @return результат
 */
let func = (x, y) => x + y;
```

Пример вызова:

```
let testLib = lib(`test`);
let ten = testLib.value();
let res = testLib.func(2, 2);
```


PL/pgSQL

Язык обработки бизнес логики на стороне базы данных. Применяется в случае если выполнение логики на сервере приложений не рационально в следствие высокой нагрузки на сеть или диск. Типичными примерами является:

- Выполнение агрегаций
- Построения аналитических запросов
- При фильтрации и сортировки больших данных

Требования к масштабируемости

При проектировании архитектуры приложения необходимо стремиться, чтобы горизонтальный рост системы не приводил к нелинейному увеличению потребностей в ресурсах.

Типичным узким местом в такой ситуации является увеличение запросов в базу данных при увеличении данных, когда код написан таким образом что при обработки бизнес логики на каждую строчку данных идет запрос в базу данных.

Типовыми вариантами сохранения масштабируемости методов являются:

- Работа через объектный кэш с возможностью его предварительной загрузки
При этом кэш должен загружаться **batch** запросами, что значительно уменьшает нагрузку на базу
- Массовая обработка объектов
При этом обработка разбивается на порции, каждая порция данных вытаскивается и сохраняется batch запросами

Общие принципы наименования

Global3-FrameWork является регистро-зависимым. Это означает что имена сущностей, свойств, методов и т.д. нужно указывать строго так, как они объявлены.

Предупреждение

Btk_Class и BTK_Class это разные сущности с точки зрения фреймворка

При написании кода очень важно соблюдать общие правила, это позволяет улучшить взаимодействие между людьми и повысить эффективность.

Общие принципы по работе с кодом:

- Помните! Код чаще читается, чем пишется, поэтому не экономьте на понятности и чистоте кода ради скорости набора.
- Не используйте подчеркивание для отделения слов внутри идентификаторов. Подчеркивание используется только после имени модуля.
- Старайтесь не использовать сокращения лишней раз, помните о тех, кто читает код.
- Старайтесь делать имена идентификаторов как можно короче (но не в ущерб читабельности). Главное, чтобы смысл идентификатора был понятен в используемом контексте.

- Когда придумываете название для нового наименования, старайтесь не использовать имена, потенциально или явно конфликтующие со стандартными идентификаторами.
- Предпочтительно использовать имена, которые ясно и четко описывают предназначение и/или смысл сущности.
- Старайтесь использовать имена с простым написанием. Их легче читать и набирать. Избегайте (в разумных пределах) использования слов с двойными буквами, сложным чередованием согласных. Прежде, чем остановиться в выборе имени, убедитесь, что оно легко пишется и однозначно воспринимается на слух. Если оно с трудом читается, и вы ошибаетесь при его наборе, возможно, стоит выбрать другое.

Конвенция для разработки на scala

Стандартный код именуется в соответствии с [руководством по языку](#)

Сокращения

В основном сокращения используют либо в горячих методах, либо в объектах базы данных, которые имеют ограничения.

Старайтесь не использовать аббревиатуры или неполные слова в идентификаторах, если только они не являются общепринятыми. Например, пишите `getClass`, а не `getCls`. (Исключением являются базовые понятия модуля, на которые имеет смысл придумать аббревиатуры, т.к. изначально сложно предсказать каким количеством коллекций они могут обладать, что может привести к вынужденным сокращениям в дальнейшем)

Не используйте акронимы, если они не общеприняты в области информационных технологий.

Широко распространенные акронимы используйте для замены длинных фраз. Например, UI вместо User Interface или Olap вместо On-line Analytical Processing.

Если имеется идентификатор длиной менее трех букв, являющийся сокращением, то его записывают заглавными буквами. Имена длиннее двух букв записывайте в стиле Паскаль `Xml`, `xmlDocument`.

Если, все-таки, сокращение необходимо из-за ограничений в длине наименования или по другим причинам, старайтесь сокращать наименее информативные части в имени и наиболее часто используемые (легко запомнить). К примеру, в таблице настроек есть поле `idEnergyVolWithLostParamType` - Тип показателя расхода электроэнергии с потерями. Что из этого можно сократить:

`Energy` - так как мы находимся в модуле энерго-учета, это поле обладает не большой информативностью и его можно сократить до `En`, по крайней мере, в контексте энерго-учета, легко вспомнить, что это электроэнергия.

`VolWithLost` — это основной смысл настройки, и его сокращать нельзя ни в коем случае

`ParamType` - тип параметра, так как в таблице настроек лежат в основном аналитики параметра учета, то на них можно придумать аббревиатуры. Название легко будет восстановить, помня о возможных аналитиках параметра учета.

В итоге, получаем сокращение `idEnVolWithLostPT`, с которым намного проще работать, чем с `idEnergyVwltParamType`

Scala типы для работы с данными

Для удобной обработки данных язык Scala расширен null типами. Null типы, добавляют null логику к стандартным java типам, а так же расширяют их функциональность.

Задачи null типов:

1. Определить стандартные операции
2. Исключить Null pointer exception
3. Сделать арифметику более компактной

Для большей наглядности достаточно взглянуть на методы типов, к примеру для NNumber:

```
def round(value: NLong): NNumber = {  
  if (this.isNull.isTrue || value.isNull.isTrue) this else NNumber(this.get.  
    ↪setScale(value.get.intValue(), RoundingMode.HALF_UP))  
}
```

Альтернативой null типам в scala является класс Option, однако в нем отсутствуют специализированные методы под каждый тип.

Пример использования:

```
//Null типы  
val a1 = 1.nn  
val b1 = None.nn  
assert(  
  (a1+b1).isNull  
)  
//Option  
val a2 = Some(1)  
val b2: Option[Int] = None  
assert(  
  (for(a<-a2;b<-b2) yield a+b).isEmpty  
)
```

Любая колонка в таблице обрабатывается соответствующим ей null типом. При этом, происходит перегрузка стандартных операторов языка, создавая, таким образом, dsl расширение.

Для создания констант используются соответствующие фабрики:

NLong

Используется для работы с идентификаторами.

Фабрика: nl

NNumber

Используется для работы с числами

Фабрика: nn

NGid

Используется для работы с глобальными идентификаторами

Фабрика: ng

NDate

Используется для работы с датами.

Фабрика: nd

NString

Используется для работы со строками.

Фабрика: ns

NBigDecimal

Используется для работы с номерами.

Внимание

Обязательно используйте BigDecimal для выполнения расчетов. Использование чисел с двоичной арифметикой, может приводить к ошибкам.

Фабрика: nn

NDuration

Используется для арифметики дат, к примеру:

```
dvNewDate = dvDate + (10.hours+5.mins)
```

Фабрики: seconds, minutes, hours, second, minute, hour

Наименование переменных для nullable типов и атрибутов

Необходимо очень четко проследить, где работа идет с dsl, а где со стандартным типом. Наиболее яркий пример возможной, это операция сравнения.

```
If ( (idvSome1 === idvSome2 ) &&(nvNumber > 0.nn) {  
}
```

В данном примере, если не использовать null логику, операция `nvNumber > 0.nn` выдаст исключение, при `nvNumber` равном `null`. Поэтому, использование null логики требует ввести эквивалент операции сравнения.

Так как, невозможно перегрузить оператор `==`, если он возвращает другой тип или используется **значимый класс**. Создан стандартный оператор `===`.

Внимание

Очень легко перепутать `===` и `==`, поэтому использование расширенных стандартов наименования для null типов обязательно.

Правила наименования переменной:

```
[Variable] [a] [t] [Scope] [Name] [Suffix]
```

где:

- **[Variable]**
Определяет тип
 - **n** - Число
 - **s** - Строка
 - **j** - Строка в формате json
 - **d** - Дата
 - **r,x** - Запись
 - **u,cur** - Курсор
 - **l,blob** - Бинарные данные (bytea, blob)
 - **c** - Символьные данные (text, clob)
 - **b** - Булево значение
 - **id** - Идентификатор
 - **gid** - Глобальный идентификатор
- **[a]**
Определяет является ли переменная последовательностью
- **[t]**
Определяет пользовательский тип
- **[Scope]**
Область действия

- **v** - Переменная процедуры
- **g** - Переменная пакета (Как различать константы)
- **p** - параметр процедуры
- **[Name]**
Имя переменной
- **[Suffix]**
Суффикс
 - **_dz** - Системные атрибуты
 - **_z** - Проектные атрибуты

Примеры задания параметров

- **dStart** - дата начала в таблице
- **dvStart** - дата начала переменной процедуры
- **dgMaxDate** - максимальная дата переменной пакета которую нельзя менять.
- **tvdaDate** - тип коллекции дат объявленный в процедуре
- **davDate** - коллекция дат объявленная в процедуре
- **uvStudents** - курсор объявленный в процедуре.

Наименования Scala-пакетов

Всегда в нижнем регистре.

Правильно:

- `ru.bitec.deepspace`
- `ru.bitec.deep_space`

Не правильно:

- `ru.bitec.deepSpace`

Правила наименования сущностей

Наименование классов

`[Имя модуля]_[Аббревиатура логического пакета][Системное имясущности]`

Имя класса привязано к таблице, поэтому необходимо согласовывать наименование классов с наименованием таблицы, что накладывает ограничение на имена.

Наименование пакетов

По аналогии с классами:

[Имя модуля]_[Аббревиатура логического пакета][Наименование сущности][pkg]

Наименование процедур и функций

Используйте глаголы или комбинацию глагола и существительных и прилагательных для имен методов.

Старайтесь избегать неоднозначных глаголов:

Не правильно:

- `checkItem` - проверить элемент
Абсолютно не сообщает, что произойдет в результате отработки данного метода: будет выдано исключения или возвращен результат.

Правильно:

- `validateItem` - гарантирует корректность элемента, в случае несоответствия - генерирует исключение
- `isItemValid` - возвращает истину если элемент корректен
- `itemErrCode` - Возвращает код ошибки
- `deleteItem` - удалить элемент
- `createOrder` - создать заказ

Наименование параметра

Из имени и типа параметра должны быть понятны его назначение и смысл.

Не усложняйте методы параметрами, которые, возможно, будут использоваться в будущих версиях.

Если в будущем понадобится новый параметр, можно использовать перегрузку методов и значения по умолчанию.

Наименование временных таблиц

[Имя модуля]_[Аббревиатура логического пакета][Системное имя][gtd]

Наименование выборок

```
[Имя модуля]_[Аббревиатура логического пакета][Системное имя]
```

Данный способ наименования позволяет более легко выполнять рефакторинг при перемещении выборки между пакетами.

Наименование отображений

```
[Системное имя]_[группа зависимости]
```

Где:

- [группа зависимости]
Опциональный параметр, если данные выборки имеют внешние зависимости

Примеры зависимостей:

- idDoc
Выборка отображает детализацию по документу
- TurnOver
Детализация к оборотной ведомости

Не следует называть группу зависимости в несоответствии с тем, что реально происходит, так как это вводит в заблуждения, к примеру, название idDoc говорит о том, что можно использовать эту детализацию в любых выборках для детализации документа, однако, если запрос заточен под конкретную выборку мастера, это невозможно.

Формат комментария в систему контроля версий

```
format ::= (attention | label)? text
attention ::= add | rem | fix
label ::= (feat| test| doc | err ..)*
text ::= Краткий текст комментария. dp (при наличии)
dp ::= #id - Документ поддержки
```

Стандартные сокращения

Odm (Object domain model)

Доменная модель сущности. Используется для описания метаданных классов фреймворка: Обозначение, наименование, тип класса, атрибутивный состав, ссылаемость и т.д.

Orm (Object-Relational Mapper)

Объектно-реляционное представление сущности. Сопоставляет объектную доменную модель и реляционную базу данных. Используется в EclipseLink

Pojo

Java объект на сервере приложения. Хранит значения атрибутов объекта класса (строки таблицы БД) в оперативной памяти.

Dpi

Доменная автономная бизнес логика (создается кода генератором). Обеспечивает для сущности базовую обвязку методами фреймворка.

Dvi

Доменная интерактивная бизнес логика (создается кода генератором). Обеспечивает для сущности базовые методы работы с пользовательским интерфейсом.

Api

Прикладная автономная бизнес-логика по классу. Содержит методы прикладной обработки данных для конкретной сущности.

Pkg

Автономная бизнес логика. Содержит методы прикладной обработки данных. В отличии от api не привязана к какой либо конкретной сущности.

Avi

Прикладная интерактивная бизнес логика. Содержит методы работы с пользовательским интерфейсом.

Dvm (Domain view markup)

Доменная разметка выборки (создается автоматически). Шаблон базового представления сущности в системе.

Avm (Application view markup)

Прикладная разметка выборки. Описывает базовое представление сущности в системе.

Ata (Application table)

Прикладная таблица. Определение таблицы на языке Scala. Что позволяет использовать подсказчик кода, и статическую проверку при написании объектных запросов.

Aro (Application row)

Прикладная строка. Определяет правила взаимодействия с сущностью (pojo) в EclipseLink. Предоставляет адаптированный для фреймворка способ взаимодействия с объектами EclipseLink.

Rop (Row provider)

Провайдер строки. Используется для гарантированного доступа к строке (Aro) через кэш. При переходе к редактору гор проверяет наличие строки в кэше и если ее нет, загружает строку из базы.

Сессия приложения

Сессия приложения создается на поток прикладной бизнес логики и предоставляет доступ к сессии базы данных, EclipseLink кэшу, серверу приложения. А так же содержит необходимые инъекции зависимости для работы прикладной бизнес логики.

Сессия приложения создается:

- на каждую mdi форму открытую в приложении
- на rest запрос к серверу приложения

Примечание

Для ускорение rest запросов возможна настройка пула сессий

Сессия приложения не привязана к сессии базы данных. Сессия базы данных выделяется на запрос либо на транзакцию.

Каждый контроллер прикладной бизнес логики содержит контекст сессии который представляет из себя фасад для быстрого доступа к сессии приложения и стандартным языковым примитивам фреймворка.

Сессия базы данных

Физическое соединение с базой.

Фреймворк использует механизм коротких транзакций в БД. Сессия БД выделяется для получения или изменения данных и возвращается в пул после окончания работы с данными.

Число прикладных сессий не равно числу соединений с БД.

Сессия базы данных выделяется из пула при:

- Начале транзакции
- Выполнении запроса из прикладного кода.

Рабочее пространство (workspace)

Рабочее пространство содержит транзакционный кэш чтения, а так же перечень изменений для отправки в базу данных.

Рабочее пространство создается при любом действии с базой данных. Чтение, создание\удаление\обновление. И удаляется при commit или rollback.

Точки сохранения

При создании точки сохранения, происходит сброс всех изменений из рабочего пространства в базу данных. При откате к точке сохранения, рабочее пространство удаляется.

Пользовательский сеанс

Пользовательский сеанс начинается в момент открытия формы в новой сессии, и заканчивается при закрытии формы.

Примечание

При открытии модального окна в той же сессии, пользовательские блокировки работают в сеансе дочерней формы.

Пользовательские сеансы можно просмотреть: Меню приложения > помощь > блокировки

Пользовательская блокировка

Пользовательская блокировка позволяет блокировать бизнес-объект вне транзакции базы данных. Такая блокировка работает в разрезе пользовательского сеанса. Элементы заблокированного бизнес-объекта не могут быть изменены каким-либо другим пользовательским сеансом.

Пользовательскую блокировку можно снять, при этом все заблокированные сессией объекты становятся доступны для блокирования.

Подробнее можно почитать [здесь](#).

Логические блокировки

Логические блокировки являются транзакционными, и снимаются на commit или rollback.

Для получения блокировки используется метод `Btk_LogicLockPkg.request()`, в который передается имя блокировки.

Сессия при получении блокировки будет ожидать фиксацию или откат данных другой сессии, которая уже получила эту блокировку до этого.

Оптимистические блокировки

По умолчанию класс использует оптимистическую блокировку, для этого в таблице создается служебное поле `nVersion_dz`. В случае, если версия загруженного в транзакционный кэш перед изменением объекта отличается от версии в строке таблицы базы данных при сохранении изменений, возникает ошибка сохранения.

Преимущества оптимистической блокировки:

- Не требует начало транзакции базы данных

Недостатки оптимистической блокировки:

- Ошибка проявляется только при сохранении изменений в базу
В момент возникновения ошибки, сложнее отловить ее источник.
- Возможно появления лишней нагрузки на базу данных
От момента правки строки, до момента возникновения ошибки может пройти существенный набор действий который в итоге будет откачен.

Пессимистическая блокировка

Транзакционная блокировка с помощью которой можно заблокировать строку базы данных. При необходимости данная блокировка может быть выполнена с помощью:

- *ATSql запроса*

Совет

Смотрите [Явные блокировки](#)

- *OQuery* с опцией `forUpdate`
- `Btk_Pkg().lockObjects`

Преимущества пессимистической блокировки

- Конфликт определяется в момент выполнения блокировки

Недостатки пессимистической блокировки:

- Требуется транзакция базы данных

Бизнес логика

На каждый контроллер бизнес логики создается объект в контексте сессии приложения. Таким образом бизнес логика работает в контексте сессии что позволяет:

- Хранить переменные объекта в разрезе сессии
- Выполнять объектные запросы в базу данных с учетом сессионного кэша
- Работать в контексте транзакции сессии

Ключевые стратегии создания масштабируемого решения

1. Минимизируйте количество запросов к базе данных

Для этого используйте:

- bulk загрузку кэша
- транзакционные индексы
- разделяемые классы
- кэширование запросов для разделяемых классов

2. Ограничивайте размер транзакции

Для этого:

- разбивайте бизнес логику на части
К примеру разбейте массив бизнес объектов на порции для обработки.
- Используйте `commitwork`
Это позволяет сбросить данные в базу без завершения транзакции, при этом происходит очистка оперативной памяти.

Api

Содержит бизнес логику по обработке данных в таблице класса(сущности). Каждому классу соответствуют два файла с кодом бизнес логики данного класса:

- `Xxx_XxxxDpi`
Содержит авто формируемый прикладной код. Данный код пересоздается системой формирования кода и не должен меняться разработчиками
- `Xxx_XxxxApi`
Содержит прикладной код, изменяемый разработчиками

Пример API

```
class Bs_GdsCostDeviationTypeApi extends Bs_GdsCostDeviationTypeDpi[Bs_
↳GdsCostDeviationTypeAro, Bs_GdsCostDeviationTypeApi, Bs_GdsCostDeviationTypeAta] {
  override protected def entityAta: Bs_GdsCostDeviationTypeAta = Bs_
↳GdsCostDeviationTypeAta

  def register(spCode: NString
               , spCaption: NString
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        , spDescription: NSString): Unit = {
    val ropSelf = get(findByMnemonicCode(spCode)).getOrElse {
        insert() :// { rop =>
            setsCode(rop, spCode)

            rop
        }
    }

    setsCaption(ropSelf, spCaption)
    setsDescription(ropSelf, spDescription)
}
```

Pkg

Пакеты создаются разработчиками по необходимости. Пакеты содержат методы бизнес-логики, которые не возможно отнести к какому-либо классу.

```
class Xxx_XxxxPkg extends Pkg{}
object Xxx_XxxxPkg extends PkgFactory[Xxx_XxxxPkg]
```

Взаимодействие с базой данных

Объектные запросы

Кроссплатформенные запросы, которые выполняются на уровне объектов класса. При выполнении запроса идет обращение к базе данных, если по классу не настроено кэширование запросов.

Пример запроса

```
new OQuery(Bs_GoodsAta.Type) {
    where(t.sSystemName === spMnemonicCode)
}
```

OQuery предоставляет подмножество JPQL и используется для выбора данных по классу объектный кэш.

Методы

- **where**
Условие запроса.
- **orderBy**
Выражение для сортировки результата
- **batchAll**
Массовая загрузка объектов. Возвращает строки с прогруженными записями всех коллекций этого класса.

- **batchIn**
Выполнение запроса с указанным перечнем коллекций, в которых требуется загрузить данные.
- **forUpdate**
Выполнение запроса с блокированием вернувшихся записей
- **forUpdateNowait**
Выполнение запроса с блокированием вернувшихся записей, без ожидания разблокирования, если записи уже заблокированы другой сессией.
- **tryCacheQueryResults**
Попытаться закешировать результат запроса. Смотри пункт «Кеширование»
- **unique**
Говорит, что запрос возвращает одну уникальную запись. Позволяет использовать cache-index'ы, указанные в orm класса

Объектный запрос с помощью выражений **EclipseLink**

Функционал, расширяющий использование объектных запросов, который позволяет запрашивать данные из неизвестных классов (когда заранее неизвестен тип Ata). Класс: `ru.bitec.app.gtk.eclipse.query.ElExpOQuery`

Пример:

```
new ElExpOQuery(api).forWhere(
    _.get("sSystemName").equal("test".ns.get)
).addOrdering(
    _.get("sCaption")
)
```

смотри примеры использования: `ru.bitec.app.btk.ElExpOQueryTest`

Объектный запрос большого списка

```
in.ru.bitec.app.btk.Btk_QueryPkg#largeInQuery
```

смотри примеры использования: `ru.bitec.app.btk.Btk_QueryPkgTest`

Кеширование объектных запросов

Кеширование запросов работает только для классов с разделяемым режимом кеширования(**Shared**).

Кэширование по полю

Кэширование через `cache-index`'ы указанные в `ogm` классе. Такой запрос возвращает одну строку и дополняется командой `unique()`. Например для атрибутов мнемокода класса в `ogm` формируется запись:

```
<cache-index>
  <column-name>SSYSTEMNAME</column-name>
</cache-index>
```

Запрос выглядит следующим образом:

```
new OQuery(entityAta.Type) {
    unique()
    where(t.sSystemName=== spMnemonicCode)
}
```

Кэширование объектных запросов

Кэширование объектных запросов возможно по требованию, в случае, если класс настроен для сохранения в разделяемом кэше.

Чтобы включить кэширование запроса, добавьте в запрос опцию `tryCacheQueryResults()`. Результат такого запроса будет кэширован, если транзакция не находится в режиме редактирования разделяемых объектов.

Пример запроса:

```
new OQuery(entityAta.Type) {
    tryCacheQueryResults ()
    where(t.sSystemName === spMnemonicCode)
}
```

Внимание

Объектный запрос не отображает удаленные строки из текущего рабочего пространства, однако он не способен отслеживать изменения в **where** условии. Если в текущем рабочем пространстве был изменен мнемокод, то объектный запрос вернет данные без учета этих изменений.

Для гарантированного получения согласованных изменений в объектном запросе необходимо вызывать перед ним **flush**.

Транзакционный индекс

Позволяет получить, согласованный с изменениями, перечень строк из базы данных по значению индексируемого атрибута.

Индекс подгружает данные из базы данных по мере обращения к ключам индекса, а так же отслеживает изменения в текущем рабочем пространстве, для получения согласованного набора строк. Это позволяет получить актуальный список строк и значение их атрибутов, даже когда изменения в кэше не сброшены в базу данных.

Примечание

В отличие от объектного запроса, транзакционный индекс видит любые изменения в рабочем пространстве по индексируемому полю.

Пример объявления:

```
lazy val idxidParent = TxIndex(Btk_GroupAta.Type)(_idParentGroup)
```

Методы

- **byKey**
Возвращает итератор по ключу индекса.
- **refreshByKey**
Возвращает итератор по ключу индекса с обновлением из базы данных.
- **queryKeys**
Кеширование ключей индекса.
- **forPartition**
Открывает секцию для массового обновления индекса. Используется для прозрачного массового обновления после очистки транзакционного кэша. Секции могут быть вложенными друг в друга, в таком случае ключи суммируются.

Реляционные запросы

Для обработки реляционных запросов в основном используются методы на базе `anorm`.

Для более удобного использования в контекст бизнес логики добавлены дополнительные функции.

Внимание

Реляционные запросы полностью игнорируют изменения в рабочем пространстве. Если требуется выполнить согласованный запрос, используйте метод `session.flush()`

ASQL

Выполнение запроса на чтение

ATSQL

Выполнение запроса с изменением данных или блокировками

ASelect

Выполнение запросов на чтение\запись с большим кол-вом колонок.

Пример:

```
for (rv <- new ASelect {
  val nParentLevel = asInt("nParentLevel")
  val gidParent = asString("gidParent")
  val gidChild = asString("gidChild")
  val idParent = asLong("idParent")
  SQL"""
  select nParentLevel,gidParent,gidChild,idParent,idChild from table
  """)
}) {
  println(rv.nParentLevel())
  //запрос поля без его предварительного объявления
  println(rv.get("idChild").asNLong())
}
```

Типичные ошибки

Некорректное использование запросов на чтение

Внимание

Даже если такой код работает в тестовом формате, это будет приводить к ошибкам в продакшене.

Пример неверного запроса:

```
ASQL"select * from table where id=$id for no key update".execute()
```

Данный запрос не регистрирует транзакцию, он может быть выполнен в режиме автокоммита, что фактически делает его бессмысленным, так как блокировка снимается сразу же после выполнения запроса.

```
ASQL"update table set a=$v where id=$c".execute()
```

Данный запрос может быть выполнен в текущей транзакции, а может быть не выполнен в транзакции на чтение, поэтому становится абсолютно непредсказуемое поведение на откат, и контроль ссылочной. Что легко может приводить к ошибкам в бизнес-логике.

Внимание

Внимание, все запросы на изменения данных должны быть транзакционными:

```
ATSQL"select * from table where id=$id for no key update".execute()
```

```
ATSQL"update table set a=$v where id=$c".execute()
```

Если вам необходимо выполнить запрос в отдельной транзакции, всегда создавайте атомарные транзакции. Если в вашем тестовом случае транзакция не начата, это не значит что она не может быть начата в другом контексте.

Некорректное использование блокировок

```
ATSQL"select * from reference where id=$id for update".execute()
```

Данная блокировка заблокирует работу всех документов по справочнику, так как блокируется не только запись, но и все внешние ключи.

Всегда пишите `for no key update`, если у вас нет четкого обоснования необходимости иной блокировки.

Некорректное формирование запроса

Будьте осторожнее с подставлением значений в строку (интерполяции) при формировании sql запроса.

```
ATSQL(s"select * from reference where gid=$gid for update")
ASQL(s"select * from reference where gid=$gid for update")
new ASelect {
  //...
  SQL(s"select * from reference where gid=$gid for update")
}
```

В данном примере значение `gid` неправильно подставится в sql выражение, т.к. технически подстановка идет в обычную строку. Никто не догадается правильно конвертировать данные в корректную для sql форму

```
ATSQL(s"select * from reference where gid={gid} for update").on(Symbol("gid") -> gid)
ASQL(s"select * from reference where gid={gid} for update").on(Symbol("gid") -> gid)
new ASelect {
  //...
  SQL(s"select * from reference where gid={gid} for update")
    on(Symbol("gid") -> gid/*, другие параметры*/)
}
```

Для случаев, когда вам нужно собрать запрос из обычной строки, используйте замену символов `on(Symbol("символ для замены") -> ваша переменная, ...)`

```
ATSQL"select * from reference where gid=$gid for update"
ASQL"select * from reference where gid=$gid for update"
new ASelect {
  //...
  SQL"select * from reference where gid=$gid for update"
}
```

Тут будет подставлено все правильно

Работа с большими данными

Postgresql не поддерживает `bulk insert\select\update`, более того, коммит закрывает любой открытый курсор. Это делает невозможной обработку больших объемов данных частями на сервере, так как очень длинные транзакции перегружают `wal` и в конечном счете сильно снижают общую производительность сервера. Поэтому необходимо выполнять партиционирование на сервере приложения.

Замеры производительности показывают, что наиболее производительным средством переноса данных является `sql` команда postgresql «`copy`».

Сравнительная таблица производительности:

Операция 100000 записей	Приблизительное время(ms) на сервере
Вставка записей одним <code>sql</code> в базе	214
Общее время запроса\вставки через <code>copy</code>	728
Вставка записей одним <code>sql</code> в базе при наличии индексов	2785
Общее время вставки через массовый <code>flush</code> при наличии индексов из сервера в базу	19562
Общее время вставки через <code>flush</code> по одной записи при наличии индексов из сервера в базу	156710
Запрос данных через <code>ResultSet</code>	396
Запрос данных через команду <code>copy</code>	330

Примечание

На десктопе разница между вставкой через `copy` и вставкой прямым `sql` может достигать различий в 10 раз. Финальные тесты производительности нужно замерять исключительно на серверном оборудовании.

Вставка через `flush` по одной записи может работать в 1000 раз медленнее, особенно это актуально для данных с аудитом.

Вставка через `batch` на чистом `sql` сравнима по порядку с `copy`, хоть и медленнее.

Массовая вставка на чистом `sql` должна быть раза в 4 быстрее за счет отсутствия затрат на кэш.

Таким образом наиболее высокую производительность при массовой работе с данными могут обеспечить следующие алгоритмы:

Массовая вставка

Через команду сору

Массовый update\delete

- Сохранение данных в файл через команду сору
- Для каждой пачки
 - Начало транзакции
 - Копирование данных в gtt
 - Выполнение dml операции
 - Завершение транзакции

Внимание

Транзакция не должна завершаться до конца обработки пачки, иначе это приведет к потере данных в gtt(при завершении транзакции сессия отпускается)

Размер транзакции

Постгресс по своей архитектуре менее зависим от длинны транзакции (смотри [наполнение базы данных](#)). Таким образом нет потребности в дроблении на мелкие транзакции.

Слишком длинные транзакции негативно влияют на базу, происходят следующие негативные эффекты:

- блокировка данных
- блокировка сборщика мусора, что может привести к падению производительности запросов на dead row и к распуханию файлов

Таким образом на сервисных операциях, если они не превышают несколько гигабайтов(зависит от базы), разбиением транзакций можно пренебречь.

Желательно, чтобы оперативные транзакции не занимали больше 10 минут (зависит от частоты транзакций и нагрузки на базу) и не вызывали блокировок.

13.2 Классы

Класс

Определяет правила хранения и обработки таблицы базы данных.

Класс позволяет существенно ускорить разработку бизнес логики ориентированную на работу с данными. Программисту достаточно объявить перечень атрибутов класса чтобы за счет кодо-генерации получить набор готовых сервисов.

Перечень генерируемых элементов:

- *Доменная автономная бизнес логика(Dpi)*
Содержит код для автономной бизнес логики

- *Каркас прикладной автономной логики*(Api)
scala класс с окончанием Api, в котором пишется автономная бизнес логика для работы с классом. Наследуется от Dpi
- *Доменная интерактивная бизнес логика*(Dvi)
- *Каркас прикладной интерактивной логики*(Avi)
scala класс с окончанием Avi, в котором пишется интерактивная бизнес логика. Наследуется от Dvi
- *Доменная разметка выборки*(dvm.xml)
Содержит сгенерированную по умолчанию разметку выборки.
- *Каркас прикладной декларации пользовательского интерфейса*(Avm)
xml файл с расширением avm.xml, в котором пишется разметка выборки
- Интеграция с Orm
 - Rojo объект для хранения данных в кэше
 - Aro объект интеграции rojo в фреймворк

Примечание

При кодо-генерации обычно создаются два элемента разных типов:

- D - Domain
Доменный элемент всегда перезаписывается при кодо-генерации и содержит бизнес логику для подключения сервисов.
- A - Application
Прикладной элемент не изменяется при кодо-генерации и служит для хранения бизнес логики написанной программистом вручную. Прикладной элемент наследуется от доменного элемента.

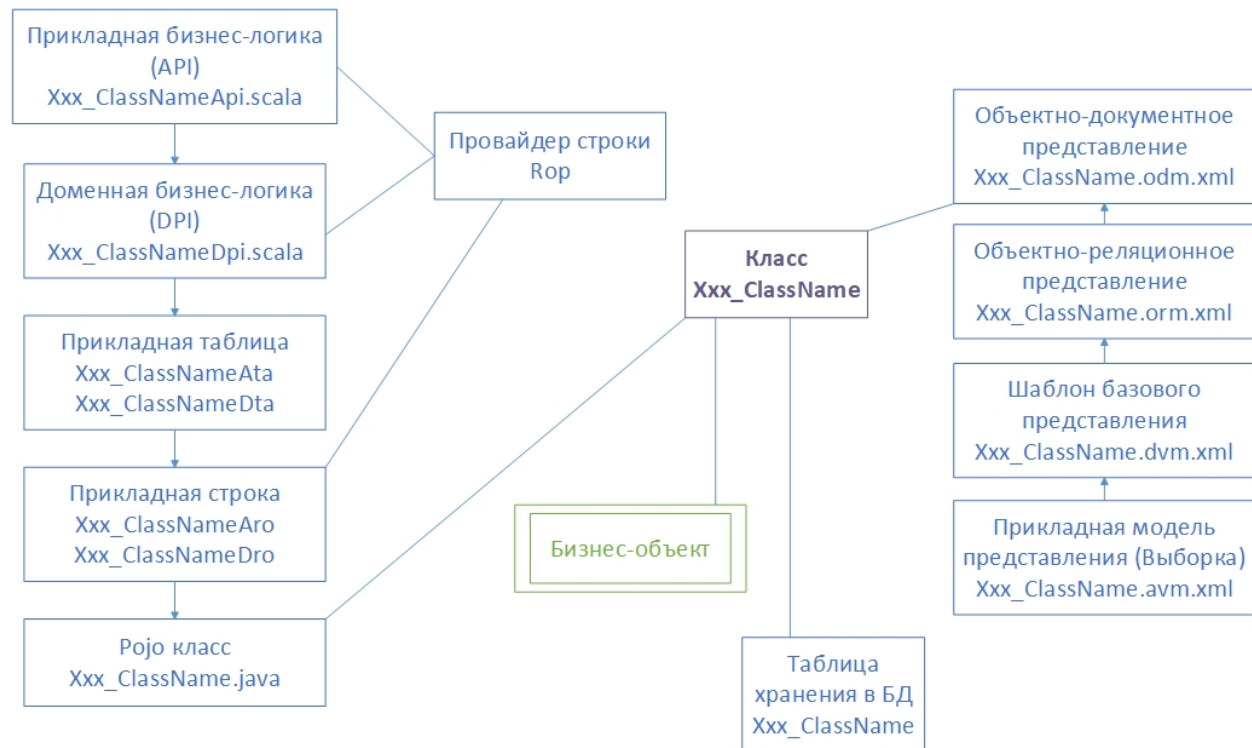
Доступные прикладные сервисы:

- Аудит
Аудит предназначен для фиксации различных событий при работе пользователей в системе
- Администрирование
Позволяет управлять доступом к автономной и интерактивной бизнес логике за счет выдачи прав на привилегии.
- Универсальная фильтрация
Позволяет пользователю строить комплексную фильтрацию списка классов на уровне базы данных. В универсальном фильтре можно использовать как поля самого класса и его коллекций так и поля классов на которые есть ссылки.
- Авто нумерация
Механизм выдачи номеров объектам класса, данный механизм позволяет переопределять алгоритм выдачи номеров на проекте.
- Копирование объектов
Коды генерация бизнес логики для копирования объектов
- Группировка
Группировка используется для систематизации хранения объектов и удобства восприятия пользователем, так же группировка позволяет массово управлять характеристиками и настройками объектов класса.

- Сервис прикрепленных файлов
Позволяет прикреплять к объектам класса произвольные файлы
- Поиск по шаблону
Возможность поиска объектов класса по частичному или полному совпадению введенного текста со значениями полей объекта или его заголовка и мнемокода
- Объектные характеристики
Возможность добавлять произвольные поля в класс на проекте
- Генерация штрих-кодов объекта
Механизм генерации штрих кодов объектам класса при создании
- Подписи объектов для печати
Используется при печати отчетов. Позволяет формировать в печатной форме список лиц с местом для подписи
- Полнотекстовый поиск
Возможность класса осуществлять быстрый поиск по значению атрибутов класса

Схема окружения

Окружение класса создается в момент кода генерации:



Общие сведения о классах

Класс сущности(далее просто класс) определяет хранилище совокупности объектов(строк), имеющих одинаковые характеристики, подчиняющихся общим настройкам и операциям, функционирующих в рамках единой логики.

Класс содержит набор атрибутов, атрибут может быть:

- Простым значением
Используется для ввода и хранения значения определенного типа данных
- Ссылочным
Используется для выбора значения атрибута из справочника или другого множества
- Переменной ссылочности
Используется для ссылочности на любую таблицу в системе
- Ссылочным на класс
Таблица классов является служебной таблицей мета данных хранящей весь перечень классов в решении.

Класс должен иметь уникальное системное имя и наименование.

Правила наименования класса:

- Системное имя должно задаваться на латинице
- Имя должно быть в формате {Module}_{Name} где:
 - Module - Имя модуля
 - Name - Имя классаИмя класса должно быть в единственном числе, именительном падеже

Пример:

- Lbr_Book

Типы данных

Система имеет специализированный набор простых и объектных типов для удобной обработки данных. Объектные типы могут быть интегрированы в контекст сессии, что позволяет обеспечить высокую производительность системы за счет минимизации операций сериализации\десериализации.

Основные типы данных, используемые для атрибутов класса:

- Целое число
- Дробное число
- Строка
- Дата
- Ссылка (на объект заданного класса)
- Ссылка на класс
- Переменная ссылка (на объект произвольного класса)
- Глобальный идентификатор gid
- Json контейнер

Простые типы

Простыми типами являются: Число, строка, дата

Тип	postgresql	odm	Рекомендация по использованию
Строка фиксированной длины	varchar	Varchar	Текст до 4000 символов
Строка переменной длины	text	Text	Текст до 15 мегабайт
Число	number	Number	Целое или дробное число
Дата	date	Date	Дата, дата и время

Ссылочные типы

Данные классов могут быть связаны между собой. Для организации связей между классами используются специальные типы.

Ссылка и переменная ссылка

Ссылка на объект и переменная ссылка используются для организации ссылочности объектов одного класса на объекты другого класса (или объекты своего собственного класса).

Тип **ссылка** может хранить объекты одного класса, который задан в настройках.

Тип **переменная ссылка** хранит ссылку на объект любого класса.

Ссылка на класс

Хранит ссылку на класс. Обычно используется совместно с типом **переменная ссылка** для хранения класса, объект которого содержится в переменной ссылке.

Глобальный идентификатор gid

Gid является уникальным идентификатором в рамках системы. Переменная ссылка на gid является альтернативой системе переменной ссылочности из двух атрибутов (ссылка на класс + переменная ссылка на объект). Для организации переменной ссылки через gid используется один атрибут.

Формат ссылки gid

Глобальный идентификатор состоит из строки:

```
gid := idClass \ id[@ idNode]
```

Где:

- idClass – идентификатор класса
Ссылка на объект в таблице btk_class
- id – идентификатор

- **idNode** – идентификатор удаленного нода
В текущей базе, задается для объекта созданного на удаленном узле (если используется механизм репликации на уровне системы)

Json контейнер

Json контейнер – это расширение объекта класса NoSQL нотацией в реляционной СУБД. Контейнер не имеет жесткой, заранее определенной схемы и основан на множестве пар «ключ-значение». Это позволяет использовать его как динамическое расширение объекта. Для добавления новых данных в контейнер не требуется перекомпиляция кода или изменение структуры СУБД.

Супертипы классов

Для проектирования существует несколько разновидностей класса, с помощью которых можно реализовать модель с необходимой бизнес-логикой. Эти разновидности класса называют супертипами.

Супертипы задают стандартное поведение для класса, а также задают стандартную обвязку методами.

Основные супертипы:

- **reference** - Справочник
Справочник — это прикладной объект, который позволяет хранить данные, имеющие одинаковую структуру и списочный характер. Пример: Справочник физ. Лиц; Места хранения; справочник ТМЦ.
- **document** - Документ
Документ – это прикладной объект, который хранит данные о событиях или операциях на предприятии. Пример: Заявка на отгрузку; Приходная накладная; Акт сверки. Документ обычно имеет атрибут состояние, который отражает его жизненный цикл.
- **collection** - Коллекция
Коллекции представляют собой классы, объекты которых не имеют права на самостоятельное существование и могут быть созданы только для объектов других классов. Коллекции применяются в качестве табличных частей документов или логических развязок между классами. Добавление коллекций в бизнес объект позволяет массово загружать данные в объектный кэш, что минимизирует нагрузку на базу данных. Так же возможно обход элементов коллекции по родителю без транзакционного индекса, что уменьшает нагрузку на процессор.
- **vcollection** – Переменная коллекция
Переменная коллекция расширяет возможности обычных коллекций и может ссылаться на родителя переменной ссылкой. Это требуется, когда для нескольких классов используется одинаковая коллекция.
- **journal** - Журнал
Журнал – это особый тип классов, приспособленный для хранения большого количества записей. Такие классы имеют ограниченную функциональную обвязку ядовыми методами фреймворка. Это позволяет увеличить быстродействие при работе с журналом. Примеры: записи по потребности ТМЦ на заказ в разрезе документов; журнал трудоемкости в разрезе операций и т.д.
- **trait** – Трейт
Абстрактный класс-предок, не имеющий собственной структуры хранения. Такой класс содержит общую логику нескольких классов-потомков и является частью механизма повторного использования кода.
- **mixin** – Миксин (класс-примесь)
Миксин – это особый вид классов, которые служат для хранения данных из разных классов.

Используются для построения общих списочных форм различных диалогов подбора в пользовательских интерфейсах, а также для удобства обработки данных в прикладной бизнес-логике. Миксин позволяет объединить несколько разных таблиц вместе что дает возможность использовать внешние ключи и индексы на данное объединение.

Бизнес-объект

Бизнес-объект (БО) - объединение нескольких классов и их коллекций в группу для более удобного манипулирования ими при работе с кэшем и конфигурировании вспомогательных сервисов.

Бизнес объект позволяет:

- Массово загружать данные в транзакционный кэш
Для бизнес объекта можно указать стратегию загрузки данных, существенно уменьшающую количество запросов в базу данных, так как запросы пойдут не по каждому объекту а по каждому классу бизнес объекта.
- Настраивать права доступа
По бизнес объекту создается административный объект на котором можно массово выдать привилегии для всех классов бизнес объекта
- Управлять электронной подписью
Можно настроить правила подписи всего бизнес объекта включая не только шапку, но и все вложенные коллекции.
- Настраивать интеграцию и репликацию

Навигация по бизнес-объекту

Навигацией является последовательное посещение элементов бизнес-объекта сверху вниз. При навигации перемещение между объектами идет по кэшу, при этом обеспечивается автоматическая дозагрузка объектов в кэш по необходимости.

В процессе навигации объекты не блокируются и могут при необходимости быть вытолкнуты из кэша, что вызовет автоматическую дозагрузку (обновление).

Примеры навигации:

```
val empApi = EmployeeApi()
empApi.load(7452) :/ { id =>
    println(id.id)
    for(idDes <- AddressApi.byParent(id)){
        println(s"address: city=${ idDes.get(_.city)}")
    }
}
```

Навигация в рабочем пространстве

Объекты загружаются из базы или из кросс-сессионного кэша, при загрузке происходит пессимистическая либо оптимистическая блокировка. Объекты находятся в рабочем пространстве до момента коммита. В момент коммита рабочее пространство очищается.

При навигации можно производить модификацию объектов. Api гарантирует корректную навигацию по мастер деталям без необходимости flush и clean кэша или немедленной загрузки коллекций.

Массовая загрузка объектов.

При массовой загрузке объектов происходит минимизация обращений к базе данных. При обходе в обычном режиме 3-х уровневой бизнес-объекта произойдет $n+2$ запросов, где n количество деталей 2-го уровня, 1 запрос, на запрос мастер объекта, 2 запрос на запрос коллекций 2-го уровня. Однако если объект запросить с помощью массового запроса, то при его обходе произойдет всего 3 запроса. Что может ускорить навигацию по объектам более чем в 10 раз.

Пример массового запроса:

```
for (rv <- new OQuery(Stk_WarrantAta.Type){
  where (t.id in idap)
  batchAll()
}){}
```

Примечание

Объектные запросы активно расходуют оперативную память. Это накладывает ограничение на использование их в процедурах бизнес-логики. Обычно объектные запросы используются для организации пользовательского интерфейса (редактирование одного объекта с коллекциями, справочники, документы и т.д.), а для программирования внутренних процедур бизнес-логики используются SQL запросы в БД.

Работа с провайдерами строк

Провайдер строки - Rop используется для работы со строкой данных (Aro), загруженных в рабочее пространство, обеспечивая гарантию того, что при доступе к строке данная строка будет находиться в рабочем пространстве.

Метод получения rop:

```
thisApi().load(идентификатор.asNLong)
```

Примеры сеттеров в файле выборки:

```
val rop = thisRop
thisApi().setidContras(rop,getVar("super$id").asNLong)
```

Работа с rop в API:

```
for (ropGrade <- new OQuery(entityAta.Type){
  where (t.idGdsGrade === idpGdsGrade)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
) {  
    setidGdsGrade(ropGrade, None.nl)  
}
```

Работа с AnyRop(rop неизвестного типа):

```
anyRop match {  
    case Btk_GroupApi(ropGroup) =>  
        ropGroup.get(_.sCaption)  
    case Btk_ClassApi(ropClass) =>  
        ropClass.get(_.sName)  
    case _ => throw AppException("Ожидали rop группы или класса")  
}  
  
// получить из списка только ропы определенного класса  
ropaAny  
    .collect {  
        case Btk_GroupApi(ropGroup) => ropGroup  
    }
```

Оптимистическая блокировка

Так как система Global Postgres ориентирована на работу с короткими транзакциями, фреймворк по умолчанию включает для классов оптимистическую блокировку.

Принцип работы оптимистической блокировки:

- При загрузке строки в кэш, запоминается версия изменения.
- Если строка изменяется, увеличивается версия изменения.
- При сохранении в базу происходит проверка того, что старая версия изменения соответствует версии изменения в базе данных, если это не так, то выдается ошибка применения изменений.

Для отключения оптимистической блокировки в классе необходимо указать свойство:

```
<class useOptimisticLocking="false"/>
```

Примечание

Для хранения версии изменений используется служебное поле `nVersion_dz`

Коллекции

Коллекцией является сущность, объекты которой не могут существовать без ссылки на объект-владелец. Классы коллекций объявляются в Odm сущности-владельце.

В бизнес-объекте предполагается использование ленивых коллекций. Элементы коллекции загружаются по необходимости, удаление добавление происходит в фоновых коллекциях и не требует немедленного запроса к базе данных.

Связывание сущностей владельца и коллекции производится путём объявления элемента `collection` в секции `collections`.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
  name="Bs_Brigade" caption="Бригада"
  cardEditor.representation="Card"
  listEditor.representation="List"
  viewOptions.openCardType="mdi"
  supertype="reference">
  <attributes>
    <attr name="id"
      attribute-type="Long"
      caption="Идентификатор"
      order="-1"
      type="basic"
      isVisible="false"/>
    <attr name="idClass"
      attribute-type="Long"
      caption="idClass"
      order="-2"
      type="basic"
      isVisible="false"/>
    <attr name="gid"
      attribute-type="Varchar"
      isVisible="false"/>
    <attr name="sCode"
      attribute-type="Varchar"
      caption="Код"
      order="10"
      isMnemonicCode="true"
      type="basic"
      isVisible="true">
      <autonum id="1">
        <dimension name="Dim1"/>
      </autonum>
    </attr>
    <attr name="sCaption"
      attribute-type="Varchar"
      caption="Наименование"
      order="20"
      isHeadLine="true"
      type="basic"
      isVisible="true"/>
    <attr name="idDepartment"
      attribute-type="Long"
      caption="Подразделение"
      order="30"
      type="refObject"
      ref.class="Bs_Department"/>
    <attr name="idForeman"
      attribute-type="Long"
      caption="Бригадир"
      order="40"
      type="refObject"

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        ref.class="Bs_Employee"/>
        <attr name="idMaster"
attribute-type="Long"
caption="Мастреп"
order="50"
type="refObject"
ref.class="Bs_Employee"/>
</attributes>
<collections>
    <collection name="Bs_BrigadeStaff"
ref.attr="idBrigade"
cascadeOnDelete="true"/>
</collections>
<dbData>
    <script name="dataInstall" version="1">
        <install>Bs_BrigadeApi.dataInstall()</install>
    </script>
</dbData>
</class>
```

Для класса коллекции указывается супертип «collection», это гарантирует наследование scala-классов от необходимых системных классов.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
name="Bs_BrigadeStaff"
caption="Состав бригады"
cardEditor.representation="Card"
listEditor.representation="List"
viewOptions.openCardType="mdi"
supertype="collection">
    <attributes>
        <attr name="id"
attribute-type="Long"
caption="Идентификатор"
order="-1"
type="basic"
isVisible="false"/>
        <attr name="idClass"
attribute-type="Long"
caption="idClass"
order="-2"
type="basic"
isVisible="false"/>
        <attr name="gid"
attribute-type="Varchar"
isVisible="false"/>
        <attr name="idEmployee"
attribute-type="Long"
caption="Сотрудник"
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        order="10"
        type="refObject"
        ref.class="Bs_Employee"
        isMnemonicCode="true"/>
    <attr name="idBrigade"
        attribute-type="Long"
        caption="Бригада"
        order="20"
        type="refObject"
        ref.class="Bs_Brigade"
        genListCollectionRep="true"/>
    <attr name="dStart"
        attribute-type="Date"
        caption="Дата начала"
        order="30"
        type="basic"
        editorType="datePick"/>
    <attr name="dEnd"
        attribute-type="Date"
        caption="Дата окончания"
        order="40"
        type="basic"
        editorType="datePick"/>
    <attr name="bisForeman"
        attribute-type="Number"
        caption="Является бригадиром"
        order="50"
        type="basic"
        editorType="check"/>
</attributes>
</class>
```

Формирование кода

При формировании кода сущности-владельца производится пересоздание кода всех коллекций. Dpi коллекции будет унаследован от ChildApi.

```
trait Xxx_XxxxDpi[T] extends ChildApi[java.lang.Long, ARO, API]
```

Dvi будет унаследован от CollectionApi.

```
trait Xxx_XxxxDvi extends CollectionApi
```

Основные методы для работы:

```
//создание по владельцу, возвращает топ созданного объекта
Api().insertByParent(ropMaster)
//загрузка всей коллекции по владельцу, возвращает обходчик записей отфильтрованных по
↳ топ предка
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
Api().byParent(ropMaster)
Api().byParent(idMaster)
//удаление по rop объекта
Api().delete(rop)
```

Отображения-детали

Для формирования отображений выборки, данные которых ограничены по значению ссылочного поля, необходимо в odm-файле, для соответствующего ссылочного атрибута указать свойство `genListCollectionRep=»true`. Будет сформировано отображение `List_{attr}`.

```
<attr name="idBrigade"
      attribute-type="Long"
      caption="Бригада"
      order="20"
      type="refObject"
      ref.class="Bs_Brigade"
      genListCollectionRep="true"/>
```

Переменная ссылочность

Часто при проектировании бизнес-логики необходимо хранить в атрибуте коллекции ссылки на объекты нескольких классов. В этом случае необходимо хранить не только идентификатор ссылочного объекта, но и идентификатор ссылочного класса.

Для реализации переменной ссылочности, в Odm коллекции необходимо создать атрибут, способный ссылаться на поле «gid» мастер-объекта.

```
<attr name="gidSrcXXXXXX"
      attribute-type="String"
      type="refAnyObject"
      genListCollectionRep="true"/>
```

Примечание

У классов, на объекты которых может ссылаться коллекция с переменной ссылочностью обязательно должен существовать атрибут «gid». Рекомендуется индексировать данное поле.

Каскадное удаление

Если у коллекции включено свойство «Каскадное удаление» (по умолчанию включено),

```
<collections>
  <collection cascadeOnDelete="true"
    name="Bs_PersonIdentityDoc"
    ref.attr="idPerson"/>
  <collection cascadeOnDelete="true"
    name="Bs_PersonProf"
    ref.attr="idPerson"/>
</collections>
```

то в `dpi` мастера формируется вызов метода `delete` из коллекций.

Для подключенных `vcollection` (переменных коллекций) код удаления необходимо писать вручную в `api`:

```
override def delete(rop: ApiRop): Unit = {
  for (crop <- Bs_BankAccApi().byParent(rop)) {
    Bs_BankAccApi().delete(crop)
  }
  for (crop <- Bs_DefSettlerAddressApi().byParent(rop)) {
    Bs_DefSettlerAddressApi().delete(crop)
  }
  for (crop <- Bs_SettlerAddressApi().byParent(rop)) {
    Bs_SettlerAddressApi().delete(crop)
  }
  super.delete(rop)
}
```

Механизмы наследования

Во фреймворке нет полноценного классического наследования классов сущностей. Понятие «наследование» используется в качестве описания логической связи двух классов.

Вместо классического наследования фреймворк использует миксины и трейты.

Миксин (Mixin)

Примесь (англ. Mix-in) — элемент языка программирования (обычно класс или модуль), реализующий какое-либо чётко выделенное поведение. Используется для уточнения поведения других классов, не предназначен для порождения самостоятельно используемых объектов.

В системе Global3 Postgres миксин — это специализированный класс, каждый объект (запись в таблице) которого соответствует одному объекту подключенного класса (записи в таблице). Mixin-класс может иметь не ограниченное число подключенных классов. Первичным ключом для mixin-объектов является поле `gidRef`. Значение поля `gidRef` равно значению поля `gid` объекта подключенного класса и заполняется в момент создания объектов.

В `Api` миксина реализуются общие методы, которые могут использоваться для всех подключенных классов.

Миксин может содержать произвольное число атрибутов. Атрибуты, имена которых совпадают с именами атрибутов подключенных классов (наследников), будут автоматически заполняться при установке значений в атрибуты.

Примечание

Все scala-классы, соответствующие миксину наследуются от системных классов D-ветви (имеют префикс «D»). Остальные scala-классы, соответствующие обычным сущностям, наследуются от системных классов S-ветви (имеют префикс «S»)

Для миксинов метод получения rop по `gidRef` называется `loadByGid`, а также реализован метод аналогичный методу `get` для `SApi`

```
getByGid(gidpRef: NGid): Option[ApiRop]
```

Создание mixin-класса

В Odm файле необходимо объявить атрибут `gidRef`.

```
<attr name="gidRef" attribute-type="Varchar"/>
```

У класса указать свойство «supertype="mixin"».

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
      name="Xxx_ClassName" supertype="mixin"/>
```

Формирование кода

При формировании кода для mixin-класса, `Dpi` будет унаследован от `MixinApi`.

```
trait Xxx_XxxxDpi[T] extends MixinApi[java.lang.Long, ARO, API]
```

`Dvi` будет унаследован от `AppMixinApi`.

```
trait Xxx_XxxDvi extends AppMixinApi
```

Основные методы для работы:

```
//создание по rop мастера, возвращает rop миксина
Api().insertByParent(ropMaster)
//загрузка миксина по gidRef, возвращает rop миксина
Api().loadByGid(gidRef)
//удаления по gidRef
Api().deleteByKey(gidRef)
```

Объявление подключенных классов (классов-наследников)

Для связывания класса с миксином необходимо в Odm файле указать имена классов-миксинов.

```
<mixins>
  <mixin name="Bs_Settler" isDpiManaged="true"/>
</mixins>
```

Формирование кода

При формировании кода сущности в Dpi будет добавлен код:

- Вставки миксин-объекта в методе `insert`
- Удаления миксин-объекта в методе `delete`
- Установки значений атрибутов миксина

Код будет добавлен в сеттера атрибутов, имена которых совпадают с именами атрибутов сущности, в сеттерах атрибутов сущности.

При необходимости управления генерацией миксина вручную необходимо отключить формирование кода в Dpi с помощью настройки `isDpiManaged`.

По умолчанию по всем подключённым миксинам генерируется код в Dpi

Системные миксины

Функционально не отличаются от прикладных миксинов. Основной системный миксин это `Btk_Object`.

При формировании кода для классов с `supertype="document"` и `supertype="reference"`, этот миксин автоматически подключается к классу. Допускается отключение миксина `Btk_Object` путём указания свойства в Odm файле:

```
<reflection isEnabled="false"/>
```

Пример смотрите в классе `Btk_Group`

Ссылочность на миксин

Для атрибутов, ссылочных на миксин, необходимо указывать тип `refAnyObject`:

```
<attr name="gidSettler"
  attribute-type="Varchar"
  caption="Контрагент"
  type="refAnyObject"
  isVisible="false"
  order="20"
  ref.class="Bs_Settler"/>
```

Для возможности автоматической генерации HL и MC атрибутов, сеттеров этих атрибутов и добавления их в `selectStatement` и `onRefreshExt` необходимо указать в настройке `ref.class` класс миксина для ссылки.

Важно

Для миксинов по ссылочным атрибутам не генерируются внешние ключи. Вместо этого генерируются индексы.

Трейт (Trait)

Trait — это механизм обеспечения повторного использования кода между классами. В отличие от наследования, класс может содержать несколько трейтов.

Трейт не имеет объектов, собственной структуры хранения данных и визуального представления.

Создание трейта:

1. Создайте Odm файл
2. укажите свойство `supertype="trait"`

При формировании кода будут созданы только Dpi и Api файлы. Dpi будет содержать только объявления сеттеров и геттеров.

Наследование сущности от трейта

Для наследования сущности от трейта, добавьте в Odm свойство класса `with="Trait_Name"`

Сущность, наследуемая от трейта, должна иметь все атрибуты, объявленные в трейте.

Перекрытие и порядок вызовов методов

Рассмотрим код трейта и наследуемого от него класса:

```
trait Gs3_TraitApi[T] extends Gs3_TraitDpi[T] {
  override def setFNumber(rop: ApiRop, value: NNumber): Unit = {
    Logger.Factory.get(getClass).info("Gs3_TraitApi.setFNumber")
    super.setFNumber(rop, value)
  }
}

class Gs3_DescendantApi extends Gs3_DescendantDpi[T] with Gs3_TraitApi[T] {
  override def setFNumber(rop: ApiRop, value: NNumber): Unit = {
    Logger.Factory.get(getClass).info("Gs3_DescendantApi.setFNumber")
    super.setFNumber(rop, value)
  }
}
```

В результате выполнения метода `Gs3_DescendantApi.setFNumber()` последовательность вызовов будет следующей:

1. `Gs3_DescendantApi.setFNumber`

2. Gs3_TraitApi.setFNumber
3. Gs3_DescendantDpi.setFNumber

Пример разметки класса

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0" name="Bs_
↪GdsCostDeviationType"
    caption="Виды отклонений в стоимости ТМЦ"
    cardEditor.representation="Card" listEditor.representation="List"
    viewOptions.openCardType="mdi" supertype="reference">
    <attributes>
        <attr name="id" attribute-type="Long" caption="Идентификатор" order="-1" type=
↪"basic" isVisible="false"/>
        <attr name="idClass" attribute-type="Long" caption="idClass" order="-2" type=
↪"basic" isVisible="false"/>
        <attr name="gid" attribute-type="Varchar" isVisible="false"/>
        <attr name="sCode" attribute-type="Varchar" caption="Код" order="10" isMnemonicCode=
↪"true" type="basic"
            isRequired="true" isVisible="true"/>
        <attr name="sCaption" attribute-type="Varchar" caption="Наименование" order="20" ↪
↪isHeadLine="true" type="basic"
            isRequired="true" isVisible="true"/>
        <attr name="sDescription" attribute-type="Varchar" caption="Описание" order="30" ↪
↪type="basic" editorType="memo"
            isVisible="true">
            <descriptionColumn/>
        </attr>
    </attributes>
</class>
```

Сервисные возможности для классов

Инфраструктурные возможности фреймворка

Возможности классов расширяются сервисным окружением, которое постоянно совершенствуется.

Служебные атрибуты

Для сервисных возможностей фреймворк добавляет в таблицы служебные атрибуты. Служебные атрибуты заканчиваются постфиксом `_dz`.

Пример:

- `nVersion_dz`;
- `screateuser_dz`;
- `dcreatedate_dz`.

Внимание

Разработчикам запрещено создавать и изменять такие атрибуты.

Отображение состояния сессии в форме

Состояние сессии можно определить по цвету системных операций Сохранить и Откатить на тулбаре формы.

Все состояния формы перечислены в таблице:

Пользовательская блокировка	Данные для отправки в базу данных	Кнопка «Сохранить»	Кнопка «Откатить»
Нет	Нет	не активна	не активна
Нет	Есть	активна	активна (крест)
Есть	Нет	не активна	активна (круг)
Есть	Есть	активна	активна (крест)

Настройки приложения

Настройки приложения позволяют задавать глобальные константы. Для получения значения настройки используйте функцию:

```
Btk_AppPropertiesTypeApi.GetbValue(  
    sType // - Системное имя настройки  
    idObjectType // Тип объекта  
) // возвращает true или false
```

Btk_AppPropertiesType

Таблица настройки приложения, содержит поля:

- sSysName — Системное имя;
- sCaption — Наименование;
- sDescription — Описание;
- idModule — Модуль.

Btk_AppProperties

Таблица значений настроек приложения, содержит поля:

- idType — Настройка;
- idObjectType — Тип объекта;
- bValue — Значение boolean;
- nValue — Значение number;
- sValue — Значение string;
- dValue — Значение date.

Поиск по шаблону

Поиск по шаблону — серверная возможность поиска объектов класса по частичному или полному совпадению введенного текста со значениями полей объекта или его заголовка и мнемокода. Поиск регистронезависимый.

Алгоритм работы поиска

1. Определяются поля, по которым требуется осуществлять поиск, и их приоритет.
2. Выполняются последовательно запросы, пока не получен результат в виде одной записи:
 - По всем полям в порядке приоритета ищутся значения по полному совпадению введенного текста (=).
 - По всем полям в порядке приоритета ищутся значения по частичному совпадению введенного текста (like).
3. Возвращается результат: первым считается первый запрос, вернувший одну запись, либо первый запрос, вернувший данные.

По умолчанию для класса включен поиск по шаблону и осуществляется по служебным полям `sheadline_dz` и `smnemocode_dz`. Сначала — по заголовку, затем — по мнемокоду.

Настройка поиска для класса

В odm-файле для класса добавьте тег `patternSearch`:

```
<class>
  <patternSearch isActive="true"
    headLineOrder="100"
    mnemoCodeOrder="200"
    searchType="startsWith"
    indexHlMc="true"/>
</class>
```

Описание настроек:

- `isActive` — Активность поиска по шаблону на классе. Если выключено, функция поиска по шаблону будет возвращать 0 записей. Умолчательное значение — `true`.
- `headLineOrder` — Порядок поиска по служебному полю `sheadline_dz`. Умолчательное значение — 1. Чем меньше значение, тем выше приоритет поиска по этому полю. Поля с наименьшими значениями порядкового номера обрабатываются в первую очередь.
- `mnemoCodeOrder` — Порядок поиска по служебному полю `smnemocode_dz`. Умолчательное значение — 2. Логика обработки аналогична `headLineOrder`.
- `searchType` — Тип поиска:
 - По началу слова (по умолчанию) — like с правым %.
 - По вхождению — like с правым и левым %.
- `indexHlMc` — Индексировать поля `sheadline_dz` и `smnemocode_dz`. При включенном свойстве будут созданы индексы по этим полям в верхнем регистре.

Настройка поиска по атрибутам

1. Откройте odm-файл.
2. Добавьте тег `patternSearch` для нужного атрибута. Если тег добавлен, атрибут участвует в поиске по шаблону.

Пример:

```
<attr name="sText"
      attribute-type="Text"
      type="basic"
      order="130"
      caption="Text">
  <patternSearch order="10"
    searchType="contains"/>
</attr>
```

Описание настроек:

- **order** — Порядковый номер поиска по полю этого атрибута. Нумерация сквозная по классу: можно настроить, чтобы атрибут обрабатывался раньше заголовка и мнемокода.
- **searchType** — Тип поиска:
 - По началу слова (по умолчанию) — `like` с правым `%`.
 - По вхождению — `like` с правым и левым `%`.

Перекрытие метода поиска

Метод реализован в одном из рутовых API:

```
ru.bitec.app.gtk.eclipse.rdb.SEntityBaseApiImpl#findByPattern
```

В наследниках его можно перекрыть и написать свою логику поиска по шаблону.

Полнотекстовый поиск

Сервисная возможность класса осуществлять быстрый поиск по значению атрибутов класса.

Данные классов хранятся в индексе полнотекстового поиска в схеме `fts`:

- **Btk_FtsReg** — индекс объектов классов;
- **Btk_FtsWords** — индекс используемых слов.

Форма поиска доступна в главном меню приложений, в меню «Сервис».

Настройка

Для включения полнотекстового поиска в Odm установите свойство класса `fts.Enabled` в значение `true`.

По умолчанию в полнотекстовом поиске участвуют все строковые значимые/автономерующиеся/вычисляемые, а также ссылочные на объект и атрибуты переменной ссылочности с указанным ссылочным классом.

Для переопределения поиска по атрибуту используется свойство `fts.Enabled`.

Обновление индекса

После обновления кода на проекте и вызова ssh-команды `init Data` происходит анализ классов. Если настройки класса изменились (класс стал индексируемым или добавились новые индексируемые атрибуты, которые до этого не индексировались; новые атрибуты не учитываются), происходит планирование обновления индекса для всех объектов класса.

При изменении объектов классов и сохранении в БД (`session.flush()`), которые индексируются, или на которые ссылаются объекты индексируемых классов, эти объекты добавляются в лог измененных объектов индекса.

Значения ссылочных полей хранятся в строковом виде и определяются как значение поля `sHeadLine_dz` ссылочного объекта.

В системе зарегистрировано задание «Синхронизация полнотекстового индекса», которое обновляет индекс. Это задание анализирует таблицу измененных объектов и таблицу измененных классов, сохраняет эти данные в общую очередь на индексацию, вычисляет и обновляет данные индекса.

Команды строки поиска

Формат описания:

- Введенная строка — Логика обработки.

Команды:

- Цепь — Найдёт текст, в котором присутствует слово «цепь», «цепи», «цепью» и т. д.
- цепь звено — Найдёт текст, в котором присутствуют оба слова и их склонения.
- Цепь И звено — То же самое, что и цепь звено. Пробел считается оператором «И».
- Цепь & звено — То же самое, что и цепь звено.
- Цепь ИЛИ звено — Найдёт текст, в котором присутствует одно из слов или оба, с учётом склонений.
- Цепь | звено — То же самое, что и цепь ИЛИ звено.
- Цепь И (Звено ИЛИ Кольцо) — Найдёт текст, в котором есть слово «цепь» и одно из слов «звено» или «кольцо».
- Цепь#40 — Найдёт текст, в котором есть похожие на «цепь» слова, например, «цеп.». Число после # показывает точность поиска — от 1 до 100, где 100 — полное совпадение.
- #Цепь — То же самое, что и Цепь#40.
- Цеп* — Найдёт текст, слова в котором начинаются на «цеп», например, «цепь», «цепная» и т. д.

Автономумерация и генерация значений

Автоматическое присвоение значений атрибутам объектов — ключевая функция платформы, обеспечивающая уникальность, последовательность и стандартизацию данных. Документ описывает механизмы автономумерации, генерации штрих-кодов и управления подписями для печати.

Автономумерация

Автономумерация — процесс автоматической выдачи значения для атрибута класса. В основе лежит выдача порядкового номера (числа), которое может быть преобразовано в значение по маске.

Автоматическая нумерация объектов системы позволяет устанавливать для созданных объектов уникальные коды. При этом создаваемые коды последовательно увеличиваются.

Существует два режима работы автономумерации:

- Без заполнения пропусков.
- С заполнением пропусков.

Определение разреза автономумерации

Автономумерация в пределах значений одного или нескольких атрибутов класса называется автономумерацией с разрезом. Значение разреза может вычисляться по правилу, указанному в теге **expression**. Если выражение не указано, будет использоваться значение атрибута.

Например, в течение одного дня выдаются номера последовательно, при наступлении следующего дня номера начинают выдаваться заново. Такой разрез называется разрез по дате.

Настройка автономумерации без разреза

Для автономумерации без разреза в odm-файле класса указываем для атрибута тип **autonum**, далее в блоке **autonum** задаём имя разреза **dimension**:

```
<attr name="sNoDep"
  caption="Автономумерация без разреза"
  attribute-type="Varchar"
  type="autoNum"
  isHeadLine="true">
  <autonum id="1">
    <mask>lpad(counter,3,"0")</mask>
    <dimension name="dim1"/>
  </autonum>
</attr>
```

Настройка автонумерации с разрезом

Для настройки разрезов `dimension`:

1. Задайте уникальные имена.
2. Укажите атрибуты разреза. Разрезы указываются из атрибутов класса.
3. При необходимости задайте порядковый номер. Если его не указывать, они будут обрабатываться в алфавитном порядке.

Количество разрезов должно быть ≥ 1 .

Указание нескольких разрезов:

```
<attr name="sTwoDepExpr"
  caption="Автонумерация с 2 разрезами"
  attribute-type="Varchar"
  type="autoNum">
  <autonum id="3">
    <mask>lpad(counter,3,"0")</mask>
    <dimension name="nNumber"
      attr="nNumber"
      order="10" />
    <dimension name="dDate_dim"
      attr="dDate"
      order="20">
      <expression>truncDate(dimValue)</expression>
    </dimension>
  </autonum>
</attr>
```

Если в настройках автонумерации указан разрез, то автоматически включено заполнение пропусков. Отключить заполнение пропусков можно через свойство `isHoleFill`:

```
<attr name="sOneDepNoHoleFill"
  caption="Автонумерация с 1 разрезом без заполнения пропусков"
  attribute-type="Varchar"
  type="autoNum"
  isHeadLine="true">
  <autonum id="4"
    isHoleFill="false">
    <dimension name="dDate"
      attr="dDate"
      order="10"/>
  </autonum>
</attr>
```

Общие настройки

Возможно использовать маски для вывода значения счётчика. Значение, указанное в элементе `mask`, будет передано в обработчик Jexl-скриптов.

Параметр, заменяемый на значение счётчика, называется `counter`.

В Jexl добавлены методы:

- `lpad(string, size, padChar)` — добавление строки символами слева;
- `rpadd(string, size, padChar)` — добавление строки символами справа;
- `truncDate(date)` — получение даты на начало дня от переданной;
- `truncYear(date)` — получение даты на начало года от переданной.

В jexl-скрипте можно обращаться к `api` и `pkg` объектам.

К значениям атрибутов разреза возможно применение выражений, они указываются в элементе `expression` для `dimension`.

На каждый разрез может быть не больше одного выражения.

Значение, указанное в элементе `expression`, будет передано в обработчик Jexl-скриптов. Параметр, заменяемый на значение атрибута разреза, называется `dimValue`.

```
<attr name="sRegNumber"
      attribute-type="Varchar"
      caption="Per. №"
      order="230"
      type="autoNum"
      isVisible="false"
      isReadOnly="true">
  <autonum>
    <mask>
      var sOTCode = Btk_ObjectTypeApi.load(self.idObjectType()).getByAttrName(\
↪ "sCode\");
      if (sOTCode eq \"OperDecisionPrj\" || sOTCode eq \"ProductDecisionPrj\") {
        counter + \"-Г\"
      } else {
        counter + \"-Г.\" + Bs_PrjVerApi.load(self.idPrjVer()).getByAttrName(\
↪ "sCode\")
      }
    </mask>
    <dimension name="idObjectType" attr="idObjectType" order="1">
      <expression>
        if (dimValue == null) {
          null
        } else {
          var sCode = Btk_ObjectTypeApi.load(dimValue).getByAttrName(\"sCode\
↪ ");
          if (sCode eq \"OperDecisionPrj\" || sCode eq \"OperDecisionWrk\" ||
            sCode eq \"ProductDecisionPrj\" || sCode eq \"ProductDecisionWrk\
↪ ")
            ) 0L else null;
        }
      </expression>
```

(продолжается на следующей странице)

```

    </dimension>
  </autonum>
</attr>

```

Для обращения к полям класса при вычислении маски используется переменная **self**, которая является объектом типа **Aro**, поэтому для получения значения атрибута необходимо добавлять в конце скобки (т.к. это метод).

Например: `self.sCaption()`

После генерации кода необходимо будет выполнить метод `*Dpi.autoNumCreateTable`.

Если с момента последнего выполнения данного метода были изменены данные автонумерации (имена разрезов, типы данных разрезов, количество разрезов), то таблицы и индексы будут пересозданы.

Также генерация окружения автонумерации выполняется перед инициализацией первоначальных данных (**init data**), которая вызывается при обновлении проектного **jar**-кода или при выполнении генерации таблиц в IntelliJ IDEA: **External Tools > Generate Tables**.

Присвоение значения автонумерации происходит в момент сохранения транзакции.

При ручной установке значения автонумерующегося атрибута (при работе из интерфейса) будет задан вопрос:

«Значение присваивается автоматически в момент сохранения, ввод данных отключит автонумерацию. Продолжить?»

Положительный ответ на вопрос установит значение и отключит автонумерацию объекта. Для повторного включения автонумерации нужно сбросить значение атрибута.

Автонумерация в миксинах

Возможно задание автонумерации в миксине. Атрибуты, которые объявлены как автонумерующиеся в миксине, но при этом как обычные в целевом классе, получают настройки автонумерации + сквозные таблицы хранения значений автонумерации для всех классов, использующих данный миксин.

Если атрибут объявлен как автонумерующийся и в миксине, и в классе, то будет использована настройка класса.

Создание окружения автонумерации

Для создания окружения автонумерации по классу:

1. Зайдите в приложение «Настройка системы».
2. Откройте список классов.
3. Переведите фокус на целевой класс. Для данного класса должен стоять признак «Есть автонумерация».
4. Выполните операцию «Создать структуры автонумерации». Данная операция находится под молоточками. При этом произойдёт вызов метода `*Dpi.autoNumCreateTable`.

Для создания окружения автонумерации по атрибуту класса:

1. Откройте карточку класса.
2. Переведите фокус на целевой атрибут.

3. Выполните операцию «Создать структуру автонумерации». Данная операция вызывает метод `*Dpi.autoNumCreateTableByAttr`.

Проектное переопределение

Для индивидуальной настройки работы автонумерации реализован механизм проектного переопределения. Вся настройка производится из пользовательского интерфейса без необходимости изменения исходного кода и перекомпиляции.

Проектные настройки хранятся в БД и не сбрасываются при установке обновлений.

Для выполнения проектного переопределения:

1. Зайдите в карточку класса.
2. Выберите автонумерующийся атрибут.
3. Откройте закладку «Настройки автонумерации».
4. На панели открытой закладки нажмите кнопку «Переопределить».
5. Настройте разрезы и маску.
6. Выполните операцию создания окружения нумерации.

Настройка счётчиков и данных разрезов

Если автонумерация без разрезов, то она и без заполнения пропусков. Счётчик такой автонумерации хранится в сиквенсе.

Автонумерация с разрезами хранит свои данные в двух таблицах:

- **Таблица разрезов.** На каждый уникальный набор значений разрезов хранится значение счётчика, маска, признак, что есть пропуски.
- **Таблица пропусков.** Для каждого разреза хранит информацию о имеющихся пропусках в виде пар значений («значение от» и «значение до»).

Таким образом, для изменения текущего значения счётчика для автонумерации без разрезов достаточно изменить значение сиквенса.

Для автонумерации с разрезами, чтобы изменить счётчик, нужно установить значение в таблице разрезов.

Для настройки:

1. Зайдите в карточку класса.
2. Выберите автонумерующийся атрибут.
3. Откройте закладку «Настройки автонумерации».
4. Откройте закладку «Данные автонумерации».
5. Настройте значение счётчика:
 - Если автонумерация без разрезов, то на детальной закладке будет карточка с одним атрибутом, который показывает текущее значение сиквенса.
 - Если автонумерация с разрезом, то на детальной закладке будут отображены данные из таблицы разрезов и таблицы пропусков. Для изменения счётчика в нужном разрезе указывается значение в поле «Максимальное значение». Если разреза нет, его можно создать. Если удалить разрез, автонумерация начнётся с 1 для этого разреза.

Автоматическая генерация штрих-кодов объекта

Сервисная возможность при вставке объекта класса формировать ему штрих-код.

Для этого в ODM в теге **Class** есть свойство **barCodeObjectTypeName**, в качестве значения указывается системное имя объектов класса **Bs_BarCodeObjectType**.

Для регистрации новой записи используется метод:

```
Bs_BarCodeObjectTypeApi().register
```

Описание работы маски

Если в маске указано: 001-[counter], при этом:

- Длина штрих-кода составляет 10 символов.
- Следующее значение счётчика: 12.

Итоговый штрих-код: 001-000012

Подписи объектов для печати

Сервисная возможность указания типа подписей объекта. Используется при печати отчётов. Позволяет формировать в печатной форме список лиц с местом для подписи.

Список подписей настраивается в коллекции для печатной формы. Список может быть переопределён у объекта. Для хранения подписей печатных бланков используется JSON-контейнер **jSign_dz**.

Приоритет отображения данных по типу подписи

1. Данные по типу подписи, хранимые в атрибуте объекта класса **jSign_dz** (формируется для всех классов).
2. Данные из настроек подписей в отчёте.

При печати отчёта, если удалось определить сохранённые подписи объекта, то будут переданы настройки подписей на самом отчёте.

Кроме этого, в базовых справочниках реализована возможность формирования комиссий, состав которых может быть передан в отчёт. Подробная настройка комиссий будет рассмотрена в разделе «Прикладная типизация объекта класса».

Данные подписей передаются в параметр **SIGNDATA_DZ**. Представляют собой JSON-объект с ключами:

- **idComission** — идентификатор комиссии. Определяется через тип объекта и подразделение.
- **data** — массив подписей объекта. Каждый элемент представляет собой JSON-объект по типу подписи. Поля:
 - **idBlankSignType** — ИД типа подписи;
 - **idBlankSignTypeMC** — системное имя типа подписи;
 - **idBlankSignTypeHL** — наименование типа подписи;
 - **sPosition** — должность;
 - **sFI0** — Фамилия Имя Отчество;

- `sBasisDocument` — документ-основание;
- `dDate` — дата подписи;
- `idDepartment` — ИД подразделения;
- `idDepartmentMC` — код подразделения;
- `idDepartmentHL` — наименование подразделения;
- `nOrder` — порядковый номер подписи комиссии.

Для работы с атрибутом `jSign_dz` реализован API в пакете `Bs_ObjectSignPkg`.

Если для класса задан **тип объекта**, для работы с подписями можно воспользоваться стандартной закладкой:

```
ru.bitec.app.bs.sign.Bs_ObjectSignAvi.List_Master
```

Управление данными объекта

Документ описывает ключевые механизмы управления данными объектов в системе: времязависимые значения, копирование, группировку и денормализацию деревьев. Все функции доступны на уровне метаданных и поддерживают конфигурацию через ODM и пользовательский интерфейс.

Времязависимые значения

Времязависимые атрибуты хранят разные значения для разных периодов времени. Реализовано через таблицу `Btk_TimeDepValue`, где каждая запись содержит значение атрибута для объекта на указанном периоде. Период ограничен датой начала (`dBegin`) и датой окончания (`dEnd`). Если дата не заполнена — это открытый период. Сам атрибут класса хранит только одно актуальное значение.

Примечание

Для каждой пары объекта и атрибута периоды не могут пересекаться.

Создание времязависимого значения

Создать времязависимое значение можно:

- Указав свойство `isTimeDepValue` у атрибута в ODM.
- Включив настройку «Времязависимое значение» в интерфейсе.

Изменение времязависимых значений

При вводе значения в поле времязависимого атрибута выполняется поиск существующих значений:

- Если значений нет — создается запись без указания границ периода.
- Если значения есть — пользователю предлагается изменить значение для текущего периода или добавить новое значение в новый период. При добавлении:
 - Новый период начинается с текущей даты.
 - Для предыдущего периода `dEnd` устанавливается как «текущая дата минус 1 день».

Редактор времязависимых значений

Для времязависимых атрибутов рядом с полем отображается кнопка, открывающая специальный редактор. В редакторе три поля:

- Дата начала.
- Дата окончания.
- Значение (обязательное).

Перед сохранением проверяется пересечение периодов. После закрытия редактора атрибуту присваивается значение, соответствующее текущей дате.

Использование в бизнес-логике

Пример получения значений

```
// Получение последовательности rop по объекту и атрибуту
Btk_TimeDepValueApi().getByObjectAndAttr("30151/146451".ng, 757.nl)

// Получение последовательности rop в отсортированном порядке
Btk_TimeDepValueApi().getByObjectAndAttrSorted("30151/146451".ng, 757.nl)

// Получение числового значения на текущую дату
Btk_TimeDepValueApi().findValueByCurrentDateByAttr("30151/146451".ng, 757.nl).asNNumber

// Получение строкового значения на определённую дату
Btk_TimeDepValueApi().getByObjectAndAttr("30151/146451".ng, 757.nl)
    .find(rop => rop.get(_dBegin) < "04.04.2025".nd && rop.get(_dEnd) > "04.04.2025".nd)
    .map(_get(_jValue))
    .getOrElse(NString())
```

Пример установки значения

```
// Добавление значения на определённый диапазон
val rop = Btk_TimeDepValueApi().insertByAttr("30151/146451".ng, 757.nl, "01.01.2025".nd,
    ↪ "31.12.2025".nd)
Btk_TimeDepValueApi().setValue(rop, "15.10.2025".nd)
```

Копирование объектов

При необходимости, для класса генерируется операция копирования. Метод позволяет создать новый объект, скопировав значения атрибутов и содержимое коллекций. Выполняется глубокое копирование — для коллекций создаются новые записи.

Также возможно добавить данные из одного объекта в другой.

Копирование управляется свойствами в метаданных (по умолчанию = true). Настройка возможна:

- На уровне класса — включение/отключение копирования.
- На уровне атрибута — исключение конкретных полей из копирования.

Уровни настройки

- **На уровне класса:** свойство `isCopyObjectEnabled`.
- **На уровне атрибута:** свойство `isCopyInCopyObject`.

Если копирование включено для класса:

- В мастер-выборке `EntityAvi#Default` создана операция «Копировать». По умолчанию пустая и неактивная. Для классов с `isCopyObjectEnabled="true"` операция активируется и содержит исполняемый код.
- В отображении `Card` создается операция `copyObjectCard`, которая создаёт новый объект и копирует в него данные по `CardRep.IdItemSharp`. Параметр `idBOParent#` используется как мастер-объект.

Настройка в ODM

Пример настройки класса:

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
  name="Mct_OperCard"
  caption="Справочник операционных карт"
  cardEditor.representation="Card"
  listEditor.representation="List"
  viewOptions.openCardType="mdi"
  supertype="reference"
  isCopyObjectEnabled="true"
  attachType="simple"/>
```

Пример настройки атрибута:

```
<attr name="sCreateUser"
  attribute-type="Varchar"
  type="basic"
  isCopyInCopyObject="false"/>
```

При генерации метода `copyObject` в мастер-классе автоматически добавляются вызовы `copyObject` из коллекций — если для их классов копирование включено.

Если свойство `isCopyInCopyObject` не указано — атрибут будет копироваться.

По умолчанию не копируются системные атрибуты:

- `id`
- `gid`
- `gidRef`
- `sImpExpKey_dz`
- `nImpExpFlag_dz`
- `sMnemoCode_dz`
- `sHeadLine_dz`
- Атрибуты связи с мастер-сущностью.

Параметры метода `copyObject`

- Для миксинов: (`gidFrom: NGid, gidTo: NGid, gidParent: NGid`)
- Для V-коллекций: (`idFrom: NLong, idTo: NLong, gidParent: NGid`)
- Для остальных: (`idFrom: NLong, idTo: NLong, gidParent: NLong`)

Метод `copyObject` создаёт объект (если `gidTo/idTo` не передан) через DPI-метод. Установка значений выполняется через DPI-сеттеры.

Группировка

Группировка используется для:

- Систематизации хранения объектов.
- Удобства восприятия пользователем.
- Массового управления характеристиками и настройками объектов.

Дерево групп — иерархическая структура, где потомки наследуют характеристики предков с возможностью переопределения.

Включение группировки

Чтобы включить группировку для класса:

1. Откройте ODM-файл класса.
2. Укажите `group.type` — тип группировки:
 - `single` — объект может входить только в одну группу.
 - `multi` — объект может входить в несколько групп.
3. Укажите `group.root` — системное имя корня группировки.
4. Добавьте в набор коллекций класса развязку групп:

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
  name="Bs_Contras"
  caption="Контрагент"
  supertype="reference"
  cardEditor.representation="MainCard"
  listEditor.representation="MainRoList"
  roListEditor.representation="ROList"
  viewOptions.openCardType="mdi"
  group.type="multi"
  group.root="Group3" />
```

```
<var-collection name="Btk_ObjectGroup"
  ref.attr="gidSrcObject"/>
```

Требования

Для подключения группировки класс должен иметь глобальный идентификатор (gid).

Добавление закладки в карточку

Для управления группами в карточке объекта можно добавить закладку:

1. Откройте AVM-файл класса.
2. Добавьте элемент:

```
<tabItems isVisible="true">
  <tabItem selection="gtk-ru.bitec.app.btk.Btk_ObjectGroupAvi"
    representation="List_Object"
    id="2"
    caption="Группы объекта"/>
</tabItems>
```

Реализация и использование

- Связь объекта с группой — через класс `Btk_ObjectGroup`.
- Денормализованное дерево групп — в таблице `Btk_FlatObjectGroup`.

Эти таблицы можно использовать в запросах для фильтрации объектов или реализации прикладной логики.

Интеграция с выборкой

При включении группировки в список объектов добавляются операции по работе с группами.

Чтобы открыть панель группировки:

1. Откройте список объектов.
2. Выполните операцию «Открыть группы» — слева появится панель с деревом групп.

После открытия:

- В основной части отображаются объекты из текущей группы и её потомков.
- Новые объекты создаются в выбранной группе.

Денормализация классов-деревьев

Для древовидных классов часто требуется вспомогательный класс денормализации, обеспечивающий быстрый доступ к иерархическим данным без рекурсивных запросов.

Реализован трейт `TreeDenormApi` (`ru.bitec.app.btk.denormalization.TreeDenormApi`), предоставляющий функциональность:

- Создание.
- Обновление.

- Удаление денормализации по объектам из класса-дерева.

Пример

Пусть класс-дерево называется `Btk_TreeExample`.

Необходимо создать класс денормализации, например — `Btk_TreeExampleDenormalization`.

Обычно такой класс содержит атрибуты:

- `id`
- `idClass`
- `idParent`
- `idChild`
- `nParentLevel`

Подключение трейта

Подключите трейт `TreeDenormApi` к API класса денормализации и реализуйте абстрактный метод `masterTableName`, возвращающий имя таблицы класса-дерева:

```
class Btk_TreeExampleDenormalizationApi extends
  Btk_TreeExampleDenormalizationDpi[
    Btk_TreeExampleDenormalizationAro,
    Btk_TreeExampleDenormalizationApi,
    Btk_TreeExampleDenormalizationAta
  ]
  with TreeDenormApi[
    EntityAbst, java.lang.Long,
    Btk_TreeExampleDenormalizationAro
  ] {

  override def masterTableName: NString = "Btk_TreeExample"
}
```

Переопределение имён полей

Если имена полей отличаются от стандартных, переопределите методы:

```
def idParentFieldName = "idParent"
def idChildFieldName = "idChild"
def nParentLevelFieldName = "nParentLevel"
```

Обновление денормализации

Запланируйте событие заполнения денормализации:

- В методе установки ссылки на предка (например, `setidParent`).
- В методе `insert` — на случай, если родительская запись изначально пуста.

Вызовите метод:

```
ru.bitec.app.btk.denormalization.TreeDenormApi#updateDenormAfterFlush
```

Метод накапливает все переданные записи и обновляет денормализацию **после** сброса данных в БД.

Удаление денормализации

Удаляйте денормализацию при удалении записи из основного класса — обычно в методе `delete`. Вызовите:

```
ru.bitec.app.btk.denormalization.TreeDenormApi#deleteDenormBeforeFlush
```

Метод накапливает записи и удаляет денормализацию **перед** сохранением в БД.

Прямое обновление (не рекомендуется)

Возможно обновление/удаление **непосредственно** при редактировании, но это замедляет работу системы.

Используйте только при крайней необходимости:

```
ru.bitec.app.btk.denormalization.TreeDenormApi#updateDenorm  
ru.bitec.app.btk.denormalization.TreeDenormApi#deleteByObject
```

Примечание

Пример реализации см. в `Bs_DepartmentTreeApi` в связке с `Bs_DepartmentApi`.

Хранение и обработка файлов

Документ описывает механизмы работы с прикреплёнными файлами в системе: настройку хранилищ, простое и версионное хранение, а также вставку изображений в документы форматов Word и PDF. Все функции управляются через метаданные и поддерживают конфигурацию без кода.

Сервис прикреплённых файлов

Сервис прикреплённых файлов позволяет прикреплять к объектам класса произвольные файлы, которые сохраняются в специальном хранилище на сервере и сопоставляются с конечным объектом-владельцем.

Файловое хранилище

Global3-Framework позволяет классифицировать места хранения файлов по функциональной принадлежности. Для этого используется справочник файловых хранилищ системы.

Файловое хранилище может быть:

- **Локальным** — расположено в директории сервера приложений.
- **Сетевым** — использует протокол SMB.

Перед работой с файлами необходимо настроить файловые хранилища:

1. Откройте приложение «Настройка системы».
2. Выполните пункт меню «Сущности > Файловые хранилища».

В системе должно быть настроено минимум два хранилища:

- **Default** — хранилище по умолчанию.
- **btkAttach** — хранилище прикреплённых файлов.

Для минимизации объёма хранилища используется алгоритм, позволяющий нескольким объектам ссылаться на один физический файл:

- Файл загружается в хранилище только один раз.
- При копировании объектов создаётся символическая ссылка (через `btk_attachitem`), а не физическая копия.

Существует два режима работы:

- Простой.
- Версионный.

Простое хранение файлов

Позволяет работать с прикреплёнными файлами без хранения истории изменений. Это частный случай версионного хранения с одной версией: любое изменение приводит к перезаписи файла.

Доступные операции:

- **Прикрепить файл** — в хранилище добавляется новый файл, в закладке отображается запись с флагом «Основной».
- **Удаление** — удаляется запись из закладки. Если запись отмечена как основная, файл удаляется из хранилища.

Внимание

Операция удаления записи из хранилища необратима.

- **Скачать файл** — из хранилища скачивается файл с именем и расширением, указанными в записи.
- **Сделать файл основным** — устанавливается флаг «Основной». У предыдущей основной записи флаг снимается.

При копировании родительской записи копируются и записи прикрепленных файлов, но **без флага «Основной»**.

Версионное хранение файлов

Версионирование сохраняет историю изменений. Любое изменение файла создаёт новую версию.

Доступны все операции простого хранения, а также:

- **Добавить новую версию** — к записи добавляется версия; предыдущие сохраняются.
- **Удалить последнюю версию** — удаляется последняя версия. Если версия одна, предлагается удалить всю запись.

Внимание

Операция удаления записи из хранилища необратима.

- **Отобразить историю изменений** — открывается список версий. Для каждой доступны операции скачивания и удаления.

При копировании родительской записи копируются записи прикрепленных файлов **со всеми версиями**, но **без флага «Основной»**.

Настройка

По умолчанию сервис прикрепленных файлов для класса отключён. Для подключения укажите тип хранения в ODM-файле класса:

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
  name="Gds_ControlDoc"
  caption="Документ контроля"
  cardEditor.representation="Card"
  listEditor.representation="List"
  viewOptions.openCardType="mdi"
  supertype="document"
  attachType="versioned"/>
```

После генерации AVM-файла во все отображения (Card, List, Tree) автоматически добавляется закладка «Прикрепленные файлы».

Вставка изображений в прикрепленные файлы типов Word и PDF

Для вставки изображений в прикрепляемые файлы документа настройте нужные изображения в коллекции-расширении «Настройки вставки изображений» для типа объекта этого документа.

В коллекции доступны следующие настройки:

- **Активность** — определяет, будет ли изображение вставлено в прикрепленный файл.
- **Печатная форма** — указывается форма типа Jasper с форматом PNG. Передаётся один аргумент: IDDOC (ID документа).
- **Изображение** — файл в формате PNG.

Примечание

Если указана печатная форма, файл изображения удаляется. При загрузке изображения ссылка на печатную форму удаляется.

- **Положение изображения по осям X, Y** — определяет координаты (точка отсчёта — нижний левый угол документа).
- **Ширина и высота изображения** — задают размер вставляемого изображения.

Как только в коллекции появится **активное изображение**, в отображении прикрепленных файлов для документов данного типа становятся доступны операции:

- Вставка изображения в один документ.
- Вставка изображения во все документы сразу.

Безопасность и системное управление

Документ описывает ключевые сервисы платформы, обеспечивающие безопасность, аудит, пообъектный доступ и гибкое управление данными. Все функции доступны через метаданные и поддерживают конфигурацию без изменения кода.

Логические выражения доступности операций и атрибутов

Для настройки редактируемости атрибутов и доступности операций используется сервис jexl-выражений доступности. Настройка выполняется в интерфейсе:
Сущности > Обзорщик проекта > {ВыборкаAvi} > Редактор атрибутов/операций > {Атрибут/Операция}.

Результат выполнения выражения определяет состояние:

- **Доступно** — если скрипт вернул `true` или `null`.
- **Недоступно** — если скрипт вернул `false`.

Структура закладки

Закладка «Выражения доступности (после/до CWA)» разделена на две части:

После checkWorkability	До checkWorkability
Выражение имеет приоритет над прикладной логикой	Прикладная логика имеет приоритет над выражением
Результат прикладной логики может быть перезаписан	Результат выражения может быть перезаписан прикладным кодом
Состояние недоступно в прикладной логике	Состояние может использоваться в прикладной логике

Примечание

Комментарии в полях ввода выражений можно удалять — они не участвуют в бизнес-логике.

Выбор типа выражения доступности

- Используйте **левую форму** (после CWA), если необходимо переопределить прикладную логику. Результат выражения установит статус «только для чтения» независимо от кода.
- Используйте **правую форму** (до CWA), если результат выражения должен быть доступен для использования в прикладной логике.

Аудит

Аудит фиксирует события в системе: вставка, изменение, удаление, выполнение операций и др. Данные хранятся в схеме Aud в таблицах формата <ИмяКлассаШапкиБО>_dzAud, создаваемых в табличном пространстве, указанном в настройке модуля btk — «Табличное пространство аудита» (по умолчанию pg_default).

Внимание

Если аудит включён для класса, но не включён или не синхронизирован для класса-шапки БО (определяется по полю gidRoot_dz), данные аудита сохраняться не будут.

Настройка аудита

1. Откройте приложение «Настройка системы».
2. Перейдите в раздел «Сущности > Классы».
3. Выберите нужный класс.
4. На закладке «Характеристики» установите флаг «Вести аудит».
5. Если класс не является шапкой БО, включите аудит для шапки.

6. Для класса-шапки выполните операцию «Создать оболочку аудита» — будет создана таблица в схеме `Aud`.
7. Очистите кэш:
Сервис > Управление решением > Очистить кэш ORM (shared кэш).

Внимание

Для активации аудита пользователь должен переоткрыть форму.

Просмотр данных аудита

- В карточке объекта: Информация > Аудит объекта.
- В списке объектов: контекстное меню → Информация > Аудит.
- Общий журнал: Настройка системы > Аудит > Аудит.

API аудита

- `Btk_AuditPkg().isEnabledAudit` — проверка, ведётся ли аудит.
- `Btk_AuditPkg().setEnabledAudit(Boolean)` — включение/отключение аудита в сессии.
- `Btk_AuditPkg().insertRow(...)` — ручное добавление записи в аудит.

Перенос таблиц аудита в другое табличное пространство

1. Откройте «Настройка системы».
2. Перейдите в «Настройки и сервисы > Настройки модулей системы > Общие настройки модулей».
3. Выберите модуль `btb`.
4. В поле «Табличное пространство аудита» укажите новое значение.
5. Нажмите операцию «Скачать файл со скриптом переноса таблиц аудита», выполните SQL-скрипт.

Пообъектный доступ

Сервис разграничения доступа на уровне отдельных объектов класса. Права выдаются ролям и пользователям. Пользователи, замещающие других, получают их права.

Права реализуются через группу доступа (`Btk_RlsAccessGroup`). На один объект может быть назначена только одна группа доступа для пользователя или роли.

Управление правами

Используется библиотека `Btk_RlsLib`:

- `manageAccess` — настройка прав для выбранных строк в стандартной выборке (через `gid` или `id/idClass`).
- `manageAccessByGid` — настройка прав для одного объекта.
- `manageAccessByGidList` — настройка прав для списка объектов.

При одновременной настройке нескольких объектов изначально отображаются пользователи и роли, имеющие права на все объекты с одинаковой группой доступа.

Работа с правами в бизнес-логике

Методы пакета `Btk_RlsPkg`:

- `grantToUsers` — выдача прав пользователям.
- `revokeFromUsers` — отзыв прав у пользователей.
- `grantToRoles` — выдача прав ролям.
- `revokeFromRoles` — отзыв прав у ролей.

Удаление объектов с пообъектным доступом

При удалении объектов необходимо очистить связанные структуры доступа:

- `clearByObject` — очистка прав для одного объекта.
- `clearByObjectList` — очистка прав для списка объектов.

Проверка прав

- **В запросах:** используйте таблицу `Btk_RlsUserRightsFlat`, условие через `EXISTS` по `gidObj` и `idUser`.
 - **В бизнес-логике:** методы `getAccessByObject`, `getAccessByObjectList`.
-

Сервис универсальных коллекций

Позволяет прикреплять произвольные коллекции к любому классу без создания межмодульных связей и участия программиста.

При удалении записей вызывается:

```
Btk_UniversalTabPkg().deleteManualCollections(rop.gid)
```

Варианты поведения при удалении

- **Каскадное удаление** — удаляются все записи коллекций, ссылающиеся на удаляемый объект.
- **Генерация ошибки** — если есть связанные записи, операция прерывается с ошибкой.

Подключение из приложения

1. Откройте «Настройка системы».
2. Перейдите в «Сущности > Классы».
3. Откройте карточку класса.
4. Перейдите на закладку «Коллекции» — здесь отображаются все коллекции и доступно добавление универсальной.
5. При необходимости добавьте универсальную коллекцию:
 - Укажите класс и атрибут ссылочности.
 - Поддерживаются типы:
 - Long + RefObject с указанием класса.
 - Varchar + RefAnyObject.
 - Допускается использование проектных атрибутов и характеристик в jsonObjAttrs_dz.

Внимание

Ссылочность возможна только для атрибутов с поддерживаемыми типами данных и видами.

6. Настройте «Действие при удалении».
7. При необходимости удалите коллекцию — доступно только для вручную добавленных.
8. Сбросьте объектный кэш — класс Btk_ClassCollection кэшируется.

Внимание

При подключении универсальных закладок убедитесь, что для класса настроена соответствующая универсальная коллекция.

Подключение из кода

Для программного подключения используйте метод:

```
Btk_ClassCollectionApi().register(  
    idpBtkClass: NLong,  
    idpRefClass: NLong,  
    idpRefAttr: NLong,  
    spDelAction: NString  
): ApiRop
```

Доступные действия определены в объекте `Btk_ClassCollectionApi`.

Справочник параметров

Для реализации локальных коллекций с техническими параметрами используются:

- Трейт `Prm_ParamCalcApi` (`ru.bitec.app.prm.techparam.Prm_ParamCalcApi`).
- Абстрактное отображение `Prm_CalcTechOpParamAbsApi` (`ru.bitec.app.prm.techparam.Prm_CalcTechOpParamAbsApi`).

`Prm_ParamCalcApi`

Трейт обеспечивает функциональность заполнения и расчёта параметров.

Типичные атрибуты коллекции:

- `idTechParameter` — ссылка на параметр `Prm_TechParameter`.
- `idRefTable` — ссылка на таблицу значений `Prm_RefTable`.
- `nValue`, `sValue`, `dValue` — числовое, строковое, датовое значения.
- `bManualInput` — признак ручного ввода.

Пример реализации

```
class Mnf_MCOperParamApi extends Mnf_MCOperParamDpi[Mnf_MCOperParamAro, Mnf_
↳MCOperParamApi, Mnf_MCOperParamAta]
  with Prm_ParamCalcApi[EntityAbst, java.lang.Long, Mnf_MCOperParamAro] {

  override protected def entityAta: Mnf_MCOperParamAta = Mnf_MCOperParamAta

  override lazy val apiParent = Mnf_MCOperApi().asInstanceOf[SApi[AnyRef, Aro[_ <: Mnf_MCOperParamAro, Mnf_MCOperParamApi]]]

  override def getsValue(rop: ApiRop): NString = rop.get(_.$sValue)
  override def getnValue(rop: ApiRop): NNumber = rop.get(_.$nValue)
  override def getdValue(rop: ApiRop): NDate = rop.get(_.$dValue)
  override def getIdValue(rop: ApiRop): NLong = rop.get(_.$idValue)

  override def setIdTechParam(rop: ApiRop, value: NLong): Unit = setIdTechParameter(rop, value)
  override def getIdTechParam(rop: ApiRop): NLong = rop.get(_.$idTechParameter)

  override def getIdRefTable(rop: ApiRop): NLong = rop.get(_.$idRefTable)

  override def getIdCalcFormulaByParent(ropParent: AnyRop): NLong =
    ropParent.asInstanceOf[Mnf_MCOperApi#ApiRop].get(_.$idCalcFormula)

  override def isManualInput(rop: ApiRop): Boolean = rop.get(_.$bManualInput).toBoolean
}
```

Если какие-либо атрибуты не используются, их геттеры можно вернуть как `null`, сеттеры — как `Unit()`.

Дополнительные методы

- `setParamValueRichAnyRef` — установка значения типа `RichAnyRef`.
- `getParamValue` — получение значения параметра.
- `clearAllValues` — очистка всех значений.
- `fillByParent` — заполнение по формуле от родителя.
- `getsParamsCodeAndValueByParent` — строка вида (код: значение).

`Prm_CalcTechOpParamAbsAvi`

Абстрактное отображение содержит два функциональных трейта:

`Default_Calc`

Используется для расчёта параметров по формуле без сохранения значений.

Примечание

Пример: `Mct_TechOpParamAvi.List_idOperCardOperationForCalculate`.

Интерфейс: *Управление данными об изделии (Судостроение) → Технология → Справочник операционных карт → Закладка Операции → Проверка формул по параметрам.*

Для использования:

- Унаследуйте от `Prm_CalcTechOpParamAbsAvi`.
- Переопределите `getFormula` и `getParams`.

`SavedParamsCalc`

Используется для коллекций с сохраняемыми значениями. Может работать вместе с `Prm_ParamCalcApi`.

Переопределяются:

- Сеттеры значений.
- `onRefresh`, `onRefreshItem`, `getAdditionalInfo`.

Необязательный метод `refreshValues` (по умолчанию `active = false`) — для вызова `refreshParamValByParent` и дополнительной логики.

Примечание

Примеры использования:

- `ru.bitec.app.mnf.manufCard.Mnf_MCOperTestParamAvi.List_idMcOper`

- `ru.bitec.app.mct.techproc.Mct_TechProcNormGroupParamAvi.List_idTechProcNorm`

Объектные характеристики

Объектные характеристики — это пользовательские свойства, расширяющие функциональность классов без изменения схемы базы данных. Документ описывает способы хранения, настройки, валидации и программного управления характеристиками.

Характеристики: понятие и назначение

Характеристика — качественное или количественное свойство объекта.

В системе реализованы как специальные атрибуты для хранения дополнительной информации.

Существует два способа хранения:

- **В JSON-контейнере объекта**
Основной способ. Представляет собой NoSQL-расширение строки таблицы. Позволяет динамически добавлять характеристики без пересборки проекта и обновления схемы БД.
- **Как атрибут класса с признаком «объектная характеристика»**
Используется в высоконагруженных сценариях, где доступ к JSON-полям недостаточно эффективен. Характеристика хранится в отдельной колонке таблицы. Требует синхронизации схемы БД при добавлении.

Интеграция с выборкой

Для вывода характеристик используется универсальное отображение `Card_ObjectAttr`. Его генерация управляется параметром `objectAttrCardType` в ODM.

Варианты настройки

Значение	Описание
<code>None</code>	Объектные характеристики не используются. Отображение не формируется.
<code>Simple</code>	Характеристики настраиваются на уровне класса или типа объекта. Если у класса есть тип объекта — настраивается на нём. Если нет — отображаются все характеристики.
<code>Group</code>	Характеристики настраиваются для групп.
<code>GroupAndOb</code>	Характеристики настраиваются для групп и дополняются теми, что заданы на типе объекта.

Отсутствие настройки эквивалентно `objectAttrCardType="Simple"`.

Что отображается в закладке

В отображении выводятся:

- Значения атрибутов с флагом `isObjectAttr="true"`.
 - Значения из JSON-контейнера `jsonAttrs_dz`.
 - Универсальные характеристики (см. раздел *Универсальные характеристики* в [Документации по модулю btk](#)).
-

Настройка характеристик на проекте

Пользователь может добавить характеристики через интерфейс:

1. Откройте приложение «Настройка системы».
2. Перейдите в раздел «Сущности > Классы».
3. Откройте карточку нужного класса.
4. Перейдите на закладку «Атрибуты».
5. При необходимости задайте значение по умолчанию для существующих атрибутов.
6. Создайте атрибут объектной характеристики.

Примечание

Имена атрибутов должны соответствовать соглашению:

- Строковые: начинаются с `s`, например `sStringAttrName`.
- Числовые: начинаются с `n`, например `nNumberAttrName`.
- Даты: начинаются с `d`, например `dDateAttrName`.
- Логические: начинаются с `b`, например `bBooleanAttrName`.

7. Сбросьте кэш метаданных:

Снимите галочку в пункте меню **Сервис > Управление решением > Использовать кэш метаданных выборок**.

8. Выполните операцию «Сбросить кэш настройки» на текущей закладке.
-

Значения по умолчанию

Значения по умолчанию подставляются при создании объекта.

Если класс использует группировку — значения берутся из основной группы.

Для базовых атрибутов

- Указываются в ODM.
- Устанавливаются через DPI при вставке.
- Могут быть переопределены на проекте.

Проектное переопределение

- В списке атрибутов укажите значение в поле «Значение по умолчанию».
- Для сброса: выполните операцию «Удалить переопределение значения по умолчанию».

Программное переопределение

Используйте метод:

```
ru.bitec.app.btk.Btk_AttributeApi#overrideDefaultValue
```

Формат значений по умолчанию

1. **Ссылочные поля:** ID ссылочного объекта.
 2. **Переменная ссылочность:** GID ссылочного объекта.
 3. **Числа:** десятичный разделитель — точка.
 4. **Дата:** формат dd.MM.yyyy HH:mm:ss.
Значение sysdate → текущая дата при вставке объекта.
-

Программная установка значений JSON-характеристик

В бизнес-логике

```
val rop = Bs_GoodsApi.load(209851.nl)  
// Установка ссылочной характеристики  
Bs_GoodsApi().setAttrValue(rop, "idGradeJson", 77016.nl)
```

В Jexl-скрипте

```
var rop = Bs_GoodsApi.load(209851L); // Установка ссылочной характеристики  
Bs_GoodsApi.setAttrValue(rop, «idGradeJson», 77016L);
```

Проверка значений объектных характеристик

Для валидации характеристик доступны два типа правил:

1. **Проверка значений атрибутов** — контроль корректности вводимых данных.
2. **Ограничение выбора ссылочных атрибутов** — фильтрация доступных значений.

Совет

Ограничивающие атрибуты

При изменении ограничивающего атрибута проверяются правила зависимых атрибутов. При нарушении — значение обнуляется.

Правила настраиваются в закладке:

Настройки и сервисы > Правила для атрибута.

Проверка значений атрибутов

Контролирует значения, устанавливаемые пользователем. Каждое правило — Jexl-скрипт, возвращающий логическое выражение:

- **true** — проверка пройдена.
- **false** — проверка не пройдена. Отображается сообщение об ошибке из карточки правила.

Приоритет применения правил

1. Уровень типа объекта (или шаблона атрибутов).
2. Уровень группы.

Примечание

Правила наследуются от родительских групп. Возможна переустановка на детализации «Правила для атрибута». Используется переопределённая версия.

3. Уровень характеристики.

Типы вызова проверки

- **На установку значения характеристики**
Проверка после завершения редактирования.
 - **На завершение редактирования объекта**
Проверка на событии `afterEdit`, после сохранения.
-

Ограничение выбора атрибутов

Ограничивает список доступных значений для ссылочных атрибутов.

Правило — Jexl-скрипт, возвращающий SQL-условие для таблицы ссылочного класса.

Если текущее значение не удовлетворяет условию — оно обнуляется.

Проверка выполняется рекурсивно: обнуление одного атрибута вызывает проверку зависимых.

Примечание

В контексте Jexl-скрипта доступна ROP объекта, которому принадлежит атрибут.

Внутренняя логика работы ограничения значений

JEXL-скрипт должен возвращать **SQL-условие в виде строки**. Это условие вставляется в WHERE финального запроса к таблице ссылочного класса.

Пример скрипта:

```
if (nNum < 5) { "upper(&sname) like '%БОЛТ%'" } else { "upper(&sname) like '%ВИНТ%'" }
```

При nNum = 7 результат: "upper(&sname) like '%ВИНТ%'".

Это условие будет использовано в SQL: `SELECT * FROM (SELECT * FROM Bs_Goods) t WHERE upper(t.sname) like '%ВИНТ%'`

Важно:

- Символ & заменяется на алиас таблицы (t.sName, g.sCaption и т.п.).
- Возвращённая строка становится частью условия WHERE.
- Правила не работают с выборками, сформированными объектным запросом — такие источники не поддерживают дополнительные SQL-условия.

Параметры правила

Позволяют переиспользовать одно правило на разных объектах. Значения параметров задаются на уровне объекта.

Пример

В классе `SupplyRequest` есть атрибут `idWarehouse`. Нужно ограничить выбор складов по региону. Регион может быть:

- В самом `SupplyRequest` (`idRegion`).
- В связанном контрагенте (`Bs_Contras.idRegion`), на который ссылается `idContras`.

Правило одинаковое:

```
"&idRegion = " + regionId
```

Скрипты получения `regionId` могут отличаться:

1. Через атрибут объекта:

```
rop.idRegion;
```

2. Через контрагента:

```
val ropContras = Bs_ContrasApi().load(rop.idContras);  
ropContras.idRegion;
```

Скрипты правила и параметров

Параметры могут быть:

- Значением.
- Ссылкой.
- **Jexl-скриптом**, исполняемым перед проверкой.

Для указания скрипта установите флажок «Jexl-скрипт» в детализации «Параметры проверки» карточки правила.

Значение скрипта можно настроить на объекте в закладке «Jexl-скрипт» детализации: Правила для атрибута > Параметр для правила атрибута.

Совет

В скрипте правила алиас таблицы можно заменять символом `&`.
Например, вместо `t.idObjectType` пишите `&idObjectType`.

Пример правила

Цель: для атрибута `idGoods` (класс `RplTst_AllDbTypes`) показывать только болты или винты в зависимости от значения `nNum`.

JEXL-скрипт правила:

```
if (nNum < 5) {  
    "upper(&sname) like '%БОЛТ%'"  
} else {  
    "upper(&sname) like '%ВИНТ%'"  
}
```

JEXL-скрипт параметра `nNum`:

```
RplTst_AllDbTypesApi.getAttrValue(rop, "nNumber");
```

Ограничивающие атрибуты

Обеспечивают согласованность значений между параметрами и коллекциями.

При изменении ограничивающего атрибута проверяются правила зависимых атрибутов — рекурсивно до конца цепочки.

Ограничивающий атрибут может быть:

- Атрибутом самого объекта.
- Атрибутом его родителя (бизнес-объекта), если зависимый атрибут — в коллекции.

Типы настройки:

- На типе объекта.
- На группе.
- На шаблоне атрибутов типа объекта.

Примечание

Обнуление происходит через стандартные сеттеры:

- Для **атрибутов БД** — сеттеры в `Api` класса.
- Для **JSON-характеристик** — `JObjectAttrApi.setObjAttrValue(rop: AnyRop, idpAttr: NLong, pValue: Any)`.

Установка и изменение правил валидации

Правила настраиваются в детализации «Правила для атрибута»:

- **На тип объекта**
В закладке **Объектные характеристики** → **Сущности** > **Типы объектов** или в карточке класса.
- **На группу**
В закладке **Атрибуты группы** при редактировании группы.
- **На шаблон атрибутов**
В закладке **Объектные характеристики** → **Сущности** > **Типы объектов** или в карточке класса.

Кэширование правил

- **Правила проверки значений:** кэшируются при загрузке выборки.
→ **Необходимо переоткрыть выборку** после изменения.
- **Правила ограничения и ограничивающие атрибуты:** кэшируются стандартно.
→ После настройки выполните **«Очистить все кэши»** в меню **Сервис** > **Управление решением**.

Внимание

Параметры правила на объекте являются **обязательными**.

Если хотя бы один не заполнен — все сеттеры в классе будут падать с ошибкой:

Заполнены не все параметры правила "<заголовок правила>" на атрибуте типа "<заголовок типа атрибута>" с id = <идентификатор атрибута>

Настройки логирования

Механизм управления логированием на уровне отдельных методов класса обеспечивает контроль над записью в журнал событий без вмешательства в бизнес-логику приложения.

Он позволяет включать логирование точно — только для нужных методов, типов документов или уровней сообщений (INFO, WARNING и т.д.).

Это используется для:

- Отладки — логирование включается только для проблемного сценария, не засоряя общий лог.
- Мониторинга ошибок — можно настроить логирование только для WARNING и ERROR, игнорируя информационные сообщения.
- Аудита — все изменения важных объектов фиксируются с привязкой к контексту и тегами.

Всё это работает без изменения кода и перезапуска — настройки применяются динамически.

Для формирования механизма управления логированием разработчик в коде:

1. Регистрирует настройки логирования для методов класса.
2. Задаёт условия применения логирования для этих настроек.
3. Использует логирующую сессию, чтобы записи в лог учитывали настройки и условия.

Совет

В стандартной поставке существует ограниченный набор классов, на которых включено логирование.

Настройки можно открыть через раздел **Сущности - Классы**, открыть карточку нужного класса, в карточке открыть закладку **Настройки логирования**.

Ключевые возможности

- Логирование можно настроить точно для конкретных методов, типов объектов или подклассов.
- Поддержка иерархического дерева методов, при котором активность настроек наследуется от родительских узлов, а условия проверяются только для текущего метода.
- Автоматическая фильтрация записей по уровню логирования (ошибка, информационное сообщение, предупреждение и др.) с учётом иерархии, заданной пользователем.
- Регистрация настроек программно через API.

Структура

Лог

Btk_Log «Лог» — V-коллекция, используемая для хранения записей о событиях, действиях пользователей и системных операциях в Global. В отличие от стандартного аудита, лог позволяет выполнить пользовательскую настройку записываемых в таблицу сообщений.

Структура данных **Btk_Log** позволяет сохранять информацию о:

- Объекте-источнике (экземпляр класса).
- Вызываемой процедуре (процедуре, инициировавшей запись в лог).
- Типе лога.
- Логирующей сессии (транзакции, в рамках которой была произведена вставка в лог).
- Теге (служит для структурирования информации).
- Тексте лога.
- Вызываемом методе (текстовое поле, в котором описана цепочка вызовов методов, приведших к записи).
Пример: `api1.proc1 -> api2.proc2`.
- Дополнительных характеристиках (в формате JSON).

Использования лога

Для обеспечения записи сообщения в лог необходимо выполнить следующие шаги:

1. Определить класс, к которому подключается **Btk_Log**, объекты которого будут выступать в качестве источника записи в лог.
2. В программный код добавить создание логирующей сессии.
3. В программный код добавить обработку, выполняющую вставку записи в таблицу **Btk_Log** при срабатывании соответствующего триггера события.
4. Выполнить настройку логирования для разграничения записей.

Для консультантов по внедрению Global ERP

Для обеспечения записи сообщения в лог необходимо создать задачу (ДП) с указанием класса, к которому необходимо подключить **Btk_Log**, и описанием содержимого лога (значения, устанавливаемые в поля лога).

Дерево настроек логирования

На закладке «Настройка логирования» для класса отображается дерево последовательно вызванных методов, на которых происходит запись в лог.

Системное имя: Тип: Модуль: Подключены системные миксины: ☐ Отслеживать изменения для включения в репиз конфигурации: ☐

Наименование: Не используется: ☐ Дата окончания использования:

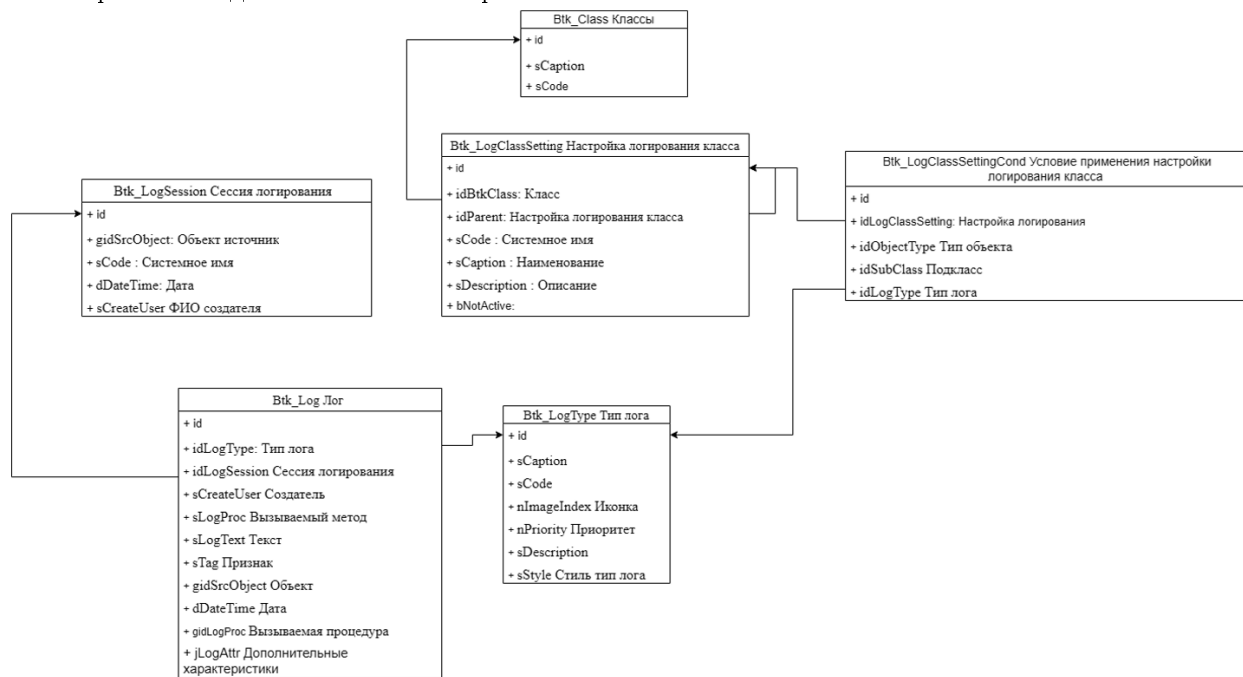
Описание:

8. Класс ссылается на ... 9. На класс ссылаются ... 10. Уникальные атрибуты класса 11. Класс входит в Бизнес объекты... 12. Функциональные настройки 13. Настройка логирования

Системное имя	Наименование	Описание	Настройка логирования	Не используется
Pm_CBStatementDocApi.processDocsByCBS			INFO	☐
Pm_CBStatementDocApi.processDocs			INFO; WARNING	☐

Pm_CBStatementDocApi.processDocsByCBS (родитель)
└ Pm_CBStatementDocApi.processDocs (дочерний метод)

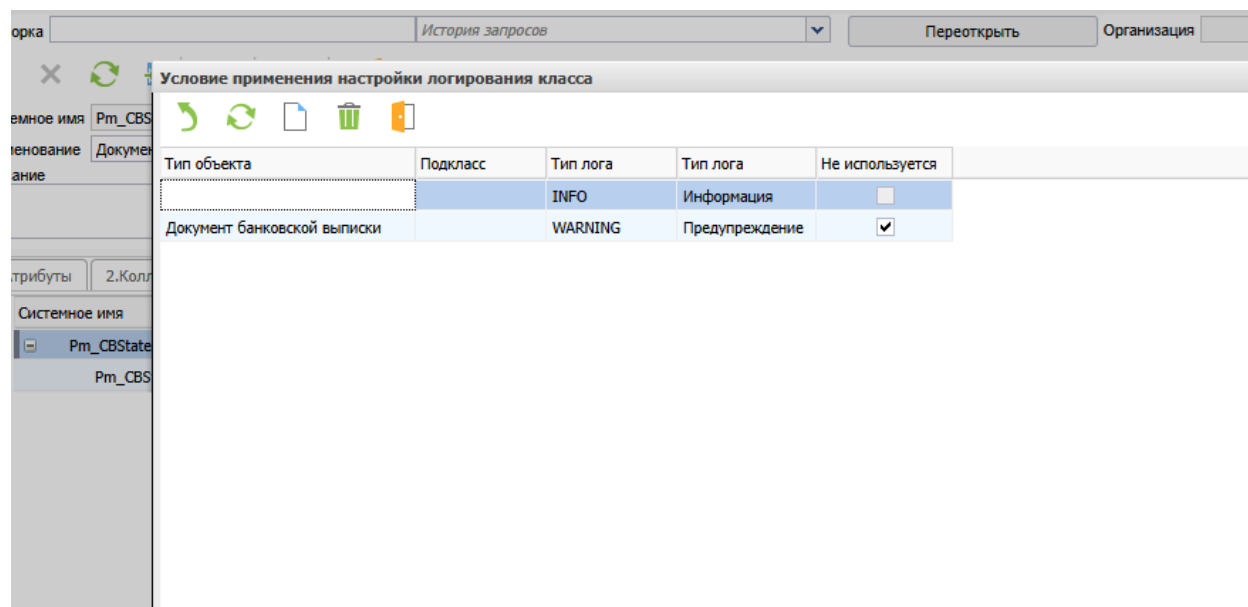
Ниже приведена ER-диаграмма связи Btk_Log, настроечных классов и непосредственно самого класса, на котором необходимо включить логирование.



Условия логирования

Условия логирования определяют, разрешено ли записывать сообщения определённого уровня (например, INFO, WARNING) для конкретного метода с учётом контекста выполнения: класса, подкласса или типа объекта.

Открыть условия можно через кнопку ... в колонке **Настройки логирования**.



Зачем нужны условия?

- Фильтрация записей в лог в зависимости от:
 - Уровня важности (INFO, WARNING, ERROR и т.д.).
 - Типа объекта (например, только для документов типа «Выписка банка»).
 - Подкласса (например, только для исходящих документов).
- Сокращение объёма логов в продакшене.
- Включение детального логирования только для определённых сценариев или объектов.

Как работают условия?

1. Регистрация условия происходит через `Btk_LogClassSettingCondApi()` или через интерфейс.
2. При записи в лог (с `bUseSetting = true` и внутри сессии) система:
 - Определяет, какой метод вызывает `insertMsg`.
 - Проверяет, есть ли для него зарегистрированные условия.
 - Сравнивает тип логирования сообщения с разрешёнными типами в условиях.
 - Применяет иерархию уровней: если разрешён INFO, то разрешены и WARNING, ERROR. Если разрешён WARNING — INFO не разрешён.
3. Условия не наследуются от родителей — каждый метод должен иметь свои условия.
4. Условия для дочернего метода проверяются только в том случае, если все вышестоящие методы в иерархии имеют активные настройки логирования (даже если для них не заданы условия).

Логирующая сессия

Логирующая сессия — это выполненная пользователем транзакция, в рамках которой система отслеживает стек вызовов методов и применяет зарегистрированные и активные для класса настройки логирования.

Без логирующей сессии:

- Запись в лог происходит напрямую, как обычный вызов `insertMsg`.
- Настройки класса игнорируются полностью, даже если указано `bUseSetting = true`.
- Нет возможности применить фильтрацию по типу объекта, подклассу или уровню логирования.

Зачем нужна сессия:

- Объединение нескольких логируемых сообщений в одну сессию.
- Изоляция настроек — сессия гарантирует, что настройки одного потока/процесса не влияют на другие. Даже если создается дочерняя логирующая сессия для записи лога, то исходная сессия не изменится.
- Определение контекста вызова — система знает, в каком методе происходит запись и кто его вызвал.
- Применение условий — только внутри сессии система проверяет, разрешено ли писать лог для текущего метода, типа объекта, уровня логирования и т.д.

Как работает:

При вызове `Btk_LogApi().withLogSession(...)` система:

- Сохраняет в контексте имя метода и его идентификатор.
- Контекст (последовательность вызовов) отслеживается для каждого класса отдельно, что предотвращает дублирование цепочек при использовании методов из других классов.
- Начинает отслеживать все последующие вызовы `insertMsg(..., bUseSetting = true)` внутри блока.
- При записи в лог:
 - Проверяет, зарегистрирован ли метод в дереве настроек.
 - Проверяет, активны ли все его родители.
 - Применяет только его собственные условия.
 - Принимает решение о записи сообщения в лог.

Ключевые API:

```
Btk_LogApi (основной API для записи логов)
├── Btk_LogClassSettingApi (регистрация методов в дереве настроек)
├── Btk_LogTypeApi (управление типами логов)
├── Btk_LogClassSettingCondApi (управление условиями логирования)
└── Btk_LogSession (управление контекстом логирования)
```

Создание

Создание настроек

Настройку можно задать через код и через интерфейс системы. По умолчанию рекомендуется создавать настройки через код — именно этот способ используется в стандартной реализации.

Настройка через интерфейс предназначена для нетиповых сценариев, когда требуется создать пользовательские методы, отличные от тех, что создаются по умолчанию (например, JEXL-скрипты).

Данный метод следует использовать исключительно при наличии обоснованного сценария и полного понимания его влияния на систему.

Регистрация методов, на которых происходит запись в лог:

```
//Регистрация верхнеуровневой (родительской) настройки для метода Pm_  
↪CBStatementDocApi.processDocsByCBS  
    val idSettingParent = Btk_LogClassSettingApi().register("Pm_CBStatementDocApi.  
↪processDocsByCBS", idClass)  
    //Регистрация дочерней настройки для метода Pm_CBStatementDocApi.processDocs_  
↪вызываемого из метода Pm_CBStatementDocApi.processDocsByCBS (с указанием родителя)  
    val idSettingChild = Btk_LogClassSettingApi().register("Pm_CBStatementDocApi.  
↪processDocs", idClass, idSettingParent)
```

Создание условий

Условия можно задать через код и через интерфейс системы.

Регистрация условий, по которым проверяется настройка логирования:

```
//Регистрация условия на классе  
    Btk_LogClassSettingCondApi().registerByClass(idSettingParent, Btk_LogTypeApi().  
↪findByMnemonicCode("INFO"))  
    //Регистрация условия на типе объекта  
    Btk_LogClassSettingCondApi().registerByObjectType(idSettingChild, Btk_LogTypeApi().  
↪findByMnemonicCode("INFO"), idOTCBStatementDoc)  
    //Регистрация условия на подклассе  
    Btk_LogClassSettingCondApi().registerBySubClass(idSettingChild, Btk_LogTypeApi().  
↪findByMnemonicCode("WARNING"), idSBCBStatementDoc)
```

Создание логирующей сессии

Логирующая сессия задается только из кода.

Пример регистрации логирующей сессии и записи в лог с учетом настроек:

```
// Открытие логирующей сессии от объекта логирования  
def processDocsByCBS(idpCBStatement: NLong): Unit = {  
    val ropvCBStatement = Pm_ClientBankStatementApi().load(idpCBStatement)  
    // Открытия логирующей сессии  
    Btk_LogApi().withLogSession("Pm_CBStatementDocApi.processDocsByCBS", ropvCBStatement.  
↪gid) {  
        // Запись в лог  
        Btk_LogApi().insertMsg(  

```

(продолжается на следующей странице)

```

        "INFO"
        ,ropvCBStatement
        ,"Начало обработки банковской выписки"
        ,"Обработка банковской выписки"
        ,bUseSetting = true // Параметр указывает, что при записи учитывается настройка
↪ на классе
    )
    // исполняемая процедура
    processDocs(refreshByParent(ropvCBStatement).map(_.get(_.id)).toList)
}
}

// Открытие логирующей сессии от класса
def processDocs(idapCBStatementDoc: List[NLong]): Unit = {
    // Открытия логирующей сессии
    Btk_LogApi().withLogSession("Pm_CBStatementDocApi.processDocsByCBS", idClass) {
        // исполняемая процедура
        // Запись в лог
        Btk_LogApi().insertMsg(
            "INFO"
            ,ropvCBStatementDoc
            ,"Завершение обработки банковской выписки"
            ,"Обработка банковской выписки"
            ,bUseSetting = true // Параметр указывает, что при записи учитывается настройка
↪ на классе
        )
    }
}

```

Примечание

Для избежания ошибок в коде настройки, регистрируемой на классе и указываемой на сессии, рекомендуется создавать на классе поле с кодом.

Пример:

```

    lazy val sLogSettingProcessDocsByCBS = "Pm_CBStatementDocApi.processDocsByCBS" //
↪ sLSProcessDocsByCBS
    lazy val sLogSettingProcessDocs = "Pm_CBStatementDocApi.processDocs" //
↪ sLSProcessDocs

```

Использование

Примеры использования

Интерпретация банковской выписки:

```

def processDocsByCBS(idpCBStatement: NLong): Unit = {
    val ropvCBStatement = Pm_ClientBankStatementApi().load(idpCBStatement)
    Btk_LogApi().withLogSession(sLogSettingProcessDocsByCBS, ropvCBStatement.gid) {

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
val sStartProcMsg = s"Запущена интерпретация выписки ${ropvCBStatement.get(_sNum)}  
→ (${ropvCBStatement.gid}) от ${ropvCBStatement.get(_dStatement).toString("dd.MM.yyyy  
→ hh:mm:ss")}. "  
Btk_LogApi().atInsertMsg("INFO", ropvCBStatement, sStartProcMsg, "", bUseSetting =  
→ true)  
Btk_InfoLogPkg().info(sStartProcMsg)  
  
val idavCBStatementDoc = refreshByParent(ropvCBStatement).map(_get(_id))  
processDocs(idavCBStatementDoc.toList)  
  
val sFinishProcMsg = s"Завершена интерпретация выписки ${ropvCBStatement.get(_  
→ sNum)} (${ropvCBStatement.gid}) от ${ropvCBStatement.get(_dStatement).toString("dd.MM.  
→ yyyy hh:mm:ss")}. "  
Btk_LogApi().atInsertMsg("INFO", ropvCBStatement, sFinishProcMsg, "", bUseSetting  
→ = true)  
Btk_InfoLogPkg().info(sFinishProcMsg)  
}  
}
```

Вынесение логики в отдельный метод:

```
def isDocProcessed(rop: Pm_CBStatementDocApi#ApiRop): Boolean = {  
    val ropvCBStatement = Pm_ClientBankStatementApi().load(rop.get(_  
→ idClientBankStatement))  
    Btk_LogApi().withLogSession(sLogSettingIsDocProcessed, rop.gid) {  
        if (rop.get(_idStateMC) > 100.nr) {  
            val sErrorMsg = s"Процедура интерпретации позиции банковской выписки ${rop.  
→ get(_sNumDoc)} по документу ${ropvCBStatement.get(_sNum)} (${ropvCBStatement.gid})  
→ от ${ropvCBStatement.get(_dStatement).toString("dd.MM.yyyy hh:mm:ss")} невозможна,  
→ так как она уже обработана пользователем"  
            Btk_LogApi().atInsertMsg("ERROR", rop, sErrorMsg, "", bUseSetting = true)  
            Btk_InfoLogPkg().error(sErrorMsg)  
            true  
        } else {  
            val sInfoMsg = s"Проверка пользовательской обработки позиции выписки ${rop.  
→ get(_sNumDoc)} по документу ${ropvCBStatement.get(_sNum)} (${ropvCBStatement.gid})  
→ от ${ropvCBStatement.get(_dStatement).toString("dd.MM.yyyy hh:mm:ss")}"  
            Btk_LogApi().atInsertMsg("INFO", rop, sInfoMsg, "", bUseSetting = true)  
            Btk_InfoLogPkg().info(sInfoMsg)  
            false  
        }  
    }  
}
```

Дополнительные примеры настройки и использования можно посмотреть в реализации:

- ru.bitec.app.pm.clientbank.Pm_CBStatementDocApi#regLogSetting
- ru.bitec.app.pm.clientbank.Pm_CBStatementDocApi#processDocs

Результат логирования

Результат логирования сохраняется в журнале событий, который настраивается со стороны заказчика.

Дата с	по	Тип лога	Создатель	Номер логируемой сессии	Вызываемый метод	Тип лога	Событие	Текст
12.04.2019 15:51:51	107303/16407	Круг h10 - 9.7 ГОСТ 14955-77 30хГСА - В - Н ...				Информация		Ответ обработан. GID запроса: 107303/16407
12.04.2019 15:51:51	107303/16407	Круг h10 - 9.7 ГОСТ 14955-77 30хГСА - В - Н ...				Информация		Начало обработки ответа. GID запроса: 107303/16407
12.04.2019 15:51:51	107303/16408	Круг				Информация		Начало обработки ответа. GID запроса: 107303/16408
12.04.2019 15:51:51	107303/16409	Круг h10				Информация		Начало обработки ответа. GID запроса: 107303/16409
12.04.2019 15:51:51	107303/16406					Информация		Начало обработки ответа. GID запроса: 107303/16406
12.04.2019 15:51:51	107303/16406					Информация		Ответ обработан. GID запроса: 107303/16406
12.04.2019 15:51:51	107303/16410	Круг h10 -				Информация		Ответ обработан. GID запроса: 107303/16410
12.04.2019 15:51:51	107303/16410	Круг h10 -				Информация		Начало обработки ответа. GID запроса: 107303/16410
12.04.2019 15:51:51	107303/16409	Круг h10				Информация		Ответ обработан. GID запроса: 107303/16409
12.04.2019 15:51:51	107303/16408	Круг				Информация		Ответ обработан. GID запроса: 107303/16408
12.04.2019 15:51:51	107303/16408	Круг				Ошибка		Ошибка при обработке ответа. GID запроса: 107303/16408 Exception wrapping
12.04.2019 16:20:17	107303/16408	Круг				Информация		Ответ обработан. GID запроса: 107303/16408
12.04.2019 16:20:17	107303/16408	Круг				Информация		Начало обработки ответа. GID запроса: 107303/16408
12.04.2019 16:21:48	107303/16408	Круг				Информация		Ответ обработан. GID запроса: 107303/16408

Все логи системы лежат в **Настройки системы - Сущности - Классы - btk_log**.

Чистка журнала логов

Очистка логов для классов с преднастроенным логированием выполняется автоматически. Она настроена в типовом задании очистки устаревших данных с окном хранения 10 дней в менеджере заданий.

Примечание

Для ручной очистки реализован API-метод `clearBtkLog()`, принимающий на вход системное имя класса и количество дней сдвига. К нему подключен джоб, автоматически очищающий таблицу.

При создании новых методов логирования необходимо зарегистрировать соответствующую запись в этом задании. Это обеспечивает автоматическое удаление устаревших записей логов и предотвращает неконтролируемый рост объема данных.

Детальнее про задания очистки смотри в разделе «Менеджер заданий»

Расширение

Для добавления новых методов в систему логирования:

1. Зарегистрируйте метод через `Btk_LogClassSettingApi().register()`.
2. Определите родительский метод в иерархии вызовов.
3. Настройте условия логирования через `Btk_LogClassSettingCondApi()`.
4. Используйте константы для имен методов во избежание ошибок.

Ограничения

- Уровень WARNING не включает INFO — фильтрация работает только на повышение уровня (INFO включает WARNING и ERROR; WARNING включает только WARNING и ERROR).
- Нет автоматического построения дерева вызовов — вся иерархия регистрируется вручную.
- Нет встроенной валидации — можно зарегистрировать несуществующий метод.
- Запись в лог происходит только при выполнении всех условий:
 - Метод присутствует в дереве настроек.
 - Все вышестоящие методы активны.
 - Для текущего метода выполняются условия логирования.

Тип объекта

Сервис предоставляет набор стандартных возможностей, доступных для всех типизируемых классов. Основным назначением типа объекта является хранение настроек по документу, которые могут быть переопределены на проекте.

Основные настройки:

- Настройка закладок
- Настройка печатных форм
- Настройка переходов состояний
- Настройка объектных характеристик

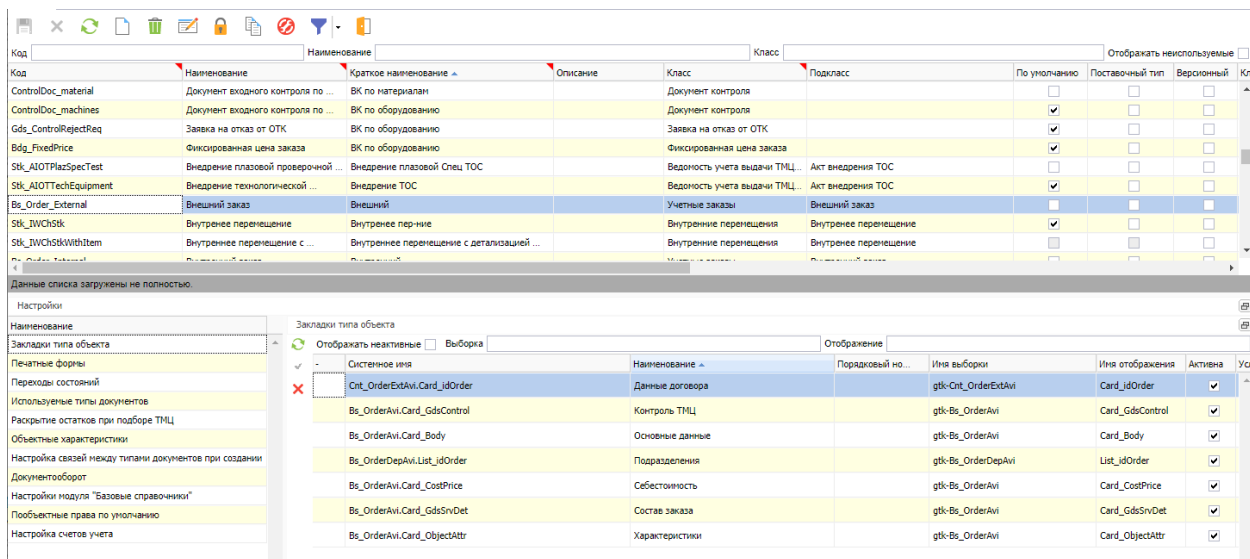
Также предусмотрены механизмы для подключения настроек для модуля или подсистемы.

Тип объекта может менять поведение документа в зависимости от бизнес процесса, в котором данный документ участвует. В случае, если необходимо запрограммировать поведение документа в зависимости от типа объект следует использовать системное имя подкласса.

Внимание

Использовать системные имена типов объекта в коде нельзя, так как типы объектов могут быть добавлены в проекте.

На базе типа объекта реализовано множество сервисных возможностей, позволяющих тонко настраивать типизированные документы не привлекая программистов.



Подключение типа объекта к классу

Примечание

Для супертипа document (Документ) ссылка на тип `idObjectType` создается автоматически

Пример объявления атрибута:

```
<attr name="idObjectType"
      attribute-type="Long"
      caption="Тип объекта"
      order="50"
      type="refObject"
      isVisible="true"
      ref.class="Btk_ObjectType"/>
```

Система имеет общий справочник типов объекта (Настройка системы \ Сущности > Типы объектов > Типы объектов)

Можно регистрировать типы объекта в коде *Регистрация в коде*

Объекты класса типизируются классом-настройкой `Btk_ObjectType`. В данной настройке можно задать несколько типов для одного класса системы, а также назначить тип по умолчанию. Обязательным условием для типизации объектов является наличие в типизируемом классе атрибута-ссылки на `Btk_ObjectType`. Атрибут обязательно должен называться `idObjectType`. Это имя зарезервировано фреймворком и не может использоваться для других целей.

Подкласс

Подкласс – это логическая типизация для разделения бизнес-логики в рамках одного класса. Подклассы добавляются разработчиками и на них можно завязываться в коде. Несколько типов объекта может быть привязано к одному подклассу

Система имеет общий справочник подклассов (Настройка системы \ Сущности > Типы объектов > Подклассы), в котором настраивается:

- Код
- Наименование
- Суперкласс (Класс к которому будет привязан данный подкласс)

Можно регистрировать подклассы в коде *Регистрация в коде*

Примечание

Системные имена подклассов можно использовать в коде.

Стандартные настройки

Закладки типа объектов

Для типизированного объекта доступна возможность управления выводом закладок.

Система имеет общий справочник закладок (Настройка системы \ Сущности > Закладки детализации), в котором настраивается:

- имя выборки
- отображение
- наименование
- условие вывода
- иконка
- перечень классов, для которых возможен вывод закладки.

Можно регистрировать закладки в коде *Регистрация в коде*

Связь с типом объекта осуществляется по списку допустимых классов. Чтобы включить отображение закладки для типа объекта нужно:

1. Откройте Настройка системы \ Сущности > Типы объектов > Типы объектов
2. Выберите настройку Закладки типа объектов
3. Отобразите все закладки
Для этого на панели фильтрации закладок снимите флажок Отображать неактивные
4. Выберите требуемую закладку
5. Установите на ней флажок «Активна»

Можно регистрировать закладки для типа в коде *Регистрация в коде*

По умолчанию в разметке выборки Avn сервис настройки закладок не используется. Программист должен указать для нужного отображения динамическое получение закладок от типа объекта.

Существует два варианта вывода детализации фрейма: в виде закладок и в виде списка

Детализация в виде закладок

Набор закладок отображается либо снизу, либо справа от карточки или списка

Для подключения в Avm нужно указать выборку с динамическими закладками

```
<tabComposer>
  <frame filter.isVisible="false">
    <card>
      <layout>
        <hBox>
          <attr name="sNumber"/>
          <attr name="sCaption"/>
          <attr name="idPrjVerHL"/>
        </hBox>
        <hBox>
          <attr name="idDepOwnerHL"/>
          <attr name="idObjectTypeHL"/>
          <attr name="idStateHL"/>
        </hBox>
      </layout>
    </card>
  </frame>
  <tabItems isVisible="true"
    selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"
    representation="List_Tab"
    selection.selectionAttr="SSELECTIONNAME"
    selection.representationAttr="SREPRESENTATIONNAME"
    selection.captionAttr="SCAPTION"
    selection.imageIndexAttr="NIMAGE"
    selection.paramsAttr="JSONPARAMS"

    />
</tabComposer>
```

Детализация в виде списка закладок

Набор закладок отображается в виде списка закладок, справа от которого выводится содержимое текущей закладки. Для подключения в Avm нужно указать выборку с динамическим содержимым.

```
<dynamicComposer>
  <frame filter.isVisible="false"
    header.isWide="true"
    toolBar.isCaptionsVisible="true">
    <card useMasterData="false"/>
  </frame>
  <dynamicItems masterAlign="top">
    <dynamicItem align="client"
```

(продолжается на следующей странице)

```

representation="DynList"
selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"/>
</dynamicItems>
</dynamicComposer>

```

Параметры динамических закладок

Дополнительные атрибуты для динамических закладок, настраиваемые на типе объекта, могут использоваться для изменения поведения детализации. Для этого им передают уникальные значения или псевдонимы, которые затем используются для получения данных.

За хранение параметров отвечает атрибут `Btk_ObjectTypeTab#sParamsClause`, использующий формат **JSON**

Параметр `sParamsClause` может быть настроен двумя способами:

- При регистрации закладки типа объекта через код в функции `Btk_ObjectTypeTabApi#registerTab`
- В интерфейсе приложения в поле Параметры выборки на детализации Закладки типа объекта в пункте меню Настройка системы \ Сущности > Типы объектов > Типы объектов

Примечание

Значение переменной `sParamsClause` явно преобразуются к типу **JSONB**, предоставляемому **PostgreSQL**, поэтому должно являться валидным **JSON**-объектом, например:

- Используя функцию `jsonb_build_object('key1', value1, 'key2', value2)`
- Строкой типа `'{"key1": value1, "key2": value2}'`

Итоговый **JSON**-объект доступен как атрибут выборки `Btk_ObjectTypeTabAvi.List_Tab` или `Btk_ObjectTypeTabAvi.DynList` и может быть получен с помощью функции `Btk_ObjectTypeLib#tabParamsJSON`, откуда по ключу можно достать нужные значения

Получение атрибутов выборки

Кроме стандартных типов данных, используемых **JSON**, значение внутри объекта может быть псевдонимом, соответствующим любому доступному параметру выборки. Так, например, для получения идентификатора объекта мастер-детали можно использовать выражение `:super$id`. Тогда итоговый **JSON** будет выглядеть так: `jsonb_build_object('idMaster', :super$id)`

Печатные формы

Позволяет настраивать список печатных форм для типа объекта. Настроенные печатные формы будут доступны под операцией печати для карточки объекта.

Печатные формы добавляются из общего справочника и могут быть активированы для типа объекта. Доступ к формированию печатной формы может регулироваться состоянием объекта.

Переходы состояний

Состояния класса позволяют управлять бизнес-логикой объекта: доступ на редактирование атрибутов, возможность выполнения операций в зависимости от состояний и т.д.

Настройка состояний подразумевает наличие в классе атрибута, ссылающегося на состояние и типа объекта.

```
<attr name="idState"
      attribute-type="Long"
      caption="Состояние"
      order="40"
      type="refObject"
      ref.class="Btk_ClassState"/>
```

Для класса, имеющего атрибут, ссылающийся на состояние, необходимо определить перечень возможных для него состояний.

Список состояний можно настроить в карточке класса, либо вручную, через метод `*Api().regState` *Регистрация в коде*. При создании состояний необходимо помнить, что у класса должно присутствовать ровно одно начальное состояние.

Возможные переходы между состояниями настраиваются для типа объекта на закладке **Переходы состояний**. Можно регистрировать переходы состояний для типа в коде *Регистрация в коде*.

Сеттеры состояний никак не проверяют эти переходы, они влияют только на выпадающий список состояний.

Выборка для выпадающего списка состояний класса, в которые возможен переход от текущего: `gtk-ru.bitec.app.btk.Btk_ClassStateAvi#Lookup_Class`

Пример разметки атрибута в Avm:

```
<attr name="idStateHL" caption="Состояние" order="30.1" isLastInLine="false">
  <editor>
    <lookup lookupQuery="gtk-ru.bitec.app.btk.Btk_ClassStateAvi#Lookup_Class"
            changeableAttr="idState"
            isLookupLazyLoad="true" lookupKeyAttr="id" lookupListAttr="sHeadLine"/>
  </editor>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
<card isControlWidthFixed="true" controlWidth="60"/>
<ref class="Btk_ClassState"/>
</attr>
```

Методы для работы с состояниями:

```
/**
 * Метод ищет состояние по сист. имени и сист. имени класса, к которому оно относится
 * @param spSystemName Сист. имя состояния
 * @param spMasterClass Сист. имя класса, к которому относится состояние
 * @param bpRaiseErr При значении = 1.пн будут создаваться исключения, если не найден
 * ↳ класс, либо нужное состояние
 */
Btk_ClassStateApi().findByNameAndClass(spSystemName:NString, spMasterClass:NString,
↳ bpRaiseErr:NNumber = 0.nn): NLong

/**
 * Возвращает начальное состояние для класса
 * @param idpMasterClass идентификатор класса
 */
Btk_ClassStateApi().getClassStartState(idpMasterClass: NLong) :NLong

/**
 * Возвращает номер состояния класса по его id
 */
Btk_ClassStateApi().getOrder(idp: NLong): NNumber
```

В прикладном коде не всегда удобно проверять ссылку на состояние, поэтому у каждого состояния есть номер. Фреймворк позволяет сохранять номер состояния в отдельном атрибуте.

Если в классе объявлен специальный атрибут с признаком `isStateMC=true`, фреймворк автоматически будет сохранять в него номер текущего состояния.

```
<attr name="idStateMC"
      attribute-type="Number"
      caption="Состояние"
      order="51"
      type="basic"
      isVisible="false"
      isStateMC="true" />
```

История состояний

Для классов, имеющих атрибут состояния (`idState`) по умолчанию, ведется история состояний. Данные хранятся в классе `Btk_ObjectStateHistory`. Через `dp`-сеттер состояния создаются записи в журнале изменения состояний.

Для отключения истории состояний необходимо воспользоваться атрибутом `isStateHistoryEnabled` тега `class`.

Для просмотра истории состояния объекта:

1. Откройте «Информацию об объекте»

2. Перейдите в закладку «История состояний» Откроется универсальная закладка `Btk_ObjectStateHistoryAvi.List_GidObj`, которая отображает историю состояний объекта. Работает от супер-параметра, в котором указан `gid`-объекта.

Возможные параметры закладки `Btk_ObjectStateHistoryAvi.List_GidObj`:

- `sGidAttrName` - имя атрибута, в котором хранится `gid`-объекта из мастер выборки. По умолчанию `gid`.
- `bWithStack` - признак необходимости отображения стека вызова. Доступные значения: 0, 1. По умолчанию 0.

Пример использования параметров, при подключении закладки через разметку выборки:

```
<tabItem selection="gtk-Btk_ObjectStateHistoryAvi"
  representation="List_GidObj"
  caption="История состояний">
  <params>
    <param name="sGidAttrName"
      value="gid"
      dataType="String"/>
    <param name="bWithStack"
      value="1"
      dataType="Number"/>
  </params>
</tabItem>
```

Настройка вызова процедур на переход между состояниями

В разрезе типа объекта возможно настроить перечень процедур, которые будут выполнены при переходе из одного состояния в другое.

Есть несколько типов событий, на которые можно настроить вызов произвольных процедур:

- **На вход в состояние** - будут вызваны, если это состояние будет конечным при любом переходе. Настраивается в детализации к состоянию.
- **На выход из состояния** - будут вызваны, если это состояние будет начальным при любом переходе. Настраивается в детализации к состоянию.
- **На переход** - будут вызваны, если будет осуществлен именно этот переход. Настраивается в детализации к переходу.

Порядок вызова событий:

1. Выполнение процедур на выход из состояния
2. Выполнение процедур на переход
3. Выполнение процедур на вход в состояние

Доступные переменные в `jexl`-скрипте:

- `rop` - `rop` объекта, для которого была вызвана смена состояния.
- `idOldState` - старое состояние объекта
- `idNewState` - новое состояние объекта

Объектные характеристики

Можно настроить вывод объектных характеристик на каждый тип объекта. На закладке характеристик учитывается настройка объектных характеристик на тип объекта.

Пообъектные права по умолчанию

На закладке можно настроить права доступа по умолчанию для пообъектного режима доступа. Настройка происходит в списке с атрибутами:

- Пользователь
- Роль
- Вид роли ОФС
- Группа доступа

Запрет редактирования объектов после создания

Механизм запрета редактирования даёт администраторам простой способ защитить критичные данные типизированных объектов (например, ключи интеграции) от случайного изменения или удаления после их первого сохранения

Флаг запрета редактирования задаётся на типе объекта: **Настройка системы > Сущности > Типы объектов > Свойства типа объекта**. Атрибут - "Запрещено редактирование объектов после создания"

Особенности реализации

Проверка происходит перед flush (сохранение дискеткой) на всех выборках, учитываются как параметры, хранящиеся в JSON, так и параметры в базе данных. Атрибуты-исключения обрабатываются по имени, без учёта регистра

Главный метод проверки находится в `Btk_ObjectTypePkg.validateReadOnly()`

Примечание

При попытке сохранения объекта с типом, запрещённым для редактирования, пользователь увидит ошибку:

*Запрещено редактировать объект класса <наименование класса> с типом <наименование типа объекта> после создания. Системные имена изменённых атрибутов:
- <системные имена атрибутов, запрещённых к редактированию>*

Атрибуты-исключения

Функционал предоставляет возможность отключить запрет на редактирование для конкретных параметров с помощью флага **Отключен запрет редактирования**, при этом проверки для остальных параметров класса сохраняются

Пути настройки исключений в интерфейсе:

- **Атрибуты класса**
Настройка системы > Сущности > Типы объектов > Управление атрибутами
- **Атрибуты типа объекта** (объектные и универсальные характеристики)
Настройка системы > Сущности > Объектные характеристики
- **Атрибуты шаблона**
Настройка системы > Настройки и сервисы > Шаблоны характеристик > Атрибуты шаблона

Примечание

Системные атрибуты (имя заканчивается на `_dz`) и `bNotUsed` - «Не используется», `dExpiryDate` - «Дата окончания использования» можно изменять даже при включённой настройкой "Запрещено редактирование объектов после создания"

Подключение индивидуальных настроек объекта

Настройки типа объекта могут быть расширены индивидуальными классами настроек. Такие настройки могут включать или отключать бизнес-логику, управлять фреймами, настраивать права доступа и т.д. в зависимости от поставленной задачи. Индивидуальная настройка добавляется в список настроек для типа документа.

Интерфейс доступен из приложения **Настройка системы**, меню: **Сущности > Типы объектов > Настройка детализации типов объектов**.

При создании собственной настройки для типа объекта можно указать список классов, у которых должна выводиться данная настройка. Если список классов пуст, то настройка выводится у всех типов объекта.

Также можно указать список подклассов, для которых требуется выводить настройку. Если список подклассов не указан, ограничения по подклассам не применяются.

Регистрация из кода

Регистрация закладок

Описание метода `Btk_TabApi().register`:

```
/**
 * Процедура для регистрации новой строки
 *
 * @param spSystemName  Необязательный параметр - Системное имя
 * @param spCaption      Название закладки
 * @param spSel          Выборка
 * @param spRep          Отображение
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
* @param idprefClass    Необязательный параметр - Класс на который регистрируется  
↪ закладка  
* @param spDescription Необязательный параметр - Описание  
* @param spCondition   Необязательный параметр - Условие появления  
* @param spManualFunction Необязательный параметр - Функция получения списка  
↪ закладок  
* @param idpDataClass  Необязательный параметр - Класс отображаемых в закладке  
↪ данных(для универсальных коллекций)  
* @param spMasterAttr  Необязательный параметр - Атрибут с мнемокодом атрибута,  
↪ через который класс из закладки будет ссылаться на мастера  
* @param nprImage      Необязательный параметр - Картинка по умолчанию  
*/
```

Пример ручной настройки Api класса `Stm_ActInApi.scala`:

```
def regTab(): Unit = {  
    session.commit()  
    session.setDefaultUOWEditType(RWSharedUOWET)  
  
    Btk_TabApi().register(  
        spCaption = "Товарные позиции"  
        , spSel = "gtk-Stm_ActInDetAvi"  
        , spRep = "List_idDoc"  
        , idprefClass = idClass  
    )  
  
    Btk_TabApi().register(  
        spCaption = "Складские ордера"  
        , spSel = "gtk-Stk_WarrantInAvi"  
        , spRep = "LinkDocs"  
        , idprefClass = idClass  
    )  
    session.commit()  
}
```

Пример регистрации скрипта в тэге `dbData` для odm класса `Stm_ActIn`:

```
<script name="regTab" version="1">  
    <install>Stm_ActInApi.regTab();</install>  
</script>
```

Регистрация состояний

Описание метода `Btk_ClassStateApi().register`:

```
/**  
* Метод регистрирует состояние для класса по сист. имени. Если остальные атрибуты не  
↪ переданы, они не изменяются  
* Переданные значения параметров записываются в объект, если он найден по id класса и  
↪ сист. имени состояния  
* @param idpMasterClass Класс, к которому относится состояние
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
* @param spSystemName  Сист. имя состояния - должно быть уникально в рамках класса, к  
→ которому относится состояние  
* @param spCaption      Необязательный параметр - наименование  
* @param bpStartState  Необязательный параметр - признак стартового состояния  
* @param npOrder       Необязательный параметр - порядковый номер состояния в классе  
*/
```

Кроме того есть перегруженная версия метода, где первый параметр – сист.имя класса

```
/**  
* @param spMasterClass Мнемокод класса, для которого регистрируем состояния  
*/
```

Пример ручной настройки Api класса Stm_ActInApi.scala:

```
def regState: Unit = {  
  session.commit()  
  session.setDefaultUOWEditType(RWSharedUOWET)  
  //Регистрация состояний  
  Btk_ClassStateApi().register(  
    idpMasterClass = idClass,  
    spSystemName = "Forming",  
    spCaption = "Формируется",  
    bpStartState = 1.nn,  
    npOrder = 100.nn)  
  session.flush() //чтоб было начальное состояние  
  Btk_ClassStateApi().register(  
    idpMasterClass = idClass,  
    spSystemName = "Executing",  
    spCaption = "Исполняется",  
    bpStartState = 0.nn,  
    npOrder = 200.nn)  
  Btk_ClassStateApi().register(  
    idpMasterClass = idClass,  
    spSystemName = "Done",  
    spCaption = "Выполнена",  
    bpStartState = 0.nn,  
    npOrder = 300.nn)  
  session.commit()  
}
```

Пример регистрации скрипта в тэге dbData для odm класса Stm_ActIn:

```
<script name="regState" version="1">  
  <install>Stm_ActInApi.regState();</install>  
</script>
```

Регистрация подкласса, типа объекта, закладок для типа и переходов состояний

Пример ручной настройки Api класса `Stm_ActInApi.scala`:

```
def regObjectType(bpNeedRefresh: Boolean = true): Unit = {
  session.commit()
  Btk_Pkg().setRWSharedUOWEditType()
  val idvInvoice = Btk_SubClassApi().register("invoiceIn", "Приходная накладная",
  ↪idClass, "").get(_id)
  session.flush()

  if (bpNeedRefresh || Btk_ObjectTypeApi().findByMnemonicCodeAndClass("Stm_ActInInvoice",
  ↪idClass).isNull) {
    //регистрируем тип
    val idvObjectType = Btk_ObjectTypeApi().register(
      spCode = "Stm_ActInInvoice".ns
      , spCaption = "Приходная накладная".ns
      , spShortCaption = "Приходная накладная".ns
      , idpRefClass = idClass
      , idpSubClass = idvInvoice
      , bpIsDefault = 1.nn
    )
    //ищем закладку и привязываем к типу
    var idvTab = Btk_TabApi().findByMnemonicCode("Stm_ActInDetAvi.List_idDoc")
    if (idvTab.isNotNull)
      Btk_ObjectTypeTabApi().registerTab(
        idpObjectType = idvObjectType
        , idpTab = idvTab
        , spCaption = "Товарные позиции"
        , npOrder = 10.nn
      )

    idvTab = Btk_TabApi().findByMnemonicCode("Stk_WarrantInAvi.LinkDocs")
    if (idvTab.isNotNull)
      Btk_ObjectTypeTabApi().registerTab(
        idpObjectType = idvObjectType
        , idpTab = idvTab
        , spCaption = "Складские ордера"
        , npOrder = 20.nn
      )

    //регистрируем переходы состояний для типа
    Btk_StateChangeApi().registerForObject(idvObjectType, List(
      "Forming" -> "Executing",
      "Executing" -> "Done",
      "Executing" -> "Forming",
      "Done" -> "Forming",
      "Done" -> "Executing"
    ))
  }
  session.commit()
}
```

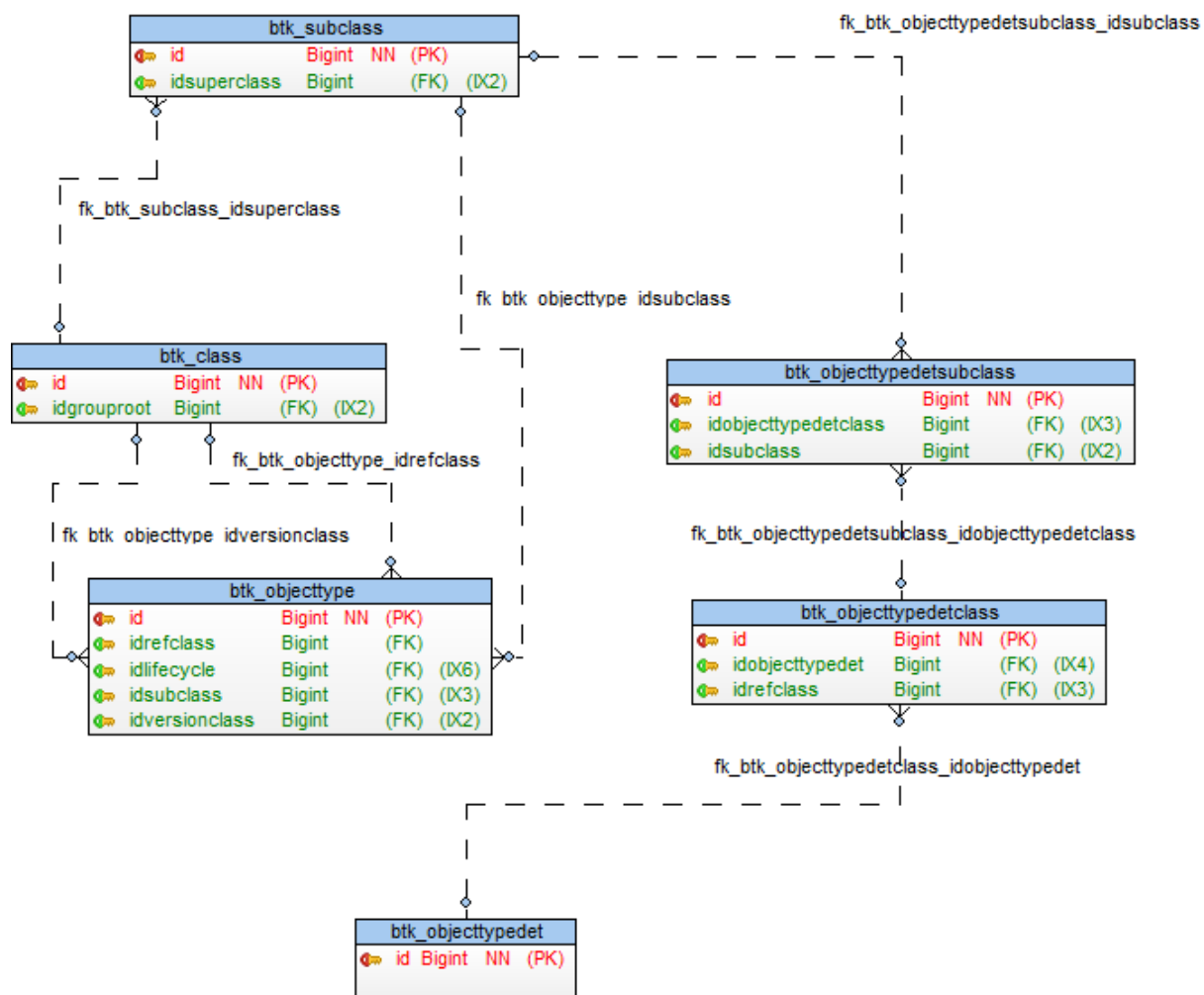
Пример регистрации скрипта в тэге `dbData` для odm класса `Stm_ActIn`:

```

<script name="regObjectType" version="1">
  <depends>
    <dep on="Stm_ActIn.regState"/>
    <dep on="Stm_ActIn.regTab"/>
  </depends>
  <install>Stm_ActInApi.regObjectType(false);</install>
</script>

```

Общая схема классов



Sql-функции

Функции getmnemocode, getheadline и getattribute

В SQL-запросе вместо получения хэдайна, мнемокода или иного атрибута через join или подзапрос можно использовать функции getmnemocode, getheadline и getattribute.

```
--getmnemocode
getmnemocode(idpclass bigint, idpobj bigint)

getmnemocode(gid text)

--getheadline
getheadline(idpclass bigint, idpobj bigint)

getheadline(gid text)

--getattribute
getattribute(idpclass bigint, idpobj bigint, spatr text)

getattribute(pgid text, pattr text)
```

Данные процедуры можно использовать, если возвращаются немногочисленные объекты, иначе на БД будет большая нагрузка.

Внимание

Нельзя использовать при формировании отчётов или отображения List.

Можно использовать для:

- получения данных через onRefreshExt для отображение Card и ему подобных
- печатных форм
- для разового анализа данных через приложение управления БД (например, dbeaver)

Примеры использования:

```
select      t.sRegNum
            , t.dReg
            , t.idService
            --для id нужно принудительно указать класс
            , getmnemocode(51001,t.idService) idServiceMC
            , getheadline(51001,t.idService) idServiceHL
            --gid можно сразу передать на вход
            , t.gidSrc
            , getmnemocode(t.gidSrc) as gidSrcMC
            , getheadline(t.gidSrc) as gidSrcHL
            --getattribute
            , getattribute('2101', t.idResourceholder, 'gid') gidResourceHolder
            , getheadline(getattribute('2101', t.idResourceholder, 'gid')) as_
↪idResourceHolderHL
from oil_tankerpos t
```

В разработке расширение методов:

```
--getmemocode  
getmemocode(sNameTb text, idpobj bigint)  
  
--getheadline  
getheadline(sNameTb text, idpobj bigint)
```

13.3 Выборки

Выборка

Выборка определяет правило получения, отображения данных и обеспечивает взаимодействие с пользователем. Выборки содержат основную часть интерактивной бизнес логики.

Выборка определяет:

- Способ получения данных
- Способ отображения данных пользователю
- Бизнес логику обработки пользовательских действий

Выборка может создаваться от класса с использованием кодо-генерации или вручную.

Пользовательский интерфейс приложения является совокупностью экземпляров отображений выборок.

Отображения

Отображение – группирует бизнес логику выборки в зависимости от способа представления данных.

Стандартные отображения для класса:

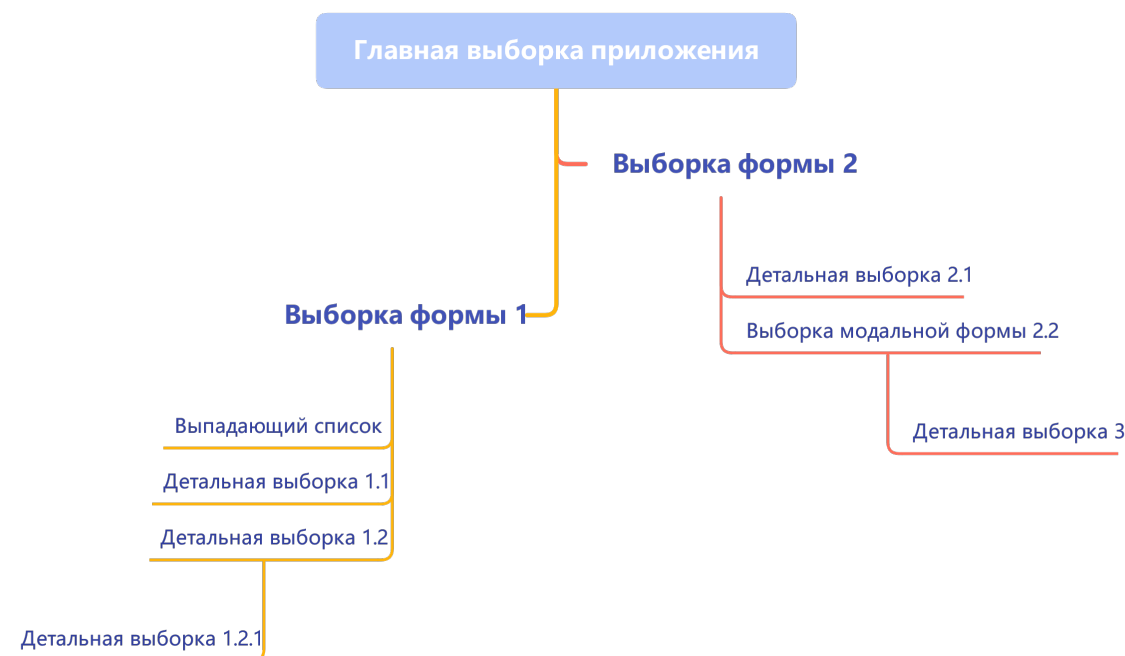
- **Default** – отображение по умолчанию
В данном отображении определена общая функциональность для данных выборки. Остальные отображения выборки обычно наследуют это отображение.
- **List** – отображение для списка объектов класса
- **Card** – отображение для карточки объекта
- и т.д.

Отношение экземпляров выборок

Экземпляры выборок, открытые в приложении, связаны друг с другом в отношении **мастер-деталь**. Это означает, что одна выборка всегда является подчиненной к другой выборке. Подчиненная выборка имеет доступ к параметрам и данным мастер выборки.

Корнем выборок экземпляра приложения – является выборка главного меню. Все остальные выборки являются подчиненными к ней.

Выборка, являющаяся корнем формы, имеет доступ к данным главного меню. Экземпляры выборок внутри формы также выстраиваются в иерархии **мастер-деталь**, в соответствии с бизнес логикой отображения.



Доступность параметров главной выборки в подчиненных

Получение значения параметра осуществляется с помощью метода `getVar()` и преобразованием результата к нужному типу.

Иерархическая структура создаваемых объектов (выборок) позволяет обеспечивать доступность и передачу параметров по отношению «мастер-деталь», а также производить автоматическое обновление при изменении используемых параметров. Параметры экземпляров выборок – это, прежде всего, атрибуты (поля) их датасета. Также это могут быть дополнительно созданные параметры, или специальные параметры фреймов.

Параметры мастер-выборок автоматически становятся доступны в детальных выборках. При этом, главная выборка является общим, наиболее высокоуровневым мастером, поэтому в ней имеет смысл создавать наиболее общие параметры, используемые всем приложением.

Обращение к параметру непосредственного мастера осуществляется прибавлением к имени параметра префикса `super$`. При этом, если необходимо обратиться к параметру мастер-выборки своей собственной мастер-выборки (т.е. перепрыгнуть через один экземпляр выборки), необходимо использовать префикс `super$super$`. В случае, если при любом из обращений к параметру он не будет найден, произойдет автоматический поиск параметра по дереву отношений «мастер-деталь» выборок вплоть до тех пор, пока мы не найдем этот параметр, или не дойдем до главной выборки приложений. Поэтому, при создании глобальных параметров на уровне главной выборки приложений, для них имеет смысл давать уникальное имя, а затем обращаться к ним из любого места приложения используя один единственный префикс `super$`.

Пример получения строкового параметра из мастер-выборки:

```
getVar("super$sCaption").asNSString
```

Передача параметров в выборку

При открытии выборки, в нее можно передать карту с дополнительными параметрами, используя метод `params()`.

Пример кода:

```
Bs_GoodsAvi.defCard.newForm().params(Map(CardRep.IdItemSharp -> idvGds, CardRep.  
↳EditingType -> EditingType.edit)).open()
```

В случае такой передачи параметров, они становятся доступны в открытом отображении.

Открытие выборок в различных режимах

При открытии выборки в любом из режимов:

- `newForm().open()`
- `newForm().openModal()`
- `newForm().openLookup()`

необходимо учитывать, что родителем для открывшейся выборки будет являться `***_MainMenu`, а сама открывшаяся выборка будет главной формой (`selection.isMainOnForm = true`). Поэтому открывать выборки, в которых установлены зависимости от параметров ("`super$...`") родительской выборки, без самой родительской выборки нельзя, тк никаким способом в открывающуюся выборку не получится передать подобные параметры.

Разметка выборки

Фреймворк использует систему декларативной разметки выборок. Это позволяет ускорить разработку визуальных форм т.к. разработчику не требуется заниматься низкоуровневым программированием.

Разметка выборки задает правила отображения данных выборки и ее дочерних элементов в `xml` файле, с расширение `.avm.xml`.

Ключевые элементы разметки:

- Отображения
- Компоновка формы
- Фрейм
Задаст тип представляемых данных (список, дерево и тд.)
- Атрибуты
- Операции
- Фильтры

Разметка карточки. Контейнеры

Существует два режима разметки отображения:

1. Разметка на основе настроек атрибутов - старый режим разметки, основанный на порядковом номере и поле `isLastInline` атрибутов.
2. Разметка на основе контейнеров - новый режим, основанный на разметке с помощью контейнеров, указанных внутри тэга `<card>`. \

Разметка атрибутов может быть задана с помощью следующих контейнеров:

- *hBox* горизонтальный контейнер: объекты внутри располагаются друг за другом горизонтально.
- *vBox* - вертикальный контейнер: объекты внутри располагаются друг за другом вертикально.
- *vGroup* - группа объединения: по сути тот же вертикальный контейнер, но визуально обводит содержимое рамкой и может иметь наименование.
- *vSection* - сворачиваемая секция: вертикальный контейнер, который имеет наименование и функцию сворачивания, которая скрывает содержимое. Можно указать, как по умолчанию будет отображаться секция - свернутой или развернутой.

Контейнеры могут быть вложены друг в друга.

Пример разметки с помощью контейнеров:

```
<representation editMode="edit" name="Card" stdFilter.isAvailable="false">
  <layout>
    <simpleComposer>
      <frame filter.isVisible="false">
        <card>
          <layout>
            <vBox>
              <hBox>
                <vBox>
                  <attr name="idCustomerHL"/>
                  <attr name="dOutDate"/>
                </vBox>
                <vBox>
                  <attr name="gidSrcDoc" visible="true"/>
                  <attr name="nOrder"/>
                </vBox>
              </hBox>
              <vGroup caption="Приемка">
                <attr name="idCustomerHL"/>
                <attr name="dExecuteDate"/>
              </vGroup>
            </vBox>
            <vSection caption="Примечание" collapsed="false">
              <attr name="sComment" caption="Комментарий"/>
            </vSection>
          </layout>
        </card>
      </frame>
    </simpleComposer>
  </layout>
</representation>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
<attributes/>
<operations/>
</representation>
```

Использование шаблонов в разметке

Файлы разметки поддерживают язык шаблонов [Thymeleaf](#). Шаблоны позволяют вставлять в выборку повторно используемые фрагменты. К примеру использовать общий фильтр в разных отображениях.

Для использования шаблонов языка шаблонов необходимо подключить в выборку пространство имен: `xmlns:th="http://www.global-system.ru/xsd/global3-view-template-1.0"`

Наследование разметки

Для наследования разметки другой выборки необходимо:

1. Подключить пространство имен языка шаблонов
2. Добавить к тегу view указание о наследовании другой разметки `th:extends="Some_Other.avm.xml"`

Полный пример xml с наследованием:

```
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
      xmlns:th="http://www.global-system.ru/xsd/global3-view-template-1.0"
      name="RplTst_AllDbTypes" th:extends="RplTst_AllDbTypes.dvm.xml">
  <representation name="Default"/>
</view>
```

Алгоритм наследования выборки:

1. При отсутствии наследования скопировать отображение предка в наследника
Копирование происходит в случае, если в наследнике нет данного отображения
2. Произвести наследование отображений
Наследование производится в случае, если и в предке, и в потомке есть отображение с тем же именем.
3. Выполнить оставшиеся команды языка разметки

Алгоритм наследования отображений:

1. При отсутствии скопировать
 - levelGroups
 - bandGroups
 - layout
 - filter

2. Добавить отсутствующие атрибуты
3. Добавить отсутствующие операции

Операции

Выборка взаимодействует с программным окружением системы с помощью операций. Операция – это некая обработка того или иного события, вызванного пользователем или каким-то процессом. Операции бывают служебными, выполняя логику обработки событий, или пользовательскими, срабатывающими в результате явного действия пользователя.

Классификация операций

- *Базовые операции*
Операции, предназначенные для работы с набором данных. Они отвечают за добавление, удаление, обновление записей, получение данных, навигацию по датасету и т.п. Эти операции, как правило, уже определены на уровне базовых выборок-предков. Некоторые из этих операций срабатывают автоматически на выполнение определенных пользовательских действий (**BeforeEdit** при начале редактирования, **AfterEdit** при завершении, **CheckWorkAbility** при перемещении с записи на запись)
- *Служебные операции*
Операции, выполняющие строго определенные вспомогательные действия
- *Системные операции*
Операции, создающиеся автоматически
- *Сеттеры*
Операции предназначены для изменения значений атрибутов (полей) выборки. При вводе значения в какое-то поле формы вызывается связанная с данным полем операция (сеттер), которая по умолчанию вызывает серверные методы изменения значений атрибутов класса. Сеттеры для атрибутов класса создаются автоматически.
- *Операции фильтров*
Операции, необходимые для работы с фильтрами.
- *Пользовательские операции*
Операции, создаваемые разработчиком самостоятельно и позволяющие осуществлять любой набор необходимых операций, не только осуществляемых над данными, но и являющихся целиком клиентскими (например, вызов другой формы, сообщения и т.п.).

Базовые операции выборки

Для автоматического срабатывания операций при различных системных событиях ключевой параметр — это их системное имя. Базовые и служебные операции выборки имеют предопределенные системные имена.

onRefresh

Операция, загружающая данные в датасет. При повторном вызове производит обновление всего набора данных (**refresh**).

onRefreshItem

Операция обновления одной (текущей) записи.

onRefreshExt

Операция с дополнительным запросом к базе для получения полей-заголовков ссылочных объектов. Вызывается только для объектных запросов в `onRefresh/onRefreshItem`.

Для использования в запросе атрибутов класса необходимо передать их в блоке `with`. При этом обязательно для `HE Long` атрибутов нужно явно указывать тип в аннотации `/*@...*/: \`

- Для `Varchar` атрибутов - «String», «NString» или «varchar», например - `/*@NString*/`
- Для `Number` атрибутов - «Number», например - `/*@Number*/`

В противном случае атрибуты попытаются преобразоваться к `bigint` типу.

beforeEdit

Операция, выполняемая перед началом редактирования записи для выполнения подготовительных действий перед началом редактирования. Автоматически выполняется в момент начала внесения изменений в объект.

afterEdit

Операция финальной проверки ввода записи. Срабатывает автоматически при попытке перейти с отредактированной записи на другую запись выборки или при закрытии выборки.

insert

Операция вставки новой записи в выборку.

delete

Операция удаления записи.

checkWorkAbility

Операция, предназначенная для осуществления проверок изменения состояния операций выборок, доступности редактирования ее атрибутов. Вызывается автоматически при переходе с записи на запись, при открытии выборки, а также после выполнения операции, для которой выставлен флаг «выполнять операцию применимости после выполнения операции».

onLoadMeta

Операция, срабатывающая при загрузке метаданных. Т.к. метаданные загружаются только один раз, при первом открытии экземпляра выборки на фрейме, то и данная операция срабатывает только один раз.

onUnloadMeta

Операция, срабатывающая при выгрузке метаданных.

beforeOpen

Операция, срабатывающая перед открытием выборки. Данная операция рекомендуется для создания дополнительных параметров, используемых в выборке.

afterOpen

Операция, срабатывающая сразу после открытия выборки. Необходимо использовать для переопределения наименований столбцов выборки и т.д.

onShow

Операция, срабатывающая сразу после того, как отрисован интерфейс формы и были созданы элементы формы.

onControllerCreated

Операция, срабатывающая сразу после того, как был создан фрейм отображения выборки. Эта операция аналогична `OnShow`, но срабатывает в том числе и при открытии интерфейсных элементов формы,

которые были скрыты при открытии формы (например, закладки, которые не были первыми в перечне закладок).

Служебные операции выборки

saveForm

Операция сохраняет данные формы.

cancelForm

Операция отменяет изменения на форме.

closeFormOk

Операция закрытия формы с подтверждением выбора

closeFormCancel

Операция закрытия формы по кнопке «Выход» или по кнопке закрытия окна «крестик»

beforeCloseForm

Операция срабатывает перед закрытием формы.

afterCloseForm

Операция срабатывает после закрытия формы.

onCloseFormQuery

Операция вызывается в начале процесса закрытия формы и позволяет отменить закрытие.

Предопределенные операции выборов

applyUniFilter

Операция применяет настроенные условия фильтрации, выполняется перезапрос данных с наложенными условиями фильтрации.

resetUniFilter

Операция отменяет все наложенные на выборку фильтры

clearUniFilter

Операция отчищает все условия фильтрации.

showAuditObject

Операция вызывает окно аудита выполняемых пользователями действий в БД, с наложенным фильтром по текущему объекту.

showAboutObject

Операция открывает окно системной информации об объекте.

copyObject

Операция копирования объекта

cardEdit

Операция открытия объекта в карточке, предварительно запросив, какую использовать выборку и отображение.

allowEdit

Разрешение/запрещение редактирования объектов в списке. По умолчанию операция доступна для редактируемых списков.

showTab

Операция открытия/закрытия детальной части

Сеттеры изменения атрибутов

Операции установки значений атрибутов (так называемые сеттеры) используются для того, чтобы установить определенное значение в поле записи объекта, соответствующее его типу. Имя сеттера состоит из префикса **set** и имени атрибута **attributeName**: **setAttributeName**.

Специальные операции контролов

onFocusedFieldChanged

Это специальная операция, срабатывающая после перехода фокуса ввода в списке или дереве с одного столбца на другой, а также при переходе с контрола на контрол в карточке или панели фильтров.

onFocusedCellChanged

Это специальная операция, срабатывающая после перехода фокуса ввода в списке с одной ячейки на другую.

onFrameActivated

Специальная операция, срабатывающая при переходе фокуса ввода в грид, дерево, карточку.

Способы визуализации операций

Существует 4 основных способа визуального отображения операций:

- На тулбаре фрейма
- В выпадающем (контекстном) списке (popup меню)
- В качестве пункта главного меню (доступно только для выборок приложений главного меню)
- На навбаре сразу под главным меню (доступно только для выборок приложений главного меню)

Для настройки отображения операции каждым из способов существует собственная настройка при помощи соответствующих свойств операции.

Пользовательские Jexl-операции

Использование подсистемы пользовательских Jexl-операций позволяет удобно добавлять операции на существующие выборки без вмешательства в прикладной код. Реализовано добавление, удаление и переименование операции

Примечание

Операции, заданные в отображении по умолчанию **Default** доступны во всех отображениях-предках в рамках выборки. Любое из свойств может быть переопределено для каждого из отображений в отдельности

Свойства Jexl-операции

В таблице описаны основные и полезные свойства операций, определяющие их поведение и отображение в интерфейсе. Все остальные свойства — системные, не редактируются вручную или не влияют на пользовательский опыт.

Свойство	Описание
Наименование	Имя операции. Оно при наведении курсора.
Описание	Пояснение назначения операции.
Порядковый номер	Числовое значение, используемое для сортировки. Чем меньше номер, тем выше (при вертикальном отображений) или левее (при горизонтальном) операция в списке меню.
Иконка	Графическое представление операции. Номер изображения, отображаемого на тулбаре.
Активность	Если свойство выключено, операция не отображается в интерфейсе и недоступна даже при вызове из кода.
Видимость	Если свойство выключено, операция скрыта в интерфейсе, но остаётся доступной при вызове из кода.
Доступность для взаимодействия	Если свойство выключено, операция отображается серым цветом и недоступна для нажатия. Для корректной работы этого свойства должны быть включены одновременно <i>Активность</i> и <i>Видимость</i> .
Флаг (checked)	Указывает, что операция является флагом. При нажатии на соответствующий пункт меню, будет установлена/снята галочка
Вызвать checkWorkability	После выполнения операции автоматически запускается проверка работоспособности.
Обновить запись после выполнения	После выполнения операции запись автоматически обновляется.
Обновить запись перед выполнением	Перед запуском операции обновляются данные записи, с которой она работает. Это полезно, если предыдущие операции не обновляли данные.
Видимость операции на всех панелях	Операция сразу отображается на всех основных панелях интерфейса.
Видимость в главном меню	Операция отображается в верхней панели интерфейса.
Видимость на панели быстрого запуска	Операция отображается на второй панели сверху.
Видимость в контекстном меню	Операция отображается в меню, вызываемом правой кнопкой мыши.
Видимость на панели управления	Операция отображается на панели управления объектом.
Тип привилегий для доступа	Обозначает тип объектной привилегии, которая должна быть выдана пользователю, чтобы он мог видеть операцию в интерфейсе и взаимодействовать с ней.
Комбинация горячих клавиш	Это клавиатурное сочетание, которое вызывает операцию.
Коллекция изображений	Название коллекции изображений, из которой присваивается иконка.
Группа на панели быстрого доступа	Определяет группу, к которой относится операция на панели быстрого запуска.
Операция-предок	Задаёт вложенность операции в дереве интерфейса. Например, если указать <i>Сущности</i> , операция будет отображаться в разделе <i>Сущности</i> вместе с другими операциями этого раздела.

Особенности реализации

Каждое свойство, в том числе jexl-скрипт и клонирующий SQL-запрос хранятся как отдельная строка в базе данных в таблице `Btk_Sel0per0ver0ride`. Однако, значения jexl-скрипта и клонирующий SQL-запроса хранятся в поле `sJexlScript`, а не в `sValue` как все остальные параметры операции, так как `sValue` имеет ограничение на длину в 255 символов

Примечание

Все системные имена jexl-операций начинаются с префикса "jexl_"

Структура таблицы:

- **sPropName**
используется для дифференцирования свойств Jexl-скрипт
- **sJexl0perScript**
jexl-скрипт операции. Имеет смысл только у свойств с `sPropName = sJexl` или `cloneQuery`
- **sValue**
значение параметра операции
- **s0perName**
системное имя операции
- **sMasterSel**
выборка, в которой задано свойство операции
- **sMasterRep**
отображение, в котором задано свойство операции

Примечание

- Операция существует в данной выборке и отображении, если существует запись с `sMasterSel` равным выборке, `sMasterRep` - названию отображения или `Default`, и именем свойства `sPropName = "sJexl"`
- При этом, чтобы операция корректно отображалась на выборке, необходимо установить имя, иконку и порядковый номер. (иконка необязательна для операций главного и контекстного меню)

Диалоги

Диалоги позволяют задавать вопросы пользователю и получать от него ответы на эти вопросы.

Для доступа к методам диалогов используется специальный объект `ru.bitec.app.gtk.gl.Dialogs`, который доступен в `Avi` через переменную `dialogs`.

Кроме стандартных диалогов, есть возможность открыть форму в режиме выбора (`openLookup`) и после обработать выбранное пользователем значение, что так же из себя представляет специализированный вид диалогов.

Стандартные диалоги

showMessage

Выводит на экран текст в модальном окне.

Пример использования:

```
dialogs.showMessage("Текст, который увидит пользователь")
```

showMsgDialog

Открывает диалог указанного типа с сообщением и указанным перечнем кнопок.

Возможные типы диалогов см. в `ru.bitec.app.gtk.gl.msgdlg.MsgDlg`

Возможные типы кнопок см. в `ru.bitec.app.gtk.gl.msgdlgbutton.MsgDlgButton`

Пример использования:

```
val dlgRes = dialogs.showMsgDialog(  
    //тип диалога "Подтверждение"  
    MsgDlg.confirmation,  
    //текст диалога  
    "Включить автонумерацию?",  
    //список кнопок: Да, Нет  
    List(MsgDlgButton.yes, MsgDlgButton.no)  
)  
if (dlgRes == MsgDlgButton.yes) {  
    //обработка нажатия на кнопку "Да"  
}  
else if (dlgRes == MsgDlgButton.no) {  
    //обработка нажатия на кнопку "Нет"  
}
```

showButtonsDialog

Открывает диалог с произвольными кнопками.

Позволяет формировать диалоги, кнопки которых будут не из перечня стандартных кнопок.

Пример использования:

```
dialogs.showButtonsDialog(  
    caption = "",  
    text = "Удалить форму или убрать ссылку на вид отчетности?",  
    colCount = 2,  
    buttons = List(List("Удалить"), List("Убрать ссылку")),  
    imageCollectionName = "ToolBarPrimaryHot",  
    focusButtonNumber = 0L  
) match {  
    case 0L =>  
        //обработка нажатия кнопки "Удалить"  
    case 1L =>  
        //обработка нажатия кнопки "Убрать ссылку"  
}
```

showPromptDialog

Открывает диалог с запросом ввода строки.

Пример использования:

```
val svDialogRes = dialogs.showPromptDialog("Заголовок окна", "Текст диалога", "Значение_
↳ по умолчанию").ns

if (svDialogRes.isNotNullOrEmpty) {
    //обработка введенного значения
}
```

showConfirmDialog

Открывает диалог с типом «Подтверждение» и кнопками «Да» и «Нет»

Пример использования:

```
val dialogRes = dialogs.showConfirmDialog("Текст диалога")
if (dialogRes){
    //обработка случая, когда пользователь нажал "Да"
}
```

showInfoForm

Отображает информационное сообщение с переданным текстом, без возможности его закрытия пользователем.

Скрывается при вызове метода `hideInfoForm`.

Применяется, когда в бизнес-логике выполняется продолжительная задача, и требуется проинформировать пользователя об ее выполнении.

Пример использования:

```
dialogs.showInfoForm("Внимание, идет загрузка Госреестра типов СИ")
try {
    //продолжительное действие
    thisApi().download()
} finally {
    dialogs.hideInfoForm()
}
```

hideInfoForm

Скрывает информационное сообщение, см. описание метода `showInfoForm`

withInfoForm

Метод включает в себя открытие информационного окна, и его закрытие после завершения выполнения переданной функции.

Пример из описания метода `showInfoForm`, можно переписать используя этот метод:

```
dialogs.withInfoForm("Внимание, идет загрузка Госреестра типов СИ") {
    //продолжительное действие
    thisApi().download()
}
```

showEditPaintStyleDialog

Вызывает диалог настройки стилей раскраски текстовых полей.

Форма в режиме выбора значения

Одной из типичных задач является предложить выбрать пользователю какой-либо объект и затем сделать обработку его выбора. Например, чаще всего ссылочные поля открывают для выбора список объектов ссылочного класса, а после выбора пользователем значения устанавливают его в текущей форме.

Для реализации такого типа диалогов используется синтаксис открытия формы в режиме `openLookup`.

Пример открытия формы и обработки выбора пользователя:

```
//создаем новую форму
val data = Btk_ClassAvi.listForChoose().newForm()
//указываем список полей, которые требуется получить из формы
.results("id" :: "sCaption" :: Nil)
//открываем форму в режиме выбора
.openLookup()

//проверяем, что пользователь подтвердил выбор
if (data.getLookupResult eq LookupResult.ok) {
  //получаем значение первого поля из списка results
  val id = data.getData(1, 0)
  //получаем значение второго поля из списка results
  val sCaption = data.getData(1, 1)
}
```

Выбор нескольких строк (мультиселект)

Для того, чтобы в форме в режиме выбора пользователь мог выбрать несколько строк используется опция `useMultiSelect`.

Пример открытия формы в режиме мультиселекта и обработка выбранных значений:

```
//создаем новую форму
val data = Btk_ClassAvi.listForChoose().newForm()
//указываем список полей, которые требуется получить из формы
.results("id" :: "sCaption" :: Nil)
//указываем возможность выбора нескольких строк
.useMultiSelect
//открываем форму в режиме выбора
.openLookup()

//проверяем, что пользователь подтвердил выбор
if (data.getLookupResult eq LookupResult.ok) {
  //цикл по выбранным строкам
  for (i <- 1 to data.size) {
    //получаем значение первого поля из списка results для строки i
    val id = data.getData(i, 0)
    //получаем значение второго поля из списка results для строки i
    val sCaption = data.getData(i, 1)
  }
}
```

Фрейм

Фреймы — это средство представления информации в выборке.

Существующие фреймы по способу представления информации можно разделить на несколько видов:

- для отображения информации в виде карточки объекта
- для отображения списка объектов в табличной форме
- для отображения списка объектов в виде древовидной структуры
- для отображения инфографики
- специализированные (для отображения графической информации, доступа к файлам и т.д.)

Основные типы фреймов

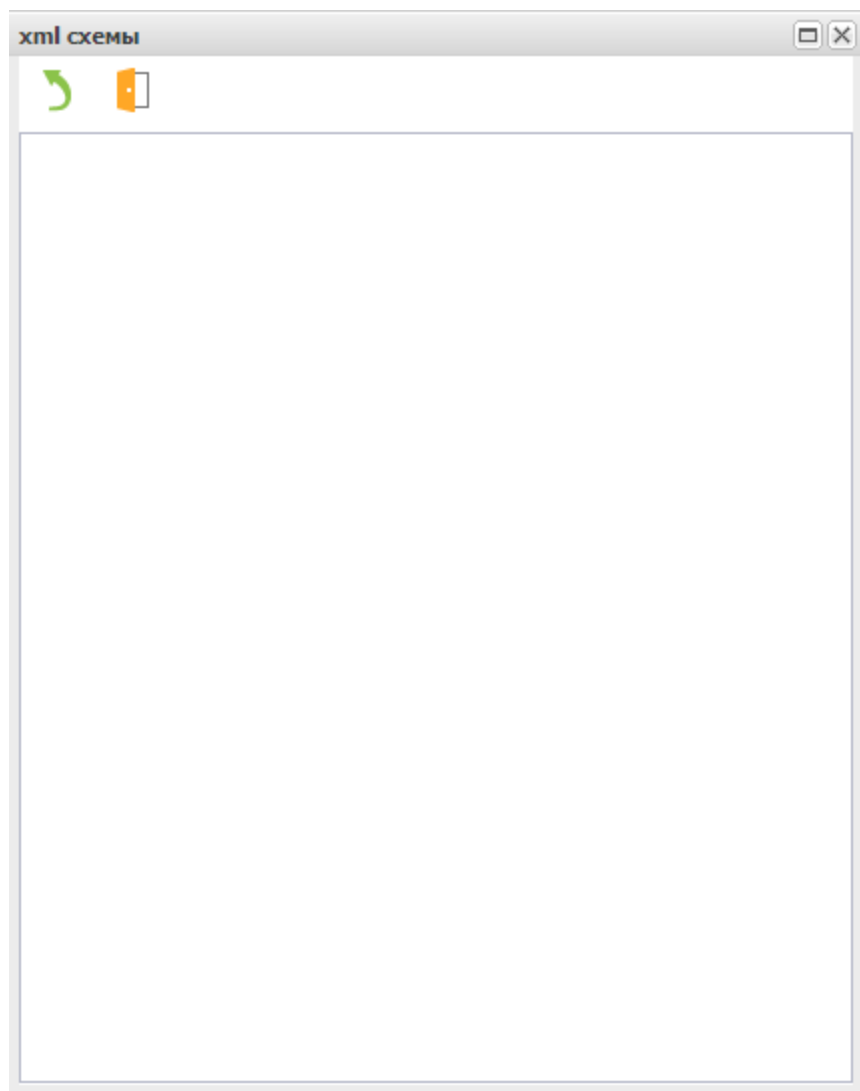
grid

Фрейм для вывода списка объектов класса в табличной форме

Группировка	Сохранить	Отменить	Обновить	Создать	Удалить	Редактировать	Копировать	Дополнительно	Экспорт	Фильтр	Помощь	Выход
Ном. №	Обозначение	Марка	ГОСТ марки	Отображать неиспользуемые								
Наименование	Наименование	ЕБЛ	Точн. округл.	Марка (текст)	Марка ТИЦ	ГОСТ марки	ГОСТ	ГОСТ	Тип сортамента	Типоразмер	Артикул	
Ном. №	Обозначение	Наименование	ЕБЛ	Точн. округл.	Марка (текст)	Марка ТИЦ	ГОСТ марки	ГОСТ	ГОСТ	Тип сортамента	Типоразмер	Артикул
993254898	ШТЦЛ 923.225-01	ШТЦЛ 923.225-01 Вал	ШТ	2								
993254897	ШТЦЛ 923.225-00	Вал ШТЦЛ 923.225-00	ШТ	2								
897860897	ТШ3265.1000-02	Корпус прав. ТШ3265.1000-02	ШТ	2								
897860896	ТШ3265.1000-01	Корпус лев. 1	ШТ	2						100		
ТУ436643	Полотнищен	Полотнищен	ШТ									
Тест-запрет Только для созданной спецификации для нсч												
ТЛШ-363621-072-01	ТЛШ-363621-072-01	Запрет на перевод idobjecttypehl	ШТ	2								
ТЛШ-363621-059-62	ТЛШ-363621-059-62	/688-03-956-01 ЖЕЛОБ С КРЫШКОМ	ШТ	2								
ТЛШ-363621-059-06	ТЛШ-363621-059-06	/688-03-949-62 ПОДВЕСКА ТИПА 2	ШТ	2								
ТЛШ-363611-080-07	ТЛШ-363611-080-07	/688-03-883-07 ПОДВЕСКА ТИПА 2	ШТ	2								
ТЛШ-363611-080-03	ТЛШ-363611-080-03	/688-03-883-07 СТОЙКА ПРИВАРНА	ШТ	2								
ТЛШ-363611-080-02	ТЛШ-363611-080-02	/688-03-883-03 СТОЙКА ПРИВАРНА	ШТ	2								
ТЛШ-363611-080	ТЛШ-363611-080	/688-03-883-02 СТОЙКА ПРИВАРНА	ШТ	2								
6449940552	ТЛШ-363173.005-11.31	/688-03-883 СТОЙКА ПРИВАРНАЯ Т	ШТ	2								
ТЕСТ МСЧ4	ТЕСТ МСЧ4	Коробка кабельная вертикальная ...	ШТ	2								
ТЕСТ МСЧ3	ТЕСТ МСЧ3	ТЕСТ МСЧ4	ШТ	2								
ТЕСТ МСЧ2	ТЕСТ МСЧ2	ТЕСТ МСЧ3	ШТ	2								
ТЕСТ МСЧ	ТЕСТ МСЧ	ТЕСТ МСЧ2	ШТ	2								
ТШД1000-362424-022	ТШД1000-362424-022	ТЕСТ МСЧ	ШТ	2								
CRST275-362281-003	CRST275-362281-003	ПОДДОН 000	ШТ	2								
CRST275-362214-047	CRST275-362214-047	РАСПЫЛИТЕЛЬ 00	ШТ	2								
CRST275-362214-015	CRST275-362214-015	МЕХАНИЗМ СКЛОНЕН НОС СТОЙКИ000	ШТ	2								
CRST275-362214-010	CRST275-362214-010	СТУПЕНЬКА	ШТ	2								
CRST275-362214-009	CRST275-362214-009	МЕХАН.СКЛОН.АНТЕННЫ.ИНМАРС.00	ШТ	2								
CRST275-362214-008	CRST275-362214-008	КРОНШТЕЙН АНТЕННЫ ИНМАРСАТ 00	ШТ	2								
CRST22-362371-001	CRST22-362371-001	КРОНШТЕЙН 00	ШТ	2								
CRST22-362211-004	CRST22-362211-004	ШУИТ С ЛАТИНС НАИМЕНОВ СУДНА00	ШТ	2								
9352304431	РФПИ.332584.009	ОПОРА	ШТ	2								
9035243604	РФПИ.332582.007-04	Пелла ЧЕРТЕЖ РФПИ.332584.009 (352-30.44 ...	ШТ	2								
РИДФ-364158-001-13	РИДФ-364158-001-13	Защайка клиновое ЧЕРТЕЖ ...	ШТ	2								
РИДФ-364151-001-33	РИДФ-364151-001-33	/216-03-342-13 КОНТРФОРС 1-	ШТ	2								
РИДФ-364151-001-19	РИДФ-364151-001-19	/216-03-290-33 СТОЙКА 1-С-1	ШТ	2								
РИДФ-364151-001-17	РИДФ-364151-001-17	/216-03-290-19 СТОЙКА 1-С-1	ШТ	2								

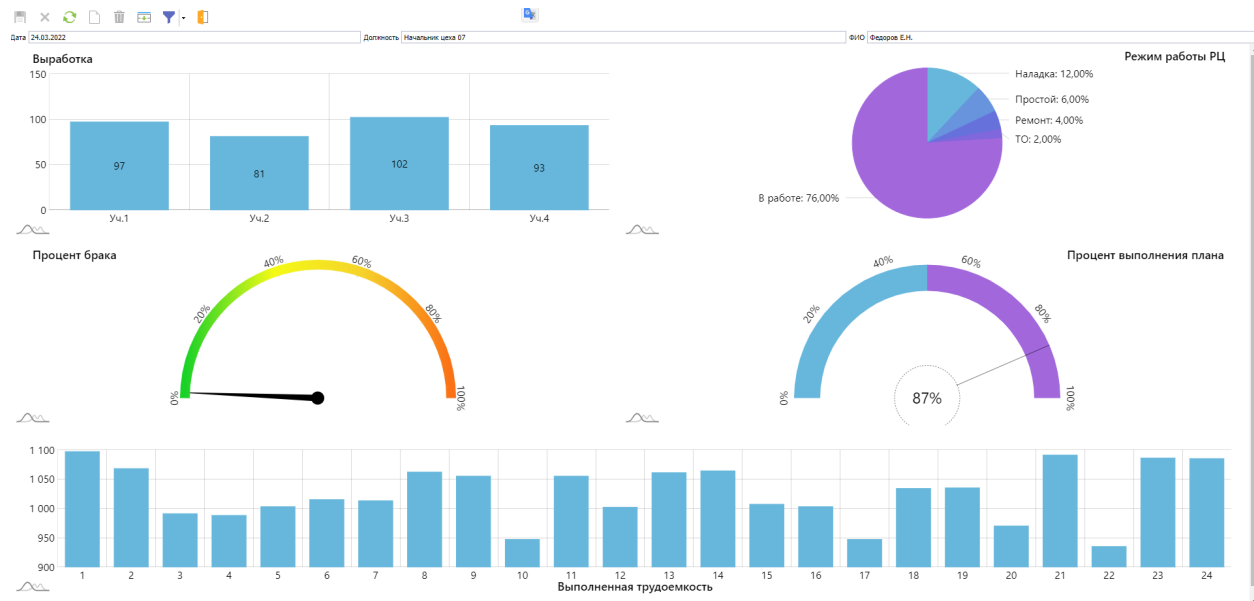
tree

Фрейм для вывода списка объектов класса в древовидной форме



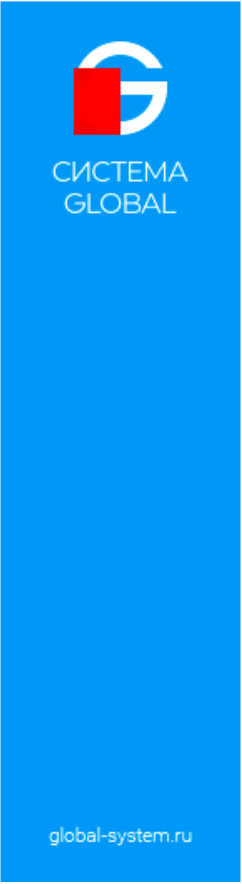
html

Фрейм, отображающий HTML



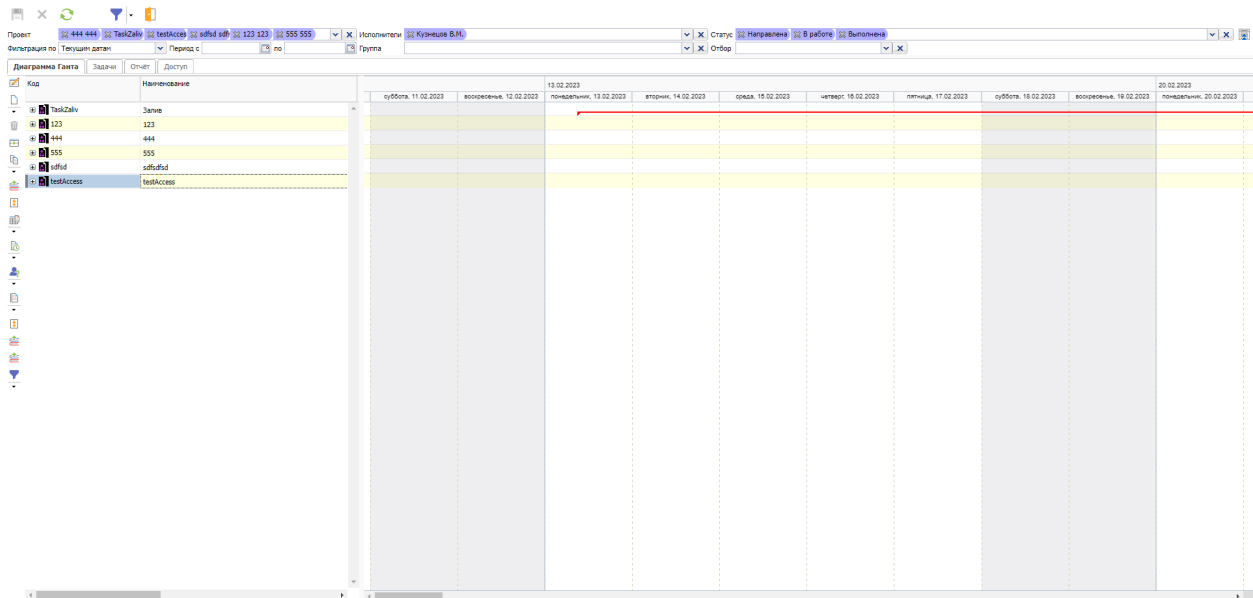
image

Изображение



gantt

Фрейм, для построения диаграмм Ганта.

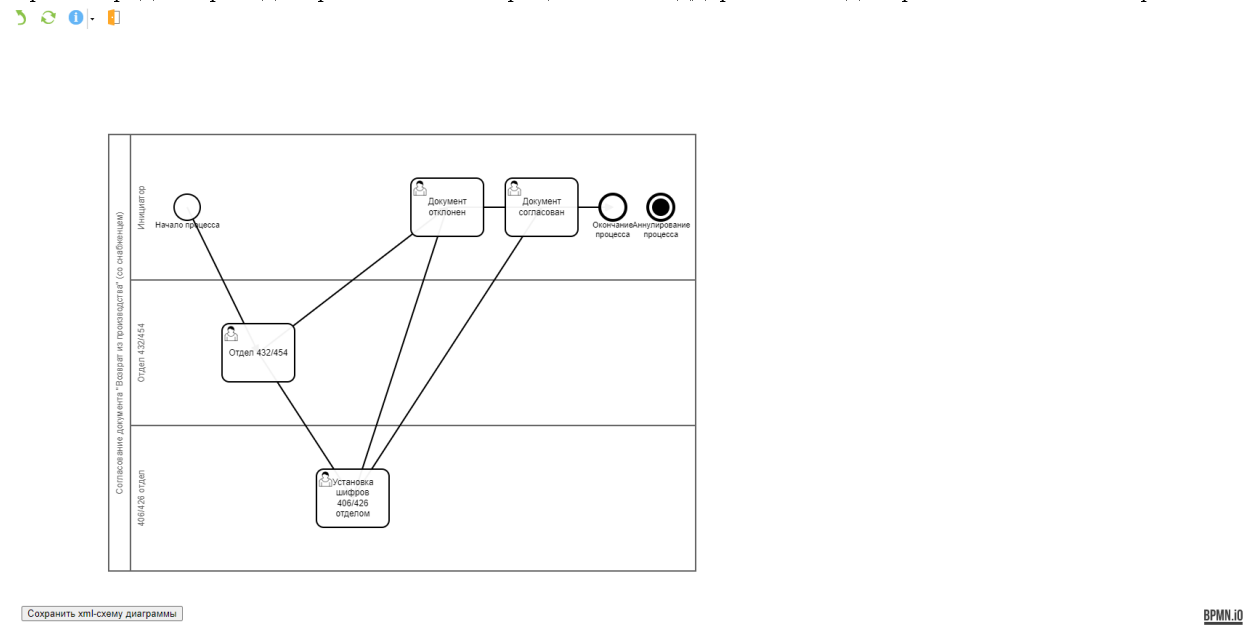


olap

Фрейм для построения сводных (OLAP) таблиц

brpmn

Фрейм редактора диаграмм бизнес-процессов. Поддерживает диаграммы BPMN версии 2.0



gridPanel

Позволяет размещать детальные фреймы в виде таблицы, для указания ширины и высоты ячеек которой можно использовать как абсолютные, так и относительные величины.

Выборка: Csc_NavValidation		История запросов		Перезагрузить												
Верхняя область - справочник ТНЦ, Нижняя область - ЕОС НСИ																
Проект	№	Обозначение	Наименование	Код ОКЗ	Наименование ОКЗ	Обозначение О.	Сопоставление	Проверка	Требуется валидация	Статус валидации	Не сопоставленные	Показать сопоставленные	Требуется валидация	Статус валидации	Дата последнего поиска в ЕОС НСИ	Обработаны все заявки в НСИ
✓	2481029001_01_1		Щит управления установкой ...													
✗	2481029001_02		Бак для гидропорхита ...													
✗	7052		гайка7052													
✗	7056		гайка7056													
✗	7057		гайка7057													
✗	2481029001_04		Установка вакуумная насоса.													
✗	2481029001_10		Цистерна сточных вод 1,0 м³.													
✗	2481029001_11		Цистерна хозяйственно-...													
✗	2481029001_15_1		Труба,ОД 50, L=150 арматур...													
✗	2481029001_01		Установка обработки сточны...													
✗	2481029001_15_1_об.1233		Труба,ОД 50, L=250 арматур...													
✗	2481029001_15_1		Труба,ОД 50, L=1000 арматур...													
✗	0163979855		Лента 60СД - С - Н - 2 x 50 ...													
✗	7053		гайка7053	279040190.010.01	Коробка соединительная...	ИТТТ.656519.0...	✓	Кузнецов В.М.		Подтверждена	✓	Включена в заявку	На классификацию			
✗	2481029001_13		Разгрузочный насос ВН-120М.													
✗	2481029001_17		Насос самотечной воды в ...													
✗	0163979855		Лента 60СД - С - Н - 0.5 x 16													
1 800																
Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ	Код ОКЗ
Код ОКЗ	Наименование ОКЗ	Базовый код	Обозначение	Код класса	Наименование класса	Сопоставленная ТНЦ										
254012000.001.0124	Патричная коробка 56-х/Т-420 с лентой 6177	56-х/Т-420	56-х/Т-420	254012000.001	Оружие боевое, кроме револьверов, ...											
254012000.001.0113	Патричная коробка 56-х/Т-420 артектическая	56-х/Т-420	56-х/Т-420	254012000.001	Оружие боевое, кроме револьверов, ...											
00185485.830.0001	Болт Н30 - 6х x 140.88.35 ГОСТ 7817-80	0185.54.83.00	0185.54.83.00	643655	ГОСТ Р ИСО 8992-2011											
00185484.830.0001	Болт Н30 - 6х x 140.82.194-3 ГОСТ 7817-80	0185.54.49.83.00	0185.54.49.83.00	643667	ГОСТ Р ИСО 8992-2011											
001846349.830.0001	Болт 2Н30 - 6х x 140.82.194-3 ГОСТ 7817-80	0184.63.49.83.00	0184.63.49.83.00	641435	ГОСТ Р ИСО 8992-2011											
001846405.830.0001	Болт 2Н30 - 6х x 140.88.35 ГОСТ 7817-80	0184.60.65.83.00	0184.60.65.83.00	26835	ГОСТ Р ИСО 8992-2011											
281412000.018.2256	Головка воздушная для двух фильтрующих ...	ВНТА.632737.007	ВНТА.632737.007	281412000.018	Краны и затворы шаровые поворотные с ...											
265141000.002.0365	Головакраспир СТ 3501 Коробка ...	СТ 3501	СТ 3501	265141000.002	Приборы и аппаратура для измерения или ...											
139229000.001.0186	Бортовая распределительная коробка типа ...	С1-3801	С1-3801	139229000.001	Изделия текстильные готовые прочие ...											
279040190.010.0007	Коробка СКС-9-25, ИЦД/2.101248.015 ...	ИЦД/2.101248.015	ИЦД/2.101248.015	279040190.010	Коробки соединительные											
279040190.010.0008	Калининградская коробка тип 3 С ...	БТСТ.617033.0997Т	БТСТ.617033.0997Т	279040190.010	Коробки соединительные											
279040190.010.0124	Коробка ОКЗ ИКСИ-421451.0607 ...	ИКСИ/421451.0607	ИКСИ/421451.0607	279040190.010	Соединительная аппаратура и ...											
279040190.010.0102	Коробка соединительная ИКСН-0-ОНС, ...	ИТТТ.656519.093-01	ИТТТ.656519.093-01	279040190.010	Коробки соединительные	7053										
281412000.018.2348	Распределительная коробка РКП	ПНКС.468353.003-208	ПНКС.468353.003-208	281412000.018	Краны и затворы шаровые поворотные с ...											
002550501.669.0001	Пруток сПН 80 x НД ОСТ 1.К2062-90 / ТУ	0255.05.01.669	0255.05.01.669	292140	сПН.3М											
002550501.674.0001	Пруток сПН 40 x НД ОСТ 1.К2062-90 / ТУ	0255.05.01.474	0255.05.01.474	292140	сПН.3М											
002550657.843.0001	Пруток сПН 150 x НД 1 - ОСТ БР-9325...	0255.06.57.843	0255.06.57.843	648552	1											
002550501.602.0001	Пруток сПН 60 x НД ОСТ 1.К2062-90 / ТУ	0255.05.01.602	0255.05.01.602	292140	сПН.3М											

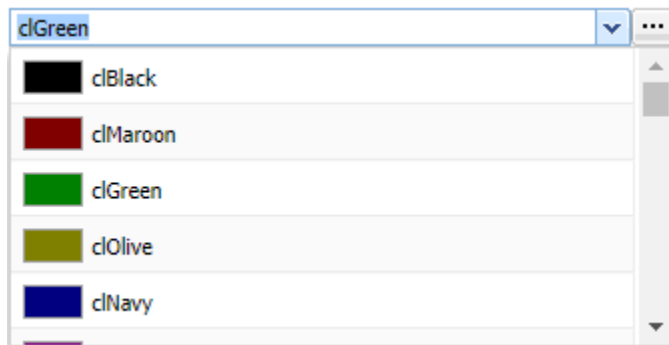
check

Редактор - Чекбокс (Галка).

Не используется ☐

colorPick

Редактор - Выбор цвета.



combo

Редактор - Фиксированный выпадающий список.

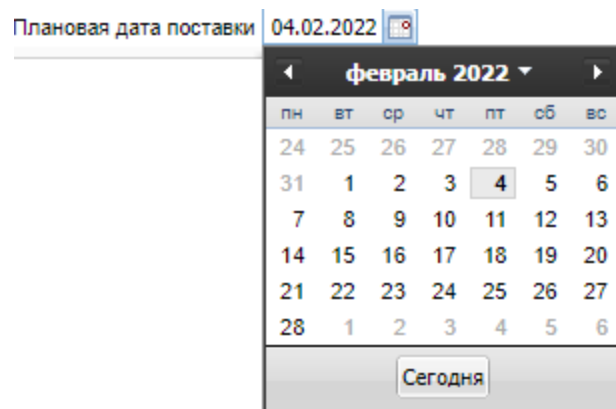
currency

Денежный редактор.

Сумма себестоимости в базовой ва...	
	1 305 000,00
	1 305 000,00
	380 833,33
	380 833,33
	319 000,00
	120 000,00
	120 000,00
	120 000,00
	116 087,50
	104 583,33
	101 750,00
	100 000,00
	96 000,00
	90 000,00
	90 000,00

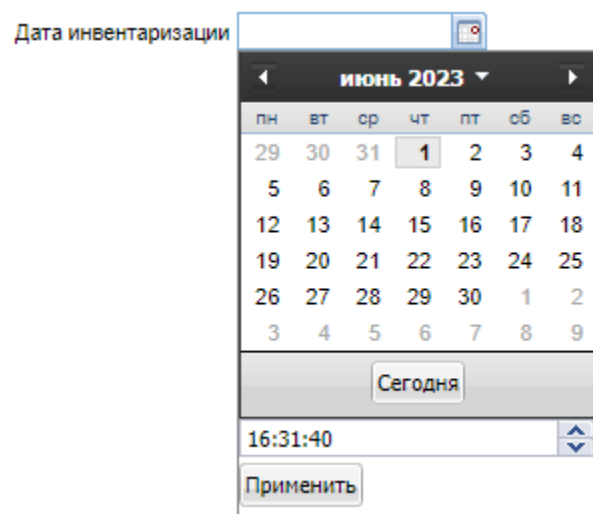
datePick

Редактор даты.



dateTimePick

Редактор даты и времени.



edit

Однострочный редактор текста.



editButton

Редактор в строке с кнопкой.



hotKey

Редактор ввода комбинации горячих клавиш.

hyperLink

Редактор адреса эл.почты и гипер-ссылки.



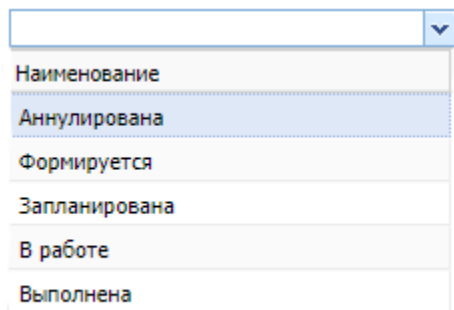
icon

Редактор - изображение. Используется для списка и дерева. Отображает изображение из коллекции изображений.



lookup

Редактор - Выпадающий список по запросу. Источником элементов могут являться: выборка или SQL-запрос, указанные в свойствах.



memo

Многострочный редактор. Предназначен для редактирования многострочных текстовых значений без разметки.

Описание

tagLookup

Редактор - Выпадающий список по запросу с возможностью множественного выбора.

Статус обеспечения ✕ Есть ожидаемое поступле ✕ Зарезервировано (синий) ▼

Наименование
Нет достаточного кол-ва (красный)
Перерезерв или перевыдача (пурпурный)
Есть достаточное кол-во для резерва (зеленый)
Потребность отсутствует (серый)

timePick

Редактор времени.

00:00:00	▲▼
	▲▼

editPassword

Редактор ввода пароля.

Пароль SSH

imageCollection

Редактор - выпадающий список изображений.

Тип
➡
➡
➡
➡
➡
➡
➡
➡

Мультиселект

Мультиселект - возможность выбрать несколько строк в списках или деревьях.

Для включения мультиселекта в отображении укажите в теге `grid` или `tree` свойство `isMultiSelectEnabled="true"`.

Пример разметки:

```
<representation name="List">
  <layout>
    <simpleComposer>
      <frame>
        <grid isMultiSelectEnabled="true"/>
      </frame>
    </simpleComposer>
  </layout>
</representation>
```

Пример обработки выделенных строк пользователем и получения значений атрибутов этих строк:

```
//цикл по выделенным строкам
for (i <- 0 until selection.selectedRecordsCount()) {
  //получение значения поля с типом NLong
  val id = NLong.fromAny(selection.selectedValueByName("id", i))
  //получение значения поля с типом NNumber
  val nOrder = NNumber.fromAny(selection.selectedValueByName("nOrder", i))
  //получение значения поля с типом NGid
  val gid = NGid(selection.selectedValueByName("gid", i).asInstanceOf[String])
  //получение значения поля с типом NString
  val sCaption = selection.selectedValueByName("sCaption", i).asInstanceOf[String].ns
  //получение значения поля с типом NDate
  val dDate = NDate(selection.selectedValueByName("dDate", i).asInstanceOf[JDate])
}
```

Примечание

Для формы в режиме выбора мультиселект регулируется опцией `useMultiSelect` при создании формы. Подробнее в главе [Диалоги](#)

Настройка стилей

Стиль - набор свойств, включающий в себя:

- `Color` - Цвет заднего фона в формате `$00BBGGRR` или `Global` константой цвета.
- `FontColor` - Цвет шрифта в формате `$00BBGGRR` или `Global` константой цвета.
- `FontSize` - Размер шрифта натуральным числом.
- `FontItalic` - Курсив. Значение 0 - нет курсива, 1 - есть курсив.
- `FontBold` - Жирность. Значение 0 - не жирный шрифт, 1 - жирный шрифт.
- `FontUnderLine` - Подчеркивание. Значение 0 - нет подчёркивания шрифта, 1 - есть подчёркивание шрифта.

- **FontStrikeOut** - Перечеркивание. Значение 0 - нет перечеркивания шрифта, 1 - есть перечеркивание шрифта.

Стиль задаётся строкой формата:

```
"param1=value1;param2=value2;param3=value3;...;paramn=valuen"
```

Пресеты стилей хранятся в классе **Btk_Registry**, их можно использовать для типовых задач.

Стили можно применять в отображении к строкам и столбцам.

Применение стиля к строке

Чтобы применить стиль к строке, нужно в теге **representation** присвоить свойству **rowStyleAttr** имя атрибута, в котором будет храниться либо готовая строка со стилем, либо системное имя пресета.

```
<representation name="List" editMode="notEdit" rowStyleAttr="sStyle1">
  <attributes>
    <attr name="sCode" caption="Код" editorType="edit" order="10" isRequired=
↪ "true"/>
    <attr name="sCaption" caption="Наименование" editorType="edit" order="20"
↪ isRequired="true"/>
    <attr name="sStyle1" caption="Первый стиль" editorType="edit" order="40"/>
  </attributes>
</representation>
```

Код	Наименование	Первый стиль
1	Здесь атрибуту присвоена непосредственно строка со стилем.	Color=\$00587920;FontColor=\$008EE4E;FontUnderline=1;FontBold=1;
2	Здесь атрибуту присвоено название пресета.	dOrangeWhite
3	Строка без стилей	

Применение стиля к столбцу

Чтобы применить стиль к столбцу, нужно в теге **attr** создать тег **style** и в нём использовать либо свойство **name**, либо **attr**. В свойстве **name** указывается непосредственно строка стиля или системное имя пресета, в **attr** указывается атрибут, где хранится либо строка, либо системное имя.

```
<representation name="List" editMode="notEdit">
  <attributes>
    <attr name="sCode" caption="Код" editorType="edit" order="10" isRequired=
↪ "true">
      <style name="Color=$00880000"/>
    </attr>

    <attr name="sCaption" caption="Наименование" editorType="edit" order="20"
↪ isRequired="true">
      <style name="BDG_URTotalRow"/>
    </attr>
    <attr name="sStyle1" caption="Первый стиль" editorType="edit" order="40">
      <style attr="sStyle1"/>
    </attr>
  </attributes>
</representation>
```

(продолжается на следующей странице)

```

    </attr>
  </attributes>
</representation>

```

Код №	Наименование	Первый стиль
1	Здесь атрибуту присвоена непосредственно строка со стилем.	Color=\$00587920;FontColor=\$008EE4E;FontUnderline=1;FontBold=1;
2	Здесь атрибуту присвоено название пресета.	dOrangeWhite
3	Строка с пустым атрибутом стиля	

Также стиль можно менять в avi используя `selection`

```

selection.attrs("Имя атрибута, на который мы хотим применить стиль").styleAttributeName
↳= "Имя атрибута со строкой стиля или с системным именем пресета"

```

Создание строки стиля. StyleBuilder

StyleBuilder - класс, позволяющий работать со стилем не как со строкой, а как с объектом, в котором хранятся параметры. Помимо этого в StyleBuilder хранятся цветовые и шрифтовые константы для удобства использования.

Внутри класса хранится изменяемый словарь, который хранит пары **название параметра GS -> (значение параметра, название параметра CSS)**. Для его корректной работы есть вспомогательный неизменяемый словарь, хранящий пары **название параметра GS -> название параметра CSS**. Все манипуляции со словарём происходят через сеттеры и геттеры. Для создания результирующей строки тоже есть отдельный метод.

Изначально класс создан для стилей, которые используются в Global, но класс также поддерживает создание результирующей строки и для CSS.

Создание экземпляра StyleBuilder

Для использования StyleBuilder нужно сначала создать его экземпляр. Можно создать как и пустой стиль

```

val builder = StyleBuilder()
builder.build() //Результирующая строка пустая

```

так и взять за основу уже существующую строку стиля

```

val stringStyle = "color=$00FF4400;font=arial;fontsize=15".ns
val builder = StyleBuilder(stringStyle)
builder.build() //В результирующей строке будут все те же параметры с теми же значениями,
                //что и в stringStyle, но возможно параметры будут расположены в другом
↳ порядке.

```

Работа с цветовыми параметрами

Для работы с цветовыми параметрами внутри `StyleBuilder` создан класс `Color`. Методы работы с цветовыми параметрами принимают на вход и возвращают экземпляры этого класса. На вход в сеттер можно подавать обычную строку типа `String`, она будет неявно преобразована в экземпляр `Color`.

При создании от строки `Color` пытается преобразовать цвет в HEX, если цвет невозможно преобразовать - бросается исключение. `Color` умеет преобразовывать строки типа:

- `$00BBGGRR`
- `$BBGGRR`
- `#RRGGBB`
- Наименование GS константы

Из `Color` можно получить строку обратно в формате HEX или GS кода:

```
val color = Color.Red //В Color есть константы Gs цветов
//ИЛИ
val color = Color("$000000FF")
//ИЛИ
val color = Color("$0000FF")
//ИЛИ
val color = Color("#FF0000")
//ИЛИ
val color = Color("clRed")
//-----
color.getGsHex // Вернётся $000000FF
color.getHex   // Вернётся #FF0000
```

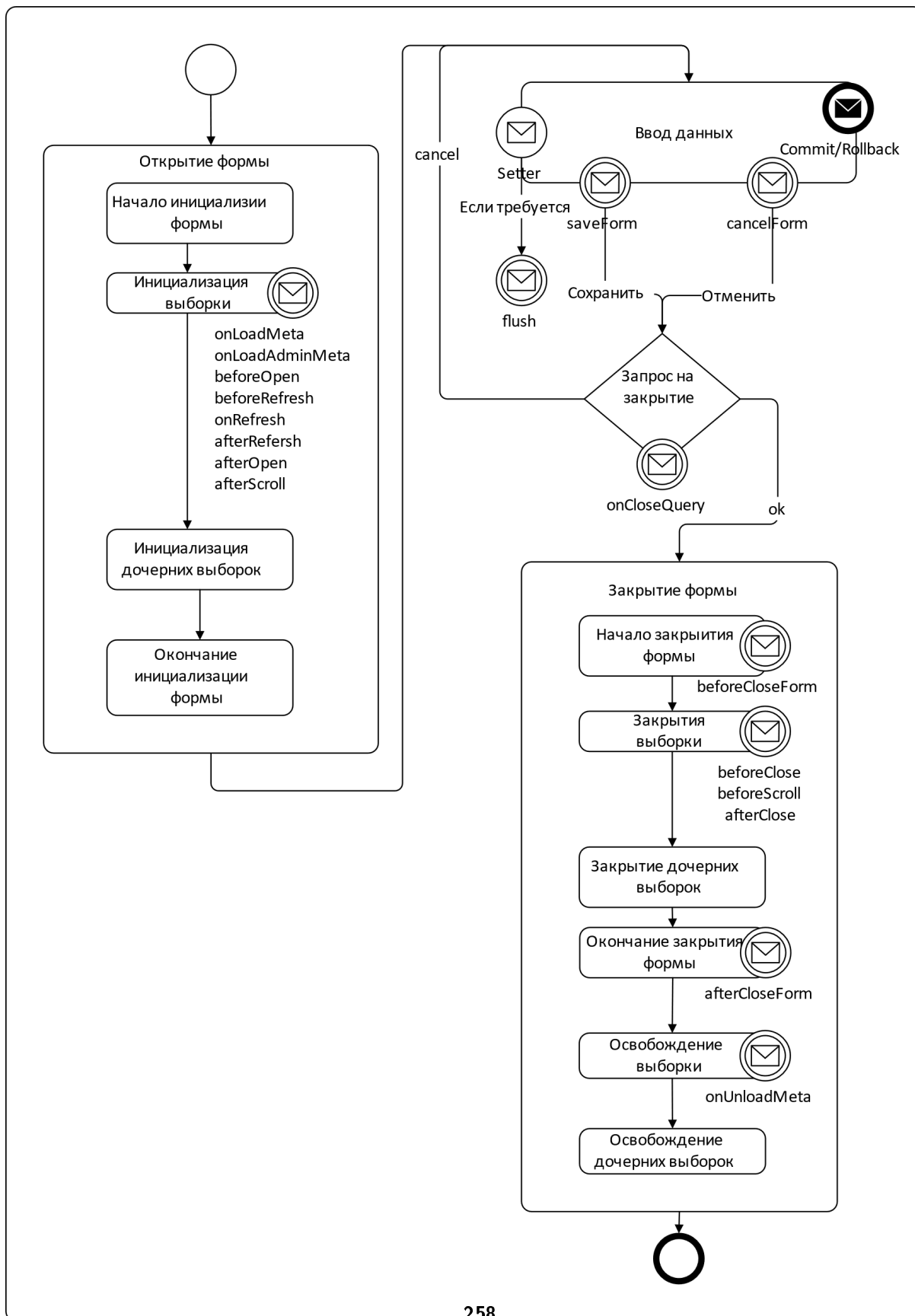
Если в `Color` подать пустую строку - это считается как отсутствие цвета.

Примеры

```
val style = StyleBuilder()
    .setBackgroundColor(Color.Blue)
    .setFontColor("#FFAAFF")
    .setFont(Font.Impact)
    .setFontUnderLine(1.nn)
    .build() // style = "color=$00FF0000;fontcolor=$00FFAAFF;fontunderline=1;
    ↪ fontname=Impact"

val style2 = StyleBuilder(style)
    .setBackgroundColor(Color("")) // Убрать параметр цвета заднего фона
    .setFontColor("$00FFFFFF")
    .setFont("") // Убрать параметр шрифта
    .setFontUnderLine(NNumber()) // Убрать параметр нижнего подчёркивания
    .setFontBold(1.nn)
    .buildCSS // style2 = "color: #FFFFFF;
              //          font-weight: bold;"
```


Жизненный цикл формы



Иконки

Фреймворк поддерживает три варианта подключения иконок:

1. Из ресурсов сервера (основной режим)
Коллекции изображений берутся из каталога с ресурсами `[G3_HOME]/resources/imagecollection`.
Каждый подкаталог является коллекцией, вложенные файлы - элементы коллекции, номера которых соответствуют именам файлов.
2. Из базы данных
Коллекции изображений загружаются из таблицы `btk_component`, поле `clobdataxml`
3. Подключение иконки по url
Для иконки указывается явный url

Использование иконок из java-ресурсов модуля

Файлы изображений размещаются в ресурсных каталогах соответствующего модуля.

На скриншоте приведён пример структуры каталогов с ресурсами модуля ВТК. Коллекции изображений размещены в каталоге ресурсов `../ru/bitec/app/btk/images`. Каждой коллекции соответствует одноимённый каталог.

Рассмотрим структуру каталога `ru/bitec/app/btk/images/toolbar`.

- В корне каталога размещаются файлы изображений с минимальным разрешением 16x16 точек.
- В подкаталоге `ru/bitec/app/btk/images/toolbar/24x24` размещаются файлы изображений с разрешением 24x24 точек. В случае необходимости могут быть добавлены каталоги изображений с большим разрешением.
- Подкаталог `ru/bitec/app/btk/images/toolbar/disabled` содержит обесцвеченные копии изображений коллекции `toolbar`, необходимые для отображения неактивных операций на панелях управления.

Пример указания изображения для операции с использованием аннотации:

```
@Oper(
    caption = "Под",
    headOperation = "references",
    imageUri = "ru/bitec/app/gs3/images/toolbar/61.png")
def open_Bs_Kind_RoList(): Unit = {
    Bs_KindAvi.roList().newForm().open()
}
```

При одновременном указании свойств `imageIndex` и `indexUri`, второе имеет приоритет.

Частично-загружаемые деревья

Введение

Частично-прогружаемое дерево – это такой вид отображения, где список в форме дерева подгружается из базы только для тех записей, которые раскрыты в дереве.

Частично прогружаемые деревья используются для отображений, в которых заведомо известно, что будет огромное количество записей, например `Mct_Structure`.

Настройка avm файла

Для отключения полной загрузки дерева в avm файле в настройках отображения необходимо указать свойство

```
fetchAllTree="false"
```

Помимо этого, в настройках фрейма необходимо указать наименование атрибута, отвечающего за отображение наличия у записи потомков

```
<tree idAttr="gid"
      idParentAttr="gidParent"
      hasChildrenAttr="bHasChild"
/>
```

Настройка refresh

Для отображения необходимых записей нужно получить список `idAttr` и `idParentAttr`, по которым будут загружаться данные из базы. Для этого нужно воспользоваться значениями параметров выборки `OPENNODEIDARRAY` и `OPENNODEID`

- `OPENNODEIDARRAY`
Параметр выборки, хранящий в себе список открытых нод через запятую в формате `ftString`
- `OPENNODEID`
Параметр выборки, передающий `idAttr` открываемой ноды в формате `ftString`

`OPENNODEIDARRAY` и `OPENNODEID` – системные параметры. Пользовательская установка значений этих параметров запрещена

Пример

Пример запроса для `Mct_StructureAvi#StructureTree`, находящегося в пакете `Mct_StructurePkg`. `getOnRefreshBs`

```
--п.1.- открытые узлы
with nodes as (
  SELECT distinct tVal.tVal as gid
  FROM regexp_split_to_table(
    concat_ws(
      ', '
    ,:OpenNodeIdArray#
```

(продолжается на следующей странице)


```

        ,:OpenNodeIdArray
    )
    ,', '
) tVal
where :OpenNodeId is null -- узлы нужны только при рефреше
union
select :gidParent as gid
where :gidCurrentGid# is not null
),
-- фильтрация объектов для вывода в дерево
gidFlt as (
    select h.gid
        ,h.gidParent
    from (
        -- п.2. блок рефреша узлов "RefreshNodes"
        select t.gid
            ,case when ll.id is null then null else l.gidParent end as gidParent
        from mct_structure t
        join mct_structurelink l on l.gidstructure=t.gid
        left join mct_structurelink ll on (
            l.gidparent=ll.gidstructure and
            l.idviewtype=ll.idviewtype
        )
        where t.idPrjVer = :flt_idPrjVer
            and l.idviewtype=:flt_idViewType
            and l.gidparent in (select n.gid from nodes n)
        -- Блок OnRefresh
    union all
        -- п.3. 1. Раскрываем заказ - корневые записи
        select t.gid
            ,null as gidParent
        from mct_structure t
        join mct_structurelink l on l.gidstructure=t.gid
        l.idviewtype=ll.idviewtype
    )
    --left join mct_structure tt on (l.gidparent=tt.gid)
where t.idPrjVer = :flt_idPrjVer
    and l.idviewtype=:flt_idViewType
    and not exists (
        select 1
            from mct_structurelink ll
            where ll.gidstructure=l.gidparent
                and ll.idviewtype=:flt_idViewType)
        and :OpenNodeId is null
    --
    union all
--п.4. - 2. Раскрываем потомков - по нажатию ноды
select l.gidstructure as gid
    ,l.gidparent
from mct_structurelink l
where l.idviewtype = :flt_idViewType
    and l.gidparent = :OpenNodeId

```

```

) h
--n.5.
SELECT
    t.id
  ,t.idClass
  ,t.gid
  ,tt.gidParent
  ,(select COALESCE(max(1),0) from mct_structurelink l where
l.gidparent=t.gid) as bHasChild
  ,t.gidDoc
  ,t.gidDocVer
  ,t.gidSourceObj
  ,t.idPrjVer
  ,t1.sCode as idPrjVerHL
  ,t.idEskd
  ,t2.sCaption as idEskdHL
  ,t.idPosType
  ,t3.sCaption as idPosTypeHL
  ,t.sCode
  ,t.sCaption
  ,t.sSysName
FROM Mct_Structure t
join gidFlt tt on t.gid=tt.gid
LEFT JOIN Bs_PrjVer t1 on t.idPrjVer = t1.id
LEFT JOIN Mct_Eskd t2 on t.idEskd = t2.id
LEFT JOIN Mct_PosType t3 on t.idPosType = t3.id
LEFT JOIN Bs_Goods t4 on t.idGds = t4.id
LEFT JOIN Msr_MeasureItem t5 on t.idMsr = t5.id
LEFT JOIN Mct_OrderSheet t6 on t.idOrderSheet = t6.id

```

- *n.1*
Получение таблицы значений открытых нод с учетом открытия ноды для новой созданной записи уровнем ниже
- *n.2*
Получение записей, которые входят в открытые ноды
- *n.3*
Получение корневых записей дерева
- *n.4*
Получение записей, которые входят в открытую ноду – блок, срабатывающий при первом раскрытии ноды
- *n.5*
Основной запрос для получения необходимых колонок.

Так же важно обратить внимание, что в основном запросе присутствуют изменения:

- Добавлен атрибут bHasChild
Признак наличие дочерних элементов.
Текст запроса атрибута:

```

(select COALESCE(max(1),0) from mct_structurelink l where
l.gidparent=t.gid) as bHasChild

```

- Добавлено ограничение получения записей по отфильтрованному списку

```
join gidFlt tt on t.gid = tt.gid
```

Загрузка дочерних записей на открытие

Если необходимо открыть выборку с уже прогруженными дочерними записями, можно при открытии передать в качестве параметров список нод для загрузки. И в качестве параметра, используемого в **refresh**-е использовать его. Однако, в таком случае все ноды будут закрыты и их нужно будет открывать вручную.

Примечание

Изменение параметров, участвующих в запросе, спровоцирует операцию **refresh**!

Для установки параметров без вызова **refresh** необходимо воспользоваться конструкцией

```
try {
  selection.ignoreParamChange = true
  setVar("gidParent", curGidParent)
  setVar("gidCurrentGid#", thisRop().gid)
}
finally {
  selection.ignoreParamChange = false
}
```

Пример разметки выборки

```
<?xml version="1.0"?>
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.global-system.ru/xsd/global3-view-1.0"
      name="Bs_GdsCostDeviationType">

  <representation name="Default" doubleClickOperation.createFormMode="CardEdit"
                  doubleClickOperation.lookupMode="CloseFormOK"
                  caption="Виды отклонений в стоимости ТМЦ">
    <filter name="Bs_GdsCostDeviationTypeFilter">
      <macros name="DefFltReferenceMacro">
        <condition logicalOperator="and" id="shownotused"
                    isExpression="true"
                    expression="(:filter$flt_bShowNotUsed = 1
                                or
                                (coalesce(t.bnotactive,0) = 0
                                 and (t.dexpirydate is null or t.dexpirydate >
↪current_date)
                                )
                    )">
          <filterAttr name="flt_bShowNotUsed" attribute-type="Long" caption=
```

(продолжается на следующей странице)

```

↪ "Отображать неиспользуемые"
                                isLastInLine="false" order="10" defaultValue="0"
↪ editorType="check"/>
                                </condition>
                                </macros>
                                </filter>
                                <layout>
                                    <simpleComposer>
                                        <frame filter.isVisible="true" toolBar.isVisible="true">
                                            <grid/>
                                        </frame>
                                    </simpleComposer>
                                </layout>
                                <attributes>
                                    <attr name="id" caption="Идентификатор" isVisible="false" editorType="edit"
↪ order="-1"/>
                                    <attr name="idClass" caption="idClass" isVisible="false" editorType="edit"
↪ order="-2"/>
                                    <attr name="gid" isVisible="false" editorType="edit"/>
                                    <attr name="sCode" caption="Код" editorType="edit" order="10" isRequired=
↪ "true"/>
                                    <attr name="sCaption" caption="Наименование" editorType="edit" order="20"
↪ isRequired="true"/>
                                    <attr name="sDescription" caption="Описание" editorType="memo" order="30"/>
                                    <attr name="bNotActive" caption="Не используется" editorType="check"/>
                                    <attr name="dExpiryDate" caption="Дата окончания использования" editorType=
↪ "datePick"/>
                                </attributes>

                                <operations>

                                </operations>

                                </representation>

                                <representation name="List" editMode="notEdit">
                                </representation>

                                <representation name="Card" editMode="edit" stdFilter.isAvailable="false">
                                    <layout>
                                        <simpleComposer>
                                            <frame filter.isVisible="false">
                                                <card/>
                                            </frame>
                                        </simpleComposer>
                                    </layout>
                                </representation>

                                <representation name="Lookup" editMode="notEdit">
                                </representation>

</view>

```

Пользовательская блокировка

Пользовательская блокировка позволяет блокировать бизнес-объект вне транзакции базы данных. Такая блокировка работает в разрезе пользовательского сеанса и даёт возможность пользователю сразу узнать о блокировке, а не столкнуться с этим при попытке сохранения изменений.

Элементы заблокированного бизнес-объекта не могут быть изменены каким-либо другим пользовательским сеансом.

Пользовательскую блокировку можно снять, при этом все заблокированные сессией объекты становятся доступны для блокирования.

Встроенная пользовательская блокировка

Пользовательская блокировка включается при взаимодействии пользователя с интерфейсом в методе `Dvi#beforeEdit`, который является результатом кодогенератора.

```
override def beforeEdit(): Unit = {  
    ...  
    val rop = defaultRep.thisRop()  
    if (rop != null && (Set(ReadRopMode, UpdateRopMode) contains rop.ropMode)) {  
        defaultRep.tryUserLock()  
    }  
    ...  
}
```

lockObject(gid)

```
Btk_FormSessionApi().lockObject(gid)
```

Установка пользовательской блокировки для документа, `gid` которого передан в параметр.

В системное поле `idlockunit_dz` документа, будет установлен номер блокировки, выданный сессии пользователя, с которой вызван метод.

Сессия пользователя представляет запись в таблице `Btk_FormSession`.

В случае, если документ заблокирован какой-то пользовательской сессией и выполняется попытка заблокировать с другой сессии, то метод вызовет исключение с описанием блокировки.

При попытке редактировать коллекцию заблокированного документа также работает пользовательская блокировка.

Потому что при изменении коллекции `beforeEdit()` вызывает пользовательскую блокировку для мастер-документа, который заблокирован.

Таким образом осуществляется блокировка всего бизнес-объекта.

lockObjectMulti(gida)

```
Btk_FormSessionApi().lockObjectMulti(Seq[NGid], riseError = true)
```

Позволяет одной пользовательской сессии заблокировать несколько записей за одну транзакцию в БД. Принцип такой же, как у метода `lockObject(gid)`, но запрос для блокировки вызывается не отдельно для каждого документа, а собирается для всех документов и выполняется 1 раз, т.е. за одну транзакцию в БД.

Можно передавать `gid`'ы различных классов.

Поле `riseError` - нужно ли поднимать исключение при наличии заблокированных записей другой сессией. Выдаст исключение, если хоть один из переданных документов уже заблокирован.

```
Btk_FormSessionApi().lockObjectMulti(Seq[NGid]) //выдаст исключение  
//проигнорирует уже заблокированные документы  
Btk_FormSessionApi().lockObjectMulti(Seq[NGid], riseError = false)
```

lockObjectMultiWithInfErr(gida)

```
Btk_FormSessionApi().lockObjectMultiWithInfErr(Seq[NGid])
```

Данный метод отличается от `Btk_FormSessionApi().lockObjectMulti(Seq[NGid])` тем, что не вызывает исключение и возвращает информацию о результате метода.

Метод возвращает кортеж:

(Seq[NGid], NString)

(Seq[NGid] - `gid`'ы уже заблокированных записей другой сессией, NString - сформированная ошибка)

Данные можно использовать для более сложной логики

Сервис группового редактирования

Основные положения

Операция группового редактирования объявлена в `EntityAvi` в отображение `Default`, что означает, что она доступна в каждой выборке.

Что бы включить операцию группового редактирования, необходимо в вашем отображении переопределить операцию, так как изначально она выключена:

```
@Oper(  
  active = true,  
  headOperation = "extraOperation"  
)  
override def groupEditAttr(): Unit = super.groupEditAttr()
```

Для включения возможности группового редактирования на атрибуте, необходимо в `odm` (Объектно-документном представлении) или `avm` (Прикладную модель представления) добавить флаг `isGroupEditAvailable`:

```
<!-- odm -->
<attr name="dDoc" attribute-type="Date" caption="Дата док-та" order="60" type="basic"
↪isVisible="true"
    editorType="datePick" isHeadLine="true" isCopyInCopyObject="false"
↪isGroupEditAvailable="true" />
```

```
<!-- avm -->
<attr name="dDoc" caption="Дата док-та" editorType="datePick" order="60"
↪isGroupEditAvailable="true" />
```

Для json-атрибутов, при регистрации так же можно включить групповое редактирование (изначально оно выключено):

```
Btk_AttributeApi().registerJsonAttr(
    idpClass = idClass,
    spSystemName = "dCreateDoc".ns,
    spCaption = "Дата создания док-та",
    spAttrType = AttributeTypes.Long.toString,
    spType = AttrTypes.Basic.toString,
    bpGroupEditAvailable = 1.nn
)
```

Если необходимо включить групповое редактирование, не изменяя проектный код, необходимо зайти в приложение "Настройки системы" -> Сущности -> Классы.

В списке атрибутов в столбце «Групповое редактирование атрибутов» поставить галочку.

Редактируемые объекты

Редактируемый объект представляет из себя ссылку на провайдер строки определённого класса. Для классов в отображении происходит поиск атрибута с наименованием `id`, для миксинов - `gidRef`. Если ваше отображение не привязано к классу, то в нём происходит поиск по следующим атрибутам: `id`, `gid`, `gidRef`. Если ни один из вышеперечисленных атрибутов не найден, будет выдано исключение.

Свой список изменяемых объектов

Если необходимо передать свой список изменяемых атрибутов, необходимо переопределить операцию группового редактирования следующим образом:

```
@Oper(
    active = true,
    headOperation = "extraOperation"
)
override def groupEditAttr(): Unit = {
    val idList = (0 until selection.selectedRecordsCount()).map { idx =>
        // Вместо id -> idTask
        NLong.fromAny(selection.selectedValueByName("idTask", idx))
    }.toList

    Btk_GroupEditPkg().execByRep(
        baseRep = this,
        listObjects = idList
    )
}
```

(продолжается на следующей странице)

```

    )
}

```

Установка групп

Если на классе установлена возможность группировки объектов *Руководство разработчика: Сервисные возможности для классов # Группировка*, то в отображении группового редактирования появится закладка «Установка групп», на которой можно задать группы для выделенных записей.

Дополнительные закладки для группового редактирования

Для отображения группового редактирования есть возможность добавить новые закладки для дополнительной обработки объектов. Для этого необходимо переопределить операцию группового редактирования, в которой вызывается `Btk_GroupEditPkg().execByRep`, с параметром `addTabs`, который принимает последовательность объектов закладок для данного отображения.

Закладки представлены в виде кейс класса `GroupEdit_TabsRow`, который принимает в себя данные об отображении и функцию обработки объекта.

Функция обработки объекта

Данная функция принимает в себя один параметр в виде кейс класса `GroupEdit_TabsRowParam`, который содержит в себе сессию выполнения группового редактирования и редактируемый объект, который может быть представлен в виде `NGid` или `NLong`.

Универсальный фильтр

Основные положения

Операции универсального фильтра объявлены в `EntityAvi`, что означает, что он доступен в каждой выборке.

Вся фильтрация построена на создании объектов `scala`-классов, которые хранятся в памяти. Сохранение всех настроек осуществляется в таблицу базы данных в виде `json`.

Универсальной фильтрацией управляет объект, который хранится в переменной `fltManager`. Для каждого экземпляра отображения создается свой экземпляр менеджера фильтра.

По умолчанию для фильтрации доступны атрибуты выборки, а так же класса, который определяется через `Avi`-функцию `thisApi()`.

Сохраненная настройка не обращает внимания на изменившиеся свойства атрибутов. Она использует те настройки атрибутов, которые были в момент добавления этого атрибута с панели. Что бы изменить поведение атрибута в настройке, необходимо его удалить, затем снова добавить и сохранить настройку.

Физические атрибуты и выражения фильтрации

- Атрибуты в макросе представлены их «физическими атрибутами». По умолчанию физический атрибут равен системному имени атрибута, но существуют атрибуты, для которых это неверно. Примером может служить технический атрибут класса «IdObject#», имеющий физическое имя «ID» (или gidRef для Mixin-ов и пр.). В макросе такой атрибут всегда представлен как ID.
- Для атрибутов фильтра можно задать выражение фильтрации. Это произвольное SQL-выражение, определяющее значение атрибута. Например, такие выражения используются в объектных характеристиках, хранящихся в json

Активация фильтра в выборке

Для использования универсального фильтра в отображении необходимо активировать операцию uniFilter. Так же удостовериться в активности операций onApplyFilter, onFilterFinalize, onFilterInit

Внимание

Фильтр не работает в выборках с объектными запросами

HL-атрибуты

HL – атрибуты (далее IdAttrHL) считаются ссылочными атрибутами и в фильтрации не участвуют. Фильтрация осуществляется по их id-атрибутам (далее IdAttr).

- Если в выборке присутствует IdAttrHL, но отсутствует IdAttr, то фильтрация по атрибуту IdAttrHL будет недоступна.
- Все расширенные настройки требуется указывать на атрибуте IdAttr
- Ссылочность определяется:
 - Через тег <ref>, например: <ref class="Rpl_IntOutSession"/>
 - через одноименный атрибут класса этой выборки

Связь выборки и класса

Для выборки класс определяется через Avi-функцию thisApi()

Связь атрибута выборки и атрибута класса

Если у выборки определен класс, то для атрибута выборки его сопоставленным атрибутом класса будет атрибут с таким же системным именем.

Тип редактора атрибута

Тип редактора атрибута в большинстве случаев такой же, как указан в выборке, исключения:

- Выпадающие списки открываются через редактор с тремя точками
- Даты фильтруются через интерфейс «Фильтрации по периоду»
- Ссылочные атрибуты открываются через редактор с тремя точками
- Специфичные типы сравнения (множественность) изменяют редактор через редактор с тремя точками .

Для атрибутов класса тип редактора определяется:

- Если указан в odm, то этот тип редактора.
- В остальных случаях определяется от типа атрибута и его типа данных.

Коллекции класса

В перечне атрибутов, доступных для фильтрации (как для выборок с классами, так и для самих классов), некоторые пункты отмечены красными иконками. Эти иконки позволяют фильтровать данные по коллекциям. Коллекции доступные для фильтрации определяются через odm- файл в теге <collections>.

Так же в карточке класса на закладке «Коллекции универсального фильтра» есть возможность добавить дополнительные коллекции для фильтрации. Например, подключить какой-то класс, который не входит в бизнес-объект, но имеет ссылки на текущий.

На этой же закладке можно переопределить и стандартные коллекции класса:

- отключить фильтрацию по коллекции, сняв признак «Активность»
- переименовать коллекцию, изменив поле «Наименование»

Доступные элементы для фильтрации

Типы доступных элементов

- Атрибут класса/выборки – доступен в случае генерации доступных атрибутов по классу или выборке
- Коллекции - добавляются при генерации по классу
- Объектные характеристики, хранящиеся в json :
 - Фильтрация:
 - * Осуществляется через выражение, которое получает значение атрибута из json
 - Заполнение на панели доступных атрибутов:
 - * Все объектные характеристики класса или класса, с которым сопоставлена выборка.
- Универсальные характеристики :
 - Фильтрация:
 - * Для единичных - осуществляется через выражение, которое получает значение атрибута из json

- * Для множественных - формирует выражение через exists или not exists, по аналогии с фильтрацией по коллекциям.
- Заполнение на панели доступных атрибутов. Все универсальные характеристики, объявленные на:
 - * группах класса, если у класса включена Группировка
 - * типах объекта класса
- Постоянные/технические атрибуты класса
 - Заполнение на панели доступных атрибутов:
 - * Всегда, при раскрытии узла, ссылоного на класс
 - * Всегда, если выборка связана с классом
 - Атрибуты
 - * sHeadLine# (Заголовок)
 - * sMnemonicCode# (Мнемокод)
 - * dModifyDate# (Дата изменения)
 - * dCreateDate# (Дата создания)
 - * sModifyUser# (Изменивший пользователь)
 - * sCreateUser# (Создатель)
 - * IdGroup# (Группа) – заполняется только для классов, имеющих группировку. Позволяет фильтроваться по группе
 - * idObject# (Объект) – специальный атрибут, позволяющий фильтроваться по идентификатору(или gidRef) объекта

Первичная генерация по выборке

В операции lazyInitFilter происходит заполнение доступных атрибутов по выборке. Ниже рассмотренные варианты.

Если у выборки указан класс, то формируются атрибуты по этому классу. Атрибутами выборки считаются все атрибуты, присутствующие в дата-сете. Типы данных таких атрибутов также определяются их типами данных в дата-сете. Атрибуты выборки будут сгенерированы только если выборка активна (загружен дата-сет). Сформированные классу атрибуты, присутствующие в выборке, будут дополнены свойствами из выборки (тип редактора, ссылоность и тд)

Генерация по классу

Для класса формируются:

- Атрибуты класса (кроме idClass), с не отключенным свойством доступности в фильтре

```
<attr name="bError">
  <uniFilter isActive="false"/>
</attr>
```

- Все объектные характеристики, настроенные пользователями и хранящиеся в jObjAttrs_dz
- Технические атрибуты (заголовок, дата создания, объект и тд)

- Коллекции, перечисленные в odm и в карточке класса на закладке «Коллекции универсального фильтра»
- Универсальные характеристики, объявленные:
 - на группах класса, если в классе включен сервис группировки
 - на типах объекта класса, если сервис группировки не включен для класса.

Примечание

Для управления формированием атрибутов класса или добавления новых используется специальная точка расширения.

Подробности в главе [Расширенная настройка - Класс](#)

Атрибуты переменной ссылочности или ссылочные на миксин.

Атрибуты переменной ссылочности фильтруются через их раскрытие в дереве доступных элементов. Такие атрибуты раскрываются на классы, доступные к фильтрации по этому атрибуту

- Атрибут ссылочный на миксин, фильтруется по всем классам, к которым подключен этот миксин
- Атрибут, ссылочный на один класс, сразу фильтруется по этому классу
- Атрибут, у которого настроен перечень ссылочных классов, фильтруется по всем классам, указанных в этом перечне. Пример настройки:

```
<attr name="gidRefDocument" attribute-type="Varchar" caption="Документ" order="20"
  type="refAnyObject">
  <uniFilter>
    <refAnyObject>
      <ref name="Wf_Doc"/>
      <ref name="Cnt_Contract"/>
    </refAnyObject>
  </uniFilter>
</attr>
```

- Атрибут с ненастроенной ссылочностью фильтруется как значимый. Приоритет обработки ссылочности:
 1. Ссылочный на класс в refClass в теге uniFilter
 2. Ссылочный на перечень классов в теге refAnyObject в теге uniFilter
 3. Ссылочный на класс в refClass в настройке самого атрибута.

Формирование макроса

Макрос формируется в операции onApplyFilter. По умолчанию в фильтре создается главная группа, имеющая признак bisMainGroup = 1 и системное имя MainGroup, по ней и формируются макросы.

Каждая фильтруема коллекция будет представлять Exists в макросе.

Режим onlyWhere

Стандартный режим работы фильтра, в результате выдается текст одного макроса, который будет представлять собой exists, примером может служить:

```
Exists (  
  select 1  
    from bs_goods t#2  
   where t#2.sName = '1'  
      and t#2.id = t.id  
)
```

Обработка сервером приложений сформированного макроса

На операции onApplyFilter сформированный макрос передается серверу, чтобы он наложил условие на запрос данных выборки:

```
selection.filters().server.alias = mainGroupAlias  
selection.filters().server.condition = fltMacro.where  
selection.filters().server.isEnabled = true
```

Фильтрация ROP-объектов в универсальном фильтре

Принцип работы

Механизм фильтрации ROP-объектов работает следующим образом:

1. Преобразование данных: ROP-объекты преобразуются во временную JSON-структуру
2. Формирование SQL-запроса: На основе настроек универсального фильтра формируется SQL-выражение
3. Выполнение фильтрации: SQL-запрос выполняется против JSON-данных
4. Возврат результата: Возвращаются только те ROP-объекты, которые соответствуют условиям фильтрации

Использование

Основной метод

Для применения универсального фильтра к коллекции ROP-объектов используется метод `applyUniFilter` из пакета `Btk_FltObjectFilterPkg`:

```
val filteredRops = Btk_FltObjectFilterPkg().applyUniFilter(rops, fltManager)
```

Параметры метода

- `rops: Seq[Rop[ID, AR0]]` - коллекция ROP-объектов для фильтрации
- `fltManager: FltManager` - менеджер универсального фильтра с настроенными условиями

Возвращаемое значение

Метод возвращает отфильтрованную коллекцию ROP-объектов, содержащую только те записи, которые соответствуют условиям универсального фильтра.

Примеры использования

Базовый пример

```
// Получаем коллекцию ROP-объектов
val rops = SomeClassApi().getList()

// Применяем универсальный фильтр
val filteredRops = Btk_FltObjectFilterPkg().applyUniFilter(rops, fltManager)

// Работаем с отфильтрованной коллекцией
filteredRops.foreach { rop =>
    // обработка отфильтрованных объектов
}
```

Использование в выборке

```
override def onRefresh: Recs = {
    // Получаем исходную коллекцию ROP-объектов
    val allRops = loadRopObjects()

    // Применяем фильтрацию, если универсальный фильтр активен
    val filteredRops = if (fltManager.isActive) {
        Btk_FltObjectFilterPkg().applyUniFilter(allRops, fltManager)
    } else {
        allRops
    }
}
```

(продолжается на следующей странице)

```

    filteredRops
}

```

Передача условий фильтра через параметры выборки

Для формирования условия используется объект класса `FltCondition`. В качестве имен атрибутов и коллекций можно использовать:

- Системное имя – в таком случае атрибут или коллекция ищется на первом уровне дерева доступных элементов.
- Путь – позволяет добавлять условия со вложенных уровней доступных элементов. Например, по атрибуту класса, на который ссылается атрибут верхнего уровня. Путь можно получить через `debugger` (атрибут `sPath`), стоя на доступном к фильтрации элементе. Пример использования:

```

val fltCondition = FltCondition()

//условия группы "отбор"
fltCondition.withMainGroup{mainGroup =>
    //добавляем условие по атрибутам выборки
    mainGroup.addAttr("nNumber", 1.nn)
    //условие "Заполнен"
    mainGroup.addIsNotNullAttr("sCaption")
    //добавляем условие по ссылочному атрибуту с раскрытием его на подуровни
    mainGroup.addAttr("attr:idPurchaseDirection/attr:sCaption", "643")

    //добавляем условие по атрибуту класса выборки, который отсутствует в дата-сете
    mainGroup.cl.addAttr("sCreateUser_dz", "admin")

    //добавляем условие по атрибуту даты
    mainGroup.cl.addDateInterval("dCreateDate_dz", "10.02.2020".nd, "10.02.2021".nd)

    //условие "в списке"
    mainGroup.addAttr("idRef", Seq(1.n1, 2.n1, 3.n1))

    //условие "Не в списке"
    mainGroup.addAttr("idRef", Seq(1.n1, 2.n1, 3.n1), FltAttributeCompareKind.notInList)

    //добавляем группу "Или"
    mainGroup.addOrGroup{orGr =>
        orGr.addAttr("sCaption", "123")
        orGr.addAttr("sCaption", "321")
    }

    //добавляем условие по коллекции
    mainGroup.withCollection("Btk_SomeCollection") {col =>
        col.addAttr("sCaption", "someValue")
    }
}

//добавляем условие по другой рутовой группе

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
fltCondition.withRootGroup("groupName"){group =>
  //условия по атрибутам и коллекциям
  //...
}

//передаем условие в выборку
Btk_ClassAvi.defList.newForm().params(Map("uniFilterCondition_dz" -> fltCondition)).
  ↪open()
```

Внимание

Если нужно добавить условия по атрибуту класса выборки, который отсутствует в дата-сете, и объединить их в отдельную группу «И»/группу «ИЛИ», необходимо добавить .cl непосредственно перед добавлением соответствующей группы, а не самого атрибута.

Корректный пример добавления группы:

```
fltCondition.withMainGroup { mainGroup =>
  mainGroup.cl.addOrGroup { orGr =>
    orGr.addAttr("attr:id/attr:jAttrDataDeb_sCaption", "123")
    orGr.addAttr("attr:id/attr:jAttrDataCr_sCaption", "123")
  }
}
```

Следующий код не будет работать (ни при компиляции, ни при работе. Вы не получите сообщение об ошибке, но переданное подобным образом условие функционировать не будет):

```
fltCondition.withMainGroup { mainGroup =>
  mainGroup.addOrGroup { orGr =>
    orGr.cl.addAttr("attr:id/attr:jAttrDataDeb_sCaption", "123")
    orGr.cl.addAttr("attr:id/attr:jAttrDataCr_sCaption", "123")
  }
}
```

Примечание

Узнать путь до определенного атрибута универсального фильтра можно при помощи отладчика (Ctrl+Alt+Shift+w). Заметим, что при формировании в коде пути до ссылочного атрибута необходимо указывать строку пути полностью, например, attr:idPurchaseDirection/attr:sCaption. Для обычных атрибутов указывается сокращенный путь, например, строка mainGroup.addAttr(«nNumber», 1.nn) добавляет условие фильтрации по атрибуту attr:nNumber.

События универсального фильтра

Работа универсального фильтра сопровождается вызовом событий — ключевых моментов жизненного цикла фильтра, которые позволяют переопределять его поведение в прикладных классах. Каждое такое событие определяется одним из наследников интерфейса `FltEvent`. Например, к событиям относится взаимодействие пользователя с объектами фильтра: установка значения ссылочного атрибута через интерфейс и т.п.

Список событий

Внимание

Все используемые типы событий должны быть добавлены в перечисление `FltEvent.Type`.

Событие	Описание
<code>FltBeforeClient</code>	Срабатывает перед установкой значения ссылочного атрибута из интерфейса. Может переопределить <code>setterParams</code> (<i>инициализируется пустым</i>) и <code>availableValues</code> (<i>создаётся от <code>availableValues</code> атрибута фильтра</i>)

Обработчик событий

Обработчиком событий УФ называется класс, который наследует интерфейс `FltEventHandler` и определяет `handleEvent(event: FltEvent)`. Экземпляр создаётся на этапе инициализации фильтра: `EntityAviDefault#onFilterInit`.

На данный момент существует единственный и стандартный обработчик — `FltAviEventHandler`:

- передаёт объект события через переменную выборки;
- находит прокси-операцию `EntityAviDefault#invokeHandleFltEvents` в целевой выборке с учётом переопределений в прикладном коде;
- выполняет обработку события.

Примечание

Можно создать собственный обработчик с изменённой логикой.

Способы использования

Переопределение свойств

Целевая выборка, в которой определен универсальный фильтр, может переопределять функцию обработки событий `EntityAviDefault#handleFltEvents`. Это основной механизм взаимодействия с событиями УФ.

- **Работают с мультивыбором:** Переопределённые параметры корректно передаются и обрабатываются для атрибутов фильтра с мультивыбором.

- **Переопределённые параметры имеют высший приоритет:** Параметры ивента имеют приоритет над всеми остальными, поэтому могут перезаписывать ранее заданные ключи (даже ключи, передающиеся по умолчанию).

Это даёт большую свободу в изменении поведения универсального фильтра и связанного с ним стандартного фильтра. Пример использования можно найти в разделе [Практические советы](#)

Описание scala-классов, используемых фильтром

Во всех отображениях присутствует переменная `fltManager`, которая является менеджером фильтров. Она хранит объект, создающийся в операции `onFilterInit`.

Менеджер управляет фильтром, загружает/выгружает настройки, формирует макрос. Основные классы фильтра находятся в пакете `ru.bitec.app.btk.flt`

Api-методы выборки

- `onFilterInit` – событие инициализации фильтра. Создается фильтр-менеджер `fltManager`
- `onApplyFilter` – первичное применение настройки по умолчанию при отрытии выборки. Формирование макроса, применение макроса.
- `onFilterFinalize` – уничтожение фильтра.
- `uniFilter` – операция фильтра, если ее активировать, то будет возможность фильтрации.
- `lazyInitFilter` – ленивая инициализация. Вызывается при обращении к фильтру. Формирует главную группу фильтра.

Расширенная настройка

Класс

Разметка класса (odm.xml)

Для атрибута доступен тег `uniFilter`:

- `isActive` - по умолчанию включено, определяет наличие этого атрибута в перечне доступных для фильтрации
- `refClass` – ссылочный класс, перекрывает настройку ссылочности самого атрибута(указанного в теге `attr`)
- тег `refAnyObject` – позволяет настроить перечень классов, на который ссылается атрибут переменной ссылочности

Пример настройки атрибута класса в `odm.xml`:

```
<attr name="bError" attribute-type="Number" caption="С ошибками" order="70" type="basic"
↪editorType="check" defaultValue="0">
  <uniFilter isActive="false" refClass = " Bs_Goods">
    <refAnyObject>
      <ref name="Bs_Goods"/>
    </refAnyObject>
  </uniFilter>
```

(продолжается на следующей странице)

```
<booleanColumn/>
</attr>
```

Управление атрибутами в бизнес-логике

Для того чтобы управлять атрибутами в бизнес-логике используется точка расширения `Btk_Ext.afterBuildFltEntityMeta`, которая позволяет изменять атрибуты, которые формируются по умолчанию, а так же добавлять свои собственные.

Для управления атрибутами используются методы пакета `ru.bitec.app.btk.flt.meta.Btk_FltMetaAttributePkg`.

Примеры использования

Коллекции

Для классов реализована возможность настройки отображения коллекций в УФ. Настройка находится в карточке класса на закладке «Коллекции универсального фильтра». Позволяет подключать дополнительные коллекции или переопределять существующие.

Если признак «Активность» снят, то такая коллекция не будет отображаться в фильтре. Коллекция метаданных считается переопределенной, если существует запись, у которой ссылочный класс и атрибуты соединения равны коллекции из мета данных класса.

Для коллекций атрибут мастера – `id`, для v-коллекций – `gid`.

Для регистрации записей в настройку используется метод `ru.bitec.app.btk.class_.flt.Btk_ClassFltCollectionApi#register`

Выборка

В авт выборки для атрибута доступен тег `uniFilter`, имеющий свойства:

- **isActive** – если снят, то такой атрибут не будет отображаться в списке доступных для фильтрации. По умолчанию включено.
- **isSelectionAttr** – если снят, то такой атрибут будет фильтроваться как атрибут класса (через join с таблицей класса по id), если у выборки нет класса, то атрибут не доступен для фильтрации. По умолчанию включено.
- **refSelection** – позволяет явно указать имя выборки, которая будет открываться при выборе значения в фильтре. Имеет приоритет перед свойством `refSelection` тега `ref`
- **refRepresentation** – позволяет явно указать имя отображения, которое будет открываться при выборе значения в фильтре. Имеет приоритет перед свойством `refRepresentation` тега `ref`

Создание дополнительных групп фильтрации

Для фильтрации сложных запросов или подключения фильтрации, не связанной с фильтруемым классом через коллекции или прямую ссылочность, есть возможность создания дополнительных корневых групп фильтрации.

Для этого требуется:

1. Инициализировать новую группу;
2. Сформировать макрос фильтрации по этой группе;
3. Полученный текст подставить в основной запрос onRefresh

Ниже приведен пример создания дополнительной группы, которая позволяет фильтровать по условиям, наложенным на класс ТМЦ

1. На операции lazyInitFilter инициализируем дополнительную группу:

```
override def lazyInitFilter(): Unit = {
  if (!fltManager.isPopulatedRootGroup) {
    Btk_FltPkg().createRootGroupByClass(fltManager, "Bs_Goods", "Bs_Goods")
  }
  super.lazyInitFilter()
}
```

2. На onApplyFilter получаем текст фильтрации по группе:

```
override def onApplyFilter(): Unit = {
  super.onApplyFilter()
  //проверяем, что группы инициализированы и фильтр активен в выборке
  if (fltManager.isPopulatedRootGroup && fltManager.isActive) {
    val alias = "tt"
    val macros = Btk_FltPkg().generateMacroByGroup(fltManager, "Bs_Goods", alias)
    //если наложено условие по этой группе, то формируем текст макроса фильтрации,
    → который будет применим к запросу данных нашей выборки
    if (macros.hasFilter) {
      selection.setMacro(
        "GdsMacro",
        s"""
          exists (
            select 1
              from bs_goods $alias
            where $alias.id = t.idGds -- join к атрибуту из дата-сета (t.idGds)
              and ${macros.where} -- условия по этой группе
          )
        """)
    } else {
      //иначе делаем макрос без наложения условий
      selection.setMacro("GdsMacro", "1=1")
    }
  }
}
```

3. В операции onRefresh используем установленный в onApplyFilter макрос:

- Через метод prepareSelectStatement

```
override protected def onRefresh: Recs = {
  prepareSelectStatement("&GdsMacro")
}
```

- В тексте sql-запроса

```
override protected def onRefresh: Recs = {
  """
  select .....
  from .....
  where ....
    and &GdsMacro
  """
}
```

Связь панели стандартного фильтра с универсальным фильтром

Для атрибутов панели стандартного фильтра можно указать режим, при котором их изменения будут отображаться в универсальном фильтре, а изменения универсального фильтра будут приводить к изменению значений на панели стандартного фильтра.

При сохранении настроек универсального фильтра будут сохранены и настройки атрибутов стандартного фильтра (только атрибутов, для которых указана связь.)

Основным принципом является то, что атрибуты стандартного фильтра задаются по определенным правилам, в сеттерах этих атрибутов используются специальные библиотечные методы.

Если атрибут на панели стандартного фильтра выведен без возможности смены типа сравнения, то в универсальном фильтре ему так же нельзя будет изменить тип сравнения.

Существует 3 режима связи атрибута стандартного фильтра и универсального:

- **Через панель доступных для фильтрации атрибутов**
Режим, при котором атрибуты стандартного и универсального фильтра полностью интегрированы. Ограничение по таким атрибутам накладывается через макрос универсального фильтра
- **Произвольный атрибут**
Режим, при котором ограничения по такому атрибуту накладываются программистом по средствам макросов, формируемых на каждый атрибут.
- **Режим "Только сохранение"**
Режим, при котором значения стандартного фильтра только транслируется в универсальный, без возможности его изменения в универсальном фильтре. Ограничения по таким атрибутам накладываются по правилам стандартного фильтра

Добавление атрибута через панель доступных для фильтрации атрибутов УФ

Если вам требуется вывести на панель стандартного фильтра один из атрибутов, по которым позволяет фильтровать УФ, то можно воспользоваться способом создания атрибута по его пути.

Фильтрация по атрибутам коллекций также доступна в этом режиме.

Настройка

1. Откройте интерфейс универсального фильтра выборки, для которой хотите создать новый атрибут на панели фильтров
2. Откройте панель доступных атрибутов
3. Найдите доступный атрибут, условия по которому хотите накладывать через панель стандартного фильтра
4. Выполните контекстную операцию **Сформировать текст разметки стандартного фильтра**. Операция доступна супер-пользователям или пользователям, обладающих объектной привилегией **Формирование текста разметки связей стандартного фильтра** пакета `Btk_FltPkg`
5. В открывшемся мастере укажите:
 - **системное имя атрибута стандартного фильтра**
По умолчанию `flt_<Имя атрибута УФ>`
 - **наименование**
По умолчанию наименование атрибута УФ
 - **порядковый номер**
Порядковый номер для атрибута стандартного фильтра.
 - **с условием сравнения**
Если признак установлен, то будут сформированы атрибуты стандартного фильтра, позволяющие изменять тип сравнения.
6. Подтвердите выбор в мастере
7. Из открывшегося окна скопируйте в `avm.xml` и `Avi` сформированный текст

Создание ссылочного атрибута с типом редактора «Выпадающий список»

1. Выполните действия описанные в главе **Настройка**.
2. Смените тип данных атрибута на `Long`
3. Сделайте атрибут не видимым
4. Добавьте `h1`-атрибут, в котором настройте выпадающий список.

Описание xml-разметки

Для атрибута стандартного фильтра добавляется тег `uniFilter`, в котором указываются следующие `xml-атрибуты`:

- **attrPath**
Путь атрибута универсального фильтра. Это уникальный путь атрибута, по которому можно однозначно определить атрибут в УФ. Если это атрибут коллекции, то перед его путем добавляется путь до коллекции.
- **attribute-type**
Тип данных атрибута
- **conditionAttr**
Имя атрибута стандартного фильтра с типом сравнения. Содержит имя атрибута, который будет отвечать за смену типа сравнения. Если не указан, то тип сравнения для этого атрибута нельзя будет изменить в универсальном фильтре.

- **conditionType**
Тип сравнения по умолчанию. Если не указан, то будет использоваться стандартный алгоритм определения типа сравнения. В качестве значений используется системное имя объекта `ru.bitesc.app.btk.flt.selected.attribute.FltAttributeCompareKind`
- **defaultValue**
Значение по умолчанию
- **defaultValueType**
Тип значения по умолчанию
- **ref.selection**
Имя выборки, которая будет открываться для выбора значений ссылочных атрибутов
- **ref.representation**
Имя отображения выборки, которое будет открываться для выбора значений ссылочных атрибутов

Добавление произвольного атрибута

Иногда требуется добавить в универсальный фильтр произвольный атрибут, которого нет среди доступных для фильтрации.

Настройка

1. Добавьте в `avm.xml` для нужного атрибута тег `uniFilter` и настройте нужные свойства
2. Добавьте в `Avi` сеттеры для атрибутов:

- атрибут значения:

```
@Setter()
def setflt_someValue(event: SetterEvent): Unit = {
    Btk_FltLib().setStdFilterValue(event)
}
```

- атрибут типа сравнения:

```
@Setter()
def setflt_someCondition(event: SetterEvent): Unit = {
    Btk_FltLib().setStdFilterConditionType(event)
}
```

3. Добавьте ограничение в запрос выборки
Для добавления ограничения добавьте в операции `onRefresh` использование макросов, сформированных по каждому из атрибутов. Имя макроса указывается в свойстве `macros` тега `uniFilter`.

Например:

```
override def onRefresh: Recs = {
    prepareSelectStatement(
        s"""&flt_stdMacroName -- условие по атрибуту, имя макроса которого равно flt_
↪stdMacroName
        """)
}
```

(продолжается на следующей странице)

```
)
}
```

Пример сложного условия с под-запросами. В этом примере, сначала проверяется есть ли наложенное ограничение по атрибуту, и если оно есть, то добавляется под-запрос

```
override def onRefresh: Recs = {
  prepareSelectStatement(
    s"""
      ${if (fltManager.hasStdAttrCondition("flt_someAttrName")) {
        ""exists (
          select 1
            from Some_class tt
            where tt.id = t.idSomeAttr -- связь между основной коллекцией и
↪таблицей      подзапроса
              and &flt_stdMacroName -- макрос с сформированным ограничением по
↪атрибуту
        )"""}
      } else {
        ""
      }
    """)
  )
}
```

Описание xml-разметки

Для атрибута стандартного фильтра добавляется тег `uniFilter`, в котором указываются следующие xml-атрибуты:

- **attribute-type**
Тип данных атрибута
- **type**
Тип ссылочности атрибута. Доступные значения:
 - `basic`
Значимый
 - `boolean`
Булево
 - `refObject`
Ссылочный на объект
 - `refState`
Ссылочный на состояние
 - `refGroup`
Ссылочный на группу
- **expression**
sql-выражение для получения значения атрибута. Например, `tt.idGs`. Будет использовано при формировании макроса атрибута

- **macros**

Имя макроса, в который будет установлено сформированное **sql-выражение** по этому атрибуту. Пример выражения: `tt.idGds = 42`. В качестве правого операнда используется выражение указанное в **expression**

- **valueMacros**

Имя макроса, в который будет установлено **sql-выражение** этого атрибута, но содержащее только часть, относящуюся к значению. Например, `42`. Если в значении указан интервал, то будет использовано выражение вида `(10, 20)`, где первое значение - начало интервала, второе значение - конец. Если одна из границ не задана, то будет указано `null`

- **ref.class**

Ссылочный класс. Используется для атрибутов:

- **Ссылочный на объект**
Содержит имя класса, объекты которого будут доступны для выбора
- **Ссылочный на группу или Ссылочный на состояние** \ Содержит имя класса, которым будет ограничен выбор групп и состояний

- **conditionAttr**

Имя атрибута стандартного фильтра с типом сравнения. Содержит имя атрибута, который будет отвечать за смену типа сравнения. Если не указан, то тип сравнения для этого атрибута нельзя будет изменить в универсальном фильтре.

- **conditionType**

Тип сравнения по умолчанию. Если не указан, то будет использоваться стандартный алгоритм определения типа сравнения. В качестве значений используется системное имя объекта `ru.bitec.app.btk.flt.selected.attribute.FltAttributeCompareKind`

- **defaultValue**

Значение по умолчанию

- **defaultValueType**

Тип значения по умолчанию

- **ref.selection**

Имя выборки, которая будет открываться для выбора значений ссылочных атрибутов

- **ref.representation**

Имя отображения выборки, которое будет открываться для выбора значений ссылочных атрибутов

Пример кода

avm.xml

```
<!-- Атрибуты условия-->
<filterAttr name="flt_idClass_idCond" attribute-type="Varchar" order="20" isVisible=
↪ "false"/>
<filterAttr name="flt_idClass_idCondH1" attribute-type="Varchar" caption="Класс" order=
↪ "21" isLastInLine="false">
  <editor>
    <lookup lookupQuery="gtk-Btk_FltSelectedItemAvi#Lookup_CompareKindStdFilter"
↪ changeableAttr="flt_idClass_idCond" lookupListAttr="sListValue" lookupKeyAttr=
↪ "sKeyValue" isLookupLazyLoad="true"/>
  </editor>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
<card controlWidth="40" isControlWidthFixed="true"/>
</filterAttr>

<!-- Атрибут для ввода значения, ссылающийся на объект-->
<filterAttr name="flt_idClass" attribute-type="Varchar" order="22" editorType="editButton"
↪">
  <editor labelPosition="none"/>
  <uniFilter conditionAttr="flt_idClass_idCondH1" attribute-type="Long" type="refObject"
↪ref.class="Btk_Class" expression="c.id" macros="flt_idClassMacros"/>
</filterAttr>
```

Avi

```
--сеттер атрибута
@Setter()
def setflt_idClass(event: SetterEvent): Unit = {
  Btk_FltLib().setStdFilterValue(event)
}

--сеттер типа сравнения
@Setter()
def setflt_idClass_idCond(event: SetterEvent): Unit = {
  Btk_FltLib().setStdFilterConditionType(event)
}

override def onRefresh: Recs = {
  /*
   *Использование макроса в запросе:
   *Если по атрибуту flt_idClass не будет пользователем выбрано значение, то в макрос
↪поставится 1=1
   *Если по атрибуту flt_idClass будет пользователем наложено условие "равно <ID>", то в
↪макрос поставится "c.id = <ID>"
   */
  """
  select t.ssystemname, count(*) as nCnt
    from btk_attribute t
   join btk_class c
     on c.id = t.idbtkclass
   where &flt_idClassMacros
  """
}
```

Режим «Только сохранение»

Режим, при котором значение атрибута стандартного фильтра транслируется в универсальный, без возможности изменения значений в интерфейсе универсального фильтра.

Такой подход позволяет сохранять значения стандартного фильтра в настройках универсального.

Настройка

1. Настройте атрибут стандартного фильтра используя *базовый подход*
2. Для атрибута добавьте тег `uniFilter`, установите свойство `saveOnly="true"`
3. Добавьте сеттер для фильтра

```
@Setter()
def setflt_someValue(event: SetterEvent): Unit = {
    Btk_FltLib().setStdFilterValue(event)
}
```

Принудительная отправка значения в универсальный фильтр

Иногда требуется заполнять атрибуты стандартного фильтра не вызывая их сеттер. Т.к. на сеттерах выполняется синхронизация значения стандартного фильтра и универсального, то такой подход требует ручного указания, что значения какого-либо атрибута требуется отправить в универсальный фильтр.

Для этого можно воспользоваться методом `Btk_FltLib.submitStdFilterValue`.

Пример использования:

```
@Setter()
def setflt_idClassHl(event: SetterEvent): Unit = {
    event.editButton match {
        case EditButton.lookup =>
            Btk_DialogLib().withLookupIdMulti(Btk_ClassAvi.listForChoose()) { ids =>
                val idss = ids.mkString(";")
                val hls = ids.map(id => Btk_ClassApi().getMnemoCode(id)).mkString(";")

                selection.clientSetSelfVar("flt_idClass", idss) // установка значения другого
                ↪ фильтра без вызова сеттера
                Btk_FltLib().submitStdFilterValue("flt_idClass") // отправка значения
                ↪ стандартного фильтра в универсальный
                selection.clientSetSelfVar("flt_idClassHl", hls)
            }
        case _ =>
    }
    Btk_FltLib().setStdFilterValue(event) // значение фильтра, для которого был вызван
    ↪ сеттер, будет отправлено в универсальный в этом вызове
}
```

Управление возможностью редактирования значения атрибута в универсальном фильтре.

Если требуется запретить редактировать значение в универсальном фильтре, то можно воспользоваться методом `Btk_FltStdLinkLib.setUniFilterReadOnly`. Часто это используется в связке вместе с блокированием атрибута и на панели фильтрации.

```
val bool = true

selection.filters().std.attrs("flt_someAttrName").isReadOnly = bool
Btk_FltStdLinkLib().setUniFilterReadOnly("flt_someAttrName", bool)
```

Примечание

Не работает для атрибутов, работающих в режиме `saveOnly`. Для таких атрибутов значение всегда не редактируется.

Значение по умолчанию

Значение по умолчанию указывается в теге `uniFilter` в свойстве `defaultValue`. Кроме самого значения может быть указан и его тип.

Тип значения по умолчанию указывается в свойстве `defaultValueType` и может принимать значения:

- **Constant**
Константа. Если тип значения по умолчанию не указан, то он считается константой.
- **Jexl**
Jexl-скрипт.

Константное значение

В свойстве `defaultValue` указывается константное значение. Например:

- Для числовых атрибутов `123`
- Для строковых `abc`
- Для даты `20.12.2023 10:30:42` Если требуется указать множественное значение, то указывается массив значений, разделенных запятой
- Для числовых атрибутов `[123,321]`
- Для строковых `[abc,cde]` Если указывается интервальное значение, то значения указывается массив, содержащий 2 значения. Первое значение - начало интервала, второе значение - конец. Если для одной из границ интервала не задано значение, то оставляется пустая строка:
- `[20.12.2023 10:30:42,20.12.2023 10:30:42]`
- `[,20.12.2023 10:30:42]`

Јех1-скрипт

В свойстве `defaultValue` указывается `jexl`-скрипт, который будет выполнен в контексте выборки для вычисления значения по умолчанию.

Если требуется указать множественное значение, то `jexl`-скрипт должен вернуть массив значений.

Если требуется указать интервальное значение, то `jexl`-скрипт должен вернуть массив, содержащий 2 значения. Первое значение - начало интервала, второе значение - конец.

Передача значений фильтра через параметры выборки

Для передачи значений используется тот же механизм, что и для *передачи значений в универсальный фильтр*

В качестве имен атрибутов используются имена атрибутов стандартного фильтра, которые являются атрибутами значений.

Пример:

```
val fltCondition = FltCondition().withStdLinkGroup{std =>
    std.addAttr("flt_idClass", Btk_ClassApi().findByMnemonicCode("Btk_Class"))
    std.addAttr("flt_bBool", 1.nn)
}

Some_Avi.defList().newForm().params(Map("uniFilterCondition_dz" -> fltCondition)).open()
```

Получение значения атрибута фильтра

Если в прикладной логике для какого-то либо атрибута стандартного фильтра требуется получить его значение, особенно это актуально для ссылочных атрибутов, необходимо воспользоваться библиотечным методом `ru.bitesc.app.btk.flt.Btk_FltStdLinkLib#getAttributeValue`, в который требуется передать имя атрибута стандартного фильтра

Пример использования:

```
val fltCond = Btk_FltStdLinkLib().getAttributeValue("flt_idBisObj")
//проверяем, что условие активно.
if (fltCond.isActive) {
    //проверяем, что выбран множественный тип сравнения
    if (fltCond.compareKind.isMultiValue) {
        //конвертируем к массиву и обходим значения
        fltCond.value.asNLongArray.foreach{v =>
            println(v)
        }
    } else if (fltCond.compareKind.isInterval) {
        //сравнение "В интервале"
        val interval = fltCond.value.value.asInstanceOf[FltLongInterval]
        println(interval.from)
        println(interval.to)
    } else {
        //единичное значение
        println(fltCond.value.asNLong)
    }
}
```

(продолжается на следующей странице)

```
}  
}
```

Формирование макроса

Строковый атрибут

- **Содержит** - накладывает условие по регистронезависимому вхождению строки.

Пример макроса:

```
upper(t.sCaption) like upper('%' || 'value' || '%')
```

- **Начинается с** - накладывает условие на регистронезависимое начало строки.

Пример макроса:

```
upper(t.sCaption) like upper('value' || '%')
```

- **Заканчивается на** - накладывает условие на регистронезависимое окончание строки.

Пример макроса:

```
upper(t.sCaption) like upper('%' || 'value')
```

- **Равно** - накладывает условие на регистронезависимое совпадение строки.

Пример макроса:

```
upper(t.sCaption) = upper('value')
```

- **Не равно** - накладывает условие на регистронезависимое несовпадение строки.

Пример макроса:

```
upper(t.sCaption) <> upper('value')
```

- **В списке** - накладывает условие на регистронезависимое вхождение в список.

Пример макроса:

```
upper(t.sCaption) in (upper('value'))
```

- **Не в списке** - накладывает условие на регистронезависимое отсутствие в списке.

Пример макроса:

```
upper(t.sCaption) not in (upper('value'))
```

- **Заполнено** - накладывает условие на наличие значения.

Пример макроса:

```
t.sCaption is not null
```

- **Не заполнено** - накладывает условие на отсутствие значения.

Пример макроса:

```
t.sCaption is null
```

- **Не содержит** - накладывает условие на регистронезависимое отсутствие подстроки.

Пример макроса:

```
upper(t.sCaption) not like upper('%' || 'value' || '%')
```

- **В интервале** - накладывает условие на вхождение в заданный интервал.

Пример макроса:

```
upper(t.sCaption) >= upper('value') and upper(t.sCaption) <= upper('value')
```

Числовой атрибут

- **Равно** - накладывает условие на точное совпадение числового значения.

Пример макроса:

```
t.nNumber = value
```

- **Не равно** - накладывает условие на несовпадение числового значения.

Пример макроса:

```
t.nNumber <> value
```

- **В списке** - накладывает условие на вхождение числового значения в список.

Пример макроса:

```
t.nNumber in (value)
```

- **Не в списке** - накладывает условие на отсутствие числового значения в списке.

Пример макроса:

```
t.nNumber not in (value)
```

- **Заполнено** - накладывает условие на наличие числового значения.

Пример макроса:

```
t.nNumber is not null
```

- **Не заполнено** - накладывает условие на отсутствие числового значения.

Пример макроса:

```
t.nNumber is null
```

- **В интервале** - накладывает условие на вхождение числового значения в заданный интервал.

Пример макроса:

```
t.nNumber >= value and t.nNumber <= value
```

- **Меньше** - накладывает условие на то, что числовое значение меньше заданного.

Пример макроса:

```
t.nNumber < 1
```

- **Меньше или равно** - накладывает условие на то, что числовое значение меньше или равно заданному.

Пример макроса:

```
t.nNumber <= 1
```

- **Больше** - накладывает условие на то, что числовое значение больше заданного.

Пример макроса:

```
t.nNumber > 1
```

- **Больше или равно** - накладывает условие на то, что числовое значение больше или равно заданному.

Пример макроса:

```
t.nNumber >= 1
```

Внимание

Для типа Long виды сравнения: Меньше, Меньше или равно, Больше, Больше или равно - не доступны!

Булевый атрибут

- **Равно** - накладывает условие на точное совпадение булевого значения.

Пример макроса:

```
and t.bBool = 1
```

- **Не равно** - накладывает условие на несовпадение булевого значения.

Пример макроса:

```
and t.bBool <> 1
```

- **Заполнено** - накладывает условие на наличие числового значения.

Пример макроса:

```
t.bBool is not null
```

- **Не заполнено** - накладывает условие на отсутствие булевого значения.

Пример макроса:


```
t.bBool is null
```

Дата

- **Равно** - накладывает условие на точное совпадение даты.

Пример макроса:

```
t.dDate = to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- **Не равно** - накладывает условие на несовпадение даты.

Пример макроса:

```
t.dDate <> to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- **Заполнено** - накладывает условие на наличие даты.

Пример макроса:

```
t.dDate is not null
```

- **Не заполнено** - накладывает условие на отсутствие даты.

Пример макроса:

```
t.dDate is null
```

- **В интервале** - накладывает условие на вхождение даты в заданный интервал.

Пример макроса:

```
t.dDate >= to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS') and t.dDate <= to_
↳timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- **Меньше** - накладывает условие на то, что дата меньше заданной.

Пример макроса:

```
t.dDate < to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- **Меньше или равно** - накладывает условие на то, что дата меньше или равна заданной.

Пример макроса:

```
t.dDate <= to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- **Больше** - накладывает условие на то, что дата больше заданной.

Пример макроса:

```
t.dDate > to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- **Больше или равно** - накладывает условие на то, что дата больше или равна заданной.

Пример макроса:

```
t.dDate >= to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

Json атрибут

- **Заполнено** - накладывает условие на наличие json значения.

Пример макроса:

```
t.jJson is not null
```

- **Не заполнено** - накладывает условие на отсутствие json значения.

Пример макроса:

```
t.jJson is null
```

Blob атрибут

- **Заполнено** - накладывает условие на наличие числового значения.

Пример макроса:

```
t.lBlob is not null
```

- **Не заполнено** - накладывает условие на отсутствие булевого значения.

Пример макроса:

```
t.lBlob is null
```

Clob атрибут

- По аналогий, как с Строковым атрибутом

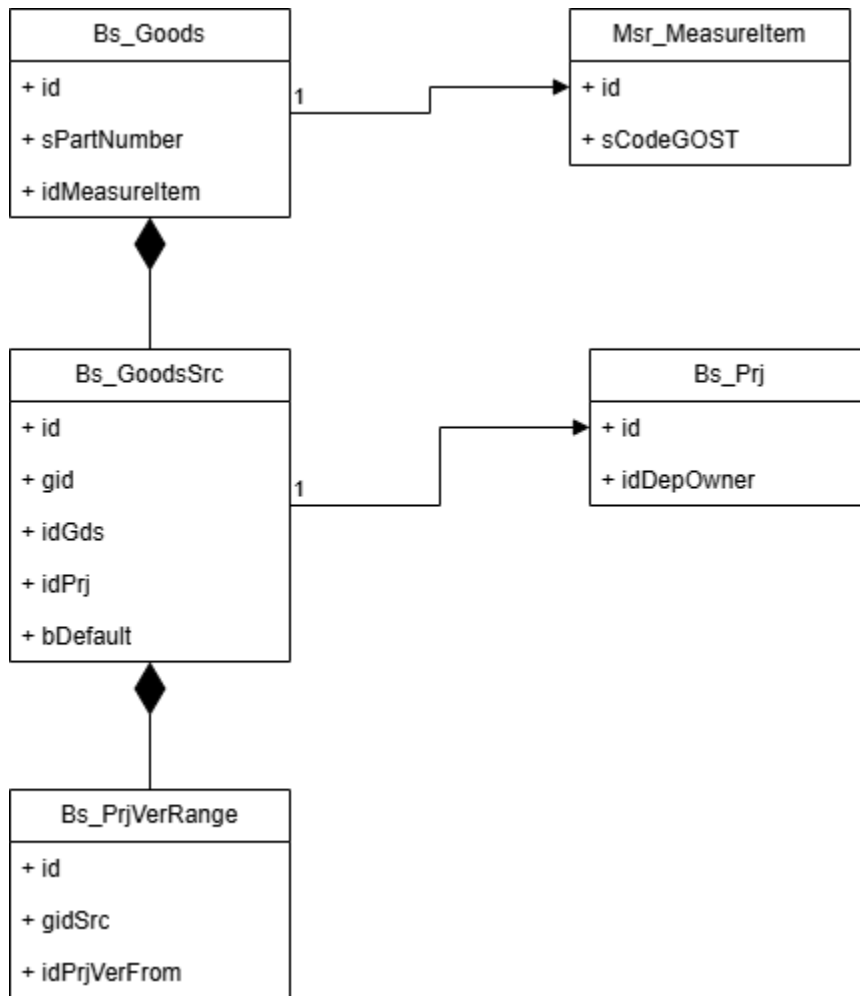
Ссылочный атрибут

- По аналогий, как с Числовым атрибутом

Примеры формируемого макроса

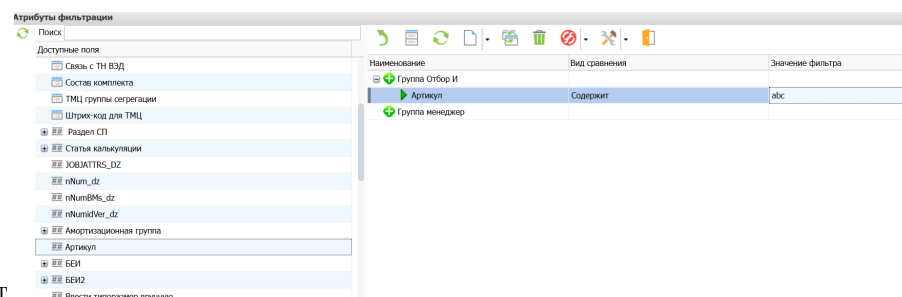
В примере будет показана фильтрация списка объектов класса `Bs_Goods`. В макросах алиас `t` обозначает исходный запрос выборки.

Структура таблиц, показанных в примере



Примеры формируемого макроса

Собственное поле класса Bs_Goods

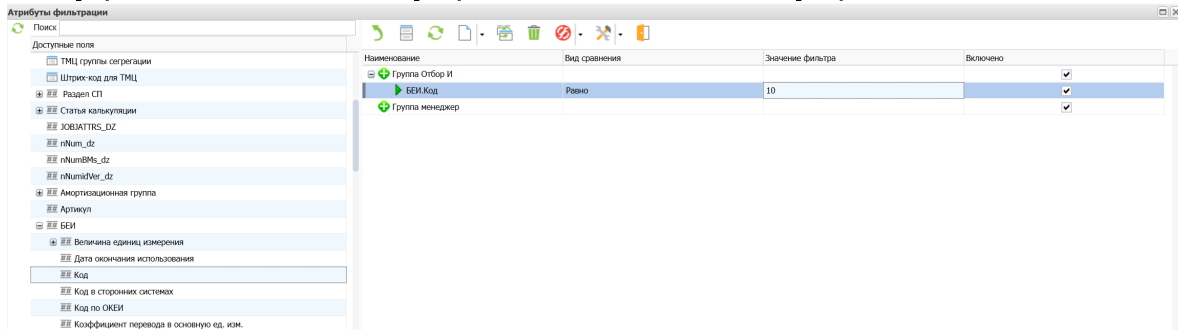


1. Добавили значимый строковый атрибут
2. Полученный макрос:

```
upper(t.sPartNumber) like upper('%' || 'abc' || '%')
```

Ссылочное поле класса Bs_Goods через join

1. Развернули ссылочный атрибут и добавили атрибут ссылочного класса

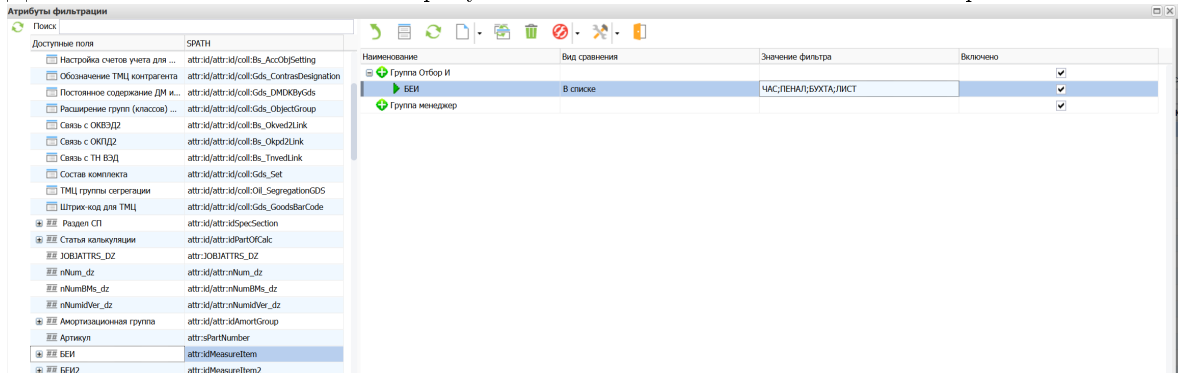


2. Полученный макрос:

```
exists(  
select 1  
  from Msr_MeasureItem t336  
 where t336.id = t.idMeasureItem  
and upper(t336.sCodeGOST) = upper('10')  
)
```

Ссылочное поле класса Bs_Goods с множественным выбором

1. Добавили ссылочный атрибут с множественным выбором значений



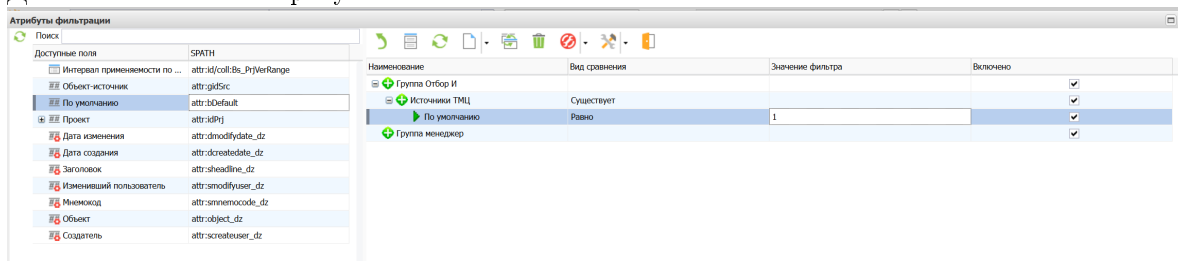
2. Полученный макрос:

```
t.idMeasureItem in (102104,101954,101967,101973)
```

Фильтрация коллекций

Фильтрация коллекции первого уровня вложенности Bs_GoodsSrc по значимому атрибуту

1. Добавили коллекцию Bs_GoodsSrc
2. Добавили значимый атрибут

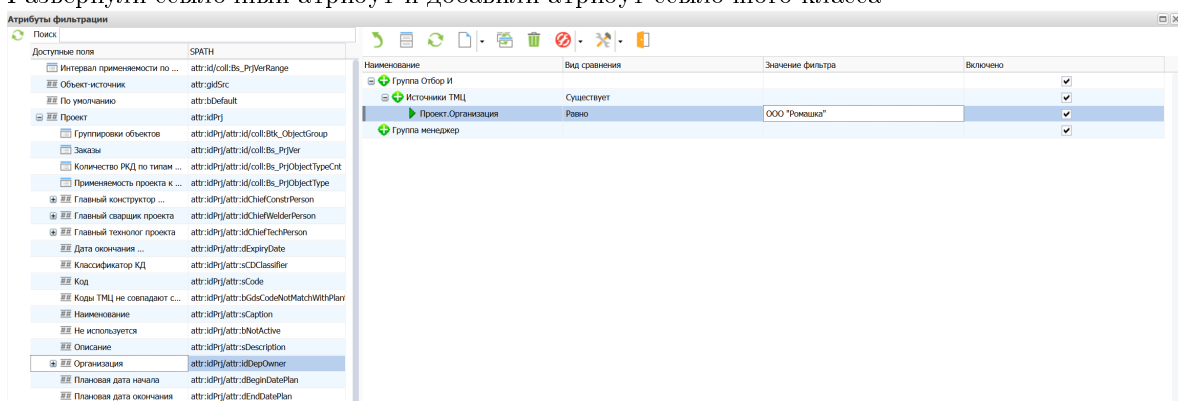


3. Полученный макрос:

```
exists(  
select 1  
  from Bs_Goods t354  
 where t354.id = t.id  
    and exists(  
      select 1  
        from Bs_GoodsSrc t370  
       where t370.idGds = t354.id  
          and t370.bDefault = 1  
    )  
)
```

Фильтрация коллекции первого уровня вложенности Bs_GoodsSrc по ссылочному атрибуту через join

1. Добавили коллекцию Bs_GoodsSrc
2. Развернули ссылочный атрибут и добавили атрибут ссылочного класса



3. Полученный макрос:

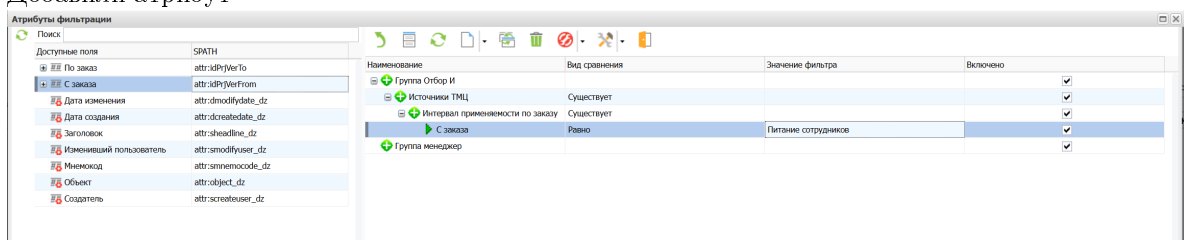
```

exists(
select 1
  from Bs_Goods t354
 where t354.id = t.id
    and exists(
      select 1
        from Bs_GoodsSrc t407
       join Bs_Prj t408 on t408.id = t407.idPrj
      where t407.idGds = t354.id
        and t408.idDepOwner = 117801
    )
)

```

Фильтрация коллекции второго уровня вложенности Bs_PrjVerRange по атрибуту без join

1. Добавили коллекцию Bs_GoodsSrc
2. Добавили коллекцию Bs_PrjVerRange
3. Добавили атрибут



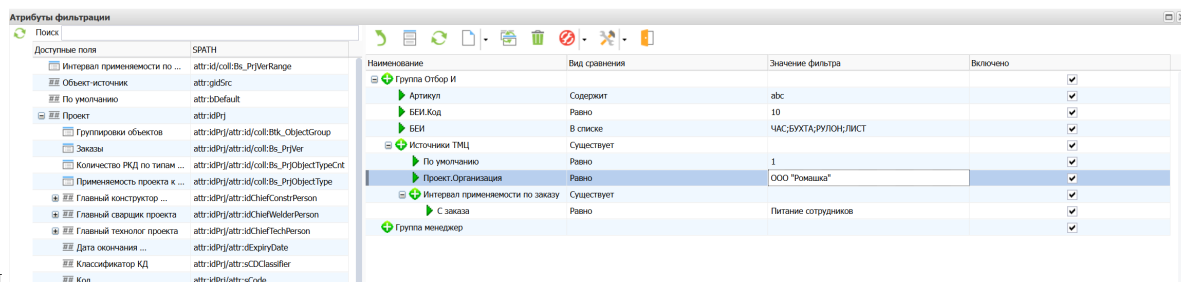
4. Полученный макрос:

```

exists(
select 1
  from Bs_Goods t354
 where t354.id = t.id
    and exists(
      select 1
        from Bs_GoodsSrc t410
       where t410.idGds = t354.id
        and exists(
          select 1
            from Bs_PrjVerRange t423
           where t423.gidSrc = t410.gid
            and t423.idPrjVerFrom = 105501
        )
      )
    )
)

```

Макрос со всеми вышеуказанными условиями



1. Условия

2. Макрос

```
t.idMeasureItem in (102104,101967,101971,101973)
and upper(t.sPartNumber) like upper('%'||'abc'||'%')
and exists(
select 1
  from Bs_Goods t354
 join Msr_MeasureItem t447 on t447.id = t.idMeasureItem
 where t354.id = t.id
   and upper(t447.sCodeGOST) = upper('10')
   and exists(
select 1
  from Bs_GoodsSrc t410
 join Bs_Prj t451 on t451.id = t410.idPrj
 where t410.idGds = t354.id
   and t410.bDefault = 1
   and t451.idDepOwner = 117801
   and exists(
select 1
  from Bs_PrjVerRange t423
 where t423.gidSrc = t410.gid
   and t423.idPrjVerFrom = 105501
   )
   )
  )
)
```

MDA-таблица

Основные положения

MDA-таблица (Multi Dimension Analysis) - средство позволяющее реализовать список, управляемый пользователем с возможностью анализа данных и их группировки.

Основной особенностью MDA-таблиц является работа с данными вне базы данных. При первом запросе данных, будет выполнен запрос к базе данных, далее, если не изменялся порядок сортировки данных, то запросы к базе не выполняются.

Для проектирования MDA-таблицы разработчик определяет запрос данных и описывает колонки этого запроса, часть колонок могут быть измерениями, другие колонки - показатели.

- Измерения - колонки, по которым данные группируются
- Показатели - колонки, к которым применяются функции агрегации

Пользователь имеет возможность управлять сортировкой данных, порядком вывода колонок, способом агрегации значений, выводить под-итоги. Сохранять эти настройки в виде приватных и публичных, назначать настройку по умолчанию.

Построение и отображение данных делится на две части:

- **Модель запроса** - описывает правила получения данных из БД, измерения и показатели
- **Представление данных** - описывает правила сортировки, агрегации и порядок вывода данных на экран.

Модель запроса

Описывает правила получения данных, измерения и показатели.

Класс `ru.bitec.app.btk.sel.mda.model.MdaModel`.

Для построения модели создан класс-builder `ru.bitec.app.btk.sel.mda.model.MdaModelBuilder`

Текст запроса

Текст запроса - строка, в которой содержится текст `sql`-запроса. Этот запрос будет получать исходные данные из базы данных.

Для использования **связанных переменных** используется синтаксис `AnormSql`. Пример связанной переменной:

```
select ...  
  from ...  
where t.id = {id}
```

Для указания значений связанных переменных используется метод `on` класса `ru.bitec.app.btk.sel.mda.model.MdaModelBuilder`

Полный пример использования запроса со **связанными переменными**:

```
val model = MdaModelBuilder()  
  .setQuery( () =>  
    s"""  
      select ...  
        from ...  
      where t.id = {id}  
    """)  
  )  
  .on(() => Seq(  
    "id" -> 100.nl  
  ))  
  .build()
```


Измерения

Определяют поля по которым доступна группировка и сортировка

Класс `ru.bitec.app.btk.sel.mda.model.Dimension`

Значимое измерение

Измерение, представляющее собой единичную колонку запроса

Пример создания:

```
val model = MdaModelBuilder()
    .addDimension(NumberColumn("nNum"), "Число")
    .addDimension(StringColumn("sString"), "Строка")
    .addDimension(DateColumn("dDate"), "Дата")
```

Первым параметром передается объект **колонка**, который описывает тип данных и имя колонки в запросе данных

Ссылочное измерение

Измерение, представляющее из себя 2 колонки запроса, одна из которых содержит `id` объекта, а вторая - `headline`.

Используется, если в качестве измерения используется какой-либо объект с прямой ссылочностью.

В отображаемых данных колонка с `id` будет невидима.

Примечание

Для ссылочных полей важно создавать такие измерения, т.к. они позволят фильтроваться в универсальном фильтре по такому измерению как по ссылочному

Пример создания:

```
val model = MdaModelBuilder()
    .addRefDimension("idRefObject", "idRefObjectH1", "Ссылочный объект", "Some_RefClassName"
    ↪)
```

Параметры в порядке следования:

- имя колонки с идентификатором
- имя колонки с отображаемым значением
- Отображаемое имя
- Имя класса, на который ссылается измерение

Измерение переменной ссылочности

Измерение, представляющее из себя 2 колонки запроса, одна из которых содержит `gid` объекта, а вторая - `headline`.

Используется, если в качестве измерения используется какой-либо объект, на который ссылочность организована через `gid`

В отображаемых данных колонка с `gid` будет невидима.

Примечание

Для ссылочных полей важно создавать такие измерения, т.к. они позволят фильтроваться в универсальном фильтре по такому измерению как по ссылочному

Пример создания:

```
val model = MdaModelBuilder()
    .addAnyRefDimension("gidRefObject", "gidRefObjectH1", "Ссылочный объект", List("Some_
    ↳RefClassName1", "Some_RefClassName2"))
```

Параметры в порядке следования:

- имя колонки с идентификатором
- имя колонки с отображаемым значением
- Отображаемое имя
- Список классов, на который ссылается измерение

Показатели

Определяют поля по которым доступна агрегация значений

Класс `ru.bitec.app.btk.sel.mda.model.Dimension`

Простой показатель

Показатель, представляющий собой единичную колонку запроса

Пример создания:

```
val model = MdaModelBuilder()
    .addMeasure(NumberColumn("nSum"), "Сумма")
```

Первым параметром передается объект **колонка**, который описывает тип данных и имя колонки в запросе данных

Составной показатель

Так же как и **простой показатель** представляет собой единичную колонку запроса, но имеет дополнительное **измерение**, в разрезе которого должна производиться агрегация. Например, сумма в договоре, где дополнительным измерением является валюта.

Пример создания:

```
val model = MdaModelBuilder()
    .addCompositeMeasure(
        NumberColumn("nSum"),
        Dimension(StringColumn("sMsr"), "Валюта"),
        "Сумма"
    )
```

Параметры в порядке следования:

- **колонка**, которая описывает тип данных и имя колонки в запросе данных
- **измерение** - дополнительное измерение, в рамках которого требуется агрегация
- отображаемое имя

Колонки

Модель обладает перечнем колонок, которые были использованы при создании **измерений** и **значений**.

Класс `ru.bitec.app.btk.sel.mda.model.Column`

Колонки определяют:

- тип данных
- имя колонки в запросе данных
- тип редактора

Типы колонок:

- `NumberColumn`
Числовая колонка
- `BooleanColumn`
Булева колонка. Наследник от числовой, с типом редактора **чекбокс**
- `MoneyColumn`
Денежная колонка. Наследник от числовой, с типом редактора **Денежный редактор**
- `LongColumn`
Целочисленная колонка
- `StringColumn`
Строковая колонка
- `DateColumn`
Колонка с типом **дата**, с типом редактора **редактор даты**
- `DateTimeColumn`
Колонка с типом **дата и время**. Наследник от `DateColumn`, с типом редактора **Редактор даты и времени**

Стандартный тип редактора может быть переопределен, используя метод `setEditorType`, который принимает строку, имеющую формат настройки динамического типа редактора.

Пример:

```
NumberColumn("nNum").setEditorType("EditorType=etEdit")
```

Представление данных

Отвечает за настройки отображения данных в MDA-таблице:

- Порядок сортировки
- Вывод промежуточных итогов
- Агрегация значений показателей
- Порядок вывода колонок на экран
- Пользовательское переопределение наименования колонки
- Изменение стилей строк итогов и промежуточных итогов
- Размеры, видимость колонок
- Параметры настройки

Класс `ru.bitec.app.btk.sel.mda.view.MdaView`.

Представлением управляет пользователь, используя пользовательский интерфейс настройки вывода колонок, а так же сохраняя и загружая настройки.

Измерения

Определяет настройки сортировки, группировки и вывода промежуточных итогов.

Показатели

Определяет показатели, по которым будет выполнена агрегация

Атрибуты

Определяет порядок вывода колонок на экран и их наименование

Описание формируемых атрибутов выборки

На каждый подобранный атрибут представления в выборке формируются следующие атрибуты:

- `idValue[<имя колонки>]` - невидимый атрибут, будет содержать `id` для ссылочных измерений или `gid` для измерений переменной ссылочности.
- `value[<имя колонки>]` - видимый атрибут, в котором будет отображаться значение, видимое пользователю. Для ссылочных измерений или измерений переменной ссылочности в этот атрибут выводится значение колонки, объявленной как `headline`
- `sEditor[<имя колонки>]` - невидимый атрибут, управляющий типом редактора

- `sStyle[<имя колонки>]` - невидимый атрибут, управляющий стилем раскраски ячейки.

Подключение MDA-таблицы к прикладной выборке

Реализация отображения сгруппированных данных

Для подключения к прикладной выборке сервиса MDA-таблиц необходимо:

1. Унаследовать выборку от `ru.bitec.app.btk.sel.mda.Btk_MdaAbsAvi`
2. Создать отображение, если требуется
3. Унаследовать нужное отображение от отображения `Grid`
4. Определить модель запроса данных, переопределив метод `buildModel`, используя строителя модели `ru.bitec.app.btk.sel.mda.model.MdaModelBuilder`
5. Переопределить метод `getSourceDataRep`, который будет определять выборку для открытия исходных данных.

Реализация отображения исходных данных

Для реализации выборки, которая будет отображать **исходные данные** (данные которые вернул запрос из базы данных), необходимо:

1. Унаследовать выборку от `ru.bitec.app.btk.sel.mda.Btk_MdaAbsAvi`
2. Создать отображение, если требуется
3. Унаследовать нужное отображение от отображения `SourceDataGrid`

Управление типом редактора

Тип редактора для колонки определяется в момент описания модели запроса данных. Каждый тип колонки имеет свой тип редактора по умолчанию, для его переопределения можно воспользоваться методом `setEditorType`

Интеграция с универсальным фильтром

Универсальный фильтр анализирует модель запроса данных и на все показатели и измерения позволяет наложить условия фильтрации. Условие фильтрации будет наложено поверх результат запроса, описанного в модели.

Правила формирования доступных для фильтрации атрибутов:

- для **ссылочных измерений** создается ссылочный атрибут фильтрации
- для **измерений переменной ссылочности** создается атрибут фильтрации переменной ссылочности
- для всех остальных **измерений и показателей** создаются значимые атрибуты фильтрации в соответствии с типом данных

Создание стандартного фильтра

Стандартный фильтр создается по тем же правилам, что и для обычной выборки.

Использование макросов

Для использования макроса выборки в тексте запроса необходимо получить его текст, используя метод `selection.getMacro`, и добавить его в нужное место запроса.

Пример:

```
val model = MdaModelBuilder()
    .setQuery(() =>
        s"""
            select ...
            from ...
            where ${selection.getMacro("SomeMacrosName")}
        """
    )
```

Описание основных методов

- `extendDynMeta` - позволяет добавить в перечень атрибутов выборки свои атрибуты.
- `extendDynRow` - позволяет установить дополнительные значения в формируемую строку выборки
- `buildModel` - метод, который должен переопределить разработчик, для описания модели данных
- `mdaModel` - возвращает модель данных
- `mdaView` - возвращает представление данных
- `needGenChooseCheckBox` - признак, что требуется формировать чекбоксы для выбора строк
- `forSelectedRows` - позволяет обойти выбранные через чекбоксы строки.
- `getSourceDataRep` - метод, который должен переопределить разработчик, для указания выборки, которая будет открыта из операции Открыть исходные данные
- `filterRow` - позволяет отфильтровать записи
- `refreshBySql` - метод, который принудительно делает перезапрос данных из базы данных, и обновляет выборку

Описание основных операций

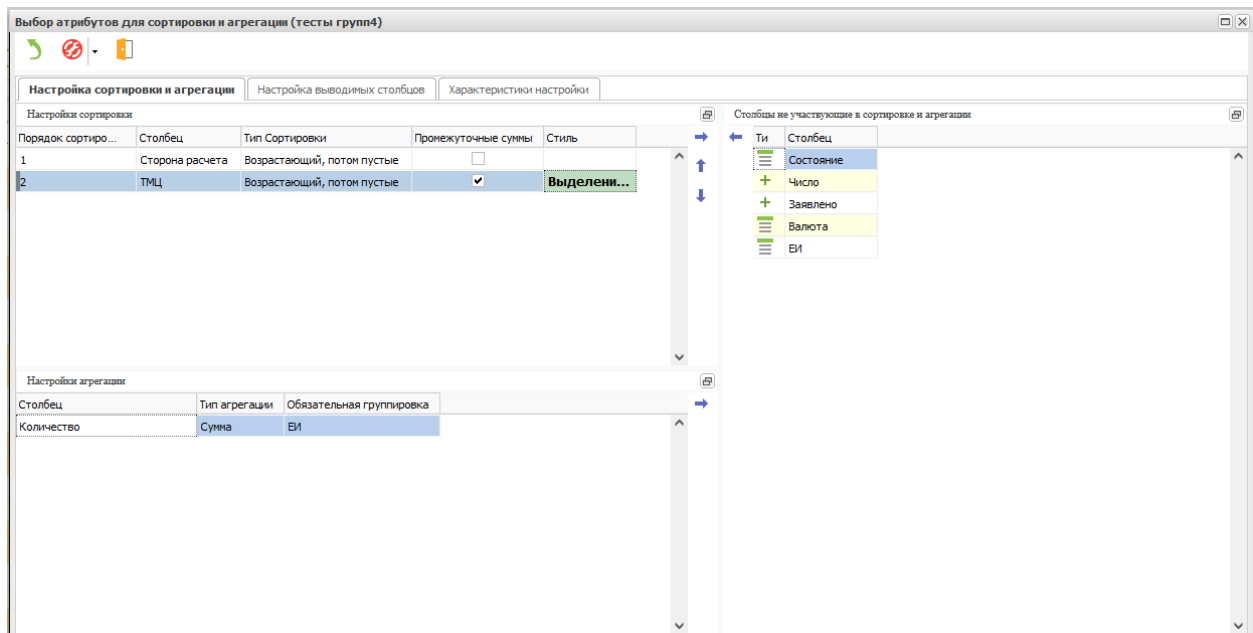
- `openSourceData` - Открыть исходные данные
Открывает выборку, указанную в методе `getSourceDataRep`, в которой будут отображены исходные данные для выбранных строк
- `chooseAll` - Выбрать все записи
Для всех записей проставляет чекбокс выбора
- `clearAllChosen` - Снять выбор со всех записей
Для всех записей снимет чекбокс выбора

- **chooseSelected** Выбрать выделенные записи
Для всех выделенных записей проставляет чекбокс выбора
- **clearSelected** Снять выбор с выделенных записей
Для всех выделенных записей снимет чекбокс выбора

Описание настройки атрибутов

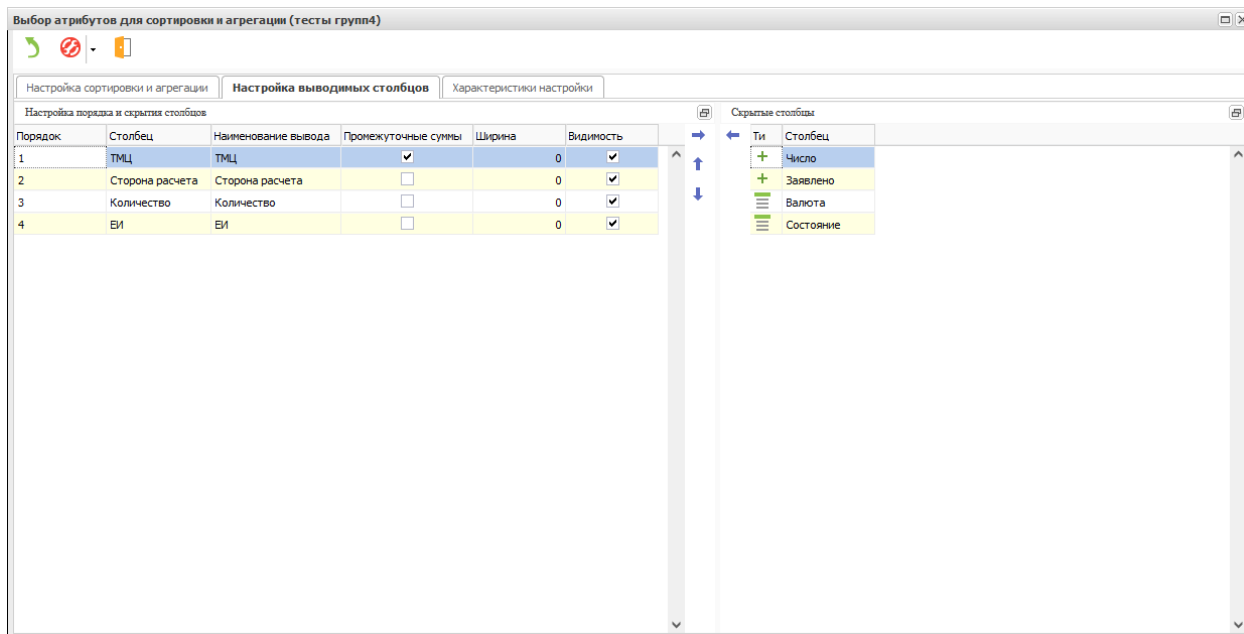
Настройка происходит через выборку `ru.bitec.app.btk.sel.mda.Btk_MdaViewEditorAvi#attrEditor`. Выборка ожидает в качестве параметра `mdaModel` в формате JSON и опционально прошлый `mdaView` для формирования нового `mdaView`. Возвращает JSON отредактированного `mdaView`.

Для работы с созданием/сохранением/загрузки настроек выборка ожидает названия выборки и репрезентации и опционально `id` прошлой настройки.



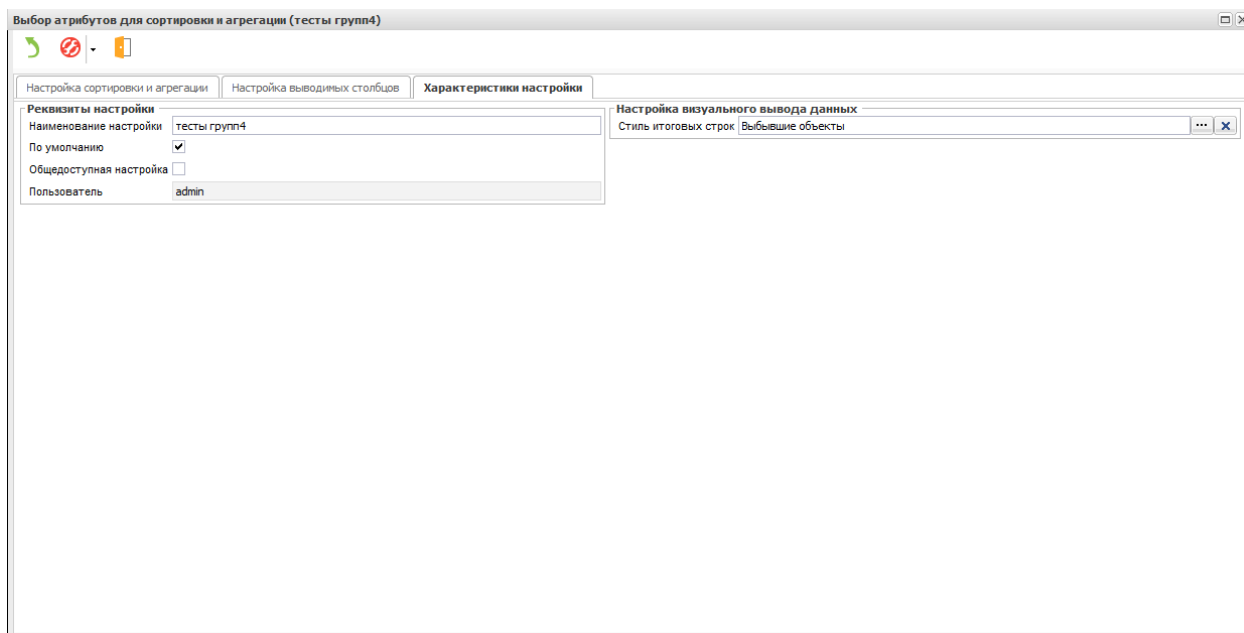
На вкладке **Настройка сортировки и агрегации** можно настроить

- Порядок и тип сортировки
- Вывод промежуточных итогов
- Тип агрегация значений показателей
- Стили промежуточных итогов



На вкладке **Настройка выводимых столбцов** можно настроить

- Порядок вывода колонок на экран
- Пользовательское переопределение наименования колонки
- Размеры, видимость колонок



На вкладке **Характеристики настройки** можно настроить

- Изменение параметров настройки
- Стили итогов

Аудит открытия форм и выполнения операций

Позволяет вести запись расширенного аудита действий пользователя при открытии форм и выполнении операций в выборке. Запись производится в автономной транзакции. Если выборка, на которой ведется аудит, будет открыта как детальный фрейм, то в аудит будут записаны все ее мастер-выборки.

Подключение настроек аудита

- В приложении **Настройка системы** перейти в **Обозреватель проектов**
- Для выбранной выборки перейдите на закладку **Настройка**

Для настройки аудита (класс `Btk_SelSetting`) можно выбрать следующие типы ведения аудита

- Не вести аудит (по умолчанию)
- Аудит только успешного открытия выборки/выполнения операции
- Аудит с сохранением ошибок

Отчет аудита выборки

- Просмотр аудита от определенного объекта. В карточке объекта под операцией **Информация**, выбрать **Расширенный аудит по объекту**.
- Аудит по всем выборкам. В приложении **Настройка системы** - **Аудит** - **Аудит открытия форм и выполнения операций**.

При включенном чек-боксе **Только главные выборки** на панели фильтров в верхнем фрейме будут выведены только главные выборки, но при этом записи от детальных выборок можно всегда посмотреть на закладке **Дерево элементов формы**.

Мастер пользовательского ввода

Мастер пользовательского ввода представляет собой базовый механизм создания или изменения/настройки объектов в системе как для пользователя системы, так и для прикладного разработчика.

Включает в себя

- *Интерфейс настройки мастеров и их шагов*
- *Интерфейс запущенного мастера*
- *Scala API*

Позволяет прикладным разработчикам иметь доступ к параметрам мастера

- *Jexl API*

Позволяет получить доступ к параметрам мастера и контексту выборок через jexl скрипты, закреплённые за каждым шагом, а также управлять переходами между ними.

Интерфейс настройки мастеров

Открыть список мастеров можно через:

Настройка системы \ Настройки и сервисы > Мастеры пользовательского ввода

Отображать неиспользуемые ☐

Код	Наименование	Описание	Модуль	Не используе...
111	Тест переходов 1. Изменение поряд...		Базовые справочники	☐
Pm_AdvRepCurWizard	Создание валютного авансового отч...		Управление денежными средствами	☐
222	Тест переходов 2. Досрочный выход		Основной системный модуль	☐
333	Мастер выбора документа		Основной системный модуль	☐

Шаг мастера

N	Наименование	Описание	Выборка	Отображение	Начальный шаг
10	Шаг 1		gtk-Bs_PersonAvi	List_WithUsers	☒
20	Шаг 2		gtk-Bs_EmployeeAvi	List	☐
30	Шаг 3		gtk-Pro_ProjectActionT...	Tree	☐

[Jexl на вход в шаг](#) [Jexl вперёд](#) [Jexl назад](#) [Jexl выход](#)

```
1 ctx.setParam('name', 'mAX');
2 ctx.getParam('name');
```

Примечание

При добавлении нового шага нумерация происходит автоматически по порядку, начиная с 10-ти с шагом 10.

Также первый добавленный шаг будет помечен как «Начальный шаг».

Интерфейс мастера

Представляет собой модальное окно, в верхней части которого расположены операции мастера, а в нижней отображается выборка, настроенная на данном шаге. Первым отображается тот шаг, у которого активен признак «Начальный шаг».

Операции мастера:

- «Предыдущий шаг» (зелёная стрелка влево)
Осуществляет выполнение *скрипта на переход назад* и переключение на предыдущий шаг.
- «Следующий шаг» (зелёная стрелка вправо)
Осуществляет выполнение *скрипта на переход вперёд*, вызов операции **afterEdit** у текущего шага и переключение на следующий шаг.
- «Выход»
Осуществляет выполнение *скрипта на досрочный выход* и закрытие мастера.

Scala API

Scala API реализуется через библиотечный класс `Btk_WizardLib` и позволяет получать/устанавливать значения параметров мастера.

Внимание

`Btk_WizardLib` работает в контексте основной выборки мастера - `Btk_WizardAvi#Card_WizardHeader`.

Получение значений

```
Btk_WizardLib().getParam(name: String): Any
```

Возвращает значение параметра `name`.

Установка значений

```
Btk_WizardLib().setParam(name: String, value: Any)
```

Устанавливает параметру `name` значение `value`.

Пример

Запишем в параметры мастера значение поля `sCode` выбранной строки.

```
val svCode = getSelfVar("sCode").asNString  
  
Btk_WizardLib().setParam("sCode", svCode)
```

Примечание

Так же для работы с параметрами существуют методы `getJParams(): JObject` и `setJParams(params: JObject)`. Они позволяют соответственно получить и установить значение json-а с параметрами мастера целиком

Jexl API

Jexl API реализует интерфейс *работы со значениями параметров мастера и данных в контексте сборки* через jexl-скрипты, настраиваемые на каждом шаге. Так же этот интерфейс позволяет *управлять переходами между шагами* мастера.

Доступные для настройки скрипты:

- Скрипт на вход в шаг (*before*)
- Скрипт на переход вперёд (*forward*)
- Скрипт на переход назад (*back*)
- Скрипт на досрочный выход из шага (*exit*)

Работа с параметрами

Через скрипты также можно работать с параметрами мастера - получать их и устанавливать.

Примечание

Работа с параметрами происходит через jexl-контекст `ctx`. Также скрипт получает значение `formResult`, которое равняется:

- "Ok" - при выходе из мастера с сохранением изменений
- "Cancel" - при выходе из мастера без сохранения изменений
- "None" - при переходе между шагами мастера

За установку значений отвечает метод `setParam(String, Any)`:

```
ctx.setParam('some_int_param', 1024);
```

За получение значений отвечает метод `getParam(String) -> Any`:

```
ctx.getParam('some_int_param'); // вернёт 1024
```

Примечание

Так же для работы с параметрами существуют методы `getJParams(): JObject` и `setJParams(params: JObject)`. Их функционал аналогичен одноимённым библиотечным методам.

Работа с выборкой

Данные скрипты выполняются в контексте выборки шага, поэтому нам доступны такие методы как `setVar`, `getVar` и т.д.

Примечание

Вызов методов, доступных для работы с данными выборки шага, осуществляется без префикса `ctx`.

Внимание

Обратите внимание, что скрипт на **вход в шаг** работает ТОЛЬКО В КОНТЕКСТЕ МАСТЕРА! Соответственно, доступа к данным выборки шага у него нет.

Переопределение перехода между шагами

По умолчанию переход между шагами идёт в порядке, определённом значением номера шага - от меньшего к большему.

Примечание

Переопределение шагов происходит через jexl-контекст `ctx`.

Установить значение шага на переход вперёд можно методом `setStepForward(String)`:

```
ctx.setStepForward('20');
```

Установить значение шага на переход назад можно методом `setStepBack(String)`:

```
ctx.setStepBack(null);
```

Так же можно получить значения шагов на данном этапе:

```
// значение текущего шага
ctx.getStep();

// значение предыдущего шага
ctx.getPrevStep();

// значение шага на переход назад
ctx.getStepBack();

// значение шага на переход вперёд
ctx.getStepForward();
```

Внимание

Методы для работы с шагами для удобства принимают/возвращают строковое значение номера шага.

Скрипт на вход в шаг

Запускается перед запуском каждого шага.

Примечание

Если на данном этапе переопределить значение следующего шага на `null`, то стрелка перехода вперёд будет недоступна. Аналогично можно поступить и с шагом назад.

Скрипт на переход вперёд

Запускается при выполнении операции «Далее».

Примечание

Если на данном этапе переопределить значение следующего шага на `null`, то будет вызвана логика на досрочный выход из шага. Так же будет осуществлена попытка вызвать метод `afterEdit()` у детальной выборки.

Скрипт на переход назад

Запускается при выполнении операции «Назад».

Примечание

Если на данном этапе переопределить значение шага на переход назад на `null`, то будет вызвана логика на досрочный выход из шага.

Скрипт на досрочный выход из шага

Запускается на операцию «Выход» и следует перед к закрытием формы.

Запуск мастера

Запуск мастера осуществляется на операцию «Запустить мастер» в интерфейсе настройки мастеров.

Так же мастер можно запустить через библиотечный метод `Btk_WizardLib().launch(idpWizard: NLong, firstStepParams: Map[String, Any])`

- `idpWizard` - Идентификатор мастера
- `firstStepParams` - Карта параметров для первого шага мастера. Эти параметры будут переданы выборке первого шага мастера при запуске мастера

Примечание

Вместо `Map[String, Any]` можно также передать `JObject`.

Методы регистрации мастера и его шагов

Регистрация мастера

```
Btk_WizardApi().register(  
    spCode: NSString,  
    spCaption: NSString,  
    spModule: NSString,  
    spDescription: NSString,  
    dpExpireDate: NSDate  
): ApiRop
```

- `spCode` - уникальный код мастера
- `spCaption` - наименование
- `spModule` - код системного модуля
- `spDescription` - описание (не обязательный атрибут)
- `dpExpireDate` - дата окончания использования (не обязательный атрибут)

Возвращает `Btk_WizardApi#ApiRop`.

Примечание

Вместо кода системного модуля можно также передать его идентификатор.

Регистрация шага мастера

```
Btk_WizardStepApi().register(  
    ropParent: Btk_WizardApi#ApiRop,  
    spCaption: NSString,  
    spSelection: NSString,  
    spRepresentation: NSString,  
    npOrder: NNumber = None.nn,  
    bpStart: NNumber = None.nn,  
    spDescription: NSString = None.ns,  
    spScriptBefore: NSString = None.ns,  
    spScriptForward: NSString = None.ns,  
    spScriptBack: NSString = None.ns,  
    spScriptExit: NSString = None.ns,  
): ApiRop
```

- `ropParent` - роп мастера
- `spCaption` - наименование
- `spSelection` - имя выборки
- `spRepresentation` - имя отображения
- `npOrder` - порядковый номер
Если передано пустое значение, присвоится автоматически

- `bpStart` - флаг, является ли шаг начальным
Если передано пустое значение, для первого шага значение будет 1, для последующих - 0
- `spDescription` - описание
- `spScriptBefore` - скрипт на вход в шаг
- `spScriptForward` - скрипт на переход вперёд
- `spScriptBack` - скрипт на переход назад
- `spScriptExit` - скрипт на досрочный выход

Возвращает `Btk_WizardStepApi#ApiRop`.

Внимание

Если переданный `npOrder` уже существует для данного `ropParent`, то будет изменена существующая запись, а не создана новая.

Пример регистрации

```
def regWizard(): Unit = {
  // регистрируем мастер
  val rvMaster = Btk_WizardApi().register(
    spCode = "createSomeObject",
    spCaption = "Создание некоего объекта",
    spModule = "bs",
    spDescription = None.ns,
    dpExpireDate = None.nd
  )

  // регистрируем шаги
  Btk_WizardStepApi().register(
    ropParent = rvMaster,
    spCaption = "Шаг 1",
    spSelection = "gtk-Btk_UserAvi",
    spRepresentation = "List",
    spScriptExit = "dialogs.showMessage('Досрочный выход из шага 1');"
  )

  // регистрируем шаги
  Btk_WizardStepApi().register(
    ropParent = rvMaster,
    spCaption = "Шаг 2",
    spSelection = "gtk-ru.bitec.app.wf.Wf_DirAvi",
    spRepresentation = "Tree",
    spScriptBefore = "ctx.setStepBack(null);",
    spScriptExit = "dialogs.showMessage('Конец');"
  )
}
```

13.4 Инструменты

Конфигуратор

Системное приложение «Конфигуратор» предназначено для автоматизации и ускорения выполнения рутинных задач по разработке проекта:

- Создание нового проекта
- Создание нового модуля
- Работа с классами
 - Создание
 - Редактирование
- Работа с выборками
 - Создание
 - Редактирование
- Управление разметкой выборок
 - Редактор карточки
 - Редактор фреймов
- Управление подсистемой распространения изменений
- Управление технической документацией сущностей и методов проекта

Конфигуратор предназначен для работы на локальном компьютере разработчика.

Конфигуратор взаимодействует с IDE IntelliJ Idea через специальный плагин.

Конфигуратор поддерживает систему контроля версий SVN.

Запуск

Для запуска конфигуратора выполните следующие действия:

1. Запустите сервер Global
Выполните скрипт `C:\Global3se\start.bat` или запустите отладку в IDE IntelliJ Idea
2. Откройте в браузере адрес `http://localhost:8080/`
3. Введите логин, пароль, базу данных
4. Разверните настройки подключения
5. Установите флажок «Конфигуратор»
6. Войдите в систему

Первичная настройка

При первом запуске конфигуратор предлагает выбрать режим работы по умолчанию: локально или через систему контроля версий. При работе через систему контроля версий необходимо далее указать:

- Логин svn
- Пароль svn
- Url svn

Настройки сохраняются в поддиректорий конфигурации сервера Global: C:\Global3se\application\config\cfg

- vcs.dat – Настройка SVN
- project.dat – Настройка проектов

Если удалить эти файлы, мастер настройки автоматически запустится заново.

Создание нового проекта

Запустить мастер создания проектов: **Проекты > Новый проект**

Общие настройки

- *Системное имя проекта*
Внутреннее имя проекта, обычно совпадает с наименованием директории проекта (пример: intern199)
- *Локальный путь до проекта*
Корневая директория, в которой будет создан проект пример: C:\SVN\ASSource\database\pgIntern\...

Настройки для локального режима

В локальном режиме работы требуется указать

- *параметры соединения с БД*
- *шаблон проекта *
 - или путь к архиву с шаблоном проекта
 - или каталог с существующим проектом

Настройки для работы через svn

Для режима работы через систему контроля версий предусмотрена расширенная конфигурация при которой надо задать параметры:

- *Путь в репозитории* Каталог в SVN, в который будет сохранен проект
пример: ASSource/database/pgIntern/...

Настройки Sbt-plugin

Sbt-plugin является расширением к системе сборки sbt (scala build tool). При настройке рекомендуется не использовать пункт «trunk-версия», а оставить предложенную версию, которая является максимальной на момент создания проекта.

Настройки Базы данных

- *Url*
Указывается тип драйвера и параметры подключения
пример: `jdbc:postgresql://v39:5432/intern199`
- *Пользователь*
Имя пользователя для подключения к БД
- *Пароль* Пароль для подключения к БД

Примечание

Параметры подключения к БД можно узнать у куратора проекта или системного администратора

Подключаемые модули

В списке подключаемых модулей, будут подобраны основные модули, для создания проекта. Их можно удалить или добавить новые. После выполнения операции «Подобрать зависимости».

Завершение создание проекта.

1. По завершению конфигурации выполните операцию «Создать проект»
При этом:
 - В случае работы с svn, в репозитории системы контроля версий будет создана структура каталогов проекта (каталог хранения проекта приложения, каталог хранения проектных модулей)
 - Отобразится диалог конфигурации сервера для запуска проекта
2. Завершите конфигурацию сервера
3. Импортируйте проект в IntelliJ Idea

Модули

Интерфейс для работы с модулями проекта расположен на панели инструментов главного меню. Позволяет выполнять настройку модулей, подключать существующие модули, создавать новые.

Создание модуля

Операция позволяет создать новый модуль и подключить его к проекту.

1. На панели инструментов выполните операцию Подключить модули > Создать новый модуль
2. В открывшемся мастере введите:
 1. *Системное имя*
Имя модуля в нижнем регистре
 2. *Наименование модуля*
 3. *Проектный модуль* (только для режима SVN)
Если установить этот признак, то модуль будет создан внутри каталога проекта в подкаталоге module, и не будет доступен к подключению в других проектах. Если признак не установить, то модуль будет создан в основном каталоге модулей (ASSource/module) и будет доступен к подключению в других проектах.
3. Подключите модули, от которых будет зависеть создаваемый модуль

Обозреватель проекта

Обозреватель проекта — это отдельное окно в интерфейсе системы, которое отображает иерархическое дерево сущностей проекта. Через него можно:

- Открыть любую выборку.
- Просмотреть состав выборки (сущности, операции и другие элементы).
- Редактировать *свойства операций*.

Сущности в обозревателе проекта:

- классы;
- выборки;
- прикладная бизнес-логика (Api);
- пакеты прикладной бизнес-логики (Pkg).

Для открытия необходимо перейти в Сущности - Обозреватель проекта.

Дерево сущностей

В дереве отображаются сущности, сгруппированные по каталогам. Для классов отображаются в качестве потомков их базовая выборка и Api

Дерево отображается в двух режимах, которые переключаются флагами на панели фильтров:

- *Логическая структура* \ Каталоги для сущностей определяются так: если в метаданных класса, выборки или пакета указан тег `logicFolder`, то каталогом этой сущности будет указанное значение, иначе физический каталог. Пример значения: `logicFolder = "audit.test"`
- *Физическая структура*
Каталоги для сущностей определяются их расположением внутри проекта. Например для scala-пакета `ru.bitec.app.btk.audit.test` каталогом будет `audit.test`

Основная закладка для всех сущностей — это «Описание», которое формируется на основе документации, которая хранится в исходном коде. Подробнее см. главу «Документирование»

Создание новых сущностей

Реализовано создание основных сущностей:

- *Класс*
Создает odm-файл
- *Выборка*
Создает avi и avm файлы
- *Пакет*
Создает Pkg и pkg.xml файлы
- *Описание каталога* Создает package-info.java в указанном каталоге.

Тегирование

Позволяет для сущностей повесить разнообразные теги, которые будут отображены в дереве обозревателя.

Настройка доступных тегов осуществляется в главном меню **Настройка > Теги обозревателя проекта**

Для установки тегов сущности используется операция **Дополнительно > Редактировать теги**

Документирование

Для сущностей на закладке «Описание» собирается их документация, хранящаяся в исходном коде.

Документация пишется в формате markdown.

Формирование описания по типам сущностей:

Класс

Документация для класса хранится в odm файле. В теге `documentation`, который доступен для:

- класса
- Атрибута

Выборка

Документация для выборки хранится в Avm файле. В тег `documentation`, который доступен для:

- Выборки
- Отображения
- Атрибута

Api

Документация хранится в scala doc в файле Api-класса, [ссылка на документацию ScalaDoc](#)

Pkg

Документация хранится в:

- теге `documentation` файла `pkg.xml`
- scala doc Pkg-класса, [ссылка на документацию ScalaDoc](#)

Каталог

Документация хранится в `package-info.java`, расположенном в каталоге на диске в scala-ветке. Документация в нем так же пишется в markdown.

Модуль

Документация хранится в `package-info.java`, расположенном в корневом каталоге scala-ветки

Интеграция с IntelliJ IDEA

Интеграция осуществляется через плагин для IDEA

Позволяет открывать файлы в IDEA, которые соответствуют записям в дереве обозревателя. Для некоторых сущностей есть возможность открывать дополнительные файлы. Если не открыт текущий проект в IDEA, то будет выдано сообщение, что проект не найден.

При открытии проекта в IDEA запускается json-грс сервер, который получает порт для прослушивания и сохраняет в структуре проекта файл `.idea/ideasocket/<номер порта>.lock`, при этом блокируя его. Конфигуратор при отправлении команды на открытие файла сканирует эту директорию на наличие заблокированных файлов и получает порт, в который требуется отправлять данные.

Для открытия файла в IDE используется операция «Открыть в IDE»

Открываемые файлы по типам сущностей:

- *Класс*
Открывает odm-файл
- *Выборка*
Открывает Avi-файл.
Дополнительная подоперация позволяет открыть avm
- *Api*
Открывает Api-файл
- *Pkg*
Открывает Pkg.scala
Дополнительная подоперация позволяет открыть pkg.xml

Разработка конфигуратора

Код конфигуратора содержится в отдельном модуле. Для разработки этого модуля необходимо его настроить в режим разработки.

Конфигурация модуля происходит в файле `global3.config.xml` через тэги:

- **database**
Конфигурация базы данных с именем «{DB_ALIAS}.CFG», где DB_ALIAS - имя базы, к которой идет подключение.
- **sbt**
Настройка модуля, содержащего исходный код конфигуратора.

Конфигурация базы данных

Если конфигурация отсутствует, она создаётся автоматически по алгоритму:

1. Скопировать конфигурацию «{DB_ALIAS}» в «{DB_ALIAS}.CFG»
2. Заменить имя sbt на значения `cfg`

```
<metaManager sbtName="cfg"/>
```

Конфигурация SBT

Если конфигурация sbt отсутствует, формируется конфигурация по умолчанию.

Шаблон конфигурации:

```
<sbt name="cfg"
  jarFolder="%GLOBAL3_HOME%\server\configurator\applib\\"
  binaryFolder="%GLOBAL3_HOME%\server\configurator\appbin\\"
  sourceMode="Jar"
  lazyLoad="true">
</sbt>
```

где:

- **GLOBAL3_HOME**
Переменная окружения, указывающая на расположения дистрибутива. Данная переменная задается автоматически при старте сервера

Конфигурация sbt для разработчика

Для разработки необходимо переопределить конфигурацию sbt по шаблону:

```
<sbt name="cfg"
  source="{PATH_TO_CFG}\configurator\application\"
  sourceMode="Dev"
  lazyLoad="true">
</sbt>
```


Алгоритм формирования для режима системной разработки

Для перехода в режим разработки:

- Задайте переменную окружения G3_ISDEV = 1
- Задайте переменную окружения G3_SHARE = «путь к jar архивам конфигулятора»

Шаблон генерации конфигурации в режиме системной разработки:

```
<sbt name="cfg"
  jarFolder="%G3_SHARE%\cfgapplib"
  binaryFolder="%G3_SHARE%\cfgappbin"
  sourceMode="Jar"
  lazyLoad="true">
</sbt>
```

Работа в IntelliJ IDEA

Подключение XSD-схем к редактору кода

Для проверки синтаксиса и работы подсказчика при редактировании xml-файлов (avm.xml, odm.xml, global3.config.xml), необходимо подключить к редактору xsd-схемы. Схемы хранятся в библиотеке C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar

xmlns	xsd -схема
http://www.global-system.ru/xsd/global3-module-1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3-module-1.0.xsd
http://www.global-system.ru/xsd/global3-view-1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3-view-1.0.xsd
http://www.global-system.ru/xsd/global3-view-template-1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3-view-template-1.0.xsd
http://www.global-system.ru/xsd/global3.config.1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3.config.1.0.xsd

Подключить схемы можно в настройках IDE IntelliJ Idea:

File > Settings > Languages & Frameworks > Schemas and DTDs

Создание сущностей без конфигулятора

Допускается создание сущностей фреймворка в ручном режиме прямо в IDE IntelliJ Idea

Создание Odm файла класса

Создание класса начинается с создания odm файла (объектно-документного представления сущности):

`application/{модуль}/src/main/resources/ru/bitec/app/{модуль}/{имя_сущности}.odm.xml`

Для создания можно воспользоваться файловым шаблоном из контекстного меню: **New > odm AllTypes**

В качестве примера, откройте файл: `../bs/src/main/resources/ru/bitec/app/bs/Bs_Contras.odm.xml`

Использование генератора кода

Для формирования базового кода для работы с сущностями предназначена утилита `ru.bitec.app.gtk.meta.SourceGenerator`. Генератор подключен к IDE IntelliJ Idea как внешняя утилита (External Tool). Доступ к генератору кода осуществляется через контекстное меню в обозревателе проекта или заголовка закладки: **External Tools > Generate sources**

Примечание

Для настройки утилиты см. главу «Начало работы».

Запуск утилиты возможен от любого файла, относящегося к сущности:

- `*.orm.xml`
- `*.odm.xml`
- `*.avm.xml`
- `*Avi.scala`
- `*Dvi.scala`
- `*Api.scala`
- `*Dpi.scala`

При работе утилиты создаются следующие файлы:

- В каталоге `../main/java/ru/bitec/app/{модуль}/`
 - `{имя_сущности}.java`
Роjo-класс сущности
- В каталоге `../main/scala/ru/bitec/app/{модуль}/`
 - `{имя_сущности}Dpi.scala`
Не изменяемая бизнес логика контроллера
 - `{имя_сущности}Api.scala` Изменяемая бизнес логика контроллера

Внимание

Создается только если нет Api файла

- `{имя_сущности}Dvi.scala`
Не изменяемая бизнес логика представления (View). Шаблон

- {имя_сущности}Avi.scala
Изменяемая бизнес логика представления (View).

Внимание

Создается только если нет Avi файла

- В каталоге ../main/resources/ru/bitec/app/{модуль}/
 - {имя_сущности}.orm.xml
Метаданные сущности дляOrm
 - {имя_сущности}.avm.xml Xml-разметка представления

Внимание

Создается только если нет Avm файла

При запуске формирования кода для сущности, у которой есть коллекции, также будут пересозданы коды для коллекций.

Для массовой регенерации файлов можно вызвать регенерацию для каталога, содержащего odm.xml файлы. В этом случае будут пересозданы исходные коды для всех odm.xml в каталоге.

При вызове регенерации от корневого каталога проекта будут пересозданы исходные коды для всех сущностей.

Тонкая настройка Orm

Для тонкой настройки формируемого orm.xml файла допускается создавать файл-шаблон с именем {имя_класса}.erm.xml, идентичный по формату файлу {имя_класса}.orm.xml, и включать в него элементы и свойства, необходимые для тонкой настройки. При формировании orm.xml будет прочитан erm.xml, и всё его содержимое будет добавлено в сформированный файл orm.xml.

Обновление схемы БД

Для корректной работы системы схема БД должна быть согласована с кодом.

В процессе разработки модуля или при подключении нового модуля к проекту требуется обновлять схему БД специальной утилитой.

Так же, как и генератор кода, утилита обновления схемы БД может быть запущена от любой сущности системы. Таблицы создаются на основе метаданных, объявленных в файлах *.odm.xml. Создание таблиц с использованием EclipseLink отключено. Утилита генерации схемы запускается из контекстного меню: **External Tools > Generate Tables**

Примечание

При обновлении бинарного кода в продакшн-режиме согласование схемы запускается автоматически после обновления jar файлов проекта.

Примечание

Для настройки утилиты смотри главу «Начало работы».

Создание главной выборки приложения

Для создания нового прикладного приложения создайте в своём модуле Avi-класс, унаследованный от `ProjectApplicationAvi`.

Если требуется создать приложение с глобальными фильтрами, то Avi-класс должен быть унаследован от `Bs_ApplicationAvi`. Для создания файла `avm.xml` используйте шаблон «avm Application» в меню создания файлов IntelliJ Idea.

Avi-класс, должен быть зарегистрирован в файле `META-INF/applications.xml` своего модуля.

Отладка приложений

Логирование на сторону клиента

Логирование позволяет отслеживать процесс выполнения запросов, подстановку параметров, последовательность вызова операций, открытия форм.

Для просмотра логов:

1. Зайдите в инструмент разработчика
В Google Chrome и Mozilla Firefox открывается по `Ctrl+Shift+i`
2. Перейдите на закладку «Console»

Для изменения настроек логирования:

1. Нажмите на кнопку настроек
Кнопка с изображением шестеренки находится в правом верхнем углу экрана.

Примечание

Отображение кнопки может быть выключено в настройках сервера.

Настройки логирования позволяют выбрать тип информации, отображаемой в логе(по умолчанию включены все):

- Операции
- SQL
- Скрипт

Для логирования можно задать уровень (предполагается, что нижестоящие уровни выводят также данные по уровням, находящимся выше, за исключением OFF):

Наименование	Описание
TRACE	
DEBUG	
INFO	
WARN	
ERROR	
OFF	Отключение логирования

Клиентское окно отладчика

Дерево выборок

Окно отладчика вызывается при нажатии сочетания клавиш `Ctrl+Shift+Alt+d` или `Ctrl+Shift+Alt+w`.

В левой части расположена иерархическая структура открытых выборок (включая выборки выпадающих списков) со столбцами:

- Выборка - Имя, отражающее функциональную принадлежность выборки.
- Отображение - Системное имя отображения.
- Наименование - Наименование выборки.
- Загрузчик активен - Признак того, что данный загрузчик классов является текущим (активным) для системы.
- Транзакция ORM - Признак того, что начата транзакция с БД.

Внимание

Наличие не завершённой транзакции с БД является ошибочным состоянием. Т.к. может приводить к бесконечным блокировкам записей БД.

- Есть изменения - Есть не сохранённые изменения данных.

В правой части отображаются закладки с данными по выбранной из левой части выборке:

- Атрибуты
- Параметры фильтра
- Параметры выборки
- Операции
- Отладочный макрос
- Выполнить запрос
- Выполнить Jexl-скрипт

- Отладочные переменные

Описание закладок

Атрибуты

На данной закладке отображаются данные по атрибутам из основного запроса выборки для того элемента, на котором находится фокус ввода. Данные не доступны для редактирования.

Атрибуты:

- *Системное имя* - Системное имя атрибута
- *Наименование* - Наименование атрибута
- *Тип данных*
- *Значение* - Значение атрибута
- *Только чтение* - Признак того, что атрибут доступен только для чтения
- *Видимость* - Флаг видимости атрибута
- *Порядковый №*
- *Ширина*

Параметры фильтра

На данной закладке отображаются данные по параметрам фильтра, если фильтр доступен в выбранном отображении. Данные не доступны для редактирования.

Атрибуты:

- *Системное имя* - Системное имя атрибута фильтра
- *Наименование* - Наименование атрибута фильтра
- *Тип данных*
- *Значение* - Значение атрибута фильтра
- *Только чтение* - Признак того, что атрибут фильтра доступен только для чтения
- *Видимость* - Флаг видимости атрибута фильтра

Параметры выборки

Закладка хранит список переменных, созданных в выборке, и параметров, переданных в выборку.

- *Системное имя* - Системное имя параметра/переменной
- *Тип данных* - Тип данных параметра/переменной
- *Значение* - Значение параметра/переменной

Операции

На данной закладке отображаются операции выборки (отображения), которые определены непосредственно в ней, либо унаследованы.

- *Системное имя* - Системное имя операции
- *Наименование* - Наименование операции
- *Активность* - Флаг активности операции
- *Видимость на ТБ* - Флаг видимости операции на панели инструментов
- *Клон* - Признак того, что операция копируемая
- *Порядковый №* - Порядковый № операции на панели инструментов
- *№ иконки* - Номер картинки из коллекции

Отладочный макрос

На данной закладке можно выполнять следующие действия:

- `open`
Открытие выборки, пример выражения:

```
ru.bitec.app.bs.Bs_PrjVer#List
```
- `openModal`
Открытие выборки в модальном окне в режиме просмотра, пример выражения:

```
ru.bitec.app.bs.Bs_PrjVer#List
```
- `doLookup`
Открытие выборки в модальном окне в режиме выбора, пример выражения:

```
ru.bitec.app.bs.Bs_PrjVer#List
```
- `call`
Вызов библиотечной операции, пример выражения:

```
ru.bitec.app.btk.Btk_GroupLib#editGroups
```

Для каждого действия можно задавать список параметров для передачи в выборку/библиотечный метод

Ctrl+F9 - горячая клавиша для запуск макроса. Запуск осуществляется от выборки, на которой находится фокус ввода.

Выполнить запрос

Данная закладка позволяет выполнить sql-запрос с использованием атрибутов выборки, на которой открыли дебагер.

Выполнить Jexl-скрипт

Данная закладка позволяет выполнить jexl-скрипт, в контексте выборки, на которой открыли дебагер.

Отладочные переменные

Список значений, содержащихся в карте `ru.bitec.common.debug.Debug#vars`. Глобальная (для сервера) потокобезопасная карта произвольных значений. Используется для целей отладки/тестирования различного нового функционала.

Отладка сервера в среде IDE

1. Запустите сервер приложения в режиме отладки.
Для настройки запуска сервера приложений смотрите раздел «Настройка рабочего места» в данной документации.
2. Установите точку останова в интересующем месте.
3. Откройте приложение в браузере.
4. Выберите необходимое приложение.
5. Выполните действие, которое вызовет отлаживаемый код.

Запуск jexl скрипта

Позволяет запускать на отладку автономную бизнес логику. Запуск jexl скрипта доступен в приложении «Настройка системы» в меню Сервисы > Библиотека JEXL

Определение причины SQL-вызова из EclipseLink

Для определения причины SQL-вызова из EclipseLink, поставьте точку останова в методе:

```
org.eclipse.persistence.queries.ObjectLevelReadQuery#executeInUnitOfWork:1219
```

Через стек вызова можно проследить, что является причиной вызова.

Определение списка загружаемых расширений (Xxx_YyExt)

Для определения списка загружаемых расширений поставьте точку остановки в методе:

```
ru.bitec.app.gtk.eclipse.api.ExtensionPoint#onCreate : 26
```

Отслеживания изменения атрибута класса.

Для того чтоб поймать изменение любого атрибута объекта в коде поставьте точку остановки в методе (сеттер атрибутов):

```
ru.bitec.app.gtk.eclipse.rdb.EntityAro#_set : 280
```

По желанию можно добавить на точку условия.

Пример: `key == 635651L && propertyName.equals("bBaseMsrItem")`

- 635651L - id объекта(Long)
- bBaseMsrItem - название атрибута

Мониторинг производительности

Для мониторинга производительности:

- jmc.exe
- jvisualvm.exe
- pg_stat_statements

Jmc

Входит в поставку JDK, имеет ограничения для коммерческого использования. Удобна тем, что позволяет видеть снимки стека со строчками кода. Позволяет искать узкие места в сервере приложения.

Jvisualvm

Входит в поставку JDK. Позволяет искать узкие места в сервере приложения.

Снятие снимка

1. Запустите jvisualvm
2. Откройте процесс для мониторинга
Для этого в окне «Application», сделайте двойной клик по нужному приложению.
3. Начните запись снимка
Для этого на закладке «Sampler» в открытом процессе, нажмите кнопку «CPU»
4. При необходимости, в открытом процессе запустите задачу, которую вам нужно профилировать.
5. Остановите запись снимка

6. Перейдите к снимку для анализа
Для этого на закладке «Sampler/CPU Sampler» нажмите кнопку «Snapshot»

Примечание

Профилирования идет по java функциям.

Получение полного списка функций для класса

Scala компилятор может генерировать служебные функции, которых не видно в исходном коде. Однако знание их имён может помочь в локализации узких мест.

Для получения соответствия анонимной функции и строчки кода в классе:

1. Запустите scala console
 1. Откройте Run/Debug configuration
Пункт «Edit configuration» в списке с лева от кнопки «запустить» в меню
 2. Нажмите добавить конфигурацию
 3. Выберете «scala console»
 4. Укажите модуль, для класса которого вы хотите посмотреть список служебных функций
 5. Запустите консоль
 6. Выполните команду :javap -l ru.bites.app.Модуль.Класс

JProfiler

Jprofiler - это комплексный профилировщик Java. Интуитивный пользовательский интерфейс Jprofiler поможет устранить узкие места производительности, точно определить утечки памяти и понять проблемы многопоточности.

Дистрибутив: «<ftp://ftp/pub/%23Distrib/IntelliJ%20IDEA/JProfiler/>»

Установщик сам интегрируется с IntelliJ IDEA

pg_stat_statements

Позволяет вести статистику запросов в базе.

Подробнее смотрите [документацию postgresql](#)

Мониторинг оперативной памяти

Для мониторинга оперативной памяти в продакшене можно использовать

```
jcmd pid GC.heap_dump dumpFile
```

Для анализа файла, рекомендуется использовать:

- [Eclipse Memory Analyzer](#)

Отладка в закрытых средах

Отладка в Visual Studio Code

Редактор VSCode с установленными расширениями является более легковесным средством отладки java-приложений по сравнению с IntelliJ IDEA и более удобным по сравнению с JDB.

Расширение vscode-gtk-debug применяется в случае, когда необходимо отладить приложение на сервере заказчика, но нет возможности предоставить доступ к исходному коду на рабочем месте.

Внимание

Не рекомендуется использовать JVM-отладчик на боевой базе. По возможности желательно запустить копию экземпляра сервера и отлаживать именно её, так как отладчик может негативно повлиять на производительность серверного приложения.

В случае необходимости отладки на боевом сервере обязательно задавайте в фильтрах поток отладки. В противном случае, возможно срабатывание точек прерывания в пользовательских сеансах, что внесёт путаницу в процесс отладки и приведёт к зависаниям на стороне пользователей.

Настройка рабочего места

1. Установите [VSCode](#)
2. Установите расширение (Extension) для отладки
Дистрибутив расширения размещён в подкаталоге, «plugins\vscode
vscode-gtk-debug-x.x.x.vsix»
3. Запустите VSCode
4. На левой панели управления нажмите кнопку Extensions
5. Нажмите кнопку с тремя точками в заголовке открывшейся панели
В открывшемся меню выберите «Install from VSIX»
6. Найдите и выберите файл дистрибутива расширения `vscode-gtk-debug-x.x.x.vsix`
7. В списке установленных расширений появится «Global for Postgres Debugger»

Настройка рабочего каталога

Для начала работы с VSCode необходимо открыть рабочий каталог.

1. Создайте пустой каталог в произвольном месте.
2. Перейдите в VSCode
3. Откройте созданный каталог
Выберете пункт меню **File > Open Folder...** и укажите созданный каталог.
4. Переключитесь на закладку отладки.
Для этого нажмите кнопку «Run and Debug» в левой панели
5. Создайте файл `launch.json`
 1. Кликните на ссылку «create a launch.json file».
 2. В открывшемся выпадающем списке выберите тип «Gtk»
В результате, в каталоге будет создан `.\vscode\launch.json`, содержащий конфигурацию подключения к отлаживаемому серверу Global по умолчанию.
6. Измените значения параметров в соответствии с настройками вашего сервера приложений.

Параметры конфигурации для отладки:

- **hostname**
Имя или IP отлаживаемого сервера
- **port**
Порт, открытый отлаживаемым сервером для подключения java-отладчика. Значение задаётся параметром при старте java-процесса сервера Global.

```
java -Xdebug -Xrunjdp:server=y,transport=dt_socket,address=4000,suspend=n
```

- **threadPattern**
RegExp-выражение фильтрует имена потоков, доступных для срабатывания точек останова. По умолчанию доступны все потоки пользовательских сессий.

Внимание

Из конфигурации подставляются экранированные слэши. Выражение должно выглядеть: `ESession_\\d*`

- **httpPort**
Порт на котором запущен сервер Global. Обычно: 80 или 8080.
- **httpUser, httpPassword**
Имя пользователя и пароль, которые будут использоваться для авторизации Rest-запросов к серверу Global при получении декомпилированных текстов прикладных Java и Scala-классов.

Подключение отладчика к процессу сервера

Для подключения java-отладчика, процесс сервера должен быть запущен с соответствующими параметрами:

```
java -Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=4000,suspend=n
```

1. Переключитесь в режим отладки нажатием кнопки «Run and Debug»
2. Запустите отладку

В случае успешного подключения к отлаживаемому процессу появится панель с командами отладчика а так же строка состояния изменит цвет. Смотрите [Документацию по отладке VSCode](#)

Получение текста прикладного класса

Для установки точки останова «Breakpoint» необходимо открыть в редакторе декомпилированный текст отлаживаемого класса. Получение текста класса не зависит от подключения отладчика к процессу сервера. Главное, что бы в конфигурации отладки были корректно указаны параметры подключения.

1. Нажмите комбинацию клавиш **Shift+Ctrl+P**.
2. В открывшемся поле ввода наберите **Find Classes**
3. Выберите найденную команду «Find Classes. . .»
4. В открывшемся поле ввода введите имя искомого класса и нажмите **Enter**
Буде выполнен Rest-запрос к серверу приложений, для получения списка имён классов, содержащих введённое имя
5. Выберите искомое имя класса из отображённого списка
Если было найдено только одно имя, будет открыт декомпилированный текст класса без промежуточного выбора.

Декомпилированный текст класса содержит расширение *.gdc (Global Decompiled Class). Это виртуальный файл, который не сохраняется на диск. Он состоит из набора строк, на которые можно ставить точки сохранения. Каждая строка содержит имена методов, вызываемых на соответствующей строке исходного кода.

Задание точки останова

Для установки точки, щёлкните мышкой слева от номера требуемой строчки.

В списке «BREAKPOINTS» появится информация об установленной точке. Точки могут устанавливаться до или после подключения отладчика к процессу сервера.

Срабатывание точки остановки

При срабатывании точки остановки в одном из потоков, имя которого соответствует паттерну, произойдёт его остановка. В соответствующих фреймах будут отображены стек вызова и значения локальных переменных. Кликами по стеку, можно перемещаться между файлами.

Перемещение между классами

Если установить курсор на имя класса или вызываемого метода и нажать **F12** или **Ctrl + Click**, будет выполнена попытка перехода к файлу, где объявлен класс.

Если не удастся однозначно определить класс по имени, будет открыт диалог выбора.

Замена декомпилированного текста на исходный код

Скопируйте исходный код класса в буфер обмена и выполните вставку в соответствующий gdc-файл через контекстное меню

После вставки исходного кода можно переключаться между исходным кодом и декомпилированным текстом кнопкой, расположенной в заголовке файла.

Также в заголовке файла находятся кнопки которые переключают режимы подсветки синтаксиса для соответствующего языка.

Отладка приложений в консоли jdb

Этот метод используется, если ошибка не повторяется в окружении с настроенной IDE и невозможна удалённая отладка.

Внимание

Штатный консольный отладчик не позволяет фильтровать поток для точки останова, так что полноценно работать на production сервере, с таким отладчиком нельзя

Внимания примеры приведены для операционной системы windows для Linux смотрите официальную документацию по jdb.

1. Убедитесь что процесс запущен с опциями позволяющими отладку:

```
-agentlib:jdwp=transport=dt_shmem,address=jdbconn,server=y,suspend=n
```

2. Запустите jdb

```
${jdk_home}\bin\jdb -attach jdbconn
```

3. Наберите help, для просмотра списка допустимых команд

Смотрите так документацию: `ru.bitec.app.gtk.debug.Jdb`

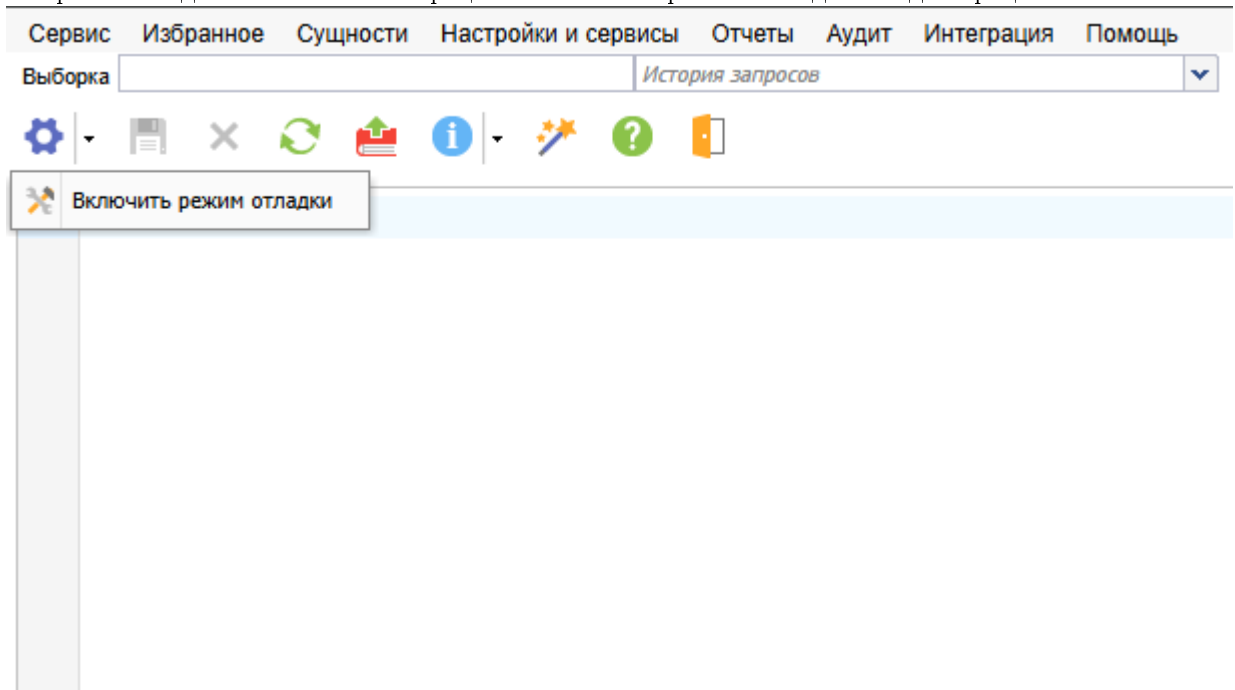
Список самых полезных команд

- **stop**
Установка точки остановки

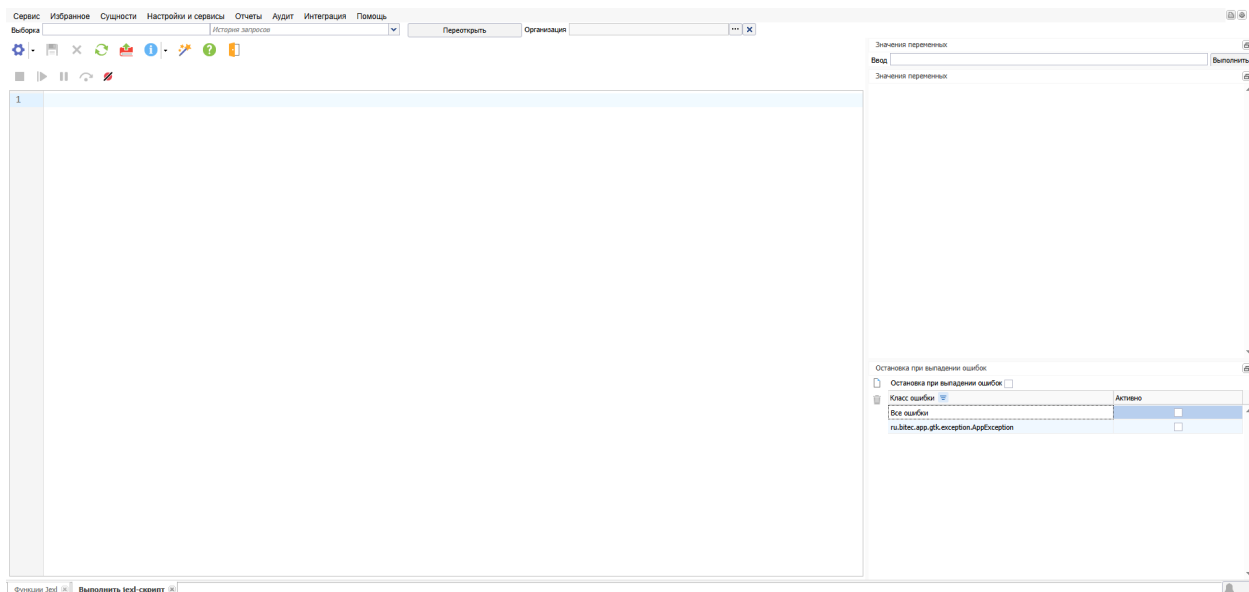
- `where`
Просмотр стэка потока
- `locals`
Печатает переменные в текущем фрейме
- `dump this`
Печатает текущий класс
- `next`
Выполняет строчку кода
- `cont`
Продолжает выполнение

Отладка jexl скриптов

Чтобы открыть отладчик выполните операцию «Включить режим отладки» под операцией «Выпол-



НИТЬ».



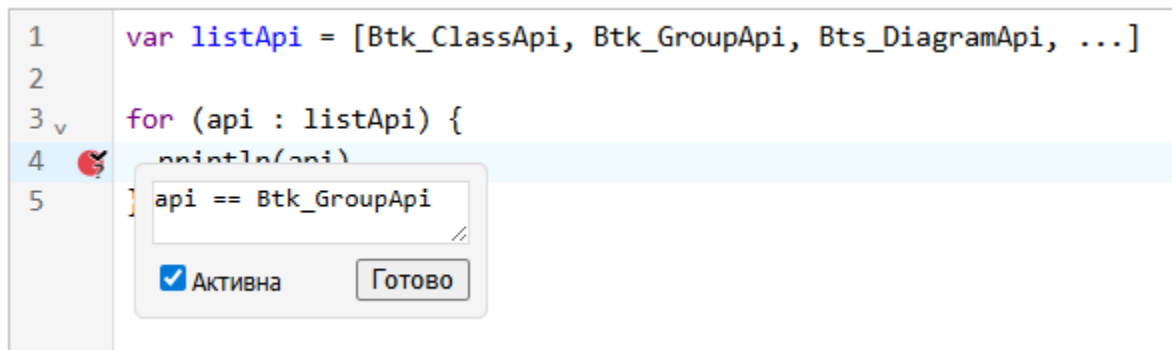
Структура отладчика

Отладчик jexl скриптов позволяет:

1. Устанавливать точки останова
2. Вычислять значения выражений, используя текущие значения переменных
3. Останавливать выполнение кода при выпадении ошибок

Установка точек останова

В окне с кодом можно ставить точки останова, нажимая на кружок слева от номера строки. Через контекстное меню точки останова (правый клик) можно установить условие остановки или временно отключить ее.



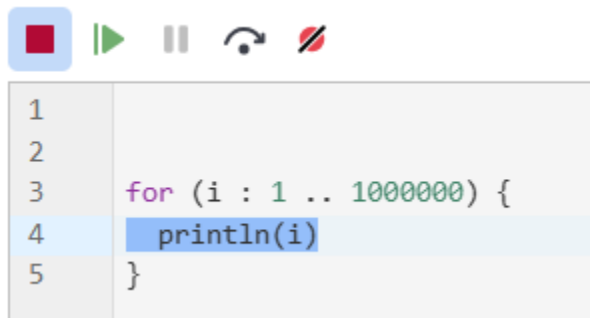
Условие остановки задаётся выражением, результатом которого является boolean.

Примечание

Точка остановки игнорируется, если условие остановки выполнится с ошибкой или вернёт не boolean значение.

Запуск отладки

После установки точек остановки можно запустить отладку (операция «Выполнить»). Во время отладки доступные следующие операции:



- **Остановить отладку**

Прерывает отладку.

- **Продолжить выполнение**

Продолжает выполнение до следующей остановки (в результате достижения точки остановки или выпадении ошибки).

- **Поставить отладку на паузу**

В процессе выполнения скрипта в режиме отладки есть возможность приостановить его выполнение. Во время паузы можно выполнять выражения и выполнять «шаг вперёд».

- **Шаг вперёд**

Продолжает выполнение скрипт на один шаг вперёд. Если встречается вызов функции (объявленной в этом же скрипте) заходит внутрь этой функции.

- **Выключить точки останова**

Когда точки останова выключены, они не оказывают влияния на выполнение скрипта.

Вычисление значений выражений

Во время отладки скрипта (во время паузы или при достижения точки останова) можно ввести выражение и нажать кнопку «Выполнить». Результаты выражения выводятся в строке «результат выражения».

Примечание

Так же в значении переменных выводится результат выполнения `jexl` скрипта или ошибка в соответствующей строке

Значения переменных

Ввод

Значения переменных

Результат выражения	ru.bitec.app.btk.Btk_GroupApi@10d4511b	ru.bitec.app.btk.Btk_GroupApi
listApi	[ru.bitec.app.btk.Btk_ClassApi@bc7e945, ...	java.util.ArrayList

Остановка исполнения кода при выпадении ошибок

Чтобы останавливать выполнение кода при возникновении ошибок, нужно установить галочку «Остановка при выпадении ошибок».

Если выбран класс ошибок «Все ошибки», выполнение прервётся при возникновении любой ошибки. Дополнительные классы ошибок можно указать в списке.

Примечание

В классе ошибок нужно указывать полное имя класса ошибки. Например: java.lang.ArithmeticException

Остановка при выпадении ошибок ☐

Класс ошибки	Активно
Все ошибки	☐
ru.bitec.app.gtk.exception.AppException	☐

Тестирование

Unit-тестирование

При необходимости проверки работоспособности отдельных частей исходного кода имеется возможность реализации unit-тестирования на основе ScalaTest.

Для создания набора тестов необходимо создать класс, который будет наследоваться от `LangFunSuite`, `ApiTest` или других классов, в которые подмешаны трейты из `ScalaTest`.

Совет

Для дополнительной информации смотрите библиотеку unit тестирования: [scalatest](#)

Создание класса с тестами

1. Перейдите в окно проекта
2. Выберите целевой модуль
3. Перейдите в папку с исходными кодами
`[module_name]/src/test/scala`

Совет

Создать недостающую папку можно из контекстного меню в idea `New > Directory`

4. Создайте пакет `ru.bitec.app.[module_name]`
5. Создайте тестовый класс

```
class Lesson1Test extends LangFunSuite{  
  test("HelloWorld"){  
    println("hello world")  
  }  
}
```

Совет

Запустить тест можно из контекстного меню, для этого:

1. Переведите курсор на декларацию функции или класса, если нужно запустить все объявленные тесты класса
2. В контекстном меню выполните операцию „Debug“ для запуска в режиме отладки или „Run“ для простого запуска

Подробнее смотрите [выполнение тестов](#) в руководстве idea.

Примечание

Существуют два специализированных базовых класса для тестовых случаев:

- **LangFunSuite**

Используется для тестов, которые не нуждаются в контроллерах бизнес логики.

Данные тесты не могут использовать `Api`, `Pkg` и не имеют подключения к базе данных по умолчанию.

- **ApiTest**

В данном тексте доступен контекст автономной бизнес логики. Тестовые случаи могут использовать `Api`, `Pkg` объекты. При этом запуск теста становится медленней из за необходимости инициализировать контекст.

Базовые Assert методы

Трейт `Assertions` содержит основные методы для проверки предположений.

1. `assert(condition: Boolean)`

Этот метод проверяет условие. Если переданное условие возвращает `true`, то метод завершается нормально, иначе выбрасывает ошибку `TestFailedException`.

Пример:

```
test("создание объекта класса Btk_SomeClass") {
  try {
    val rop = Btk_SomeClassApi().insert()
    assert(Btk_SomeClassExtApi().findSomeClassExt(rop).isDefined)
  } finally {
    session.rollback()
  }
}
```

Данный тест проверяет, что при создании объекта `Btk_SomeClass` для него создается расширение `Btk_SomeClassExt`. Если расширение не было создано, то тест не будет пройден. В консоль будет выведен текст ошибки

```
Btk_SomeClassExtApi.apply().findSomeClassExt(rop).isDefined was false
```

В `assert` можно передать дополнительный комментарий, который будет добавляться к тексту ошибки:

```
test("создание объекта класса Btk_SomeClass") {
  try {
    val rop = Btk_SomeClassApi().insert()
    assert(rop.get(_.dBeginDate).isNotNull, "При создании не установилась дата начала.
↪")
    assert(Btk_SomeClassExtApi().findSomeClassExt(rop).isDefined, "При создании не
↪зарегистрировалось расширение.")
  } finally {
    session.rollback()
  }
}
```

2. `assertResult(expected: Any)(actual: Any)`

Сравнивает ожидаемое ожидаемое значение с переданным

```
test("parseGtkSessionClientFullName") {
    val x = WorkSessionClientHelper.parseGtkSessionClientFullName("DESkTOP-23565:8080
↪#E1@123-45")
    assert(x.isDefined)
    assertResult("DESkTOP-23565:8080")(x.get._1)
    assertResult("E1")(x.get._2)
    assertResult("123-45")(x.get._3)
}
```

3. `assertThrows[T <: AnyRef](f: => Any)`

Используется, когда нужно удостовериться, что при определенных методах тестируемый код выдаст ошибку.

```
test("data.oaObjQty.isEmpty") {
    val data = new Bs_DistrData()
    assertThrows[AppException] {
        Bs_DistributionPkg().distribQty(data)
    }
}
```

4. `assume(condition: Boolean)`

Метод аналогичен методу `assert`, но в случае, когда не выполняется условие, не завершается ошибкой, а отменяет тест. Так же может содержать пояснительный комментарий.

```
assume(database.isAvailable, "База данных не доступна.")
assume(database.getAllUsers.count === 9)
```

5. `intercept[T <: AnyRef](f: => Any)`

Метод аналогичен методу `assertThrows`, но позволяет получить ошибку ожидаемого типа для более детальной проверки.

```
val vEx = intercept[AppException] {
    _api.validateOnBeforeManualInsert(ropDuplicate)
}
assert(vEx.getMessage.startsWith("Ошибка. Уже существует запись с классом типа_
↪Объектов:"))
```

Matchers

Трейт `Matchers` позволяет использовать в тестах комбинаторы вроде `shouldBe` вместо обычного `assert`, улучшая читаемость кода.

По умолчанию к `ApiTest` и `LangFunSuite` `Matchers` не подключен, поэтому чтобы использовать его методы, добавьте его в объявлении класса:

```
class Bs_AccApiTest extends ApiTest with Matchers {
    test("set level on insert") {
        val rop = api.insert()
        rop.get(_nLevel) should ===(1.nn)
    }
}
```

Для дополнительной информации смотрите в руководстве пользователя: [matchers](#)

Некоторые примеры:

1. Проверка размера и длины

Для проверки размера и длины используйте:

- `<проверяемый объект> should have length (<ожидаемое значение>)`
- `<проверяемый объект> should have size (<ожидаемое значение>)`

Пример:

```
test("проверка размера массива") {  
    val result = getArgs()  
    result should have length (3)  
}
```

2. Проверка строк

- на наличие подстроки

```
string should startWith ("Hello")  
string should endWith ("world")  
string should include ("seven")
```

- с использованием регулярных выражений

```
string should startWith regex ("Hel*o")  
string should endWith regex ("wo.ld")  
string should include regex ("wo.ld")
```

- проверка всей строки на соответствие регулярному выражению

```
string should fullyMatch regex ("^(-)?(\d+)(\.\d*)?$")
```

3. Сравнения больше/меньше/равно

Можно сравнивать объекты любого типа, кроме тех, которые могут быть преобразованы в `Ordered[T]`. Для сравнения списков, используйте `greater than`, `less than`, `greater than or equal`, `less than or equal to a value of type T`

```
one should be < (7)  
one should be > (0)  
one should be <= (7)  
one should be >= (0)
```

- с использованием регулярных выражений

```
string should startWith regex ("Hel*o")  
string should endWith regex ("wo.ld")  
string should include regex ("wo.ld")
```

- проверка всей строки на соответствие регулярному выражению

```
string should fullyMatch regex ("^(-)?(\d+)(\.\d*)?$")
```

Массовое тестирование модулей

Из консоли sbt

- Для запуска тестов по имени/модулю воспользуйтесь командой `testOnly`

```
testOnly {путь до класса}.{имя класса}
```

Например:

```
testOnly ru.bitec.app.pro.diagram.Pro_DiagramApiTest
```

Символ `*` заменяет любое количество символов в наименовании теста, таким образом можно запустить выполнение тестов не по полному имени, а по части наименования. Например, запустить тесты по модулю можно так:

```
testOnly ru.bitec.app.{ваш модуль}.*
```

Например:

```
testOnly ru.bitec.app.tax.*
```

- Для запуска тестов всего проекта воспользуйтесь командой `test`

```
test
```

Примечание

Не рекомендуется использовать, запускайте этот сценарий только на тех проектах, где точно известно, что все тестовые классы не делают нежелательных изменений в файлах или в БД.

- Для запуска только тех тестов, которые связаны с последними изменениями в коде, используйте команду `testQuick`, которую можно вызывать с такой же фильтрацией, как и `testOnly`.

Запуск сценариев тестирования с использованием JMeter

- Для запуска скачайте и установите **JMeter**
- Скачайте пример сценария, запускающего тестирование, и откройте его в **JMeter**:
Обратитесь к дистрибьютору документации за файлом `files/unitTest.jmx`
- Данный проект содержит три примера сценария:
 - `TestByModule` - тестирование по модулю
 - `TestByClass` - тестирование по классу
 - `TestAll` - запуск всех тестов проекта
- Выберите нужный тред и сделайте его активным (операция контекстного меню `Enable`. `Disable` - сделает выбранный элемент неактивным).
- Развернув выбранный тред, отредактируйте параметры компонента `run test`:
 - При необходимости замените выбранную рабочую директорию на ту, где лежит исполняемый файл `sbt` нужного вам проекта.

- В параметрах замените исполняемую команду (Примеры см. выше), исправьте модуль или имя файла.
- Запустите тестирование через операцию Start на тулбаре. Выполнение тестирования может занять продолжительное время. Дождитесь окончания.
- Результаты выполнения будут отображены в View Results Tree. Для более детального отчета можно посмотреть логи на вкладке «Response data».

Выборочное исключение тестов из массового тестирования

Иногда возникает необходимость временно отключить тест, для этого нужно вызов метода test заменить на ignore. В этом случае при тестировании в результатах будет записано, что этот тест проигнорирован.

Jexl-тесты

В меню Настройки и сервисы - Сервисы Jexl - Jexl-тесты можно писать и выполнять jexl-тесты.

Создание

При создании jexl-теста ему необходимо задать план, который будет состоять из разных шагов, выполняемых друг за другом. Для каждого шага необходимо задать наименование и текст скрипта, который должен выполняться. Порядковый номер шага проставится автоматически.

На закладке «Глобальные переменные» можно задать переменные, к которым можно обращаться в тесте через методы `tst.setGlobalVar` и `tst.getGlobalVar`. Для их создания необходимо указать системное имя переменной, которая должна быть уникальна для теста, и ее значение в виде jexl-скрипта, который будет выполняться перед стартом теста.

Доступные методы

tst.setVar

Записывает в контекст выполнения jexl-теста переданное значение. Информация об изменении переменной записывается в лог выполнения теста с типом SETVAR

Пример:

```
tst.setVar("param_name", param_value); //param_name - имя параметра, param_value - значение параметра
```


tst.getVar

Достает из контекста значение параметра по его имени.

Пример:

```
var p = tst.getVar("param_name"); //param_name - имя параметра
```

tst.setGlobalVar

Записывает в глобальные переменные выполнения jexl-теста переданное значение. Доступ к этим переменным есть на любом шаге теста, без необходимости получать нужный контекст. Информация об изменении переменной записывается в лог выполнения теста с типом **SETVAR**

Пример:

```
tst.setGlobalVar("param_name", param_value); //param_name - имя параметра, param_value - значение параметра
```

tst.getGlobalVar

Получает значение глобальной переменной по ее имени.

Пример:

```
var p = tst.getGlobalVar("param_name"); //param_name - имя параметра
```

tst.findParentContext

Возвращает контекст предыдущего шага выполнения теста. Из него можно получить параметры, записанные в контекст на предыдущем шаге.

Пример:

```
var parent = tst.findParentContext();  
var parentParam = parent.getVar("param_name");
```

tst.info

Записывает в лог выполнения теста переданный текст с типом **INFO**.

Пример:

```
tst.info("Текст с информацией");
```

tst.warning

Записывает в лог выполнения теста переданный текст с типом **WARNING**.

Пример:

```
tst.warning("Текст с предупреждением");
```

tst.error

Записывает в лог выполнения теста переданный текст с типом **ERROR**.

Пример:

```
tst.error("Текст с ошибкой");
```

tst.raise

Выбрасывает ошибку с переданным текстом, а также записывает ее в лог выполнения с типом **ERROR**.

Пример:

```
tst.raise("Текст с ошибкой");
```

tst.assertTrue

Проверяет, что переданное условие истинно. Если оно ложно, то прекращает выполнение теста и записывает в лог выполнения ошибку. Если выполняется группа тестов и проверка не пройдена, то переходит к следующему тесту.

Пример:

```
tst.assertTrue(1 == 2); //запишет в лог "assertTrue не прошло проверку"

tst.assertTrue(1 == 2, "Число 1 не равно числу 2"); //запишет в лог переданный текст
↪ "Число 1 не равно числу 2"
```

tst.shouldBeTrue

Проверяет, что переданное условие истинно. Если оно ложно, то записывает в лог выполнения ошибку.

Пример:

```
tst.shouldBeTrue(1 == 2); //запишет в лог "shouldBeTrue не прошло проверку"

tst.shouldBeTrue(1 == 2, "Число 1 не равно числу 2"); //запишет в лог переданный текст
↪ "Число 1 не равно числу 2"
```

tst.sqlRows

Возвращает количество строк, полученных в результате выполнения переданного запроса

Пример:

```
var count = tst.sqlRows(`select t.id from RplTst_AllDbTypes t where t.idObjectType =  
↪120`);
```

Выполнение

Выполнение теста осуществляется в помощью операции **Выполнить тест** в карточке теста или списке тестов. Для выполнения тестов целой группы необходимо в списке тестов выполнить операцию **Выполнение всех тестов текущей группы**.

Выполнение с помощью Rest сервиса

Выполнение тестов также возможно с помощью Rest сервиса.

Путь для выполнения одиночного теста: `http://{имя_сервера}/app/sys/rest/ss/pkg/Bts_TestPkg/runSingleTest`

Путь для выполнения группы тестов: `http://{имя_сервера}/app/sys/rest/ss/pkg/Bts_TestPkg/runTestForGroup`

Тело обоих запросов является json-объектом, который состоит из ключа **name** и его значения, в виде системного имени теста или группы, соответственно.

Запрос выполнения одиночного теста возвращает json-объект, который содержит в себе следующие параметры: имя теста **name**, наличие ошибок **hasErrors**, наличие предупреждений **hasWarnings**, количество ошибок **countErrors** и количество предупреждений **countWarnings**.

Пример для одиночного теста:

```
//пример обращения  
curl -H Database:pgDev -u admin:admin -d {"name":\\"PRs_PurchReqTest\\"} -H "Content-  
↪Type: application/json" -X POST http://localhost:8080/app/sys/rest/ss/pkg/Bts_TestPkg/  
↪runSingleTest  
  
//пример результата  
{  
  "hasErrors":true,"hasWarnings":false,"countErrors":1,"countWarnings":0,"name":"Тест_  
↪заявок на закупку услуг"}  
}
```

Запрос выполнения группы тестов возвращает json-объект, который содержит в себе следующие параметры: наличие ошибок **hasErrors**, наличие предупреждений **hasWarnings**, количество ошибок **countErrors**, количество предупреждений **countWarnings** и результаты выполнения всех тестов **tests** в виде массива json-объектов таких же, как при выполнении одиночного теста.

```
//пример обращения  
curl -H Database:pgDev -u admin:admin -d {"name":\\"Test1\\"} -H "Content-Type:  
↪application/json" -X POST http://localhost:8080/app/sys/rest/ss/pkg/Bts_TestPkg/  
↪runTestForGroup  
  
//пример результата  
{  
  "hasErrors":true,"hasWarnings":false,"countErrors":2,"countWarnings":0,"tests":[{"  
    "hasErrors":false,"hasWarnings":false,"countErrors":0,"countWarnings":0,"name":"test_ok  
↪"
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
→"}, {"hasErrors": true, "hasWarnings": false, "countErrors": 1, "countWarnings": 0, "name":  
→ "test333"}, {"hasErrors": true, "hasWarnings": false, "countErrors": 1, "countWarnings": 0,  
→ "name": "test22"}]}}
```

Статический анализ

Использование

Статический анализ происходит во время компиляции кода

1. Частичная сборка - покажет где в измененных файлах есть ошибки
2. Полная пересборка - покажет где в исходниках есть ошибки

Добавление правил

Уже существующие [правила](#)

Правила добавляются в build.sbt, или для всех модулей в sbt-plugin CommonSettings

```
val settings = Seq(  
  wartremoverWarnings += Seq(  
    Wart.custom("ru.bitec.warts.NetWart"),  
    Wart.ListAppend  
  )  
)
```

Написание правил

1. Кастомные правила пишутся в этот [проект](#)
2. Официальная документация [wartremover](#)
3. Документация по обходу синтаксического дерева [scala](#)
Главный помощник для написания правил это функция [reify](#)

```
import scala.reflect.runtime.universe._  
val tree = reify {  
  println(2)  
}.tree // это дерево не содержит информацию о типах  
  
/*  
  Apply( вызов метода  
    Select(  
      Select(This(TypeName("scala")), TermName("Predef")), на чем вызывают метод  
      TermName("println") что вызывают  
    ),  
    List(Literal(Constant(2))) параметры  
  )  
*/  
val raw = showRaw(tree)
```

Горячие клавиши

Глобальные горячие клавиши

Горячие клавиши определенные для любого отображения в системе:

Сочетание клавиш	Назначение
Ctrl+L	Размещение всех колонок на экране
Ctrl+Alt+A	Автофильтр
Ctrl+W	Подобрать ширину всем колонкам
Shift+Ctrl+=	Сумма значений по выделенным записям
Esc	Клавиша отмены редактирования
Num-	Свернуть всё дерево
Ctrl+Alt+H	Настройка видимости колонок
Shift+Ctrl+Alt+D	Вызов окна отладки
Shift+Ctrl+Alt+W	Вызов окна отладки
Shift+Ctrl+Alt+V	Открытие карточки выборки
Ctrl+F9	Выполнение отладочного макроса
Enter	Клавиша окончания редактирования
Shift+Ctrl+F	Построитель фильтров
Ctrl+F7	Добавить значение ячейки в фильтр
Ctrl+Alt+F	Отфильтровать выделенные записи
Ctrl+Alt+Left	Зафиксировать по левому краю
Ctrl+Alt+Right	Зафиксировать по правому краю
Ctrl+Alt+T	Область итогов
Ctrl+Alt+G	Область группировки
Ctrl+Del	Скрыть колонку
Shift+Ctrl+T	Промежуточные итоги
Ins	Множественный выбор
Ctrl+P	Вывести на печать...
Ctrl+R	Сервис замены
Ctrl+Alt+D	Восстановить настройку по умолчанию
Shift+Ctrl+S	Сохранить как...
	Отобразить выделенные записи
Ctrl+Enter	Клавиша открытия редактора
Enter	Клавиша начала редактирования
Ctrl+-	Дерево - Заккрыть текущий узел
Ctrl+=	Дерево - Открыть текущий узел
Ctrl+Alt+=	Дерево - Открыть текущий узел и его подузлы
Shift+Ctrl+F7	Универсальные фильтры - Отмена
Ctrl+Alt+F7	Универсальные фильтры - Очистить
F7	Универсальные фильтры - ОК
Ctrl+O	Универсальные фильтры - Открыть...
Shift+Ctrl+O	Универсальные фильтры - Открыть общие настройки...
	Универсальные фильтры - Сохранить
	Универсальные фильтры - Сохранить как...
Ctrl+Alt+U	Пользовательская настройка - Открыть...
	Пользовательская настройка - Сохранить
	Пользовательская настройка - Сохранить как...
	Загружать данные полностью

Примечание

Сочетания глобальных горячих клавиш можно посмотреть в файле «ru/bitec/app/btk/HotKeys.xml» и в настройках системы, открыв выборку Btk_HotKeysAvi#Default

Горячие клавиши

Правила указания сочетания клавиш:

- *Символы в верхнем регистре* - [A-Z]
- *Цифры* - [0-9]
- *F клавиши* - [F1-F12]
- *Специальные клавиши* - [Ctrl Alt Shift Esc]

Клавиши в сочетании указываются через символ +

Совет

Не используйте слишком сложные сочетания клавиш, расположенные далеко друг от друга, а также сочетания, которые используются в браузере.

Сочетания горячих клавиш в Avi

Для задания сочетания горячих клавиш в Avi на операции, требуется определить атрибут hotKey:

```
@Oper(hotKey = "Ctrl+S")
override def saveForm(): Unit = super.saveForm()
```

Сочетания горячих клавиш в Avm

Также задать сочетание клавиш можно и в Avm, определив атрибут hotKey:

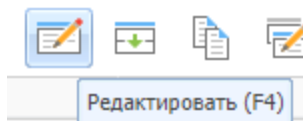
```
<operations>
  <oper name="SHOWTAB" hotKey="Ctrl+Alt+T"/>
</operations>
```

Внимание

Горячие клавиши заданные в Avm перекрывают сочетания заданные в Avi

Примечание

Сочетания горячих клавиш будут отображаться на всплывающей подсказке с наименованием операции в скобках



Примечание

Для невидимых операций можно задавать горячие клавиши, которые будут наследоваться. Но пока они невидимы, операция не будет срабатывать при использовании горячих клавиш.

Наследование горячих клавиш

Горячие клавиши у операций наследуются также, как и другие атрибуты. Если у потомка в отображении атрибут не был переопределен, он будет иметь значения предка.

Последовательность срабатывания горячих клавиш

При нажатии сочетания клавиш поиск нужного сочетания будет выполнен в следующем порядке:

- *Глобальные горячие клавиши приложения*
- *Горячие клавиши главной выборки*
- *Горячие клавиши детализации*

Если сочетание найдено, выполняется соответствующая операция и поиск прекращается. То есть, если сочетание клавиш повторяются, то выполнится операция, которая определена в этом списке раньше.

Совет

Чтобы посмотреть порядок, в котором будут проверяться соответствие горячим клавишам, можно открыть `debugger` и посмотреть дерево отображений. Срабатывать будут горячие клавиши более верхнеуровневых отображений.

Типовые горячие клавиши

Для некоторых операций стандартных отображений определены горячие клавиши

Горячие клавиши для Entity

Сочетание клавиш	Операция	Системное имя
F2	Выбор	CLOSEFORMOK
Esc	Выход	CLOSEFORMCANCEL
Ctrl+S	Сохранить	SAVEFORM
Ctrl+B	Откатить	CANCELFORM
F12	Обновить	REFRESH
F11	Детализация	SHOWTAB
F1	Помощь	HELP
Ctrl+Shift+Alt+A	Аудит объекта	SHOWAUDITOBJECT
Ctrl+Shift+Alt+F	Расширенный аудит по объекту	SHOWAUDITSEL
Ctrl+Alt+I	Информация об объекте	SHOWABOUTOBJECT
F4	Редактировать	CARDEDIT

Горячие клавиши для CollectionClass

Сочетание клавиш	Операция	Системное имя
F2	Выбор	CLOSEFORMOK
Esc	Выход	CLOSEFORMCANCEL
Ctrl+B	Откатить	CANCEL
Ctrl+Q	Отменить	ROLLBACK
Ctrl+S	Сохранить	COMMIT

Горячие клавиши для Standalone

Сочетание клавиш	Операция	Системное имя
F12	Обновить	REFRESH

Горячие клавиши для Application

Сочетание клавиш	Операция	Системное имя
Ctrl+Alt+Shift+E	Поиск по глобальному идентификатору	MM_BTK_FINDBYGID

Проектное переопределение горячих клавиш

Проектное переопределение глобальных горячих клавиш

На данный момент переопределить проектно глобальные горячие клавиши нельзя.

Проектное переопределение горячих клавиш

Если требуется переопределить горячие клавиши проектно, то в соответствующем переопределенном файле Avn или Avi для операции изменяется значение атрибута `hotKey`.

Проектное переопределение типовых горячих клавиш

На данный момент переопределить проектно типовые горячие клавиши так, чтобы у потомков оставались прежние сочетания, нельзя.

13.5 Отчеты

Построение отчета

Отчеты системы GlobalFramework хранятся в базе данных — это позволяет легко адаптировать их под проекты. Для управления отчетами перейдите в приложение Global - Настройка системы.

Существует четыре типа отчетов:

- произвольный;
- динамический;
- отчет с селекционным экраном;
- экранная форма.

Варианты построения отчета

Операция «Построить отчет»

Стандартное построение отчета выбранного типа с заполнением параметров.

Для построения отчета нужно перейти в **Отчеты - Настройки отчетных форм**, выбрать нужный отчет, нажать **Построить отчет** и заполнить необходимые параметры.

Операция «Построить отчет с текущими параметрами»

Построение отчета с уже настроенными ранее параметрами.

Для построения отчета нужно перейти в **Отчеты - Настройки отчетных форм**, выбрать нужный отчет, перейти на закладку «Журнал построения отчета», выбрать одну из предыдущих версий отчета и нажать **Построить отчет с текущими параметрами**. Система создаст новый отчет, используя параметры выбранного варианта.

Работает аналогично операции «Построить отчет».

Операция «Построить отчет в фоновом режиме»

Для работы отчета в фоновом режиме необходимо

- Включить параметры:
 - «Вести журнал построения отчета» (закладка «Объектные характеристики»).
 - «Доступен фоновый режим» (закладка «Печатные формы отчета»).
- Убедиться, что планировщик Btk_Job запущен и корректно настроен в системе. Через него работает фоновая печать.
- Отчет не должен быть интерактивным. Печать интерактивного отчета в фоновом режиме невозможна.

1. Работает аналогично операции «Построить отчет», но в фоновом режиме. Запускается той же кнопкой.
2. При запуске (после окна с указанием параметров) открывается модальное окно с выбором:
 - Режима выполнения.
 - Даты и времени старта.
3. После завершения система показывает уведомление с перечнем построенных отчетов.
4. Результаты сохраняются:
 - В Журнале построения отчета.
 - Как прикрепленные файлы (если выбрано сохранение документа).
5. Если отчет запланирован, то в журнале можно удалить запись в состоянии «Запланирован к фоновой печати».
6. Подробности выполнения смотрите:
 - На вкладке **Журнал построения отчета**.
 - В меню **Сервис → Фоновые задания** (отображает все фоновые процессы).

Механизм построения отчета

Для произвольных отчетов

1. **Механизм поиска отчета:** Система ищет печатную форму, указанную на закладке «Объектные характеристики».
2. **Открытие выборки с параметрами:**
 - Вычисляет расширение файла (если выбрано несколько расширений по умолчанию), в котором будет создан отчет.
 - Открывает выборку в режиме `lookup`, в которой заполняется карта параметров `Rpt_Pkg(). docReportParamsMap` (системное имя параметра → параметр).
3. **Печать отчета:** Отчет формируется через метод `printReport`.

Для остальных типов

1. **Механизм поиска отчета:** Система ищет печатную форму, установленную «По умолчанию», указанную на закладке «Печатные формы отчета».
2. **Открытие выборки с параметрами:**
 - Вычисляет расширение файла (если выбрано несколько расширений по умолчанию), в котором будет создан отчет.
 - Форма с параметрами указывается напрямую названием выборки и отображения.
3. **Печать отчета:** Отчет формируется через метод `printReport`.

Механизм поиска отчета

- **Динамический отчет:** Поиск по `sSelectionName` и `sRepresentationName`.
- **Отчет с селекционным экраном:** Поиск по `sSelectionName` и `sRepresentationName`.
- **Экранная форма:** Поиск по `sSelectionName` и `sRepresentationName`.
- **Произвольный отчет:** Механизм отличается (см. раздел *Описание операций*).

Название выборки `sSelectionName` и отображения `sRepresentationName` указываются в карточке в объектных характеристиках отчетной формы `Rpt_Entity`.

Пример:

- `sSelectionName` (имя выборки): `Asf_InvNumbInSelScreenAvi`
- `sRepresentationName` (имя отображения): `SelectionScreenCardDeprTax`

Проверка дубликатов отчетов по параметрам

При построении нового отчета система определяет его ключевые параметры (все параметры указаны в разделе «Параметры отчета»), вычисляет их хэш и ищет готовые отчеты с такими же параметрами. При совпадении предлагается выбрать существующий отчет вместо создания нового.

Условия работы:

- Параметр «Тип проверки отчета» выбран: «При построении» или «При построении и сохранении» (закладка «Объектные характеристики»).
- Включен параметр «Вести журнал построения отчета» (закладка «Объектные характеристики»).

Если условия не выполнены — система всегда создает новый отчет.

Сохранение отчетов

Вариант сохранения определяется параметрами на закладке «Объектные характеристики»:

1. «Сохранять отчет в документ».
 - Создается документ формата **Wf_Doc** с прикрепленным PDF-отчетом.
 - В журнале появляется запись о построении отчета с указанием версии документа.
2. «Сохранять отчет в журнал как прикрепленный файл».
 - Создается документ **Wf_Doc** с PDF.
 - В журнале появляется запись о построении отчета + документ прикрепляется к этой записи.

Оба варианта используют документ общего типа **Wf_Doc**, к которому прикрепляются файлы отчетов. Разница заключается в способе связи между записью в журнале и созданным документом.

Закладки справочника

Параметры отчета

- Содержит параметры, настроенные на отчете.
- Создание и удаление параметров по умолчанию выполняется в классе **Печатные формы** на соответствующей закладке.
- Получить параметры можно с помощью операции **Получить параметры отчета**.

Объектные характеристики

Настройки печати отчетов:

- Тип проверки отчета при построении.
- Ведение журнала построения отчета.
- Сохранение отчета в журнал как прикрепленного файла.
- Сохранение отчета в документ (прикрепление отчета к документу класса **Wf_Doc**).

Журнал построения

- Содержит ранее сформированные отчеты.
- Хранит параметры и метаданные отчетов.
- Запись создается, только если включен соответствующий флаг на закладке **Объектные характеристики**.

Дополнительные настройки

- **Тип документа:** Определяет тип документа, создаваемого при построении отчета.
- **JEXL для даты версии:** Содержит Jexl-выражение, которое вычисляет дату версии отчета.
- **Роли доступа отчета:** Определяет, кто может просматривать или запускать отчет.
- **Процедуры интерактивного отчета:** В закладке создаются процедуры, которые можно выполнить прямо в выборке построения отчета.
- **JEXL расширения сеттеров отчета:** Позволяет задать кастомную логику сеттера для интерактивного отчета.

Печатные формы отчета

Для произвольных отчетов

- Допустима только одна печатная форма.
- Из этой формы берутся все параметры отчета.

Для остальных типов

- Отображается список всех печатных форм.
- Можно установить флаг печатной формы по умолчанию.

Печатная форма

Печатная форма (сокр. ПФ) определяет правила построения отчета и его доступность для приложения, *типа объекта* и пользователя. При выполнении печатной формы система формирует отчет в файле заданного формата.

Печатную форму можно подключить к выборке на проекте без изменений в программном коде.

Создание и настройка печатной формы

Создание печатной формы

1. Перейдите в приложение **Настройка системы**.
2. Откройте раздел: **Отчеты** → **Печатные формы**.
3. Нажмите **Создать**.
4. Заполните поля:
 - Системное имя;
 - Наименование;
 - Модуль.
5. Создайте версию печатной формы.
6. Выберите тип шаблона.
7. Загрузите файл шаблона — выполните операцию **Загрузить файл в базу**.

Типы шаблонов печатных форм

Определяют формат шаблона:

- **jasper** — zip-архив с XML-файлами, созданными в *Jasper Studio*;
- **xlsx** — файл XLS со спец. тегами;
- **docx** — файл DOCX со спец. тегами.

Версия печатной формы

Позволяет:

- Безопасно вернуться к предыдущей реализации.
- Запускать разные версии в зависимости от периода.

Содержит:

- Тип шаблона печатной формы.
- Бинарный файл для построения отчетов.
- Дату версии.
- Описание.

Доступные форматы

Если тип шаблона печатной формы поддерживает конвертацию в другие форматы, можно указать список доступных форматов. Если указано несколько форматов, при печати печатной формы система запросит у пользователя выбор формата для построения отчета.

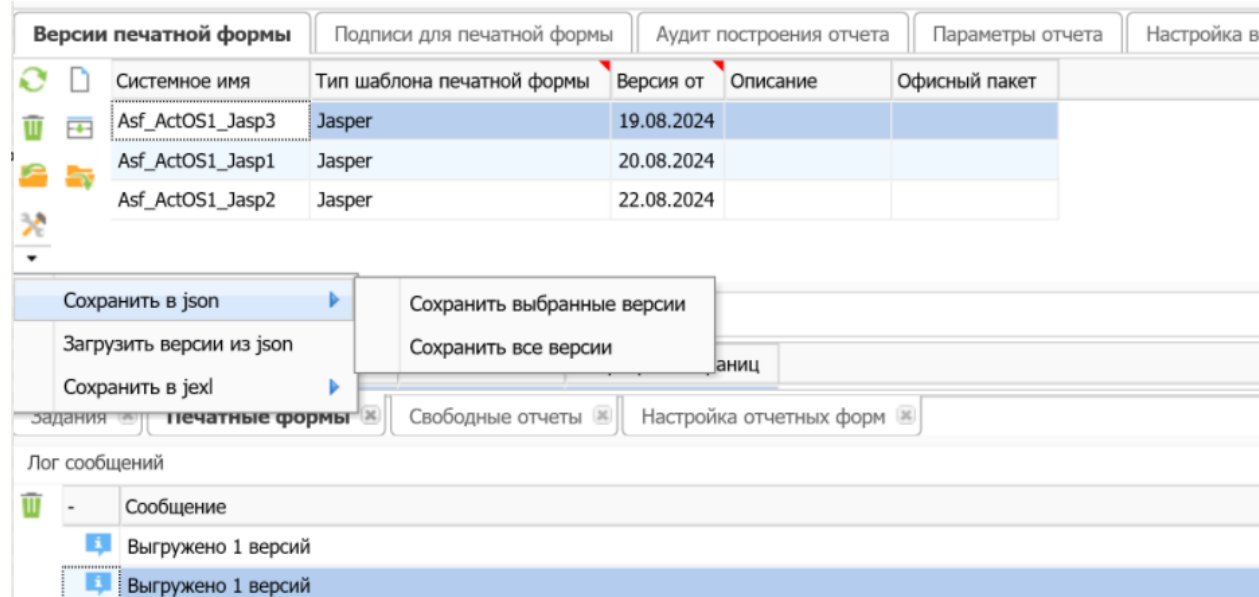
Сохранение и перенос версий печатных форм

В интерфейсе есть кнопка **Дополнительно**, которая открывает функционал для сохранения и загрузки версии отчетов.

Ключевые возможности:

- Экспорт версий - можно скачать версии отчета в виде файла
- Импорт версий - перенести сохраненные версии в другую базу данных
- Форматы переноса - для удобства работы реализован перенос с использованием:
 - JSON - структурированный формат данных
 - JEXL - язык выражений для сложных преобразований

Это упрощает перенос настроенных печатных форм между различными базами данных системы.



Настройка вставки изображений

Настройка вставки изображений в печатную форму типа DOCX и XLSX выполняется в коллекции к печатной форме. Все поля обязательны.

- **Активность:** Поле определяет, будет ли вставлено изображение.
- **Шаблон (Тег):** Тег, по которому будет вставлено изображение. Тег должен содержаться в печатной форме, иначе изображение не будет вставлено. Существует два вида тегов:
 - **Общий:** Тег не имеет динамической части. Изображение будет вставлено при нахождении данного тега в документе. Пример:
 - * В поле: SomeTag.
 - * В документе: [SomeTag].
 - **Подпись:** Тег для подписи состоит из общей части для всех тегов (которая задается в данном поле) и имени пользователя (подпись которого необходимо вставить в данный документ). Пример:
 - * В поле: SomeSignTag.
 - * В документе (с динамической частью): [SomeSignTagAdmin].
- **Печатная форма изображения:** В поле указывается печатная форма с типом Jasper и форматом PNG. В зависимости от типа изображения данной печатной форме будут переданы следующие аргументы:

– **Подпись:**

- * «PSDATE» - дата подписания документа;
- * «PSSERNUMBER» - № сертификата подписи;
- * «PSDBEGIN» - дата начала действия подписи;
- * «PSDEND» - дата окончания действия подписи;
- * «PSFIO» - ФИО сотрудника, подписавшего документ;
- * «IDDOC» - ID документа;
- * «IDDOCV» - ID версии документа (если документ версионный, то подписывается именно версия а не сам документ);
- * «JOB» - JSON объект подписи документа.

– **Общий:**

- * «IDDOC» - ID документа;
- * «IDDOCV» - ID версии документа (если документ версионный, то подписывается именно версия а не сам документ).

- **Масштаб и Разрешение:** Размер изображения высчитывается по формуле: Масштаб * Measures.DOTS_PER_INCH / Разрешение.
- **Тип:** В поле указывается тип изображения:
 - **Общий:** Изображение вставляется при печати в любом случае.
 - **Подпись:** Изображение вставляется, если на документе есть хотя бы одна подпись.

Вставка изображений

Для вставки изображений в печатную форму типа docx должны быть выполнены следующие условия:

1. Печатная форма должна содержать теги вида [SomeTag].
2. Тип шаблона печатной формы - DOCX или XLSX.
3. Формат печатной формы - PDF.
4. В коллекции к печатной форме(Настройки вставки изображений в печатную форму) настроены необходимые изображения для вставки.

Если все требования выполнены корректно, то при печати такой ПФ будет получен PDF-файл (для DOCX) или XLS с изображениями на месте тегов.

Примечание

В печатной форме изображения можно использовать параметры той ПФ, в которую происходит вставка.

Вызов печатной формы

Печатные формы можно вызывать:

- из операции выборки;
- из запроса к сервису печати;
- из *REST*-сервиса;
- от произвольного объекта системы;
- через интерфейс **Свободные отчеты**.

Совет

Проверка полномочий в печатных формах

Из операции выборки

Для вызова отчета из операции выборки используйте функцию `ru.bitec.app.gtk.gl.Reports#createReportEx`:

```
/**
 * Выполняет построение отчета по системному имени отчета.
 *
 * @param reportName      Имя отчета
 * @param reportVersionDate Дата
 * @param postBuildAction Действие, которое необходимо произвести после заполнения
↳ отчета
 * @param propertyMap     Карта входящих параметров
 */
@throws[ApplicationException]
def createReportEx(reportName: String, reportVersionDate: Date, postBuildAction:
↳ PostBuildAction, propertyMap: Map[String, Any]): Unit
```

Данная функция может быть вызвана только в контексте интерактивной бизнес-логики (интерфейса пользователя).

Пример:

```
reports.createReportEx("Mct_OrderSheetMaterials2", null, PostBuildAction.print,
  Map[String, Any]("IDSRBJECT" -> getVar("id").asNLong,
    "GIDSRBJECT" -> getVar("gid").asNString,
    "SDESIGNATION" -> getVar("sCode").asNString)
)
```

От произвольного объекта

При открытии карточки объекта отображаются стандартные операции печати.

Чтобы добавить ПФ к типу объекта:

1. Откройте приложение **Настройка системы**.
2. Откройте типы объектов **Сущности > Типы объектов > Типы объектов**.
3. Перейдите на вкладку **Печатные формы**.
4. Добавьте необходимые печатные формы.

Из свободных отчетов

Свободные отчеты позволяют настроить для пользователя и приложения перечень отчетов, которые можно построить без привязки к каким-либо типам объектов.

Для вызова интерфейса построения свободных отчетов:

1. Откройте приложение, в котором есть пункт меню **Отчеты**.
2. Откройте свободные отчеты **Отчеты > Свободные отчеты**.
3. Выберите нужный отчет.
4. Заполните параметры.
5. Напечатайте отчет. Для этого выполните операцию **Печать**.

Для того чтобы ПФ могла быть вызвана из свободных отчетов:

1. Откройте приложение **Настройка системы**.
2. Откройте печатные формы **Отчеты > Печатные формы**.
3. Выберите требуемую печатную форму.
4. Включите признак **Свободный отчет**.
5. Настройте параметры отчета.
6. Укажите приложение для печати.
7. Укажите требуемые роли.

Примечание

Если роли не указаны, печатная форма будет доступна всем пользователям.

Параметры и данные для построения отчета

Параметры отчета

Параметры отчета передаются в запрос, формирующий данные отчета. Параметры настраиваются на печатной форме в соответствующей закладке.

Перед выполнением отчета пользователь может ввести данные в эти параметры.

Служебные параметры

Формируются автоматически

- SIGNDATA_DZ - Блок подписи.
- IDUSER - Пользователь.
- IDSRCOBJECT - Объект источник. Идентификатор объекта, от которого выполняется печатная форма.
- IDSRCCLASS - Класс источник. Идентификатор класса объекта, от которого выполняется печатная форма.

Совет

Видимость кнопки **Печать отчета** настраивается в закладке «Тип объектов» с помощью параметра «jexl-скрипт фильтрации ПФ», который определяет, будет ли кнопка доступна для данного типа объекта.

Подписи для печатных бланков

Если отчет должен быть подписан после печати, в запрос печатной формы следует добавить атрибуты, необходимые для формирования блока подписания.

```
with signs as(select * from jsonb_array_elements( ${SIGNDATA_DZ}::jsonb ->'data'))
select t.*
, (select s.value ->>'sFIO' from signs s WHERE s.value->>'idBlankSignTypeMC' =
↪'matching1') as sMatching1
, (select s.value ->>'sPosition' from signs s WHERE s.value->>'idBlankSignTypeMC' =
↪'matching1') as sMatching1Pos
```

На печатный бланк необходимо вывести блок с подписями.

Блок с подписями формируется перед печатью и передается в отчет в формате JSON, в параметр SIGNDATA_DZ. Пример:

```
{
  "data": [
    {
      "idBlankSignTypeMC": "HeadOfDepartment",
      "bReadOnly": 0,
      "idBlankSignTypeHL": "Руководитель подразделения",
      "nOrder": 1,
```

(продолжается на следующей странице)

```

        "dDate": "10.12.2020",
        "idBlankSignType": 95401,
        "sPosition": "Начальник отдела",
        "sFIO": "Дьяченко Евгений Владимирович"
    }
]
}

```

Где:

- idBlankSignType - идентификатор тип подписи;
- idBlankSignTypeMC - код типа подписи;
- nOrder - порядковый номер подписи;
- sPosition - должность;
- sFIO - ФИО;
- dDate - дата.

Блок формируется на основе настроек данной закладки.

Настройки могут быть переопределены\уточнены на конкретном объекте в закладке **Подписи** Bs_ObjectSign.

Аудит построения отчета

Содержит информацию о построении отчетов пользователями.

Формирование файла с отчетом

Для формирования файла с отчетом используйте функцию `ru.bitec.app.rpt.Rpt_Pkg#getReportStreamEx`:

```

/**
 * Выполняет построение отчета по системному имени отчета.
 * Если для версии отчета указано несколько доступных для печати форматов, то будет
↳ выдана ошибка построения.
 * Требуется явно указать формат построения, указав параметр [[Rpt_Pkg.
↳ ParamFormatType]]
 *
 * @param reportName      Имя отчета
 * @param reportVersionDate Дата
 * @param propertyMap      Карта входящих параметров
 * @return InputStream, содержащий результат построения отчета. ByteArrayInputStream
↳ не требует закрытия.
 */
def getReportStreamEx(reportName: String, reportVersionDate: Date, propertyMap:
↳ Map[String, Any]): Option[InputStream]

```

Функция работает в контексте автономной логики (REST-сервиса).

Внимание

- Если получен поток — его необходимо закрыть.
- При формировании файла с отчетом в потоке переменные выборки недоступны.

Формирование имени файла отчета

В настройках печатной формы (кнопка **Редактировать**) можно выбрать режим формирования имени файла отчета. Это определяет логику, по которой система будет генерировать названия файлов для данной печатной формы.

Системное имя: 12321123
Наименование: 123
Описание:
Модуль: Оперативное производственное планирование
Вести аудит построения: ☐
Выводить мастер настроек: ☐
Свободный отчет: ☐
Режим формирования имени файла:
Указывать дату и время создания в имени файла:
Процедура получения дополнительных параметров:
Функция пост-обработки отчета:
Не используется:
Дата окончания использования:
Версии печатной формы:
Подписи для печати:
Тип шаблона печатной формы:
Вес от:
Описание:
Объемный пакет:
Наименование:
Заголовок объекта:
Наименование ПО:
Наименование ПО + заголовок объекта:

Если режим не выбран явно, по умолчанию будет использоваться наименование самой печатной формы.

Профили печати

Профили печати позволяют группировать несколько печатных форм в единый отчет для выборок или классов. Это позволяет создавать комплексные документы, состоящие из разных разделов с индивидуальным форматированием в заданной последовательности.

Профили печати отображаются под операцией печати и позволяют выводить настроенные печатные формы для определенных выборок или классов данных.

Выборка: Gds_ArticleContrasAvi#ReportConformity
История запросов
Переоткрыть
Организация
Тип документа
Тест множественной интерактивной печати (профиль)

Тема	Описание	Папка	Дата доку...	Дата регистрации	Внешний номер	Внешняя дата	Тип
Тест передачи параметра с мультивыбором ...		Избранное - admin	13.10.2025				Wf
Документация по JEXL v5 Луценко А.В. ...			10.10.2025				Wf
TEST2 (10)			09.10.2025				Wf
Test			09.10.2025				Wf
Null 09_11_39_40			09.10.2025				Wf
Null 09_10_58_20		Расшифровка к акту услуг СНГ	09.10.2025				Wf
Null 09_10_20_16			09.10.2025				Wf
Test			08.10.2025				Wf
Test			08.10.2025				Wf
Null 08_16_22_11		Расшифровка к акту услуг СНГ	08.10.2025				Wf
Test			08.10.2025				Wf
serializeAttr			08.10.2025				Wf

Принцип работы

Профиль печати связывает с выборкой или классом одну или несколько печатных форм. Каждая форма может обрабатывать определенные параметры из выборки, что позволяет создавать разнотипные отчеты из одних данных.

Пример использования

В профиль можно добавить 3 разные печатные формы, работающие с разными атрибутами выборки. При печати создается единый отчет с разными разделами.

Создание профиля печати

1. Откройте раздел **Отчеты - Профили печати**.
2. Создайте новый профиль печати.
3. Добавьте нужные печатные формы.
4. Укажите классы и выборки, для которых будет доступен этот профиль.

Настройка печатных форм

Для каждого профиля можно настроить печатные формы, которые будут выводиться при выборе данного профиля в операции печати.

Для каждой формы можно указать:

- порядковый номер в очереди печати;
- количество копий;
- доступность формы.

При использовании профиля в отчет включаются только доступные печатные формы в заданном порядке, каждая печатается указанное количество раз.

Печатные формы	Классы	Выборки
Отчёт		
JasperReport Samples: Одноуровневый отчёт...	10	2 JasperReport Samples: Одноуровневый отчёт...
JasperReport Samples: Мастер-Деталь	20	2 JasperReport Samples: Мастер-Деталь

Отображение профиля печати

Профиль печати доступен для выборок или классов, указанных в его настройках.

Выборка	Отображение
Wms_TaskTypeAvi	List
Gds_ArticleContrasAvi	ReportConformity
Bs_ContrasAvi	MainList
Ddp_ReportByRKDTypeBlockAvi	ListObj

Системное имя	test
Наименование	test
Модуль	Печатные формы и отчеты
Не используется	☐
Дата окончания использования	

Печатные формы	Классы	Выборки

Класс	Активен
Контрагент	☒
Кодировка ТМЦ контрагента	☒
Отгрузка	☒
Тип задачи	☒
Документ графика поставки документации	☒

Получение дополнительных параметров печати

В печатной форме `Rpt_Report` есть возможность указать процедуру `Bts_Procedure` для получения дополнительных параметров печати (поле «Процедура получения дополнительных параметров»). Процедура получает на вход 3 параметра:

- `reportName` - имя отчета;
- `reportVersionDate` - дата версии отчета;
- `propertyMap` - изменяемая карта параметров, которые будут переданы в отчет.

Процедура не возвращает никаких значений. Ее предназначение - редактирование карты параметров `propertyMap`. Процедура может вызывать как функцию из прикладного кода, так и выполнять Jexl-функцию. Таким образом можно добавлять дополнительные параметры без изменения прикладного кода.

Пример функции, вызываемой из прикладного кода:

```
def getAddReportParamTest(reportName: NString, reportVersionDate: NDate, propertyMap:   
↳ mutable.HashMap[String, Any]): Unit = {  
    propertyMap.addOne("idBisObj", 15723.nl)  
}
```

Пример jexl-функции:

```
//создаем карту дополнительных параметров  
var addParamsMap = {"idBisObj", 15723L};  
//к переданной карте propertyMap добавляем карту доп.параметров, приведенную к scala   
↳ map  
propertyMap.addAll(asScala(addParamsMap));
```

Функция пост-обработки

Внимание

Работает с версии сервера 1.25.0-rc45

Функция пост-обработки отчета позволяет выполнить дополнительные действия с уже сформированным отчетом, такие как:

- Блокировка вывода отчета — при обнаружении критических ошибок отчет можно не показывать пользователю.

- Дополнение отчета — динамическое добавление информации (например, данные из внешних источников).
- Проверка корректности отчета — например, контроль наличия обязательных данных, сбалансированности сумм и соблюдения бизнес-правил.
- Взаимодействие с прикладным кодом — вызов бизнес-логики, логирование, интеграция с другими модулями системы.

Данная функция вызывается в момент окончания заполнения отчета, но до вывода его пользователю. Чтобы использовать функцию пост-обработки, необходимо в печатной форме `Rpt_Report` в поле «Функция пост-обработки отчета» указать имя нужной процедуры `Bts_Procedure`, при этом у указанной процедуры должен быть установлен тип — «Функция пост-обработки отчета».

Обработка ошибок

- **Критические ошибки:**
 - Блокируют вывод отчета.
 - Вызываются с помощью `raise(<текст ошибки>)`.
 - Останавливают выполнение пост-обработчика.
- **Некритические ошибки** (информационные)
 - Не блокируют вывод.
 - Вызываются с помощью `event.writeError()`.
 - Не останавливают выполнение пост-обработчика.
 - Выводят информационное сообщение в журнал печати.

Параметры процедуры

- `propertyMap` - карта параметров, по которой строился отчет;
- `event` - событие класса `JexlEndFillTemplateEvent`.

Описание класса `JexlEndFillTemplateEvent`

Событие, доступное из jexl-процедуры в пост-обработчике печатных форм.

Структура класса

- `fileName: String` - возвращает имя файла, с которым результат экспорта отчета будет передан клиенту.
- `inputStream: InputStream` - возвращает поток байтов с результатом экспорта заполненного шаблона в файл, который будет передан клиенту.
- `setInputStream(value: InputStream)` - устанавливает поток байтов с данными файла.
- `writeInfo(sInfo: NString)` - устанавливает значение в поле «Сообщение» журнала построения отчетных форм (`Rpt_EntityExec`). При повторном указании сообщения оно перезапишется.

- `writeError(sError: NString)` - устанавливает значение в поле «Текст ошибки» журнала построения отчетных форм (`Rpt_EntityExec`). При повторном указании сообщения оно перезапишется.

Пример scala функции

```
def postProcessing(paramMap: Map[String, Any], event: JexlEndFillTemplateEvent): Unit = {
    val img = session.sbtClassLoader.getResource("ru/bitec/app/rpt/imgForTest/meme.png")

    Using.Manager { using =>
        val imgBytes = using(img.openStream()).readAllBytes()

        val res = Btk_PDFUtilPkg().insertWithiText(
            pInpStream = event.inputStream
            , pImages = List((Btk_PDFUtilPkg.ImageInfo(name = "0",
                x = 300.nr,
                y = 300.nr,
                width = 100.nr,
                height = 100.nr,
                page = 1.nr,
            ), imgBytes))
            , bInAllPages = true
        )
        event.setInputStream(new ByteArrayInputStream(res.toByteArray))
    }.get
    event.writeInfo("Изображение успешно вставлено");
}
```

В примере в поток pdf отчета производится вставка изображения при помощи пакета `Btk_PDFUtilPkg`. Таким образом можно наложить водяной знак на все страницы отчета.

Пример проверки корректности построения отчета

```
def postProcessing(paramMap: Map[String, Any], event: JexlEndFillTemplateEvent): Unit = {
    Using(Loader.loadPDF(new RandomAccessReadBuffer(event.inputStream))) { pdfDoc =>
        val nPageCount = pdfDoc.getNumberOfPages
        if (nPageCount == 0) {
            throw AppException("Получен пустой отчет")
        } else {
            event.writeInfo(s"Количество страниц: $nPageCount")
        }
    }.get
}
```

В данном примере поток отчета парсится как pdf и проверяется наличие страниц у документа. Если оно = 0, то поднимается ошибка, иначе в поле «Информация» журнала построения записывается кол-во страниц

Типы отчетов

Произвольный

Произвольный отчет — это отчет, параметры которого берутся напрямую из существующей Печатной Формы (ПФ) с возможностью выбора только нужных параметров.

Как это работает?

В системе существует Печатная Форма (ПФ), содержащая множество параметров. При создании произвольного отчета можно выбрать только те параметры из ПФ, которые действительно необходимы для конкретной задачи.

Итоговая форма предзаполнения и параметры для фоновых операций при построении отчета зависят от выбранных параметров исходной ПФ.

Это позволяет создавать отчеты с произвольной структурой и нужными полями, используя только часть параметров из комплексной печатной формы.

Для чего используется?

- Построение отчетов с произвольной структурой на основе существующих ПФ.
- Упрощение сложных печатных форм путем выбора только релевантных параметров.
- Создание специализированных версий отчетов из универсальных ПФ.

Динамический

Динамический отчет — это отчет, параметры которого формируются через построитель запросов в интерфейсе системы, без необходимости программирования.

Для открытия построителя отчетов необходимо выбрать отчет с динамическим типом - **Редактировать - Объектные характеристики - Построитель запросов**.

Как это работает? Через интерфейс системы необходимо открыть построитель запросов и задать необходимые колонки отчета, доступные поля для заполнения, условия и параметры, по которым будет строиться отчет.

На предпечатный экран можно вывести нужные поля для заполнения и указать дополнительные условия. Система автоматически формирует итоговый отчет на основе заданной конфигурации.

Для чего используется?

- Настройка и изменение отчетов бизнес-пользователями через интерфейс.
- Быстрое построение типовых и уникальных отчетов без необходимости создания новых печатных форм и выборок.

Экранная форма

Экранная форма — это тип отчета, который позволяет отобразить данные системы в виде отчета. Можно отобразить любой другой отчет или таблицу из системы в заданном виде, без необходимости программирования.

Как это работает? Через интерфейс системы в Объектных характеристиках экранной формы необходимо указать выборку данных и тип ее отображения — например, таблица, карточка, древовидный список. После чего система напрямую отображает данную выборку (списки классов, логи, справочники и т.д.) в указанном виде.

Этот подход позволяет отображать любой элемент системы независимо от того, в каком модуле он находится и как называется в интерфейсе.

Для чего используется?

- Прямой вывод данных системы без промежуточных преобразований.
- Единый интерфейс для просмотра информации из разных модулей.

Отчет с селекционным экраном

Отчет с селекционным экраном — это отчет, где данный экран генерируется из параметров выборки, которую программист задает в коде.

Перед печатью данного отчета открывается сам *селекционный экран* — интерфейс для заполнения полей. На основе заполненных данных строится отчет.

Как это работает? Разработчик в коде создает выборку, где прописывает необходимые параметры. Эти параметры определяют структуру селекционного экрана и логику обработки данных.

Разработчик может закодировать что угодно: создать любой селекционный экран с любыми полями, а также добавить дополнительную логику для проверки, расчета или пересчета данных.

Для чего используется?

- Реализация сложной бизнес-логики проверок и расчетов.
- Разработка универсальных отчетов с максимальной гибкостью настройки.
- Автоматизация специализированных процессов формирования отчетности.

Выборка для селекционного экрана

Выборка для селекционного экрана указывается в **Настройка отчетных форм → Объектные характеристики** через поля:

- Имя выборки;
- Имя отображения.

Создание выборки

Выборку для селекционного экрана нужно унаследовать от базовой выборки `ru.bitec.app.rpt.Rpt_ScreenCardAbsAvi.BaseSelectionScreenCard`. Эта выборка по умолчанию содержит фильтр с отчетами отчетной формы, который определяет, какая печатная форма будет напечатана. Если этого фильтра нет или он не заполнен, будет напечатана печатная форма по умолчанию.

Валидация введенных значений

Для валидации значений полей нужно переопределить метод `validateFields`, который вызывается перед печатью.

Параметры выборки

Для передачи параметров в селекционный экран используется параметр `jParams#`, который содержит JSON следующего вида:

```
{
  "attribute_1_name": "attribute_1_value",
  "attribute_2_name": "attribute_2_value"
}
```

Карта параметров отчета

Для указания логики получения карты параметров отчета переопределите метод `getPropertyMap`. По умолчанию получает значения всех фильтров `StdLink` и стандартных фильтров. Для редактирования полученной карты параметров можно переопределить метод `preparePropertyMap`.

Селекционные экраны

Селекционные экраны используются для создания отчетов с динамической фильтрацией. Они предоставляют возможность пользователю задавать параметры для фильтрации данных перед генерацией отчета. Эти параметры используются для формирования условий, которые затем передаются в отчетную форму.

Селекционные экраны позволяют:

- Задавать фильтры для отчетов.
- Выбирать типы отчетов и их параметры.
- Автоматически формировать условия для выборки данных.

Как использовать селекционные экраны

Шаг 1: Наследование от базовой выборки

Для начала разработчику необходимо отнаследоваться от базовой выборки `Rpt_ScreenCardAbsAvi#BaseSelectionScreenCard`.

Шаг 2: Переопределение методов `setReportName` и `setPropertyMap`

После создания наследника, необходимо переопределить два метода:

- `setReportName` — определяет системное имя отчета, которое будет использоваться при генерации.
- `setPropertyMap` — в этом методе заполняется логика формирования результирующей карты параметров (`reportParamsResultMap`), которая будет передаваться в метод печати.

Пример:

```
override def setReportName: String = "MyCustomReport"

override def setPropertyMap: Unit = {
  fltMacroGeneratorPkg.updateResultMap(
    fltMacroGeneratorPkg.getAllFltValuesMap,
    {
      case (key, value) =>
        val newKey = key.stripPrefix("flt_").toUpperCase
        newKey -> value
    })
  fltMacroGeneratorPkg.updateResultMap(Map("WHERECOND" -> fltMacroGeneratorPkg.
    generateReportMacro()))
}
```

Шаг 3: Работа с результирующей картой параметров

В методе `setPropertyMap` разработчик заполняет результирующую карту `reportParamsResultMap`, которая передается как параметры в метод `printReport`. Для помощи в формировании этой карты используется пакет `ru.bitec.app.btk.FltMacroGeneratorPkg`, который содержит необходимые методы для работы с фильтрами и формирования карт значений.

Пример использования пакета `fltMacroGeneratorPkg` (см. выше):

Шаг 4: Дополнительные возможности — выбор печатной формы

Разработчик может позволить пользователю самостоятельно выбирать нужный отчет. Для этого необходимо вставить фильтр для выбора отчета в свой `.avm`-файл:

```
<condition id="idReport">
  <filterAttr name="flt_idReport" attribute-type="Long" caption="Отчет" order="10"
  isVisible="false"/>
  <filterAttr name="flt_idReportHL" attribute-type="Varchar" caption="Отчет" order="10.
  1">
    <card isControlWidthFixed="true" controlWidth="60"/>
    <editor>
      <lookup changeableAttr="flt_idReport" lookupQuery="gtk-Rpt_EntityAvi#Lookup_
      SelReport"
      lookupListAttr="sHeadLine" lookupKeyAttr="id" isResetButtonVisible=
      "true"/>
    </editor>
  </filterAttr>
</condition>
```

(продолжается на следующей странице)

```
</filterAttr>
</condition>
```

Шаг 5: Метод печати отчета

Метод `printReport` отвечает за формирование и печать отчета. Он вызывает метод генерации отчета с передачей системного имени отчета и параметров, сформированных в методе `setPropertyMap`.

Интерактивные отчёты

Интерактивный режим отчёта представляет собой предпросмотр с возможностью редактирования полей, заранее определённых разработчиком отчёта.

Изменённые пользователем данные либо передаются как параметры в отчёт, либо применяются к объектам, на основе которых строится отчёт. Это настраивается отдельно для каждого поля. Интерактивный режим есть только у Jasper-отчётов.

Интерактивные отчёты позволяют пользователям удобно настраивать параметры отчёта через интерактивную форму перед печатью.

Основные возможности

- Редактирование параметров отчёта через интерактивную форму.
 - Расширение логики сеттеров с помощью jexl-скриптов.
- Пользовательские Jexl-операции для интерактивных отчётов:
 - Указание условий доступности операций.
 - Использование данных отчёта в операциях.
- Настройка пользовательской выборки для переопределения бизнес-логики.

Объявление отчёта интерактивным

Для активации интерактивного режима в интерфейсе системы:

- Перейдите в **Отчёты > Настройка отчётных форм**.
- Найдите нужную отчётную форму.
- На детализации **Печатные формы отчёта** поставьте галочку «Интерактивный».

Отчёт определяется как интерактивный с помощью атрибута `bInteractive`.

Примечание

Построение интерактивных отчётов в фоновом режиме невозможно.

Настройка интерактивного отчёта

Предпочтительный способ

Создайте поле типа `TextField` и установите параметр `net.sf.jasperreports.export.html.class` со значением по шаблону:

```
"[#" + Гид объекта + "#" + Наименование редактора + "#" + Наименование параметра/Название атрибута изменяемого объекта + "#" + Значение, которым будет инициализирован редактор + "/"#]"
```

Компоненты шаблона:

- **Гид объекта:** Гид изменяемого объекта, если указан атрибут для изменения.
- **Наименование редактора:** Тип редактора.
- **Наименование параметра/атрибута:**
 - Если строка начинается с «P_», то значение считывается как наименование параметра, который передаётся в печатную форму. В печатной форме его можно получить из карты параметров `REPORT_PARAMETERS_MAP`.
 - В остальных случаях значение интерпретируется как название атрибута объекта, гид которого был указан ранее, и при запуске операции печати в атрибут объекта устанавливается введенное значение.
- **Значение инициализации:** Начальное значение редактора.

Примечание

Обязательное требование: В `TextField` должно находиться какое-то значение (хотя бы пробел).

Преимущества предпочтительного способа:

- Проще в реализации и поддержке.
- Меньше вероятность ошибок позиционирования.
- Более чистая структура отчёта.

Типы редакторов

- `textArea` — текстовый редактор (многострочный текст);
- `checkBox` — флаговая кнопка (булево значение);
- `date` — редактор даты;
- `dateTime` — редактор даты и времени;
- `ref` — ссылочный редактор (выбор объекта).

Примечание

Примечание: в именах редакторов учитывается регистр — используйте точные названия как в примерах.

Альтернативный (устаревший) способ настройки

Используйте только для совместимости со старыми отчётами:

1. Создайте два поля в одном фрейме, дублирующих размер и положение.
2. Первое поле — выводит финальное значение из параметра/объекта.
3. Второе поле — содержит строку интерактивности с условием видимости: `$P{FILENAME}.equals(".html")`.
4. Для второго поля установите `TextAdjust = ScaleFont`.
5. Оба поля должны содержать значение (хотя бы пробел).

Изменение параметров интерактивного отчёта

Чтобы сделать поле изменяемым в выборке интерактивного отчёта, оно должно содержать строку в следующем формате: `[#{gid}#{тип_редактора}#{имя}#{значение}/#]` (вводится в одно из полей в JasperSoft Studio).

Если указан `gid`-объекта и имя не начинается с `P_`, то при изменении значения параметра в выборке будет вызван стандартный сеттер `ru.bitec.app.btk.class_.attr.AttrApi#setAttrValue`.

Чтобы получить введённые пользователем параметры в Jasper Reports, нужно добавить новый параметр отчёта в списке `Parameters` или получить его из `REPORT_PARAMETERS_MAP` типа `java.util.Map`, при чём имя в поле должно совпадать с именем в списке параметров.

Примечание

- Если имя атрибута начинается с `P_`, карта параметров обновляется, минуя сеттер по `gid`.
- В JasperSoft Studio все параметры указываются в **верхнем регистре**.

Для примера, нескольких параметров:

- `IDSRCOBJ` - вызывающий сеттер по gidу параметр:
 - `$P{IDSRCOBJ}` - получение значения изменённого пользователем параметра.
 - `"[#" + $P{GIDOBJ} + "#textArea#IDSRCOBJEDITED#" + $P{IDSCROBJ} + "/#]"` - содержание редактируемого поля в JasperSoft Studio.
 - * `GIDOBJ` - параметр отчёта, содержащий `gid` объекта с параметром `IDSRCOBJ`, в который будет устанавливаться его значение при изменении поля интерактивного отчёта.
 - * `IDSCROBJ` - параметр отчёта, содержащий изначальное значение изменяемого поля интерактивного отчёта.
 - * `IDSRCOBJEDITED` - имя, по которому можно обратиться к изменённому пользователем в интерактивном отчёте параметру.
- `$P{P_NTOTAL}` - не вызывающий сеттер параметр.
- `(java.util.Date) $P{REPORT_PARAMETERS_MAP}.get("P_DDATE")` - получение параметра `P_DDATE` и приведение его к нужному типу `java.util.Date`.

Функция парсинга изменяемых полей

Изменяемые поля из полученной от Jasper Reports HTML-разметки парсятся в функции `ru.bitesc.app.rpt.interactive.Rpt_HtmlReportHelper#replaceTags`.

Расширение сеттеров

Можно задать кастомную логику для сеттера параметра отчёта `setByTagId` с помощью JEXL-скриптов. Скрипты выполняются до вызова сеттера `setByTagId`.

В контексте JEXL-скрипта доступны параметры атрибута:

- `gid` - глобальный идентификатор объекта;
- `sAttrName` - имя атрибута.

Пользовательские JEXL-операции

Операции могут быть добавлены на тулбар формы интерактивного отчёта через детализацию Процедуры интерактивного отчёта в Отчёты > Настройка отчётных форм. Это происходит с помощью клонируемой операции `userDefinedProcedures`. Настройка привязывается к `Rpt_EntityReport` и содержит:

- № п/п (порядок отображения).
- Иконка (номер иконки).
- Наименование.
- Признак «Выводить текст операции на панели».
- Ссылка на `Bts_Procedure`, в которой указан JEXL-скрипт (исполняется в контексте выборки).

Контекст операций

В настраиваемых операциях интерактивных отчётов доступны:

- **Значения изменяемых полей** - по имени параметра в селекционном экране \ параметра произвольного отчёта.
- **Значения неизменяемых полей Jasper-отчёта** - для полей, отмеченных специальными тегами.
- **Список всех имён доступных параметров** - по алиасу `PARAMS`.
- **Методы выборки и сессии** - `dialogs`, `commit()`, `flush()` и др.

Получение неизменяемых параметров

Чтобы получить неизменяемые параметры из Jasper-отчёта и использовать их в пользовательских операциях, необходимо задать следующие свойства для поля в Jasper Reports:

- `net.sf.jasperreports.export.html.class = parse` - обозначает, что поле будет доступно в jexl-операциях.
- `net.sf.jasperreports.export.html.id = {имя атрибута}` - имя, по которому можно будет получить значение поля.

Условия отображения

Условия доступности операций задаются в детализации **Условие доступности процедуры**, находящейся в форме **Настройка отчётных форм**, в детализации **Процедуры интерактивного отчёта**.

Условия указываются как JEXL-скрипт, который должен возвращать:

- `true` - операция активна;
- `false` - операция недоступна.

Скрипт может использовать JSON с параметрами из селекционного экрана или свободного отчёта.

Переопределение выборки интерактивного отчёта

Для интерактивных отчётов предусмотрена возможность указания пользовательской выборки для переопределения бизнес-логики.

Примечание

В зависимости от типа объекта `Rpt_EntityReport`, для которого вы хотите переопределить выборку и отображения интерактивного отчёта, нужно наследоваться от:

- Отчёт с селекционным экраном: `InteractiveSelReportHtml`.
- Произвольный отчёт: `InteractiveFreeReportHtml`.

1. Указание кастомной выборки:

- В классе `Rpt_EntityHtmlReportPkg` реализован метод `hasCustomInteractiveSelection`, который проверяет наличие кастомной выборки для интерактивного отчёта.
- Если кастомная выборка указана, она открывается с переданными параметрами.

2. Пример использования:

- Для задания кастомной выборки необходимо указать её имя и представление в объектных характеристиках `sSelectionInteractive` и `sRepresentationInteractive` класса `Rpt_EntityReport`.
- При вызове метода `enterInteractiveMode` кастомная выборка будет автоматически открыта, если она указана.
- Параметры в выборку передаются через `selection.form.params` и сохраняются в JSON.

Параметры для передачи в отображение интерактивного отчёта

Отчёт с селекционным экраном

Обязательные:

- IDENTITY# - идентификатор отчёта;
- IDREPORT# - идентификатор печатной формы;
- BWITHDOC# - флаг: сохранение в документообороте/истории печати;
- DREPORTVERSION# - дата версии печатной формы.

Параметры для построения и печати отчёта:

- JPARAMETERS - параметры отчёта для истории (наименования как в селекционном экране);
- LOGJPARAMS# - параметры отчёта для печати (наименования как в Jaspersoft Studio).

Произвольный отчёт

Обязательные:

- IDENTITY# — идентификатор отчёта;
- IDREPORT# — идентификатор печатной формы.

Параметры для построения и печати отчёта:

- JPARAMETERS# — параметры отчёта для печати (наименования как в Jaspersoft Studio).

Ошибки отображения интерактивного отчёта

Внимание

При отображении особенно крупных интерактивных отчётов, браузерная вкладка может быть принудительно завершена из-за нехватки памяти.

22.08.2025 проведены испытания производительности. Результаты испытаний отображены на графиках ниже.

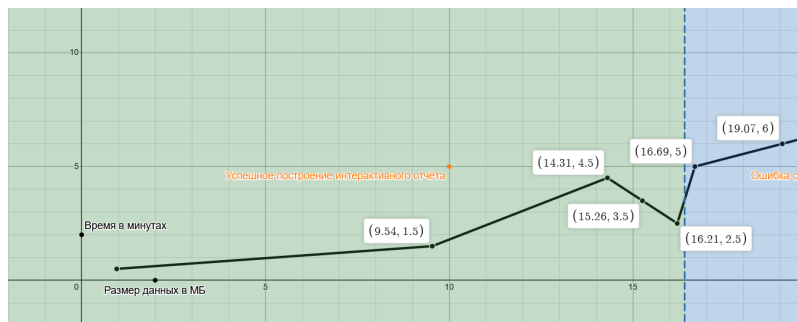


График ошибок отображения интерактивного отчёта:

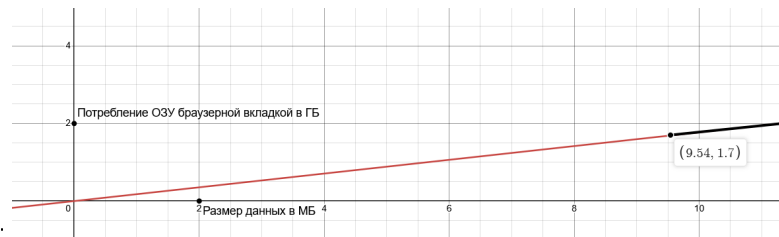


График зависимости размера отчёта и памяти вкладки:

Система, на которой проводились испытания:

- Выпуск: Windows 11 Pro.
- Версия: 24H2.
- Процессор: 13th Gen Intel(R) Core(TM) i9-13900K (3.00 GHz).
- Видеоадаптер: Intel(R) UHD Graphics 770.
- Тип системы: 64-разрядная.
- Браузер: Яндекс 25.8.0.1908 (64-bit).

Испытания проводились при объёмах оперативной памяти: **6, 8, 16 и 64 GB**.

Смещение критических точек (границ зон) в зависимости от объёма ОЗУ **не зафиксировано**.

Левая граница определена с точностью **0,47 MB**.

Правая граница определена с точностью **2,38 MB**.

SQL запрос, который использовался для генерации данных и настройки размерности отчёта:

```
SELECT lpad(gs::text, CAST($P{MULTIPLY} AS int), '0') AS uniq_str
FROM generate_series(1, $P{ROWCOUNT}) gs;
```

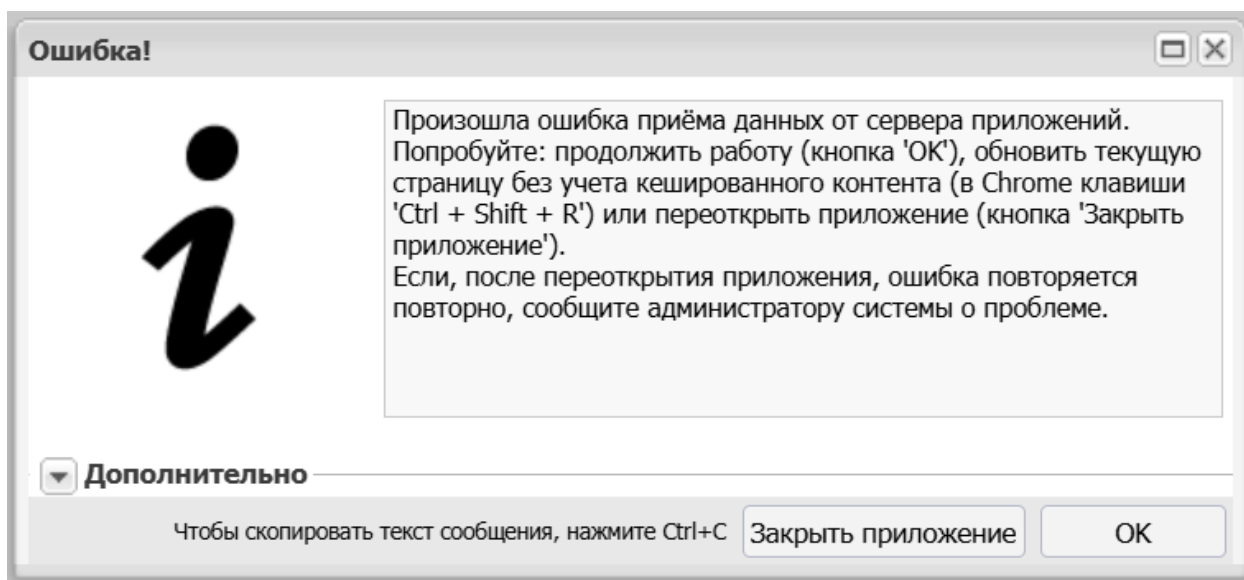
Примечание

При объёме отчёта в 16.21 MB, браузерная вкладка потрeбует 2,9 GB ОЗУ, а сам браузер более 3 GB. Это максимальное зафиксированное значение. Объём памяти, используемый в момент критических ошибок, определить не удалось.

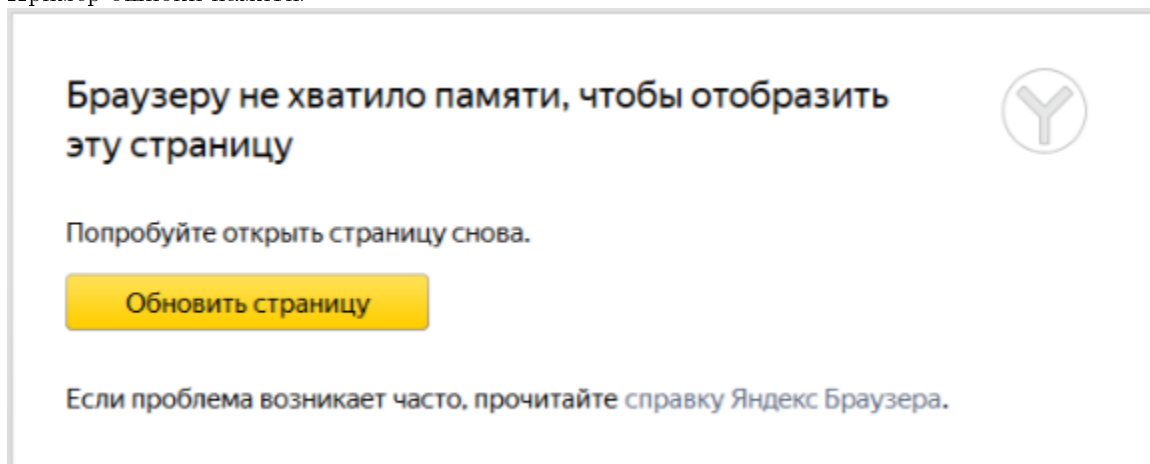
Примечание

- 16,21 MB = 17 миллионов символов (1 символ = 1 байт).
- Это соответствует 170 тысячам строк текста по 100 символов.
- Для сравнения: все 4 тома «Война и мир» содержат около 3 миллионов символов.

Пример ошибки сети:



Пример ошибки памяти:



Jasper Reports

Java-библиотека для построения печатных отчетов. Позволяет строить:

- Готовые к печати PDF-файлы в интерактивном динамическом HTML с навигацией внутри или за пределами отчета.
- Высококачественные документы PowerPoint, RTF, Word.
- Электронные таблицы или необработанные CSV, JSON или XML.

Примечание

Данный раздел не является полным руководством к средству построения отчётов JasperReport. Документ содержит описания базовых принципов создания шаблонов и способов взаимодействия инфраструктуры Global 3 SE с строителем отчётов JasperReports.

Дистрибутив

Скачать дистрибутив можно с ftp по адресу <ftp://ftp.bitec.ru/pub/%23Distrib/JasperStudio/>.

Для разработки шаблонов отчётов доступны две IDE:

- Jaspersoft® Studio;
- iReport Designer.

Примечание

Данный раздел касается только Jaspersoft Studio.

Jaspersoft Studio

Программное обеспечение для создания и редактирования шаблонов JasperReports. С помощью Jaspersoft Studio возможно:

- разрабатывать и запускать шаблоны отчетов;
- создавать запросы к отчетам;
- писать сложные выражения;
- компоновать визуальные компоненты.

Внимание

Перечень шрифтов в дизайнера Jaspersoft Studio может отличаться от перечня шрифтов на сервере приложения. Используйте стандартные шрифты, к примеру Arial.

Внимание, быстродействие отчета сильно зависит от шрифтов. Если в вашем отчете выводятся данные со множеством страниц, например, отчет с тысячами строк, то рекомендуется разрабатывать его на шрифте Arial, после отладки менять шрифты на нужные и обратить внимание на изменения по быстродействию формирования отчета.

Пример:

- отчет с 100 тысячами строк на Arial: 3 минуты;
- на Times New Roman: 20 минут.

Обучающее видео

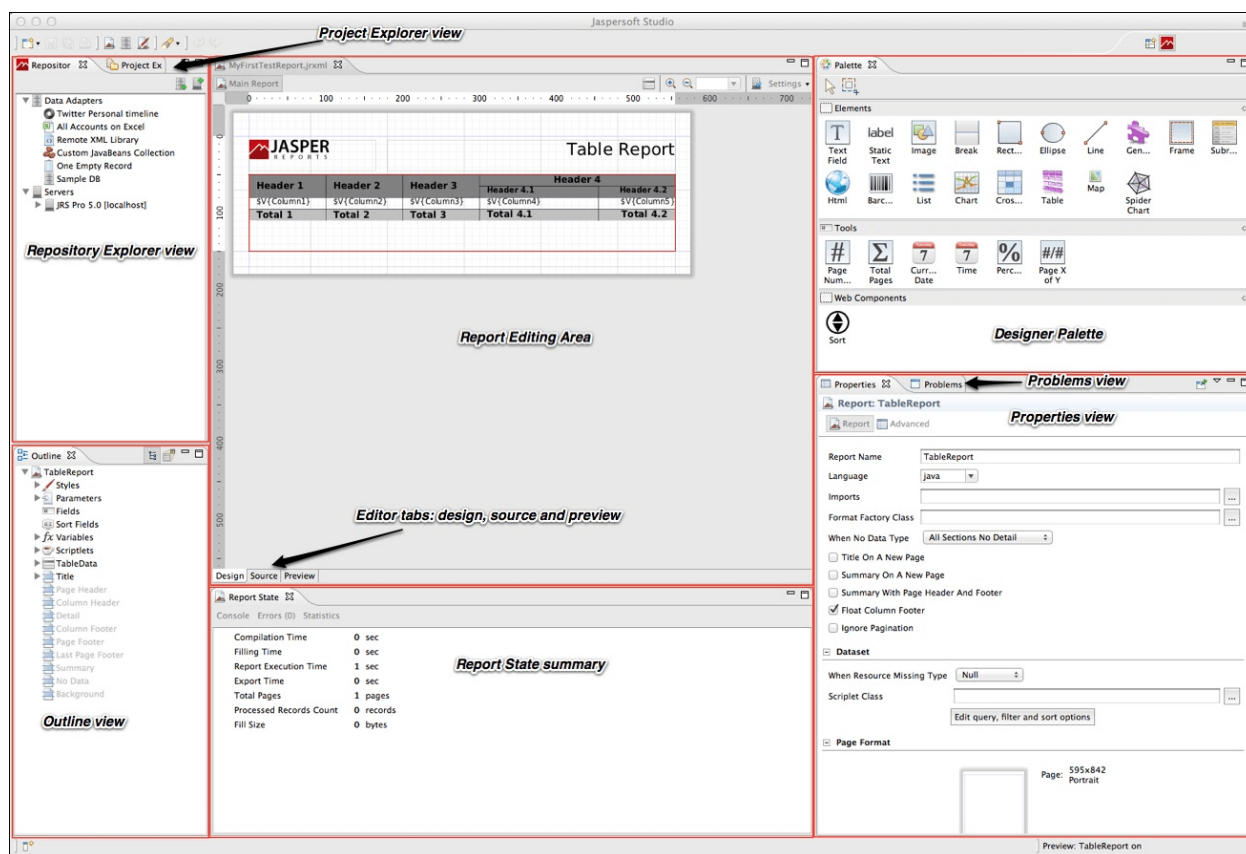
Видео выложено на Ftp по адресу [ftp://ftp.bitec.ru/pub/#Global/Video/Обучение/Создание ПФ отчетов в Jasper](ftp://ftp.bitec.ru/pub/#Global/Video/Обучение/Создание_ПФ_отчетов_в_Jasper).

Документация JasperReports

- Ресурсы с jaspersoft;
- jaspersoft Wiki;
- Руководство пользователя по адресу <ftp://ftp.bitec.ru/pub/%23Distrib/JasperStudio/>.

Чтобы открыть примеры Jaspersoft Studio выполните **File > New > Project > JasperReports Samples**. В выбранном рабочем каталоге будет создано множество примеров отчётов.

Пользовательский интерфейс



Repository

Содержит перечень подключений к разным базам данных. Подключения к базе данных необходимо настраивать для того чтобы была возможность тестировать отчеты из Jaspersoft Studio.

Outline

Отображает структуру открытого отчета.

Properties

Отображает свойства элемента выбранного в **outline**.

Editor

Отображает открытый отчет. Содержит следующие закладки:

- Design;
- Source;
- Preview.

Колонки

Страницы отчета содержат банды (группы визуальных элементов), независимые от данных (например, заголовки или нижние колонтитулы страницы), и банды, которые печатаются только при наличии одной или нескольких записей из данных для печати.

Банды с данными можно разделить на вертикальные столбцы, чтобы максимально использовать доступное пространство на странице. Количество колонок можно задать в диалоге разметки страницы(`properties\reports\page format` для `outline\{Report}`).

Bands

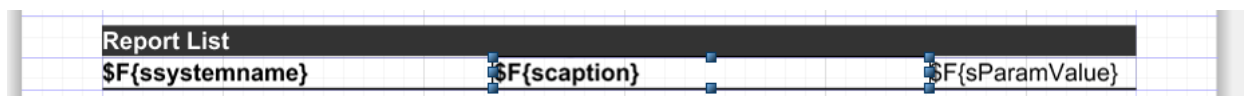
Отчет делится на банды, в которые добавляются визуальные элементы. Список основных предопределенных бандов:

- **Title:** Это первый видимый банд. Он создается только один раз и может быть распечатан на отдельной странице.
- **Page Header:** Позволяет определить заголовок страницы.
- **Column Header:** Печатается перед началом отрисовки колонки.
- **Column Footer:** Печатается в конце каждой колонки.
- **Page Footer:** Появляется в конце каждой страницы.
- **Summary:** Содержит итоги по отчету.

Detail band

Печатается для каждой записи данных отчета. Детальных бандов может быть несколько, поэтому по умолчанию создается детальный банд с именем „detail 1“.

В детальной части можно расположить «Text Field» и задать для каждого выражения получения данных.

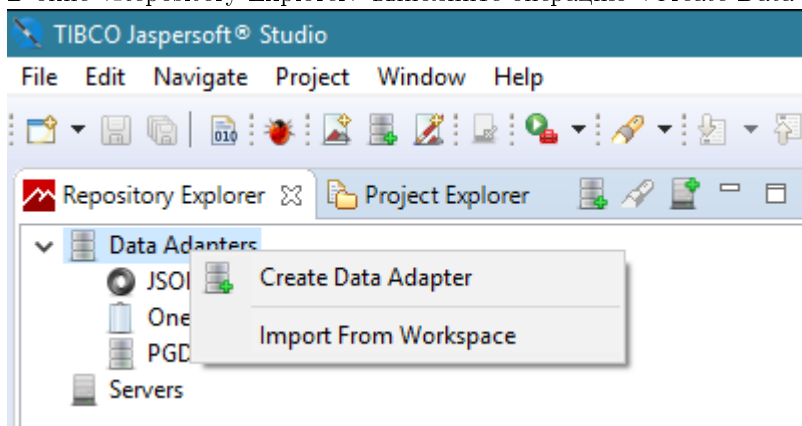


Настройка среды Jaspersoft Studio

Запустите среду разработки отчетов Jaspersort Studio

Создание подключения к ДБ

1. В окне «Repository Explorer» выполните операцию «Create Data Adapter».



2. В открывшемся диалоге выберите «Data base JDBC connectivity».
3. Укажите реквизиты подключения к базе Postgres:
 - name=PGDEV;
 - JDBC driver=org.postgresql.Driver;
 - JDBC url=jdbc:postgresql://{host}/{db};
 - Username ={username}.

Примечание

Подключение к БД необходимо для того, чтобы тестировать отчет из студии.

Проект отчета

Проект отчета создается на группу печатных форм, на усмотрение аналитика.

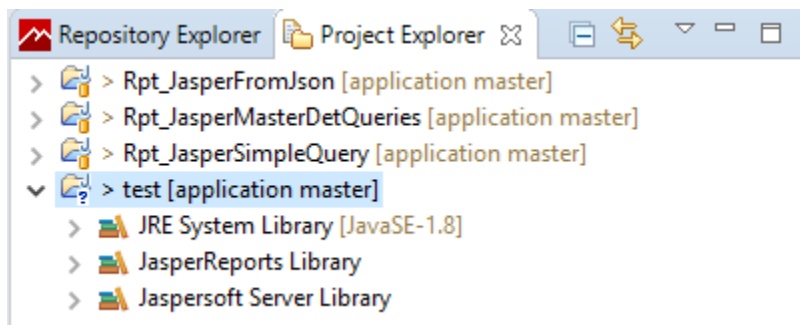
Совет

Печатные формы проще создавать и редактировать в одном проекте, если они не содержат скриптов.

Для создания проекта отчёта:

1. Выполните **File > New > Project**
2. Выберите **Jaspersoft Studio / JasperReports Project**.
3. Задайте имя отчёта в следующем диалоге.

По завершении на закладке **Project Explorer** появится новый проект, пока ещё не содержащий шаблонов отчёта.



Создание простой печатной формы

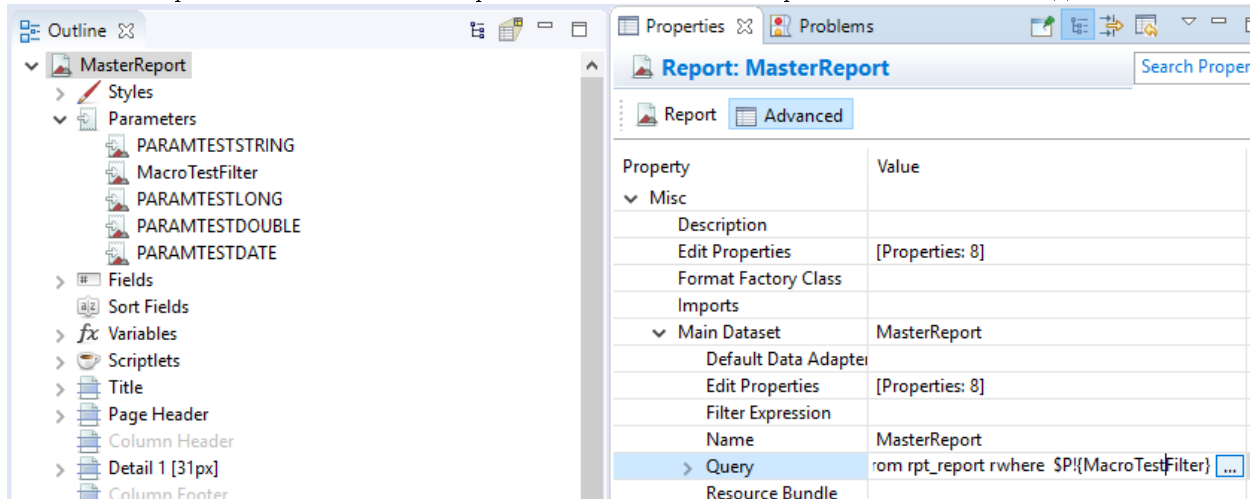
Простейший отчёт, выводящий список на основе SQL-запроса.

1. В окне **Project Explorer** выберите требуемый проект.
2. Создайте каталог для печатной формы. В контекстном меню выполните пункт **New > folder**.
3. Создайте отчет. В контекстном меню выполните пункт **New > Jasper Report**.
4. Задайте имя шаблона отчета **main.jrxml**. Файл с этим именем будет искаться при построении отчёта из Global 3 SE.
5. Нажмите кнопку **next**.
6. Выберите адаптер базы данных.
7. Нажмите кнопку **next**.
8. Введите запрос.
9. Нажмите кнопку **next**.
10. Добавьте требуемые поля в соответствующие банды.
11. Нажмите кнопку **finish**.
12. Проверьте корректность отчета на закладке **Preview** в окне **Editor**.
13. *Опубликуйте отчет в БД.*

Редактирование запроса с данными:

1. Откройте отчет.
2. В окне outlet выберите корень.
3. Нажмите кнопку **Edit query, filter and sort options**. Кнопка находится в окне Properties по адресу **Report \ DataSet**.

Также запрос можно посмотреть в окне Properties на закладке **Advanced**



Двухуровневый отчёт (мастер-деталь)

Документация в wiki

1. Создайте файл мастер-шаблона **main.jrxml** (имя обязательно). Аналогично одноуровневому шаблону.
2. Создайте файл детального-шаблона **subreport.jrxml**. В данном шаблоне будут отображаться детальные данные. Имя может быть любым.
3. Добавьте в мастер **Subreport**. Откройте мастер шаблон и в окне outlet в секцию **Detail 1** добавьте элемент **Subreport**, в свойствах которого укажите:
 - Имя файла детального шаблона. **Expression="subreport.jrxml"**.
 - Подключение к БД. **Connection Expression = \$P{REPORT_CONNECTION}**.
4. Свяжите запросы двух шаблонов параметрами:
 1. Создайте в детальном шаблоне параметр **ID_MASTER**.
 2. Параметризируйте детальный запрос. Обращение к параметру в детальном запросе **\$P{ID_MASTER}**.
 3. Свяжите параметр детального шаблона с мастером:
 1. Зайдите в раздел «Subreport» свойств мастер-шаблона.
 2. Вызовите диалог редактирования свойства «Edit Parameters».
 3. Задайте связь. Для этого создайте параметр **ID_MASTER** с выражением **\$F{id}**.

Совет

Для вывода актуальных данных в предпросмотре мастер отчета, после редактирования детального отчета, следует вызвать перекомпиляцию.

Отчет книга (Report Book)

Отчет книга – это .jrxml, который объединяет несколько отчетов в единый объект.

[Документация в wiki](#)

1. Создайте новый отчет-книгу:
 1. Кликните правой кнопкой мыши в Project Explorer и выполните пункт New > Jasper Report.
 2. В новом мастере создания отчета выберите Report Books \ Empty Book.
 3. Укажите имя main.jrxml (имя обязательно) .
2. Создайте отчет, который будет отображаться в книге. Отчет может быть как одноуровневым, так и двухуровневым.
3. В отчете-мастере вызовите контекстное меню и выполните пункт «Add new book part».
4. Добавьте отчет, созданный в пункте 2.
5. Свяжите отчеты между собой. Связь запросов двух отчетов параметрами происходит аналогично двухуровневому отчету.

Внимание

Отображение номеров страниц в отчетах, отображаемых в книге осуществляется через переменную `$V{MASTER_CURRENT_PAGE}`, а общее количество страниц – через `$V{MASTER_TOTAL_PAGES}`. В настройках текстового поля должно быть указано `evaluationTime="Master"`.

Параметризация шаблонов

[Документация в wiki](#)

Параметры отчета находятся в окне Outline по адресу {Report} \ parameters. Параметры отчёта используются для следующих целей:

- В SQL-запросах. Для ограничения данных по переданным извне параметрам.
- Связи между частями шаблонов.
- Для связи между движком отчёта и внешним окружением.

Параметры могут быть любого java-типа.

Например:

- REPORT_CONNECTION тип `java.sql.Connection`;
- REPORT_PARAMETERS_MAP тип `Map<String, Object>`.

Шаблон отчёта содержит множество служебных параметров. В дереве элементов шаблона они отображаются серыми. Их невозможно удалить, или изменить значение из дизайнера. Их значения могут быть переданы извне или изменены в коде.

Параметр имеет значение по умолчанию. Его удобно использовать при разработке и тестировании отчёта, заменяя передачу значения параметра извне.

Внимание

При задании строкового значения по умолчанию, его необходимо брать в двойные кавычки.

Флаг `Is For Prompting` означает, что при тестовом построении отчёта из среды Jaspersoft Studio будет запрошено значение параметра.

Внимание

Создавайте параметры с именами в верхнем регистре. Это связано с тем, что имена параметров, переданных извне, приводятся к верхнему регистру автоматом.

Передача параметров в отчёт

В отчёт можно передать значение параметра с любым типом данных, в том числе служебных параметров. Также, через параметры можно передать значения макросов фильтра, сортировки или весь запрос целиком.

```
@Oper(
  caption = "Печать",
  description = "Печать",
  imageIndex = 17,
  order = 100
)
def printReport(): Unit = {
  createReportEx(
    getVar("sSystemName").asString(),
    null,
    PostBuildAction.print,
    Map[String, Any](
      "ParamTestLong" -> 100500.nl,
      "ParamTestDouble" -> NNumber.fromAny(100500.100500D),
      "ParamTestString" -> "Тестовая строка",
      "ParamTestDate" -> NDate.now(),
      "MacroTestFilter" -> "(1 = 1)"
    )
  )
}
```

Карта значений всех переданных извне параметров доступна в служебном параметре `REPORT_PARAMETERS_MAP`.

Значение можно получить выражением:

```
$P{REPORT_PARAMETERS_MAP}.get("ИМЯ_ПАРАМЕТРА")
```

Если в шаблоне создать параметр MY_PARAMETER, имя которого в верхнем регистре равно имени переданного в отчёт параметра, значение MY_PARAMETER будет проинициализировано переданным значением. Это позволяет использовать переданные значения параметров в SQL-запросах.

Пример использования, в отчёте: Rpt_JasperSimpleQuery

Использование параметров в SQL-запросах

Подстановка значения

Подстановка значения параметра в запрос выполняется выражением \$P{ИМЯ _ПАРАМЕТРА}. Следите за регистром имён параметров, в случае несовпадения будет ошибка компиляции.

```
select o.*, $P{MY_ORDER_ID} as idMy
  from orders o
 where order_id = $P{MY_ORDER_ID}
```

В данном случае будет выполнен sql запрос:

```
select o.*, ? as idMy from orders o where order_id = ?
```

Значение параметра отчёта будет передано как значение sql-параметра.

Подстановка выражения (макроса)

Для подстановки в sql-запрос выражения (или полного текста sql-запроса) необходимо использовать синтаксис \$P!{ИМЯ _ПАРАМЕТРА}.

```
select * from orders order by $P!{ИМЯ _ПАРАМЕТРА}
select * from orders where $P!{ИМЯ _ПАРАМЕТРА}
$P!{MY_QUERY}
```

Использование параметров, содержащих список значений

В отчет можно передавать массив значений в одном параметре. Для этого в REPORT_PARAMETERS_MAP в значении параметра нужно передать массив `Array[java тип]()`.

Например:

```
"TEST" -> Array[Long](23553,23552)
```

Настройка параметра с множеством в Jasper Studio

В Jasper Studio нужно выбрать тип для параметра, которому будет передаваться массив значений:

1. Перейдите к параметрам отчета.
2. Откройте свойства параметра.
3. Задайте «Class» значением `ArrayList`.
4. Задайте «Nested type name». К примеру `Long` или `String`.

Использование параметров с null

Для использования в SQL параметров которые могут быть принимать значение null необходимо использовать синтаксис `$X{ИМЯ_ФУНКЦИИ, ИМЯ_КОЛОНКИ, ИМЯ_ПАРАМЕТРА}`.

Где ИМЯ_ФУНКЦИИ

- `EQUAL`;
- `NOTEQUAL`;
- `LESS`;
- `GREATER`;
- и др.

Для запроса

```
select * from orders where $X{EQUAL, num_column, num_param}
```

Будут сгенерирован текст:

```
select * from orders where num_column = 1
```

Или если параметр null

```
select * from orders where num_column IS null
```

Управляющие параметры

На процесс построения и экспорта отчёта могут влиять:

- служебные параметры отчёта;
- свойства контекста построения отчёта.

Служебные параметры отчёта.

Имя	Возможные значения	Описание
REPORT_LOCAI	Locale.forLanguageTag(«ru-RU»)	Устанавливает локаль, используемую при построении отчёта

Свойства контекста построения отчёта.

Имя свойства контекста построения отчета начинается с `net.sf.jasperreports` Полный список можно найти в документации [JasperReports](#)

Значения по умолчанию определены в ресурсах JasperReports:

```
jasperreports-x.x.x.jar\default.jasperreports.properties
```

Значения могут быть переданы через параметры метода `createReportEx()` или определены с свойствах шаблона («Properties \ Advanced \ Misc \ Edit Properties»)

Дополнительные управляющие параметры

Параметры, передающиеся в отчет стандартной операцией печати:

- `idSrcObject` - id объекта от которого создан отчет;
- `idSrcClass` - id класса объекта от которого создан отчет;
- `idUser` - id пользователя;
- `signData_dz` - json, подписи печатных бланков для отчетов. Пример значения:

```
[
  {
    "sFIO": "Калинин В.М.",
    "sPosition": "Транспортировщик",
    "idEmployee": 25712,
    "idDepartment": 101672,
    "sBasisDocument": " №036/426-06-2020-ОЛПС от 01.01.2020",
    "idBlankSignType": 95351,
    "gidBasisDocument": "139800/5433"
  }
]
```


Экспорт

По умолчанию, отчёт экспортируется в pdf. Формат, в который будет выполнен экспорт, определяется параметром `FILENAME`.

На экспорт влияют *параметры контекста построения*, имена которых начинаются с `net.sf.jasperreports.export`.

Описание параметров:

- **FILENAME**

Определяет имя и формат файла с результатом построения отчёта. Отчёт может быть экспортирован в следующие форматы: pdf, xls, docx, xlsx, pptx, xml, json, html, ods, odt, rtf, txt, jpg, jpeg, gif, png.

По умолчанию, экспортируется в pdf. Возможные значения с AS 1.18: \

`{name} | {name.} | {name.ext} | {.ext}`

Возможные значения до AS 1.17: \

`{name.ext} | {.ext} | {ext}`

Растровые изображения

Дополнительные параметры контекста, отсутствующие в списке стандартных свойств контекста JasperReports.

- `net.sf.jasperreports.export.graphics2d.page.index` тип `Integer` - номер экспортируемой страницы.
- `net.sf.jasperreports.export.graphics2d.zoom.ratio` тип `Float` - коэффициент увеличения. По умолчанию, страница формата A4 имеет размеры 842x595px, 72dpi. При экспорте в растровое изображение с такими параметрами, читаемость текста крайне низка. Параметр позволяет пропорционально увеличить размеры изображения без потери качества.
- `net.sf.jasperreports.export.graphics2d.offset.x` тип `Integer` - сдвиг по оси X.
- `net.sf.jasperreports.export.graphics2d.offset.y` тип `Integer` - сдвиг по оси Y.

GIF и PNG с прозрачностью

Для экспорта страницы отчёта в gif или png с прозрачным фоном необходимо отключить заливку фона белым цветом. Параметр `net.sf.jasperreports.export.graphics2d.white.page.background` тип `boolean` управляет заливкой фона страниц отчёта белым цветом.

Построение отчёта на основе JSON-данных

[Документация в wiki](#)

Для построения отчёта на основе JSON-данных:

1. Создайте xxx.json файл содержащий данные в json формате. Файл должен находиться в каталоге с шаблоном.

Пример содержимого:

```
{
  "Northwind": {
    "Customers": [
      {
        "Phone": "030-0074321",           // nonstandard unquoted field name
        "PostalCode": 12209,             // nonstandard single-quoted field name
        "ContactName": "Maria Anders",   // standard double-quoted field name
        "Fax": "030-0076545",
        "Address": "Obere Str. 57",
        "CustomerID": "ALFKI",
        "CompanyName": "Alfreds Futterkiste",
        "Country": "Germany",
        "City": "Berlin",
        "ContactTitle": "Sales Representative"
      }
    ]
  }
}
```

2. В «Repository Explorer» создайте JSON-источник данных.
3. Перейдите в свойства «Main DataSet \ Query» шаблона.
4. выберите созданный источник JSON-данных.
5. Установите Language = JSON.
6. Заполните выражения запроса.

Пример:

```
Northwind.Customers(Country == $P{Country})
```

Более подробную информацию по синтаксису смотрите в вики. Для автоматического формирования запроса кликните по требуемому узлу JSON дерева(слева от выражения запроса).

7. Нажать «Read Fields» для создания полей на основе JSON-данных.

Теперь к полям источника данных можно обращаться как к полям SQL-запроса `$F{ПОЛЕ}`.

Для передачи данных в функцию `createReportEx()` используйте параметр `JSON_INPUT_STREAM` с типом `java.io.InputStream` содержащим JSON-данные.

пример:

```
val json = ""{json-данные}""
createReportEx(
  getVar("sSystemName").asString,
  null,
  PostBuildAction.print,
  Map[String, Any](
```

(продолжается на следующей странице)

```

        "JSON_INPUT_STREAM" ->
            IOUtils.toInputStream(json, "UTF-8")
    )
)

```

Также доступны параметры:

- `JSON_LOCALE`;
- `JSON_TIME_ZONE`;
- `net.sf.jasperreports.json.date.pattern`;
- `net.sf.jasperreports.json.number.pattern`.

Скриптлет (Scriptlet)

Скриптлет - это Java-класс, позволяющий кастомизировать обработку событий построения отчета, а также добавить вспомогательные функции.

Дополнительная информация:

- [all-you-want-know-about-scriptlets](#);
- [hello-world-sample-example](#);
- [sample reference](#).

Скриптлет наследуется от одного из классов:

- `net.sf.jasperreports.engine.JRAbstractScriptlet`;
- `net.sf.jasperreports.engine.JRDefaultScriptlet`.

Экземпляр данного класса создаётся при заполнении отчёта, и его методы вызываются на различных этапах в качестве обработчиков. Также скриптлет может содержать пользовательские методы, которые можно вызвать через выражения «Expression» полей, переменных и т.п.

Создание скриптлета

Для создания скриптлетов необходимо иметь Java-проект, к которому подключена библиотека:

```
"net.sf.jasperreports" % "jasperreports" % "6.5.1"
```

Перечень действий:

1. Создайте Java-класс.

```

public class JRMyExampleScriptlet extends
    net.sf.jasperreports.engine.JRDefaultScriptlet {
}

```

2. Переопределите необходимые методы-обработчики событий.
3. Если требуется добавьте необходимые функции.

Для создания и компиляции класса скриптлета, можно использовать проект **Application**. Для этого необходимо подключить к модулю, в котором будет создан класс скриптлета, библиотеку:

```
settings(
    libraryDependencies += Seq(
        "net.sf.jasperreports" % "jasperreports" % "6.5.1" %
        "compile"
        excludeAll(*ExclusionRule("com.lowagie"),*ExclusionRule("org.olap4j"))
    )
)
```

При этом, если построение отчёта будет запущено из приложения отлаживаемого в IDEA, будет возможна отладка кода скриптлета.

Подключение скриплета к шаблону

Видеоурок

1. Соберите Jar, содержащий данный класс.
2. Скопируйте полученный Jar в каталог проекта версии отчёта. Также допустимо не создавая Jar, скопировать структуру каталогов с class-файлом скриптлета в каталог проекта отчёта.
3. Подключите скриплет к дизайнеру JasperReport Studio:
 1. Кликните правой кнопкой мыши на каталоге проекта и выберите «Build Path\Configure Build Path».
 2. Если нужно, подключите jar. Для этого выполните «Add JARs» и выберите jar-файл из каталога проекта версии отчёта.
 3. Если нужно подключите каталог. Для этого выполните «Add Class Folder...» и выберите каталог.
4. Создайте новый скриплет в отчете. Новый скриплет создается из списка скриплетов отчета в окне **outline** по адресу **{Reports} \ Scriptlets**.
5. В свойствах скриплета укажите класс созданного скриптлета.

Теперь при выполнении компиляции и просмотра отчёта в JasperReports Studio будут вызываться методы созданного скриптлета.

Пример скриплета для реализации суммы прописью

Rpt_JRScriptletNumberToStrSample

Сумма прописью реализована в библиотеке numbertostr: «ftp://ftp.bitec.ru/pub/%23Global/Utils/NumberToStr»

Эта библиотека поставляется с сервером global. Для работы в Jasper Studio необходимо подключить эту библиотеку к проекту отчета.

Скриплет – обертка

Создание скриплета:

1. Создайте в проекте файл скриплета. Файл создается в проекте отчета в каталоге **src**:

```
import net.sf.jasperreports.engine.JRDefaultScriptlet;
import net.sf.jasperreports.engine.JRScriptletException;
import ru.bitec.numbertostr.MoneyToStr;
import ru.bitec.numbertostr.settings.Currency;
import ru.bitec.numbertostr.settings.PennyShowType;
import ru.bitec.numbertostr.settings.Language;

import java.math.BigDecimal;

public class JRMoneyToStrScriptlet extends JRDefaultScriptlet {

    public String moneyToStrRu(BigDecimal npSum) throws
    JRScriptletException{
        return MoneyToStr.spellout(
            npSum,
            Currency.RUR,
            Language.RUS,
            PennyShowType.TEXT
        );
    }

    public String moneyToStrByCurrencyAndLanguageAndPennyShowType(
        BigDecimal
        npSum,
        String currency,
        String language,
        String pennyShowType
    ) throws JRScriptletException {
        return MoneyToStr.spellout(
            npSum,
            Currency.valueOf(currency),
            Language.valueOf(language),
            PennyShowType.valueOf(pennyShowType)
        );
    }
}
```

2. Добавляем новый скриплет в отчет:
 1. Зайдите в свойства отчета. Для этого кликните правой кнопкой мыши по каталогу отчета.
 2. Перейдите в раздел «Java build path».
 3. Перейдите на закладку «source».
 4. Добавьте каталог. Нажмите «Add folder» и укажите каталог **src**.
3. В свойствах сервлета укажите:
 - Name = JRMoneyToStrScriptlet;
 - Class = JRMoneyToStrScriptlet.

После этого функции скриплет становятся доступными в expression editor в разделе Parameters

Пример выражения:

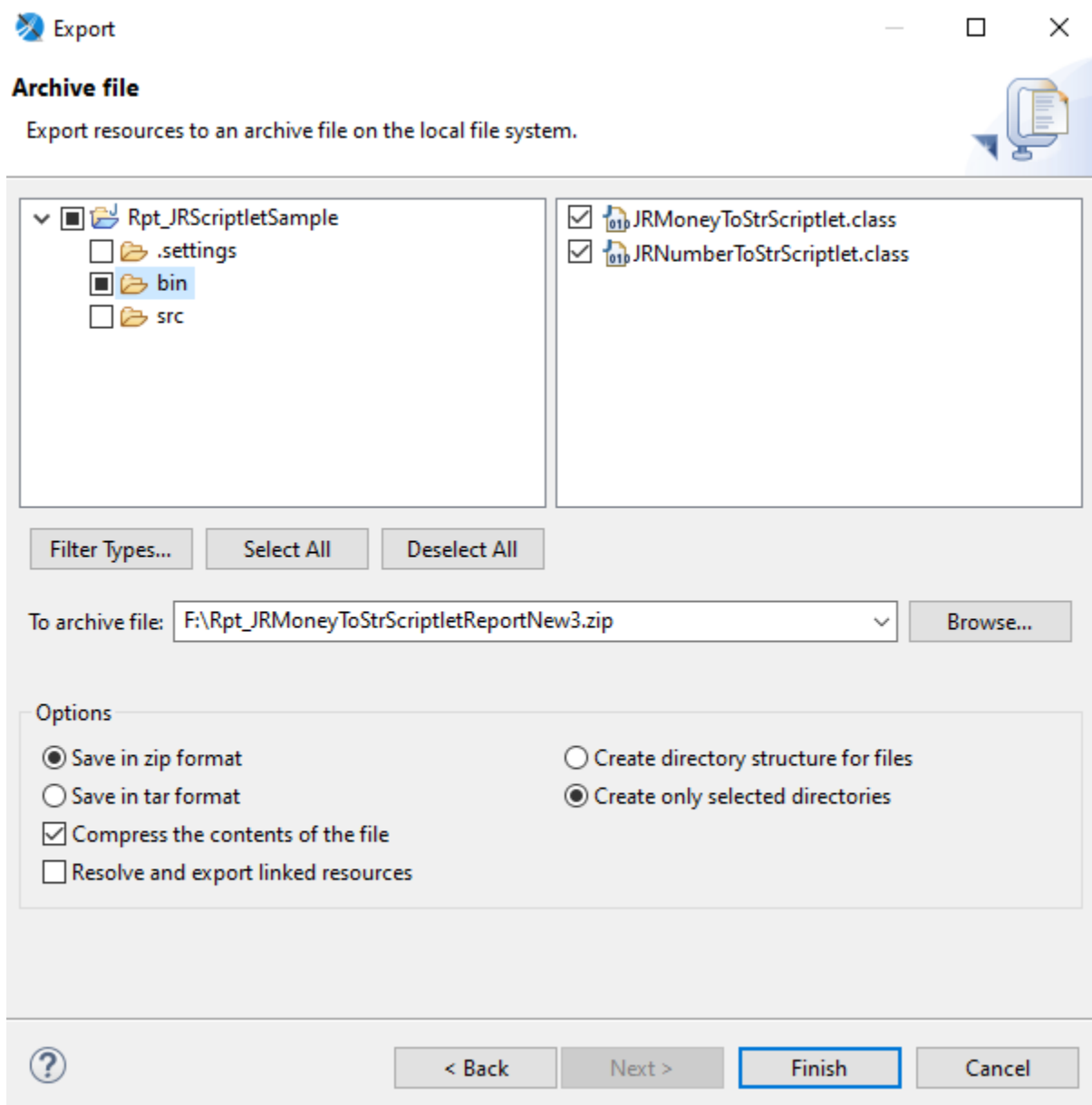
```
$P{JRMoneyToStrScriptlet_SCRIPTLET}.moneyToStrRu($F{nSum})
```

После завершения редактирования отчета необходимо собрать проект.

При экспорте кроме *.jspxml файлов надо экспортировать *.class файлы скриплетов.

Внимание

Именно файлы, а не папку bin, чтобы они оказались в корне архива, иначе jasperg их не найдет при печати из глобал.



Скриптлет для скрытия колонок таблицы с сохранением ее ширины

При разработке отчетов возникает необходимость скрывать колонки таблицы в зависимости от определенных условий. Т.к. движок Jasper не позволяет динамически изменять ширину элементов в момент построения отчета, при скрытии колонок с помощью выражения `printWhenExpression`, оставшиеся видимыми элементы не изменяют свои размеры, из-за чего на месте скрытых элементов появляются пустоты. Скрыть колонки и сохранить ширину таблицы можно с помощью скриптлета `ru.bitec.app.rpt.ColumnResizerScriptlet`.

Примечание

Скриптлет позволяет колонки для элемента Таблица, не элементы `HeaderColumn` и `Detail`.

Для скрытия колонок необходимо создать группу, в `Header` или `Footer` которой нужно поместить элемент Таблица. Для того, чтобы указать скриптлету, какие колонки нужно скрыть, необходимо в отчете создать параметр `HIDDEN_COLUMN`, в котором указать нужные колонки в формате строки `"GroupName1_ColumnName1;GroupName2_ColumnName2;...;GroupNameN_ColumnNameM"`, где:

- `GroupNameN` - имя группы, в которой расположена нужная таблица (автоматически группы именуются `Group1`, `Group2` и т.д.).
- `ColumnNameM` - имя колонки в таблице (автоматические именуются `Column1`, `Column2` и т.д.). Имена групп и колонок таблицы можно поменять в отчете.

Внимание

Для корректной работы скриптлета в `Header` или `Footer` должна содержаться только таблица и ничего больше.

Публикация отчётов в БД

Для обеспечения доступа сервера Global 3 к шаблону отчёта, zip-архив с шаблоном необходимо загрузить в базу данных через приложение «Настройка системы», пункт меню **Отчёты > Печатные формы > Версии печатной формы > Загрузить в базу**.

Шаблон `JasperReports`-отчёта может состоять из нескольких файлов (мастер-деталь, несколько страниц) и содержать различные ресурсы (картинки, скриптлеты).

Поэтому, для сохранения в базу, шаблон приходится архивировать в zip-архив. Файл `main.jrxml` должен находиться в корне архива, иначе он не будет найден. Остальные файлы могут быть во вложенных каталогах, главное, чтобы в шаблоне были указаны верные относительные ссылки.

Архив не должен содержать бинарных `*.jasper` файлов.

При построении, zip-архив будет разархивирован во временный каталог.

Ручная загрузка отчетов

1. В **Project Explorer** выберите папку содержащую отчеты печатной формы.
2. В контекстном меню выполните **Export**.
3. Выберите тип **Archive file**.
4. выберите файлы для экспорта.

Внимание

Допустимы только следующие расширения:

- `.jrxml`;
- `.class`;
- `.jar`.

5. Укажите опцию **save in zip archive**.
6. Укажите файл куда сохранить шаблон.
7. Откройте приложение **Настройки системы**.
8. Откройте печатные формы. Пункт меню **Отчеты > Печатные формы**.
9. Создайте версию печатной формы.
10. Загрузите шаблон печатной формы.

Внимание

Убедитесь что файлы экспортируются в корень архива. Для этого при экспорте должна быть установлена опция **Create only selected directories**.

Загрузка отчета на dataInstall

Загрузку отчета в базу можно добавить в `dataInstall` получив `blob` отчета с помощью метода `Rpt_ReportVersionApi().compress`.

Пример:

```
val blobData = Rpt_ReportVersionApi().compress(  
    List(  
        "subreport1.jrxml",  
        getClass.getResourceAsStream("/reports/Mes_PlanReport/subreport1.jrxml")  
    ),(  
        "subreport2.jrxml",  
        getClass.getResourceAsStream("/reports/Mes_PlanReport/subreport2.jrxml")  
    ),(  
        "order.jrxml",  
        getClass.getResourceAsStream("/reports/Mes_PlanReport/order.jrxml")  
    ),(  
        "main.jrxml",
```

(продолжается на следующей странице)


```

    getClass.getResourceAsStream("/reports/Mes_PlanReport/main.jrxml"))
    )
)

```

Полностью пример можно посмотреть в `ru.bitec.app.mes.Mes_DataInstallPkg#reportInstall`.

Отчет с содержанием

Документация на [sourceforge](#)

Возьмем Report Book, в котором 2 отчета:

- `TableOfContents`;
- `Content`.

Для создания содержания с гиперссылками в начале документа

1. Укажите свойства отчета `TableOfContents`:

- если отчет стоит перед `Content` то `EvaluationTime = Report`;
- `net.sf.jasperreports.bookmarks.data.source.parameter = REPORT_DATA_SOURCE`.

2. При необходимости установить галочку «Use Cache». Галочка находится на `Pages`.

3. Настройте поле для `Anchor` в отчете `Content`:

1. Перейдите на поле которое хотите видеть в содержании.

2. Задайте свойства на закладке `Hyperlink`:

- «Anchor name expression» = `$F{sposition}`. Можно указывать имя в виде "Позиция: " + `$F{sposition}`, но нужно учитывать, что если тут будет число – нужно привести его к `.toString`.
- «Bookmark level» = 1.

4. Создайте поля в `Dataset`'е отчета `TableOfContents`:

- `level` тип `java.lang.Integer`;
- `pageIndex` тип `java.lang.Integer`;
- `label` тип `java.lang.String`.

5. Для отображения имени гиперссылки:

1. Создайте или перейдите на соответствующий `TextField` в `TableOfContents`.
2. Задайте значением `$F{label}`.
3. Свяжите поле с уровнем `anchor`. Укажите свойство «Appearance \ print when \ print when expression» = `$F{level}==1`.

6. Для отображения номера страницы:

1. Создайте или перейдите на соответствующий `TextField` в `TableOfContents`.
2. Задайте значение `$V{PAGE_NUMBER} + $F{pageIndex} + 1`.
3. Задайте свойство «Evaluation Time» = `Auto`.
4. Свяжите поле с уровнем `anchor`. Укажите свойство «Appearance \ print when \ print when expression» = `$F{level}==1`.

7. При необходимости можно сделать поля с именем анкера и номером страницы гиперссылками:

1. Откройте свойства поля.
2. Перейдите на закладку Hyperlink.
3. В разделе HyperLink задайте:
 - «Link Target» = Self;
 - «Link Type» = LocalAnchor;
 - «Hyperlink Anchor Expression» = `$F{table}`.

Шаблоны отчетов

Шаблоны отчетов позволяют создавать документы различных форматов на основе данных системы. Поддерживаются форматы XLSX, DOCX и TXT с единым принципом разметки через специальные тэги.

Принцип разметки

Шаблоны документов имеют одинаковую разметку. Научившись размечать шаблоны одного типа, можно создавать отчёты других типов.

Примечание

Раздел не относится к специализированным отчетам, таким как Jasper Reports.

Основным элементом разметки шаблона является **тэг** — текст, окружённый спецсимволами, который управляет формированием отчёта.

Все тэги имеют следующий вид: `[#{значение}#]`. Простейшим случаем является тэг, возвращающий значение атрибута строки: `[#{имя атрибута}#]`.

С помощью таких тэгов обозначаются места, которые должны быть заполнены данными. Если в данных не будет найден атрибут или параметр с именем `{имя атрибута}`, текст будет заменён на пустую строку.

Для печати в отчёте набора данных предназначены переменные части, позволяющие выполнять запросы и выводить в отчёт данные с изменяющимся количеством строк.

Переменной частью называется область шаблона, ограниченная открывающим и закрывающим тэгами. Переменная часть будет выведена в отчёт столько раз, сколько записей находится в выборке.

Форматы тэгов для печати данных из запроса:

- Открывающий тэг:

```
[#{произвольное системное имя}={SQL-запрос}#]
```

- Закрывающий тэг:

```
[#{/произвольное системное имя}#]
```

Переменная часть обязательно должна заканчиваться закрывающим тэгом, системное имя которого совпадает с системным именем открывающего тэга. Все системные имена открывающих тэгов начинаются с символа `&`, а закрывающие тэги — с символа `/`.

При выводе данных в отчёт из запроса открывающий тэг переменной части состоит из произвольного уникального (в рамках шаблона) имени и SQL-запроса. В этом случае для данной переменной части будет создан набор данных с запросом, указанным в тэге.

Переменные части могут быть вложенными друг в друга, образуя связку «мастер-деталь». Если из вложенного уровня (уровень «деталь») необходимо получить значение атрибута из запроса верхнего уровня (уровень «мастер»), следует использовать маркер следующего вида: `[#super$sField#]`, где `sField` — имя атрибута из SQL-запроса верхнего уровня.

При этом стоит отметить, что маркер `[#sField#]` тоже сработает, если атрибут `sField` присутствует только в запросе верхнего уровня. Если же он встречается в запросе нижнего уровня, то в печатной форме будет выведено значение из него.

Совет

1. Располагайте открывающие и закрывающие тэги на отдельных строках.
2. Выносите запросы переменных частей в выборки, если их длина превышает 1000 символов.

Шаблон xlsx

Шаблоном `xlsx` является документ в формате `xlsx`, размеченный по общим правилам разметки шаблонов, описанным в предыдущем разделе.

[#&Person=Select * from LBR_Person#]																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								

Обратитесь к дистрибьютору документации за файлом `report_template/ReportExample.xlsx`

Шаблон docx

Шаблон `DOCX` является документ в формате `docx`, размеченный по общим правилам разметки шаблонов.

Если создать отчёт с шаблоном DOCX и скопировать в него разметку из отчёта XLSX, результат построения будет таким же, как для XLSX. Отличия будут только в форматировании — данные будут теми же.

Шаблон TXT

Шаблон на основе текстового файла был реализован для быстрой печати на матричных принтерах, поскольку печать текста происходит намного быстрее, чем печать графики или векторных шрифтов. Пример:

```
[#&Person=Select * from LBR_Person#]
Карточка читателя библиотеки №[#sMnemoCode#]
Фамилия: [#sSecondName#] Имя: [#sFirstName#]
Отчество: [#sMiddleName#] Дата рождения: [#dBirthDate#]
Телефон: [#sPhone#]
Список заказов
Пер. №          Дата          Состояние
[#&Orders=select o.*,
lbr_UserOrderAPI.GetIdState_HL(o.idState) as IdState_HL
from lbr_UserOrder o
where o.idUser = :super$id#]
[#nRegNumber#] [#dDate#] [#idState_HL#]
[#/Orders#]
[#/Person#]
```

Внимание

Открывающий тэг с запросом должен располагаться в одной строке!

13.6 Организация разработки

Основные понятия

Данная глава описывает основные понятия необходимые для понимания организации разработки

Решение

Дистрибутив решения это набор скомпилированных библиотек (jar файлов) устанавливаемых в сервер приложения. Решение предоставляет весь перечень прикладной бизнес логики необходимой для работы сервера приложения.

Системные требования

- Установленный сервер приложения GlobalFramework
- Доступ к базе данных PostgreSQL

Дистрибутив сервера приложения

Дистрибутив сервера приложений представляет из себя архив содержащий набор файлов необходимый и достаточный для запуска **решений**.

Системные требования

- Java на базе OpenJDK 1.8

Модуль

Неделимый набор файлов для сборки решения. Набор модулей определяет доступную функциональность развернутую в сервере приложения.

Модуль содержит:

- **Классы**
Описание атрибутов таблицы, и бизнес логика обработки строк.
Подробнее смотри:
 - *Классы*
 - *Сервисы класса*
 - *Тип объекта*
- **Выборки**
Выборка определяет правило получения, отображение данных и обеспечивает взаимодействие с пользователем.
Подробнее смотри:
 - *Выборка*
- **Поставочные данные**
Данные и скрипты для установки в БД при первом запуске решения.
- **Скрипты миграции**
Скрипты комплексного обновления схемы базы данных и модуля при первом запуске новой версии решения

Зависимости

Модуль может содержать зависимости от другим модулей:

- **Статическая зависимость**
При статической зависимости в программе дочернего модуля можно использовать весь код родительского модуля, как если бы этот код был в самом дочернем модуле. При этом невозможен параллельный выпуск родительского и дочернего модуля. Для исключения ошибок бинарной несовместимости необходимо выпустить сначала родительский модуль и на основе него дочерний. Либо оба модуля должны выпускаться одновременно в одной сборке.
- **Динамическая зависимость**
При этом доступ к дочернему модулю возможен только через jexl скрипты. Jexl скрипты ограничивают возможности совместного использования кода, часто неявно требуя специализированных

фасадов для взаимодействия. Однако это позволяет выпускать релизы модулей независимо друг от друга.

Проектный модуль

Модуль содержащий в себе конфигурацию решения, данный модуль содержит:

- Обработку прикладных событий
Прикладные модули могут предоставлять набор событий на которые можно подписаться для расширения или склейки поставочной бизнес логики
- Конфигурацию проектных переопределений
Можно переопределить контроллеры классов и выборов
- Проектную бизнес-логику
Если нет специальной необходимости выделять отдельный модуль для бизнес-логики, ее можно расположить непосредственно в проектном модуле.

Репозиторий библиотек

Репозиторий библиотек хранит повторно используемые артефакты. Для корректной сборки решения необходимо предоставить доступ к следующим репозиториям:

- Репозиторий библиотек
- Репозиторий комплектов сборки
Используется в случае если *проект* собирается на основе комплекта сборки

Примечание

Используются разные репозитории, так как это облегчает сброс локального кэша, что может быть полезно при использовании **snapshot** версий.

Комплект сборки

Набор одновременно выпускаемых артефактов, достаточный для сборки проекта. Базовый комплект сборки содержит:

- Сервер приложений
- Плагин сборки
Используется для расширения стандартной функциональности **sbt**
- Базовые модули

Модули публикуются группами в комплекте сборки. Что существенно снижает нагрузку на программистов при разработке модуля. Так как позволяет не отслеживать бинарную совместимость. Это позволяет существенно увеличить допустимые изменения в рамках одной версии релиза или **snapshot** версии.

База данных

Для хранения транзакционных данных используется база данных PostgreSQL

Системные требования

- PostgreSQL 12 и выше
- *Расширения БД*

Конфигурация решения

Позволяет настраивать поведение решения без привлечения программистов. В основном содержит:

- *Настройки типов*
- *Печатные формы*
- Настройки электронных подписей для пользователей
- Права доступа
- Универсальные характеристики
- Универсальные справочники

Создание базы данных для разработки

Программист может в любой момент создать себе локальную копию базы данных (*Начало работы*).

При необходимости распространять конфигурацию по локальным базам разработки, возможно:

- Использование полных релизов конфигурации
- Клонирование базы разработки с настроенной конфигурацией решения
В простейшем случае достаточно снятия и установки дампа схемы `public`

Создание тестовой базы данных

Тестовая база может быть создана от рабочей базы. В простейшем случае достаточно снятия и установки дампа схемы `public`.

Шаблон ландшафта

- `common_dev`
Центральный проект разработки. На нем разрабатываются базовые версии модулей. И выпускаются релизы коробочных решений.
- `project_dev`
Проектная база разработки, осуществляется детальная настройка модулей и интеграций по ТЗ заказчика. Производится тестирование.
- `project_test`
Тестирование решения на мощностях заказчика.

- `project_prod`
Рабочий контур заказчика

Файловое хранилище

Используется для хранения прикрепленных файлов отдельно от БД. Это позволяет существенно снизить нагрузку на базу данных.

Доступ к файловым хранилищам настраивается в приложении: **Настройка системы \ Сущности > Файловые хранилища**

В дополнительных операциях есть следующие операции:

- **Проверить чтение и запись в хранилище**
Запрашивается размер файлов в Килобайтах, дефолт 1. После чего запускается тест:
 1. В течение одной минуты создаются файлы указанного размера. Наполнение файлов — случайный текст, файлы все одинаковые. Генерация наполнения файла не включается во время теста (текст генерируется до создания файлов). Фиксируется количество созданных файлов.
 2. Чтение созданных файлов. Содержимое читается в переменную и так по всем файлам. Фиксируется время чтения всех созданных в п.1 файлов.
 3. Удаление созданных файлов. Фиксируется время удаления всех созданных в п.1 файлов.

Проект

Директория на диске, хранящая файлы для сборки решения.

Структура проекта

- `module1` - прикладной модуль.
- `module2` - прикладной модуль.
- `moduleN` - прикладной модуль.
- `projectModule` - проектный модуль.
- `project` - метаинформация по сборке.
- `build` - каталог, находящийся вне системы контроля версий и содержащий данные, специфичные для локальной сборки:
 - `lib` - локальные JAR-библиотеки для запуска сервера и утилит прикладной разработки.
 - `config.yml` - создаётся при необходимости локального переопределения конфигурации проекта.
- `build.sbt` - декларация сборки для `sbt`.
- `project.yml` - конфигурация проекта, подробнее можно посмотреть [тут](#).

Конфигурация проекта

Конфигурация проекта описывает правила сборки прикладного решения и располагается по пути `application/project.yaml`

В ней перечисляются модули, используемые в сборке решения, а также их источники. В случае необходимости источник модуля можно переключить, что позволяет отлаживать исправления ошибок непосредственно в текущем проекте.

Внимание

После отладки исправления необходимо создать `pull request` в ветку разработки соответствующего модуля и дождаться нового релиза. Если решение требуется выпустить без ожидания релиза комплекта сборки, следует переключиться на режим сборки по исходному коду для всех дочерних модулей.

Источники модуля

`local`

Код модуля находится в системе контроля версий исходного проекта. Подключается по исходному коду.

`git`

Код модуля находится в отдельном `git` репозитории. Подключается по исходному коду.

Совет

Для массовой синхронизации модулей используйте утилиту `gsf-cli`.

`base`

Модуль выпускается в базовом комплекте сборки. Подключается в виде JAR-артефакта.

`buildKit`

Модуль выпускается в прикладном комплекте сборки. Подключается в виде JAR-артефакта.

Пример конфигурации

```
#Комплект сборки, куда будет происходить публикация прикладных модулей
buildKit:
  name: ru.bitec.app
  version: 1.0.1

#Комплект сборки на основе которого работает проект
#Базовый комплект сборки обычно содержит сервер приложений и базовые модули
baseBuildKit:
  name: ru.bitec.base
  version: 1.0.1

#Источник сервера приложений
#Используется утилитой gs3-cli для подготовки проекта к работе
applicationServer:
  source: "base://"

#Источник плагина для сборки
sbtPlugin:
  source: "base://"

modules:
  #Подключение модуля через jar артефакт из maven репозитория
  - gtkjpa:
    source: "base://"
  - gtk:
    source: "base://"
  - btk:
    source: "base://"
  #Подключение модуля через исходный код из удаленного репозитория
  - bts:
    source: "git://{host}/bts.git"
    branch: main
    isPublish: true
```

Примечание

Подробнее ознакомится с конфигурацией можно [тут](#)

Конфигурация репозиториев

Чтобы обеспечить возможность сборки проекта в различных локальных сетях, настройка репозиториев вынесена в отдельный файл и может быть переопределена при необходимости.

По умолчанию настройка репозиториев берется из файла: `application/project/repositories/default.yaml`.

Пример настройки:

```
repositories:
- name: maven-global
  url: "https://repo.global-system.ru/artifactory/libs-release"

publishRepository:
  name: app-global
  url: "https://repo.global-system.ru/artifactory/build-kit"
  #Запрашивать данные авторизации из менеджера учетных данных для данного репозитория
  isLoginRequired: true
```

Работа в изолированной сети

Если нужно настроить проект на использование только локальных сетевых репозиториях, необходимо:

1. Настроить глобальную конфигурацию репозиториях `sbt` - для этого необходимо добавить локальные репозитории в файл `~/.sbt/repositories`.

```
[repositories]
maven-proxy: http://host/repository/maven/
buildkit: http://host/repository/buildkit/
```

2. Включить изоляцию репозиториях - это позволит обращаться только к локальным репозиториях, игнорируя репозитории вшитые в программный код. Для этого в файле проекта `application/.sbtopts` добавьте флаг `-Dsbt.override.build.repos=true`.

Примечание

Включение данного флага не перекрывает настройки авторизации.

Смотрите: Прокси репозитории

Интеграция в vcs

Исходный код проекта может располагаться в произвольной системе контроля версий.

Модули могут находиться как в репозитории самого проекта, так и в удалённом репозитории. При инициализации проекта код из удалённых репозиториях добавляется в каталог основного проекта.

Совет

Модули из удалённого репозитория рекомендуется добавлять в файл `.gitignore` (или аналог) основного репозитория проекта.

Стратегии ветвления

После инициализации проекта можно пользоваться стандартными средствами ветвления используемой vcs.

Предупреждение

При внесении изменений, нарушающих обратную совместимость схемы базы данных, рекомендуется:

- Вносить такие изменения в основную ветку (**master/main**), чтобы минимизировать конфликты схемы БД между разработчиками.
- Если изменения слишком масштабны, для их тестирования следует использовать локальную базу данных разработчика.

Публикация решения

Публикация решения происходит средствами CI, например [jenkins](#).

Команда sbt для публикации решения в локальный каталог:

```
sbt publishSolution
```

Публикация комплекта сборки

Публикация комплекта сборки происходит средствами CI.

Команда sbt для публикации решения в репозиторий:

```
sbt publish
```

Релизы

Релиз представляет из себя снимок файлов определенной версии.

Решение

Релиз решения содержит перечень модулей и их версии.

Конфигурация

Релиз конфигурации содержит данные и скрипты обновления **конфигурации решения**. База выпуска релизов выбирается по согласованию с заказчиком.

Сервер приложения

Содержит в себе ядровой функционал необходимый для запуска решения. Использует **семантическое версионирование**.

Комплект сборки

Содержит скомпилированные прикладные модули. Версия формируется по следующему правилу:

- **GENERATION**
Модули должны содержать одно поколение.
- **MAJOR**
Перещелкивается при любом изменении мажорной версии модуля в составе комплекта.
- **MINOR**
Перещелкивается при любом изменении минорной версии модуля в составе комплекта.
- **BUILD**
Перещелкивается при любом изменении билда версии модуля в составе комплекта.
- **PATCH**
Перещелкивается при любом изменении патч версии модуля в составе комплекта.

Snapshot версия

Используется для распространения артефактов из ночной сборки или внеплановой сборки. Артефакты snapshot версии могут повторно закачиваться в репозиторий.

Для обновления Snapshot версий разработчику необходимо выполнить следующие команды sbt

```
sbt clean
sbt update
sbt updateClassifiers
sbt publishDevDependencies
```

Релиз прикладного модуля

Исходный код, согласованный на момент выпуска релиза, а также скрипты необходимые для установки обновления модуля в базе данных.

Версия модуля

Версия модуля задается в формате: Поколение.МажорнаяВерсия.МинорнаяВерсия.Билд.Патч

Типы релиза

- **Поколение** - Крупные изменения, которые ломают обратную совместимость. Старшая версия может означать полное переписывание кода. Так же в этом типе релиза происходит очистка миграционных скриптов предыдущей версии. Устанавливается только на предыдущее поколение.
- **Мажор** – Крупные изменения, которые могут ломать обратную совместимость. К примеру переключение сервера приложений или обновление SBT плагина.
- **Минор** – Плановые изменения в рамках развития модуля, а так же изменение конфигурации модулей проекта (добавление, удаление модулей).
- **Билд** – Сквозная нумерация. Увеличивается при выпуске любой из версий, не считая патчей.
- **Патч** – Изменения критических ошибок на уже выпущенных версиях. От патчей нельзя создавать другие версии, кроме последующих патчей.

Состав релиза

Релиз состоит из:

- Исходного кода
- Первичных данных, указанных в `odm.xml` и `pkg.xml`
- Первичных и миграционных данных, указанных в пакетах миграции (`<Имя модуля>_MigratePkg`)
- `Sql`-скриптов миграции

Схема базы данных модуля

Схема базы данных, прикладного модуля создается в базе данных при установке модуля.

Объекты схемы, которые были сформированы ранее, и были удалены из исходного кода приложения, не удаляются из БД автоматически. Этим занимается администратор используя инструмент «Корзина удаленных объектов схемы».

Доменная схема

Доменная схема, представляет из себя таблицы, индексы и последовательности создаваемые по описанию классов в `odm`-файлах.

Расширенная схема

Расширенная схема, может быть описана в `odm`-, `pkg`-файлах, в тэге `dbSchema`.

Поставочные данные модуля

Поставочные данные прикладного модуля устанавливаются в базу данных при установке модуля.

Миграционные данные

Миграционные данные делятся на 2 типа:

- **Sql-скрипты** - используются для работы с изменением схемы напрямую в БД, без `eclipse-link`. Хранятся в структуре модуля в ветке `resource/migration/dbSchema`
- **Миграционные задачи** – скрипты, которые написаны с использованием `Api` и `Pkg`. Хранятся в миграционных пакетах модуля.

Миграционные пакеты

Общие сведения

В каждом модуле возможно создание миграционного пакета, который должен иметь имя `<Имя модуля>_MigratePkg`, и находится в рутовом `scala`-каталоге модуля. Например, полное имя пакета модуля `btm` будет иметь вид: `ru.bites.app.btm.Btm_MigratePkg`

Этот пакет используется для написания кода по установке первичных данных или выполнения миграции данных, которое можно выполнить с помощью `scala`-кода.

Миграционные пакеты требуется наследовать от `MigratePkg`, переопределяя в них методы `onUpMigrate` и `onDownMigrate`, в этих методах требуется создавать соответствующие задачи миграции через методы `upTask` и `downTask`. Вызов миграционных пакетов осуществляется после выполнения команды установки первичных данных (`init data`).

Порядок вызова

Вызов миграционных пакетов осуществляется в 2 прохода:

- первый раз осуществляется вызов метода `upMigrate`, при обходе модулей от низкоуровневых (например, `gtk`), к высокоуровневым (например, `stk`). В этом методе можно писать логику по созданию новых данных
- при втором проходе вызывается метод `downMigrate`, при проходе модулей от высокоуровневых к низкоуровневым. В этом методе можно удалять данные.

Миграционные задачи

Задачи имеют имя и версию. Задача выполняется только один раз, для того чтобы выполнить задачу повторно, требуется изменить ее версию или имя. Задачи делятся на 2 вида:

- `upTask` - используется в `upMigrate`
- `downTask` – используется в `downMigrate`

UpTask

Задачи метода `upMigrate`. Позволяют указывать зависимости от других задач, от скриптов `dbData` из `odm.xml` и `pkg.xml`. Могут содержать секцию, которая будет выполнена, если задача уже выполнялась на целевой базе. А также могут быть анонимными и не содержать тело.

Простая задача

Содержит только тело. Пример объявления задачи:

```
upTask("taskW0Else", 1){  
    //Код задачи  
}
```

Задача с секцией `orElse`

Если задача выполняет установку каких-то данных, например, типа объекта, то можно указать ей секцию `orElse`, которая будет вызвана, если тело задачи уже было выполнено, например, в этой секции можно вызвать метод поиска типа объекта по его системному имени. Пример объявления такой задачи:

```
upTask("installType", 1){  
    Btk_AuditActionApi().register("someName", "someCaption", "SomeDesc")  
}.orElse{  
    Btk_AuditActionApi().findByMnemonicCode("someName")  
}
```

Анонимная задача

Задача без тела, выполняется для группировки зависимостей. Пример объявления:

```
upTask("dependencyGroup")  
    .dependsOnDbData("Wf_Doc", "dataInstall", 25)  
    .dependsOn(installStkTypeTask())
```


Зависимости

Для каждой задачи можно указать зависимость как от другой задачи, так и от данных, устанавливаемых секцией dbData.

Пример объявления зависимостей.

```
val someOtherTask = upTask("someOtherTask", 1){
    //Код задачи
}

upTask("taskWOElse", 1){
    //Код задачи
}

    .dependsOnDbData("Wf_Doc", "dataInstall", 25) //зависимость от dbData
    .dependsOn(someOtherTask) // зависимость от другой задачи
```

Зависимости между модулями.

Для того чтобы сделать зависимость задачи одного модуля (module1) от задачи другого (module2) требуется:

1. В Module1_MigratePkg задачу вынести в отдельный метод и вызывать его из upMigrate.

```
def someTask(): UpTask[NLong] = {
    upTask("someTask", 1){
        1.nl
    }
}

override def onUpMigrate(): Unit = {
    someTask()
}
```

2. В Module2_MigratePkg для задачи указать зависимость от задачи из module1.

```
override def onUpMigrate(): Unit = {
    val someTask = Module1_MigratePkg().someTask()
    upTask("someDepTask", 1){
        //Получаем значение, которое вернула задача, от которой зависим
        val someTaskValue = someTask.get()
    }
    .dependsOn(someTask)
}
```

DownTask

Задачи метода `downMigrate`. Содержат только тело задачи. Пример объявления задачи:

```
downTask("Task_Down", 1) {  
    //код задачи  
}
```

Sql-скрипты миграции

Общие сведения

Используются для внесения изменений в схему БД, до инициализации `eclipse-link`. Позволяют гарантированно выполняться в схеме, идентичной той, что была на момент выпуска релиза.

Описание файла скриптов

Миграционные скрипты размещаются внутри файла `resource/migration/trunk.script.jexl`. А при выпуске релиза копируются в подкаталог `resource/migration/dbSchema`.

Скрипты пишутся в формате `jexl`-скрипта, и могут быть выполнены до обновления схемы или после.

Исполняются внутри специального `jexl`-контекста, который обладает функциями:

- **before** – скрипт будет выполнен до обновления схемы
- **after** – скрипт будет выполнен после обновления схемы.

Если при выполнении скрипта миграции будет получена ошибка, то процесс установки релиза и обновления базы будет прерван. Для того, чтобы выполнить какой-либо скрипт с игнорированием ошибок нужно добавить к нему директиву `force()`.

Пример файла скриптов

```
//выполнение скрипта после обновления схемы  
after("scriptName1", `  
    delete from btk_class where 1=2 asd  
`);  
  
//выполнение скрипта перед обновлением схемы  
before("scriptName2", `  
    delete from btk_class where 1=2  
`);  
  
//выполнение скрипта с игнорированием ошибок.  
//Если в таком скрипте произойдет ошибка, то обновление не будет остановлено  
  
after("scriptName3", `  
    delete from btk_class where 1=2  
`);  
).force();
```

Инструкции

Создание задачи миграции

1. Зайдите в IDE
2. В нужном модуле в `scala`-ветки найдите или создайте пакет с именем `<Имя модуля>_MigratePkg`
3. Создайте задачу миграции, используя методы `upTask` или `downTask`

Создание `sql`-скрипта миграции

1. Зайдите в IDE
2. В нужном модуле в ветке `resource/migration` найдите или создайте файл `trunk.script.jexl`
3. Добавьте в него скрипт используя методы `before` или `after`

Выпуск новой версии

Выпуск с помощью плагина для IDEA Выпуск с помощью GSF-CLI

Установка релиза

1. Сервер приложений Global должен быть запущен.
2. Подключитесь к серверу SSH-клиентом. И выполните скрипт.

```
attach db {dbAlias} as sys
upgrade
detach
exit
```

`dbAlias` – алиас базы данных. Если не указан, будет использоваться значение по умолчанию из `global3.config.xml`

Обновление будет выполняться под суперпользователем `SYS`

Алгоритм работы

1. Shh-команда `attach db as sys` переключает консоль в режим работы с обновлениями.
2. Команда `upgrade` запустит процесс обновления, состоящий из следующих действий:
 1. Формирование очереди выполнения `sql`-скриптов миграции.
 2. Последовательное обновление схемы с промежуточными выполнениями скриптов
 3. Установка первичных данных (`dbData`)
 4. Выполнение задач миграции
3. Отключение от базы
4. Выход из SSH

Сброс состояния обновления

Если требуется откатить информацию об установленных первичных данных или миграционных задачах, то используется **ssh-команда** **reset upgrade**

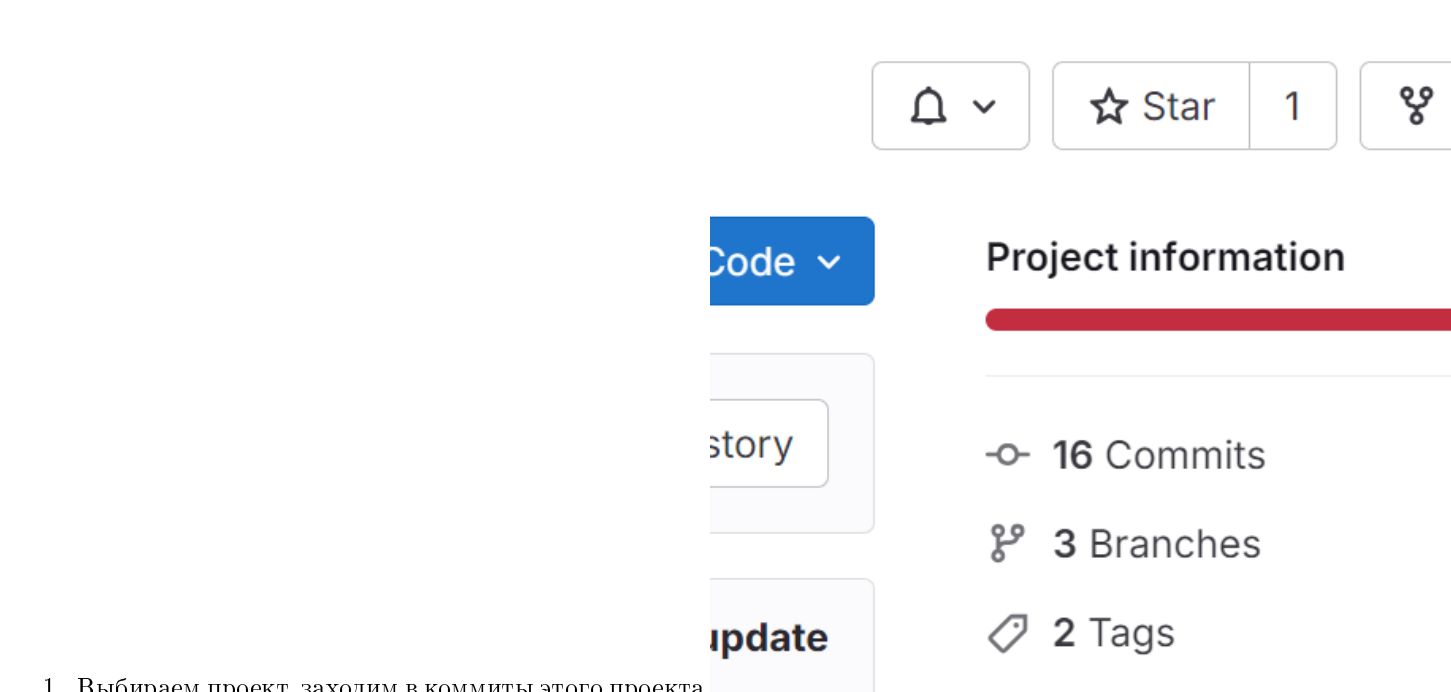
Если в нее не передан идентификатор обновления, то будет сброшено последнее выполненное обновление.

```
attach db {dbAlias} as sys
reset upgrade
detach
exit
```

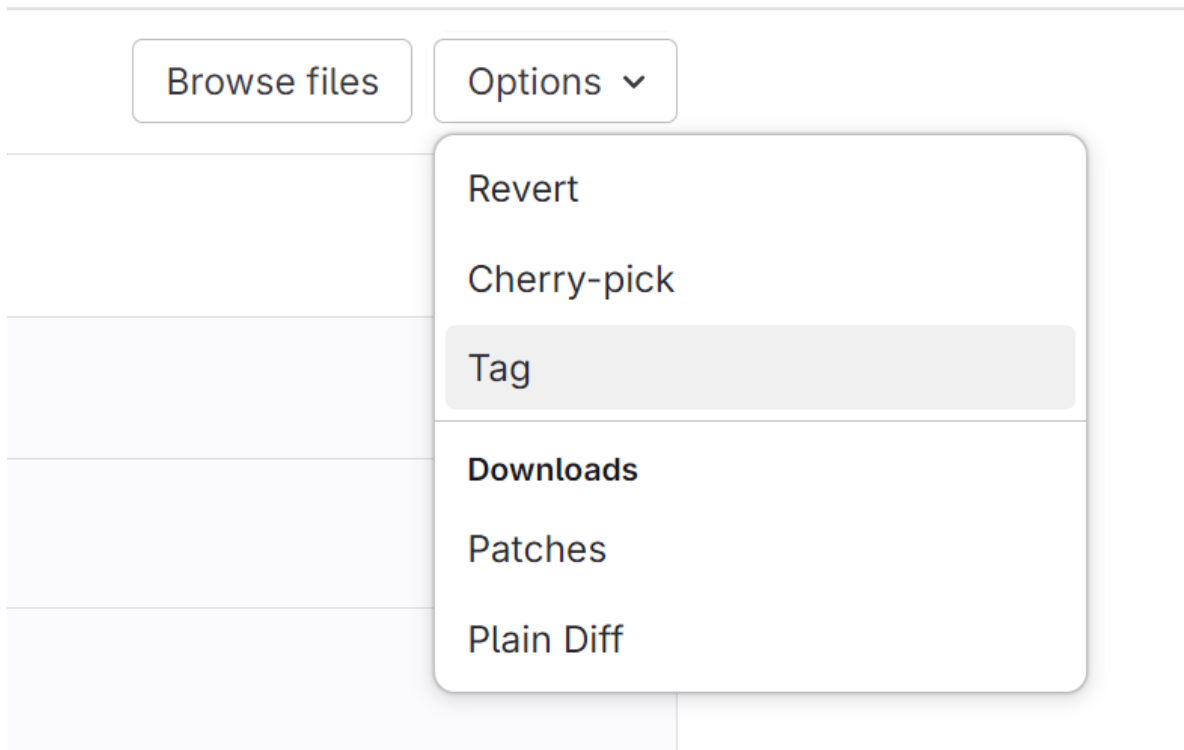
или

```
attach db {dbAlias} as sys
reset upgrade 10202
detach
exit
```

Создание нового тега



1. Выбираем проект, заходим в коммиты этого проекта
2. Выбираем коммит, от которого хотим создать тег, нажимаем на вкладку «Options» в правом верхнем углу, там выбираем «Tag»



3. В поле «Tag name» вводим номер следующего тега. В поле «Message» то же, только перед этим

New tag

Do you want to create a release with the new tag? You can do that in the [release](#) page.

Tag name

Create from

 ▾

Existing branch name, tag, or commit SHA

Message

Optionally, add a message to the tag. Leaving this blank creates a [lightweight tag](#).

слово «Release»

4. Выбираем «Create tag»

Работа с Git

Введение

Предисловие

Назначение

Руководство по изучению и практическому применению системы управления версиями **Git**.

Цель — сформировать у читателя систематизированное понимание принципов работы **Git**: от базовых концепций до организационных аспектов командной разработки и внутренней логики системы.

Аудитория

Руководство предназначено для:

- системных и прикладных разработчиков;
- внешних партнёров и заказчиков;
- технических специалистов (дизайнеров, технических писателей).

Содержание

Руководство включает:

- объяснение ключевых концепций и терминологии систем управления версиями (VCS);
- практические примеры команд с пояснением параметров и флагов;
- рекомендации по организации процесса разработки;
- ответы на часто задаваемые вопросы новичков;
- описание типовых проблем и алгоритмы их решения.

Руководство разделено на две части:

1. **Базовый курс.** Для разработчиков без предварительных знаний о системах контроля версий. Материал рассчитан на углублённое изучение.
2. **Организационные аспекты и правила работы.** Описание `workflow`, соглашений и корпоративных стандартов для работы с репозиториями. Обязательно к применению сотрудниками ООО «Бизнес Технологии»; рекомендовано внешним партнёрам и контрибьюторам.

Предварительные требования

Обязательно установите систему управления версиями `Git`.

Актуальные дистрибутивы для всех поддерживаемых ОС доступны на [официальном сайте](#).

Дополнительные инструменты (рекомендуемые):

Совет

Часть перечисленных инструментов имеет встроенные аналоги в операционных системах, поддерживаемых корпоративным стандартом GlobalERP (например, Блокнот в Windows или `nano` в UNIX-подобных системах).

Их использование не обязательно, но рекомендуется: это упрощает, ускоряет и делает нагляднее взаимодействие с `Git`.

Все дополнительные инструменты, упоминаемые в руководстве, явно обозначаются как опциональные при первом использовании.

Перечень рекомендуемого ПО:

- `GitExtensions` — кроссплатформенный графический интерфейс с открытым исходным кодом для управления репозиториями `Git`; визуализирует историю изменений и упрощает сложные операции;

- **GitToolBox** (плагин для **IDEA** или **Giga IDE**) — расширяет функциональность **Git** непосредственно в интерфейсе **IDE**: отображает имя текущей ветки, статус файлов в реальном времени и другое;
- **VS Code** — современный кроссплатформенный редактор кода с мощной встроенной поддержкой **Git**; позволяет выполнять большинство операций через интуитивный графический интерфейс, минимизируя взаимодействие с консольными редакторами, такими как **Vim**;
- **LazyGit** — консольная TUI-утилита для работы с **Git**; сочетает мощь командной строки и удобство визуального интерфейса прямо в терминале.

Git

Git — современная, высокопроизводительная, распределённая система управления версиями (**VCS**). Она отслеживает изменения в файлах и управляет ими. Применяется не только для исходного кода, но и для любых текстовых файлов: документации, конфигураций и т.д.

Git сохраняет «снимки» состояния проекта на разных этапах. Вы можете:

- вернуться к любому снимку;
- сравнить изменения между ними;
- создать отдельную ветку для экспериментов, не затрагивая основную версию.

Основные определения

- **Система управления версиями (VCS):** отслеживает изменения в файлах проекта (кода, документации, конфигураций и других артефактов); позволяет восстановить любую версию, сравнить изменения, определить автора и причину правок;
- **Распределённая система:** в отличие от централизованных систем (например, **SVN**), где история хранится на одном сервере, в **Git** каждый участник имеет полную локальную копию репозитория; это обеспечивает высокую скорость операций, отказоустойчивость и гибкость рабочих процессов, так как разработчики могут обмениваться изменениями напрямую.

Ключевые особенности

- **Полное хранение истории:** **Git** сохраняет снимки состояния проекта, а не разницу между файлами; это создаёт неизменяемую хронологию всех изменений;
- **Поддержка параллельной разработки через ветвление и слияние:**
 - ветвление — изолированные линии разработки для новой функциональности, исправлений или экспериментов, не затрагивающие основную ветку (**main** или **master**);
 - слияние — механизм интеграции завершённой функциональности из одной ветки в другую;
- **Распределённая архитектура:** каждый разработчик работает с полной локальной копией репозитория; это позволяет создавать коммиты, переключаться между ветками и просматривать историю без сетевого подключения; синхронизация с удалёнными репозиториями (например, на **GitLab**) выполняется отдельными командами (**git fetch**, **git push** и т.д.);
- **Целостность данных:** все объекты в **Git** (коммиты, файлы, теги) идентифицируются по криптографическому хешу (**SHA-1**), вычисленному от их содержимого; любая попытка изменить историю нарушает хеши и становится заметной.

Взаимодействие с Git

Руководство фокусируется на работе через CLI (командную строку), так как это:

- даёт полное понимание внутренних механизмов **Git**;
- обеспечивает универсальность навыков между операционными системами (**Windows**, **macOS**, **GNU/Linux**);
- предоставляет доступ ко всем функциям, включая низкоуровневые команды, недоступные в GUI.

Для обучения некоторые операции демонстрируются через GUI, например **VS Code**. Большинство команд также доступны в популярных IDE:

- **JetBrains IntelliJ IDEA** (Giga IDE);
- **Microsoft Visual Studio Code** (VS Code);
- другие аналогичные среды.

Глоссарий (ключевые термины и концепции)

Для изучения материала необходимо понимание следующих терминов:

- **Репозиторий (Repository)**: хранилище проекта со всей историей изменений и служебной информацией; физически представлен папкой `.git` в корне проекта;
- **Коммит (Commit)**: атомарная единица истории — неизменяемый снимок состояния всех файлов на момент сохранения; содержит:
 - уникальный идентификатор (**SHA-1 хеш**);
 - ссылку на родительский коммит(ы);
 - автора;
 - дату создания;
 - сообщение (**commit message**), описывающее изменения;
- **Тег (Tag)**: статическая именованная ссылка на конкретный коммит; используется для маркировки релизов (например, `v1.0.0.0`, `v2.7.14.127`);
- **Ветка (Branch)**: подвижная именованная ссылка на коммит; при создании нового коммита в ветке она автоматически перемещается на него; ветки изолируют потоки разработки; стандартное название основной ветки — `main`.

Установка и настройка

Система **Git** — кроссплатформенное программное обеспечение. Процедура установки зависит от операционной системы.

Установка Git в Windows

Для Windows рекомендованы два способа установки.

Способ 1: Установка с официального сайта

Внимание

Процесс установки описан для версии GIT 2.48.1.windows.1

1. Скачайте `Git x64 Setup` на [официальном сайте Git](#).
2. Запустите загруженный исполняемый файл `exe`. На экране появится мастер установки `Setup Wizard`.
3. Укажите путь: `C:\programs\Git\`. Нажмите `Next`. (Это рекомендованный путь, но не обязательный.)
4. Настройте компоненты, которые будут установлены. Оставьте выбор по умолчанию.
5. На этапе `Choosing the default editor used by Git` выберите любой редактор, кроме `Vim` (если вы с ним не знакомы). Для новичков рекомендуется: `Use Visual Studio Code as Git's default editor` или другой знакомый редактор, например `Notepad++`, `Sublime Text`, `Atom`.
6. На этапе `Adjusting the name of the initial branch in new repositories` выберите опцию `Override the default branch name for new repositories` и укажите `main`. Это соответствует отраслевым стандартам ООО "Бизнес Технологии".
7. На этапе `Adjusting your PATH environment` выберите опцию: `Git from the command line and also from 3rd-party software`. Это позволит использовать `Git` из командной строки `Windows` (`cmd.exe`) и из терминала `Git Bash`.
8. Выбор `SSH`: оставьте опцию по умолчанию — `Use bundled OpenSSH`.
9. Выбор сервера `HTTPS`: оставьте опцию по умолчанию — `Use the OpenSSL library`.
10. Настройка окончаний строк (`Line Endings`): выберите рекомендованный вариант — `Checkout Windows-style, commit Unix-style line endings`. `Git` будет конвертировать окончания строк в формат `LF` (стандарт для `Unix-систем`) при коммите и обратно в формат `CRLF` (для `Windows`) при извлечении файлов.
11. Настройка эмулятора терминала: выберите опцию `Use MinTTY`. Терминал `MinTTY`, используемый в `Git Bash`, обладает более широкими возможностями по сравнению со стандартной консолью `Windows` (`cmd.exe`).
12. Выбор поведения команды `git pull`: установите значение `Rebase`, чтобы не засорять репозиторий лишними коммитами.

Внимание

Либо указывайте значение по умолчанию (`Fast-forward or merge`), либо читайте руководство по видам слияния, чтобы понять, какой подход вам лучше подходит.

Далее, в ручной настройке рассказано, как сменить выбранный тип настройки.

13. Выбор **Credential-Helper**: оставьте значение по умолчанию — **Git Credential Manager Core** (позволяет безопасно хранить учётные данные для удалённых репозиториях на ОС Windows).
 14. Дополнительные опции: оставьте включёнными **Enable file system caching** и **Enable symbolic links**.
 15. Завершите установку: нажмите **Install**, затем **Finish**.
-

Способ 2: Установка через пакетный менеджер Winget

Внимание

Для опытных пользователей, так как настройка производится отдельно от установки.

1. Откройте **PowerShell** или командную строку (**cmd.exe**) от имени администратора и выполните команду:
`winget install -e --id Git.Git`
2. Проверьте установку, открыв терминал (**cmd.exe**) или **Git Bash** (через меню «Пуск»), и выполните команду:
`git --version`.

Установка Git в ОС семейства GNU/Linux (на примере ALT Linux)

Операционные системы **ALT Linux** используют пакетный менеджер для установки программного обеспечения.

1. Откройте терминал.
2. Обновите информацию о пакетах (этот шаг может требовать прав суперпользователя):

```
su -  
apt-get update
```

3. Установите Git: `apt-get install git`
4. Для проверки установки выполните: `git --version`. В терминале должна отобразиться версия Git.

Установка дополнительных инструментов (рекомендуемых)

- Установка **Git Extensions** (только для Windows): загрузите установщик с [официального сайта](#) и запустите его.
- Установка **Visual Studio Code** (Windows): загрузите установщик с [официального сайта](#) и запустите его.
- Установка **LazyGit**: следуйте инструкциям на [официальной странице](#) проекта.

Первичная настройка Git

После успешной установки необходимо выполнить глобальную настройку окружения и идентификационных данных (эта информация будет добавляться к каждому вашему коммиту).

Откройте терминал и выполните следующие команды, подставив свои реальные данные:

```
# Установка основного редактора для написания сообщений коммитов.
# Для VS Code
git config --global core.editor "code --wait"
# ИЛИ
# Для Nano (GNU/Linux)
git config --global core.editor "nano"
# ИЛИ (GNU/Linux) (если вы уверены в своих силах, так как в данном руководстве не
→ рассказывается как выйти из `VIM`. p.s. esc + :wq! + enter)
git config --global core.editor "vim"

# Установка имени ветки по умолчанию для новых репозиториях.
git config --global init.defaultBranch main

# Включение цветного вывода в терминале для лучшей читаемости.
git config --global color.ui auto

# Настройка поведения команды 'git pull' по умолчанию на '--rebase'.
git config --global pull.rebase true
```

Аутентификационные данные GIT:

Внимание

Эти данные используются Git для идентификации автора коммитов и привязываются ко всем создаваемым коммитам в репозиториях на данном компьютере.

Не рекомендуется применять данные команды, так как инструмент командной строки для GlobalERP (gsf-cli) автоматически их подставляет, но они приведены в данном руководстве для продвинутых пользователей.

```
# Установка вашего логина и email. Используйте те же данные, что и на основном хостинге
→ Git (GitLab или др.).
git config --global user.name "login"
git config --global user.email "your.email@example.com"
```

Проверка конфигурации GIT:

Чтобы просмотреть все примененные настройки, выполните команду:

```
git config --list
```

Дополнительная документация

Данный раздел содержит подборку источников, предназначенных для самостоятельного углублённого изучения **Git**. Материалы варьируются от официальной документации до интерактивных учебных курсов и рекомендованы для расширения кругозора.

Основные материалы по системе управления версиями **Git**

Внимание

Настоятельно рекомендуется ознакомиться с данными источниками совместно с данным руководством.

- **Pro Git (ru)** – официальная книга по **Git**, авторства Скотта Шакона и Бена Штрауба, являющаяся де-факто стандартом в изучении системы. Книга полностью переведена на русский язык и доступна в онлайн-формате. Содержит исчерпывающее описание системы: от фундаментальных концепций (коммит, ветвление, слияние) до продвинутых техник (перебазирование, работа с подмодулями, внутреннее устройство **Git**). Используйте как энциклопедию или учебник для структурированного, глубокого изучения.
- **Git Hot To (ru)** - интерактивный вводный курс, построенный на основе пошаговых практических заданий, выполняемых в имитации терминала прямо в браузере. Цель курса - освоение базовых рабочих процессов и команд **Git** через прямое применение. Идеальный ресурс для самого начала практического знакомства с системой. Позволяет быстро получить первый опыт, не отвлекаясь на установку и настройку окружения.
- **Официальная документация Git (en)** - исчерпывающая справка по всем командам и их опциям. Представлена в виде **manual pages**. Является первоисточником и главным справочным материалом.

Материалы по работе с командной строкой (**Terminal/Shell**)

Внимание

Поскольку настоящее руководство ориентировано на использование **CLI**, уверенное владение базовыми принципами работы с терминалом является важным навыком. Приведённые ниже материалы помогут сформировать и укрепить этот навык для операционных систем, поддерживаемых корпоративным стандартом **GlobalERP**.

- **Windows Terminal (ru)** - руководство по установке, настройке и использованию терминала от Microsoft, который рекомендуется к использованию вместо устаревшего **cmd.exe**. Документация включает руководства по работе с профилями, настройке внешнего вида (темы, шрифты) и использованию различных оболочек, таких как **PowerShell**, **Command Prompt** и **Git Bash**.
- **Введение в PowerShell (ru)** - официальное вводное руководство по оболочке и языку сценариев **PowerShell**, который является стандартом для автоматизации и управления в современных экосистемах **Windows**.
- **The Linux Command Line (en)** - фундаментальная книга Уильяма Шоттса, признанная одним из лучших вводных руководств по **shell**-скриптингу и работе в **bash**.

- [Linux Terminal \(en\)](#) - официальная документация Ubuntu по работе с терминалом Linux.

Документация по интеграции Git со средами разработки (IDE) и редакторами кода

- [Официальная документация \(VS Code\) \(en\)](#) - полное руководство по встроенным возможностям работы с системами контроля версий (с акцентом на Git) в редакторе VS Code. Подробно описывает интерфейс Source Control, индикаторы изменений, интерактивное разрешение конфликтов слияния и работу с удалёнными репозиториями непосредственно из редактора.
- [Официальная документация: Использование интеграции с Git в IntelliJ IDEA \(en\)](#) - исчерпывающее руководство по мощному встроенному инструментарию для работы с Git в IntelliJ IDEA. Документация охватывает весь спектр операций: от клонирования репозитория и просмотра истории изменений до сложного перебазирования и разрешения конфликтов с помощью визуального merge-инструмента среды разработки.

Дополнительные рекомендованные ресурсы

- [FAQ GlobalERP](#) - набор часто задаваемых вопросов, созданный специально в рамках данного руководства.
- [Oh Shit, Git!?! \(en\)](#) - неформальное, но чрезвычайно полезное руководство, описывающее простым языком, как исправить типичные ошибки, допущенные при работе с Git (например, случайный коммит не в ту ветку, потерянный коммит и т.д.).
- [GitHub Skills \(en\)](#) - интерактивные курсы от GitHub, позволяющие на практике освоить не только основы Git, но и специфические workflow, используемые на платформе GitHub (например, работу с Pull Requests, GitHub Actions и т.п., что позволит расширить кругозор использования Git в современной открытой среде разработки).
- [Atlassian Git Tutorials \(en\)](#) - подборка качественных учебных материалов и руководств по Git от компании Atlassian (создатели Vcs - Bitbucket).

Основные понятия Git

Структура репозитория

Описание структуры репозитория Git, как с точки зрения физического размещения служебных файлов и каталогов на диске, так и с точки зрения ключевых логических концепций, определяющих принципы его работы.

Введение: что такое репозиторий Git

Репозиторий Git — это сердце системы контроля версий. Его можно представить как специальную базу данных, которая хранит не только все файлы вашего проекта, но и полную историю их изменений, а также всю служебную информацию, необходимую для работы. Физически эта «база данных» представляет собой скрытую папку `.git` в корне вашего проекта.

Существует два основных типа репозитория:

- **Локальный репозиторий:** Полноценная копия репозитория, расположенная на вашем компьютере. С ней вы работаете напрямую (совершаете коммиты, просматриваете историю, создаёте ветки и т.п.).

- Удалённый репозиторий: Версия вашего проекта, размещённая на сервере (например, на GitLab или внутреннем корпоративном сервере). Используется для обмена изменениями между разработчиками и для резервного копирования кода.

Любая работа начинается с создания или получения локальной копии репозитория.

Логическая структура: состояния файлов

Прежде чем углубляться в физическое устройство, важно понять логику работы Git. Любой файл в вашем проекте может находиться в одном из четырех основных состояний (три из которых являются непосредственно отслеживаемыми Git):

1. Не отслеживаемый (**Untracked**) - Файл, который существует в вашей рабочей папке, но Git никогда о нём не знал (к примеру только что созданный файл) или вы явно сказали Git его игнорировать. Может не являться состоянием файла в репозитории, так как он только что создан, либо является игнорируемым файлом.
2. Изменён (**Modified**) - Файл был изменён в вашей рабочей папке, но Git ещё не знает, что вы планируете сохранить эти изменения. Другими словами, изменения есть, но они не подготовлены к сохранению, но зафиксированы VCS. Является состоянием уже существующего в репозитории файла.
3. Подготовлен (**Staged**): - Вы сообщили Git, что конкретная версия файла должна войти в следующий «снимок» проекта (Это делается с помощью команды `git add`). Файл помещается в специальную промежуточную область, называемую в техническом контексте **Индекс (Staging Area)**.
4. Зафиксирован (**Committed**): - Подготовленные изменения были окончательно сохранены в локальную базу данных репозитория (в локальную папку `.git`) в виде нового коммита. Файл становится частью истории проекта.

Основные состояния соответствуют трём ключевым физическим секциям проекта Git, которые будут рассмотрены далее:

- Рабочий каталог (**Working Tree**): Ваши файлы и папки проекта, с которыми вы работаете напрямую.
- Область подготовленных файлов (**Staging Area / Index**): Промежуточная зона, где формируется следующий коммит.
- Каталог `.git` (**Git Directory / Repository**): База данных, хранящая всю историю и метаданные.

Рабочий процесс работы с состоянием файлов

Процесс перемещения изменений из рабочего каталога в базу данных репозитория - это перемещение изменений между этими областями: `untracked` или `modified` -> `staged` -> `committed`. Он может осуществляться через различные интерфейсы, два основных из которых - командная строка (**Terminal / CLI**) и интегрированная среда разработки (**IDE**).

Работа через командную строку (Terminal / CLI)

- Отслеживание состояний: - Основным инструментом для определения состояния файлов служит команда `git status`. Она выводит отчет, разделяющий файлы по их текущим состояниям:

```
C:\programs\gsf-cli\workspace\sources\pgTest\application\btkkst git:[main]
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   src/main/java/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttr.java
    modified:   src/main/resources/orm/all.xml
    new file:   src/main/resources/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttr.avm.xml
    new file:   src/main/resources/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttr.dvm.xml
    new file:   src/main/resources/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttr.odm.xml
    new file:   src/main/resources/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttr.orm.xml
    new file:   src/main/scala/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttrApi.scala
    new file:   src/main/scala/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttrApi.scala
    new file:   src/main/scala/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttrDpi.scala
    new file:   src/main/scala/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttrDvi.scala

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   src/main/scala/ru/bitec/app/btkkst/usr/ryazanov/Btkkst_ReferenceForObjAttrApi.scala

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    src/main/scala/Btkkst_FileUnBatck.java

C:\programs\gsf-cli\workspace\sources\pgTest\application\btkkst git:[main]
```

- **Staged** - Перечисляются файлы в секции **Changes to be committed**, т.е. подготовленные зафиксированные в локальном репозитории.
- **Modified** - Перечисляются файлы в секции **Changes not staged for commit**, т.е. файлы изменённые, но не добавленные в фиксацию следующего коммита.
- **Untracked** - Перечисляются файлы в секции **Untracked files**, т.е. не отслеживаемые файлы, к примеру только что созданные.

Зафиксированные файлы (Состояние **Committed**) и файлы, которые игнорируются репозиторием

- Ключевые операции:

Работа через интегрированную среду разработки (IDE)

Физическая структура

При инициализации репозитория в корне вашего проекта создаётся скрытая папка с именем `.git`. Эта папка и есть репозиторий Git. Все остальные файлы и папки в вашем проекте (кроме `.git`) составляют рабочую копию (каталог).

```
gtk/
├── .git/           # Вся служебная информация Git (сам репозиторий)
└── hooks/*        # Директория для клиентских и серверных скриптов-хуков
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

		— info/*	# Глобальные файлы исключений (не отслеживаемые файлы)
		— objects/*	# Всё содержимое репозитория (коммиты, файлы, дерево каталогов, ┘
		→ аннотированные теги)	
		— refs/	# Указатели (ссылки) на коммиты (ветки, теги)
		— heads/*	# Ссылки на вершины веток (локальные указатели)
		— remotes/*	# Ссылки на вершины веток (удалённые указатели)
		— tags/*	# Ссылки на теги
		— COMMIT_EDITMSG	# Файл, содержащий сообщение последнего выполненного коммита
		— config	# Конфигурационный файл именно для этого репозитория (локальная ┘
		→ конфигурация)	
		— description	# Файл с описанием репозитория
		— FETCH_HEAD	# Файл, который хранит информацию о последних ветках, ┘
		→ загруженных с удаленного сервера	
		— HEAD	# Файл-указатель на текущую ветку или коммит
		— index	# Файл, который содержит набор подготовленных файлов для ┘
		→ следующего коммита	
		— src/*	# Ваши файлы проекта (рабочий каталог)
		— build.sbt	# Ваши файлы проекта (рабочий каталог)
		— README.md	# Ваши файлы проекта (рабочий каталог)

Внимание

Это не полный пример каталога, в нём так же содержатся дополнительные папки и файлы, которые не рассматриваются в данном руководстве.

Рассмотрим ключевые элементы папки `.git`:

Скрипты-хуки

Глобальные файлы исключений

База данных объектов `objects/*`

Это самая важная часть репозитория. `Git`— это, по сути, контентно-адресуемая файловая система, которая содержит следующие данные:

- снимки файлов `blobs`,
- структура каталогов `trees`,
- коммиты `commits`,
- теги `tags`,

Они хранятся в виде объектов, каждый из которых идентифицируется своим `SHA-1` хешем.

Объекты

- **BLOB (Binary Large Object)**: Объект, хранящий содержимое одного файла на момент сохранения, то есть снимок относительно предыдущего изменения. Не хранит метаданные.
- **TREE (Дерево)**: Объект, представляющий собой один каталог. Он содержит список вложенных в него файлов (указывая на **BLOB-объекты**) и других каталогов (указывая на другие **TREE-объекты**), а также их имена.
- **Commit (Коммит)**: Объект, который связывает всё вместе. Он содержит:
 - Указатель на **TREE-объект** корневого каталога проекта на момент коммита.
 - Указатели на родительский(ие) коммит(ы): родительский коммит - это предыдущее состояние, на основе которого сделаны текущие изменения.
 - Автора (коммитера), дату и время.
 - Сообщение коммита.

Эта структура формирует направленный ациклический граф, где коммиты связаны друг с другом, что и представляет собой историю проекта.

Система ссылок

Хранит историю в виде графа коммитов, идентифицируемых только хешами, что неудобно для человека. Для этого существуют ссылки (**refs**) - простые файлы, хранящие внутри **SHA-1** хеш определённого коммита, но имеющие человеко-читаемые имена.

- **refs/heads/**: Здесь хранятся ссылки на ветки. Имя файла - это имя ветки, а его содержимое - хеш коммита, на который эта ветка в данный момент указывает (её вершина). Например, файл **refs/heads/main** содержит хеш последнего коммита в ветке **main**.
- **refs/tags/**: Здесь хранятся ссылки на теги. В отличие от веток, теги - неизменны и всегда указывают на один и тот же коммит, отмечая определённые моменты в истории (например, релизы).

Указатели

Это специальный файл, который обычно содержит ссылку на текущую активную ветку (т.е. ссылается на файл в **refs/heads/**).

HEAD указывает на коммит, который в данный момент извлечён в ваш рабочий каталог. Когда вы создаёте новый коммит, он становится потомком того коммита, на который указывает текущий **HEAD**, а затем он же (**HEAD**) перемещается на этот новый коммит.

Конфигурационные файлы

- **config**: Файл конфигурации для данного конкретного репозитория. Здесь хранятся настройки, специфичные для этого проекта (например, URL удалённого репозитория, настройки хуков и т.п.).
- **.gitconfig**: Глобальный файл конфигурации, применяемый ко всем вашим репозиториям на этом компьютере (например, ваше имя и почта, настройки **core.editor** и т.п.).

Прочие важные элементы

- **index**: Этот файл (также называемый областью подготовленных файлов) является промежуточной зоной между вашим рабочим каталогом и репозиторием. Когда вы выполняете команду `git add`, Git вычисляет хеш файла и сохраняет эту информацию в **index**. При выполнении `git commit` состояние **index** фиксируется в виде нового **tree-объекта**, который затем становится частью создаваемого коммита.

Логическая структура

Любой файл в вашем рабочем каталоге может находиться в одном из трёх основных состояний, определяемых Git:

1. **Modified (Изменён)**: Файл был изменён в рабочем каталоге, но эти изменения ещё не были отмечены для включения в следующий коммит (не проиндексированы). Git видит изменения, но пока не готов их сохранить в текущий стейдж.
2. **Staged (Подготовлен)**: Изменённый файл был отмечен для включения в следующий коммит с помощью команды `git add`. Его текущая версия помещена в область подготовленных файлов - тот самый **index-файл**.
3. **Committed (Зафиксирован)**: Файл безопасно сохранён в вашей локальной базе данных (в папке `.git/objects/`). Это состояние файла в последнем сделанном коммите.

Эти три состояния соответствуют трём основным секциям проекта Git: рабочему каталогу, области подготовленных файлов (Индексов) и каталогу `.git`.

Коммит

Коммит - Это отдельное сохранение (снимок) состояния ваших файлов в определенный момент времени. Является фундаментальной единицей истории Git, которая сохраняет изменения и позволяет отслеживать эволюцию проекта.

Правила наименования коммитов

Наименования постоянных коммитов являются фиксированными.

Общий формат коммита: `Label(Scope): Subject #issue [Issue-Number]`

Label - тип изменений в коммите:

- **feat** - реализация нового функционала.
- **fix** - исправление ошибок.
- **hotfix** - оперативное исправление критических ошибок.
- **test** - добавление тестов.
- **refactor** - рефакторинг кода без изменения функциональности.
- **docs** - обновление документации.
- **chore** - прочие изменения, не подпадающие под указанные категории.

Scopes - опциональная область, в которую вносят изменения (компонент, страница, модуль или пакет).

Subject - короткое описание изменений. Оно начинается с заглавной буквы и краткого страдательного причастия. Например: «Изменено», «Исправлены», «Указаны», «Реализован».

Issue-Number - номер задачи или документа поддержки, при наличии, либо `notask`.

Примеры названия коммита:

- `fix(rpl)`: Убрано поле "Источник" у входящих сообщений интеграции `#issue 182831T`
- `feat(btk)`: Указаны подробности в описании БД `#issue 217081D`
- `refactor(gtk, btk)`: Перенесена логика кэширования `#issue notask`

Метаданные коммита

Каждый коммит содержит следующие метаданные:

- Уникальный хеш (SHA-1) - 40-символьный идентификатор, вычисляемый на основе содержимого коммита.
- Автор - пользователь, создавший изменения в файлах.
- Коммитер (Автор фиксации коммита) - пользователь, зафиксировавший изменения в репозитории.
- Дата создания - временная метка фиксации.
- Сообщение коммита - описание внесённых изменений.
- Ссылки на родительские коммиты - указатели на непосредственных предков в истории (есть у всех коммитов, кроме первого).

Основные свойства

1. Неизменяемость

После создания, коммит не может быть изменён. Любая модификация приводит к созданию нового коммита с уникальным хешем. Это обеспечивает целостность и достоверность истории проекта.

2. Атомарность

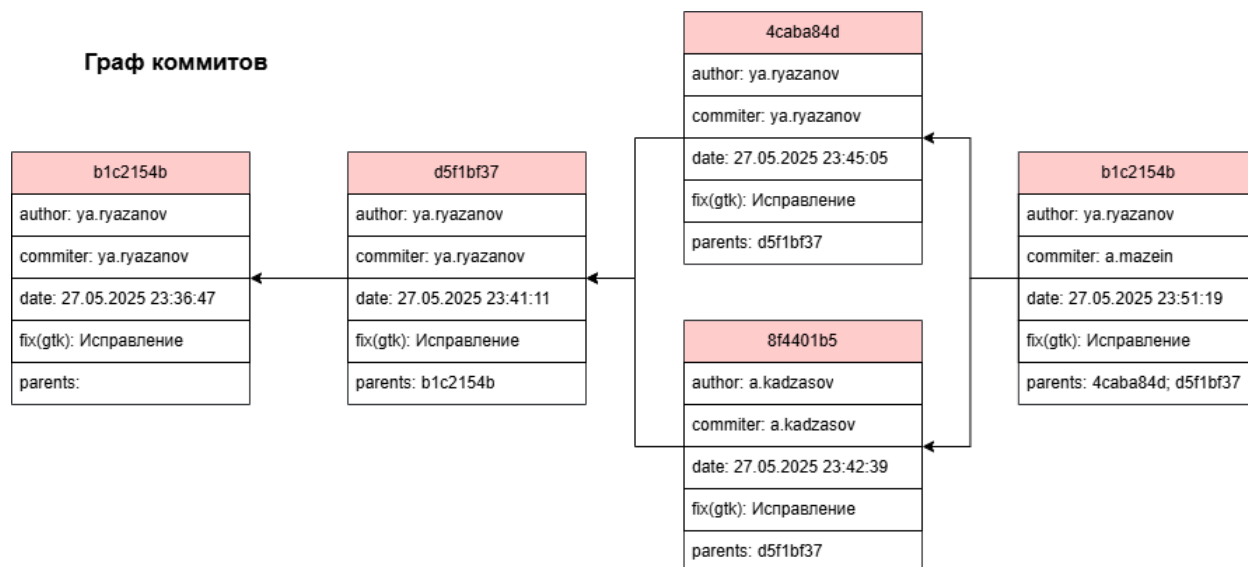
Коммит представляет собой логически завершённый набор изменений, который должен быть функционально целостным.

3. Ссылочная целостность

Каждый коммит (кроме начального) ссылается на один или несколько родительских коммитов, формируя граф.

Коммит и ветвление

История Git представляет собой направленный ациклический граф, где коммиты являются узлами, а связи между ними - рёбрами, указывающими на предков.



На изображении выше отображена типичная структура коммитов с демонстрацией их взаимосвязей и ключевых атрибутов.

Ссылки

Ссылки в Git — механизм для именованного указания на конкретные коммиты в истории репозитория.

Тег

Тег — статическая именованная ссылка, указывающая на конкретный коммит.

Используется для маркировки значимых моментов в развитии проекта, таких как релизы.

Типы тегов

- легковесные (обычные) теги — простые указатели на коммиты;
- аннотированные теги — полноценные объекты Git со следующими метаданными:
 - наименование тега;
 - сообщение с описанием;
 - информация об авторе;
 - дата создания.

Ветка

Ветка — динамический указатель на коммит, который автоматически перемещается при создании новых коммитов.

Особенности веток

- динамичность — ветка всегда указывает на последний коммит в цепочке;
- изоляция — позволяет вести параллельную разработку (локальные и удалённые ветки);
- лёгкость — ветки являются лёгковесными объектами без метаданных.

Ветвление веток

Ветвление происходит путём создания новой ветки от существующего коммита.

Ветвление истории коммитов и создание веток-указателей — разные, но связанные концепции.

HEAD и detached HEAD

- HEAD — специальная ссылка, указывающая на текущий активный коммит (обычно через ветку);
- detached HEAD — состояние, когда HEAD указывает непосредственно на коммит, а не на ветку.

Ветка

Ветка (**branch**) представляет собой изолированную линию разработки в системе контроля версий. Ее основное назначение — обеспечить возможность параллельного внесения и тестирования изменений без воздействия на стабильное состояние основного кода. Ниже приведены ключевые аспекты использования веток.

1. Изоляция изменений. Ветки обеспечивают изоляцию процессов разработки. Все изменения, вносимые в рамках отдельной ветки, не оказывают влияния на другие ветки до момента выполнения операции слияния (**merge**). Это позволяет вести разработку и экспериментирование, не затрагивая стабильную версию проекта, расположенную в основной ветке.
2. Организация параллельной разработки. Использование веток является стандартным методом для организации параллельной работы над несколькими функциональностями или исправлениями. Каждая задача может выполняться в собственной ветке, что исключает конфликты между усилиями различных разработчиков на ранних стадиях и позволяет независимо управлять жизненным циклом каждого изменения.

Правила наименования веток

Основные ветки используют фиксированные имена: `main`, `master`, `release-candidate`, `release`, `dev` и другие.

Ветки для задач именуются по формату: `Group/[SubGroup]/Label/Some-name-[Issue]`.

Group - обозначение группы веток:

- **wip** (Works in progress) - для долгосрочных задач с неопределенным сроком завершения.
Пример: `wip/feat/config-manager-203636t`.
- **iss** (Issue) - для краткосрочных веток, связанных с конкретной задачей.
Пример: `iss/fix/db-generator-update-class-exist-field-210782t`
- **junk** - для экспериментальных разработок. Подлежит обязательному удалению после завершения экспериментов.
Пример: `junk/chore/update-sbt-version`
- **usr/<user_name>** - для персональных экспериментальных веток.
Пример: `usr/ya.ryazanov/chore/java-21-support`

[SubGroup] - опциональная подгруппа.

Пример: `usr/ya.ryazanov/wip/java-21-support`, где `wip` является подгруппой.

Label - тип изменений в ветке:

- **feat** - реализация нового функционала.
- **fix** - исправление ошибок.
- **hotfix** - оперативное исправление критических ошибок.
- **test** - добавление тестов.
- **refactor** - рефакторинг кода без изменения функциональности.
- **docs** - обновление документации.
- **chore** - прочие изменения, не подпадающие под указанные категории.

Some-name-[Issue] - краткое описание и номер задачи.

Совет

Наименование должно быть на английском языке и в нижнем регистре, что бы избежать конфликтов при работе на разных операционных системах.

Слияние веток

Слияние веток - это процесс интеграции изменений из одной ветки в другую, позволяющий объединить параллельно развивающиеся линии разработки.

Типы слияния

Fast-Forward слияние

Происходит, когда целевая ветка не содержит изменений, отсутствующих в сливаемой ветке. Git просто перемещает указатель целевой ветки вперёд.

Условия:

- Отсутствуют расхождения в истории.
- Целевая ветка является прямым предком сливаемой.

Преимущества:

- Сохраняется линейная история.
- Не создаётся дополнительных коммитов.

Recursive слияние (трехстороннее)

Применяется, когда обе ветки содержали изменения после момента расхождения. Git создаёт новый коммит слияния, объединяющий изменения.

Характеристики:

- Создаётся коммит слияния с двумя родителями.
- Сохраняется полная история изменений.
- Требует разрешения конфликтов при необходимости.

Слияние с отклонением fast-forward (`-no-ff`)

Принудительное создание коммита слияния даже когда возможно fast-forward слияние.

Применение:

- Сохранение информации о факте слияния в истории.
- Улучшение читаемости истории проекта.
- Отслеживание функциональных веток.

Процесс слияния

1. Определение базового коммита - поиск общего предка веток.
2. Анализ различий - сравнение изменений в обеих ветках.
3. Автоматическое объединение - применение непротиворечивых изменений.
4. Разрешение конфликтов - ручное устранение противоречий (при необходимости).
5. Создание коммита слияния - фиксация результата объединения.

Стратегии слияния

- **ort** (По умолчанию) - оптимизированная стратегия для сложных слияний.
- **resolve** - трёхстороннее слияние с разрешением конфликтов.
- **octopus** - для слияния более двух веток одновременно.

Лучшие практики

- Регулярное слияние изменений из основной ветки в feature-ветки.
- Использование **--no-ff** для сохранения информации о слиянии feature-веток.
- Тщательное тестирование после слияния.
- Использование Merge Request для код-ревью перед слиянием.

Команды слияния

```
# Слияние с fast-forward (по умолчанию)
git merge feature-branch

# Слияние с созданием коммита слияния
git merge feature-branch --no-ff

# Прерывание слияния в случае конфликтов
git merge --abort
```

Merge request

Merge Request (MR) — запрос на слияние изменений из одной ветки репозитория в другую. Это ключевой механизм в системах контроля версий (таких как GitLab), предназначенный для проверки, обсуждения и интеграции кода перед его включением в основную ветку проекта.

Порядок выполнения Merge Request от разработчика:

- Создает отдельную ветку для задачи (фичи, исправления).
- Формирует запрос на слияние этой ветки с целевой (например, main, develop).
- Организует проверку кода (code review) командой.
- Вносит правки на основе комментариев.

После одобрения и успешных автоматических проверок (тесты, сборка) изменения сливаются в целевую ветку.

Зачем он нужен

- **Контроль качества:** Главная цель — коллективный просмотр кода (code review). Другие разработчики могут найти ошибки, предложить более оптимальные решения и убедиться, что код соответствует стандартам проекта.
- **Обмен знаниями:** Команда видит, какие изменения вносятся в кодовую базу, и узнает о новых функциях или исправлениях.
- **Обсуждение:** Платформа для обсуждения архитектурных решений прямо в контексте кода.
- **Автоматизированные проверки:** Запуск CI/CD пайплайна (тесты, линтеры, сборка) гарантирует, что изменения не сломают сборку или функционал.
- **Документирование:** История всех обсуждений и утверждений изменений сохраняется и привязана к коду.

Процесс работы с Merge Request

Шаг 1: Подготовка перед созданием MR

- **Локально:** Убедитесь, что ваша ветка готова к мержу.
- **Запустите ветку:** Отправьте свою ветку в удаленный репозиторий.

Шаг 2: Создание Merge Request

1. Перейдите на страницу вашего репозитория на GitLab/GitHub.
2. Система сама предложит создать MR/MR для недавно запущенной ветки.



3. Заполните форму:
 - **Title:** Заголовок MR.
 - **Description:** Описание. Что было сделано? Зачем? Какие ключевые изменения? Прикрепите скриншоты, если это UI.
 - **Assignee:** Назначьте ответственного (тимлида или коллегу) на ревью.
 - **Reviewers:** Укажите нужных людей для ревью.
 - **Labels:** Добавьте соответствующие метки (bug, feature, frontend, backend).

Шаг 3: Ревью и процесс обсуждения

1. Ревьюеры изучат ваш код, оставят комментарии (общие или привязанные к конкретной строке кода). Комментарии бывают:
 - Comment: Просто вопрос или замечание.
 - Approval: Одобрение изменений.
2. Если нужны правки:
 1. Внесите правки локально в ту же ветку.
 2. Сделайте коммит (с понятным сообщением, например, fix: Учтены правки ревью #issue 123456T).
 3. Запустите изменения. MR обновится автоматически.
3. Обсуждение продолжается до тех пор, пока все правки не будут учтены и не будет получено одобрение (approve).

Шаг 4: Слияние (Merge)

После получения одобрений и успешного прохождения всех CI-проверок (зеленая галочка) вы можете выполнить слияние нажав кнопку Merge.

 **Ready to merge!**

☒ Delete source branch ☐ Squash commits  ☐ Edit commit message

- The source branch is **3 commits behind** the target branch. [Rebase source branch](#)
- 1 commit and 1 merge commit will be added to main.

Merge

- Delete source branch: Автоматическое удаление ветки, из которой был создан Merge Request, сразу после успешного слияния. Это помогает поддерживать репозиторий в чистоте, избавляя его от устаревших веток. Рекомендуется включать эту опцию всегда, если вы не планируете продолжать работу в этой ветке.
- Squash commits: Объединяет все коммиты из вашей ветки в один новый коммит перед слиянием в целевую ветку. Это упрощает историю проекта, делая ее линейной и легко читаемой, и скрывает промежуточные, технические или «черновые» коммиты. Идеально подходит для задач, где было сделано множество мелких коммитов.
- Edit commit message: Редактирование сообщения финального коммита, который будет создан при слиянии. Если вы используете «Squash commits», это сообщение станет описанием всего объединенного изменения. Если нет — оно будет применено к коммиту слияния. Рекомендуется включать эту опцию при использовании «Squash commits».

Идеальный Merge Request

- Содержит небольшие, логически завершенные изменения. (Большие MR сложно ревьюить).
- Имеет четкий заголовок и описание, объясняющее мотивацию изменений.
- Вся история коммитов в ветке тоже написана по правилам.
- Связан с задачей в issue tracker (например, Jira, YouTrack).
- Включает в себя тесты для нового функционала или исправлений.
- Не ломает существующие тесты (CI пайплайн успешен).
- Не содержит закомментированного «мусорного» кода.
- Соответствует code style проекта.
- Следование этому процессу делает разработку предсказуемой, качественной и командной.

Ответы на частые вопросы

Примеры рабочих процессов

Разработка несрочного функционала или исправлений

```
# Переход в целевой модуль
cd gtk

# Переключение на основную ветку разработки
git switch main

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Создание ветки для нового функционала
git switch -c "iss/feat/create-any-reference-123123T"

# ИЛИ создание ветки для исправления
git switch -c "iss/fix/bug-in-any-reference-123123T"
```

Внимание

Перед созданием ветки разработки обязательно:

1. Переключиться на ветку main командой `git switch`.
2. Синхронизировать локальную копию командами `git fetch` и `git pull`.

После внесения изменений необходимо:

1. Создать коммиты.

2. Отправить изменения в удаленный репозиторий через `git push`.
3. Сформировать merge request в ветку `main`.

Экстренные исправления (hotfix) или срочный функционал для релизной ветки

```
# Переход в целевой модуль
cd gtk

# Переключение на стабильную ветку
git switch release

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Создание ветки для hotfix
git switch -c "iss/hotfix/fix-any-reference-123123T"

# ИЛИ создание ветки для срочного функционала
git switch -c "iss/feat/urgent-functionality-123123T"
```

Внимание

Перед созданием ветки разработки обязательно:

1. Переключиться на ветку `release`.
2. Синхронизировать локальную копию.

После внесения изменений необходимо:

1. Создать коммиты.
2. Отправить изменения в удаленный репозиторий.
3. Сформировать merge request в ветку `release`.

Копирование экстренных исправлений в основную ветку разработки

Для переноса используется команда `git cherry-pick -x`.

Внимание

Обязательно используйте параметр `-x`, который добавляет в сообщение коммита строку `cherry picked from commit <hash>`.

Копирование одиночного коммита

Скопируйте `hash` коммита из GitLab'a или IDE.

```
# Переход в целевой модуль
cd gtk

# Переключение на основную ветку разработки
git switch main

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Создание ветки для копирования исправления
git switch -c "iss/hotfix/cp-functionality-123123T"

# Копирование коммита
git cherry-pick -x <hash>
```

При возникновении конфликтов:

1. Разрешите их (например, средствами IDEA).
2. Создайте merge request в ветку main.

Перенос диапазона коммитов

Скопируйте `hash` первого и последнего коммитов из GitLab'a или IDE.

```
# Переход в целевой модуль
cd gtk

# Переключение на основную ветку разработки
git switch main

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Создание ветки для копирования исправления
git switch -c "iss/hotfix/cp-functionality-123123T"

# Скопировать коммит
git cherry-pick -x <start_hash>^..end_hash
```

Внимание

Запись `<start_hash>^..end_hash` включает все коммиты в интервале `[start_hash; end_hash]`.

Без символа `^` диапазон становится `(start_hash; end_hash]`.

При возникновении конфликтов:

1. Разрешите его (например, средствами IDEA).
2. Продолжите копирование командой: `git cherry-pick --continue`.
3. Создайте merge request в ветку main.

Перенос изменений в релизную ветку через rebase

Если ваши изменения попали в основную ветку разработки `main`, но они так же должны переехать в `release`, то не обязательно использовать команду `cherry-pick`, можно воспользоваться ребейсом. Для этого необходимо использовать команду `git rebase`. Для начала необходимо скопировать коммит, от которого была создана ветка разработки.

Внимание

Учтите, что при ребейсе в ветке разработки у коммитов будет изменён хэш. Если вы не один используете данную ветку, то у других она сломается.

Если вы сомневаетесь, что сможете сделать правильно, лучше использовать `cherry-pick`.

```
# Переход в целевой модуль
cd gtk

# Переключение на основную ветку разработки
git switch main

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Переключение на релизную ветку
git switch release

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Переключение на ветку с задачей
git switch feature

# Выполнение rebase, где <hash> - коммит, от которого была создана ветка feature
git rebase --onto release <hash> feature
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
# При конфликтах:
#1. Разрешите конфликты
#2. Продолжите процесс: git rebase --continue

# Принудительная отправка изменений (с перезаписью истории)
git push --force-with-lease
```

Актуализация долгосрочных веток

Если у вас есть ветка `wip/feat/...`, то её необходимо периодически актуализировать. Для этого лучше всего подойдёт `rebase`, так как он, в отличие от `merge` - не создаёт дополнительные мусорные коммиты с решением конфликтов.

```
# Переход в целевой модуль
cd gtk

# Переключение на ветку с задачей
git switch wip/feat/task-123123T

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Ребазирование на актуальную main
git pull origin main --rebase

# Принудительная отправка изменений
git push --force-with-lease
```

Организация совместной разработки

При совместной разработке необходимо создать ветку под задачу:

```
# Переход в целевой модуль
cd gtk

# Получение актуальных изменений из удаленного репозитория
git fetch

# Синхронизация локальной копии с удаленным репозиторием
git pull

# Создание ветки под совместную разработку
git switch -c wip/feat/task-123123T
```


Создание индивидуальных веток

Каждый разработчик от этой ветки создаёт ветку под свою часть задачи.

```
# Создание ветки под задачу от общей ветки
git switch -c wip/feat/task-part1-123123T`
```

Синхронизация изменений

Перед продолжением работы в ветке, её необходимо актуализировать:

```
# Переход в целевой модуль
cd gtk

# Переключение на ветку с задачей
git switch wip/feat/task-part1-123123T

# Синхронизация локальной копии с удалённым репозиторием
git fetch origin

# Ребазирование на актуальную общую ветку разработки
git rebase wip/feat/task-123123T

# Принудительная отправка изменений
git push --force-with-lease
```

Интеграция изменений в основную ветку

```
# Переход в целевой модуль
cd gtk

# Переключение на общую ветку с задачей
git switch wip/feat/task-123123T

# Получение актуальных изменений из удалённого репозитория
git fetch

# Синхронизация локальной копии с удалённым репозиторием
git pull

# Попытка смержить свою ветку (при конфликтах merge - не пройдёт)
git merge --ff-only wip/feat/task-part1-123123T
```

Если смёржить не получилось, то в общую ветку кто-то успел слить свои изменения и необходимо ещё раз выполнить синхронизацию изменений.

Приложение

Global GitFlow

Global GitFlow — модель ветвления Git. Данная модель регламентирует процесс разработки, устанавливая четкие правила создания и именования веток, а также процедуры слияния и создания merge request'ов (MR).

Внимание

Настоящее руководство применяется исключительно для модулей gtk и btk.

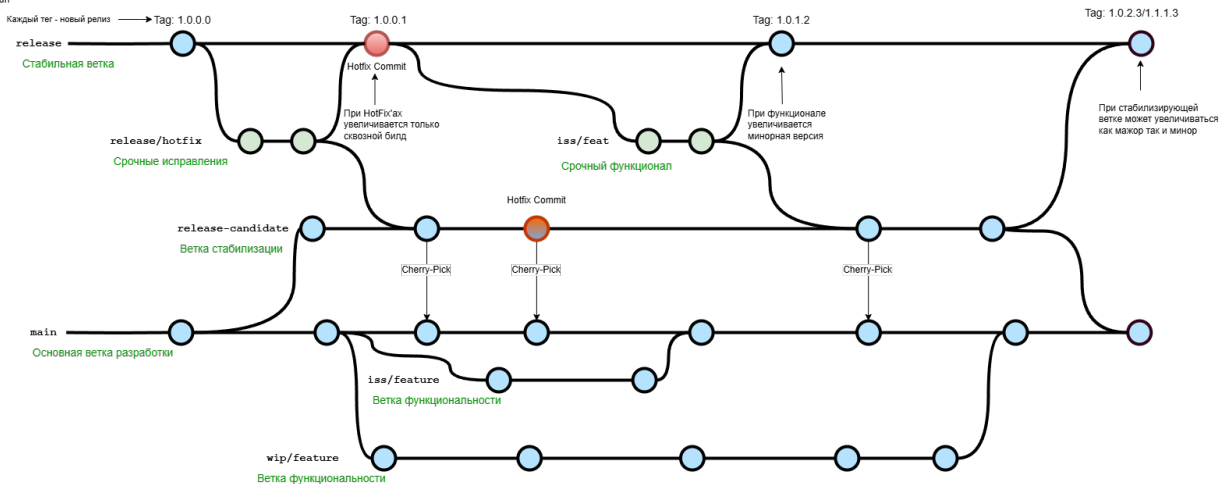
Рекомендации по настройке окружения

Для обеспечения эффективной работы рекомендуется использовать редактор VS Code в качестве стандартного git-редактора. Установка производится следующей командой:

```
git config --global core.editor "code --wait"
```

"Global-Git-Flow" :

See: url



Основные ветки

В основные ветки запрещены прямые пуши. Все изменения сливаются через Merge Request'ы (MR) и проходят Code Review. Все новые изменения или доработки существующего функционала сначала отправляются в main ветку, после чего, если необходимо перенести их в последующие ветки, создаются ветки переноса, куда черри-пикаются изменения. Уже от этих веток изменения сливаются в release-candidate и в release.

- **release** — ветка production-окружения. Содержит исключительно стабильный код, соответствующий выпущенным версиям продукта. Обновляется только через слияние ветки release-candidate или черри-пиками хотфиксов.
- **release-candidate** — ветка стабилизации релиза. Формируется как ответвление от main. Подвергается недельному циклу тестирования и исправлению обнаруженных ошибок. После чего изменения сливаются в ветку release.

- **main** — основная ветка разработки. Используется для интеграции нового функционала. Все **feature**-ветки создаются от **main** и в нее же сливаются посредством **MR**.

Вспомогательные ветки

- **feature** — ветка для разработки нового функционала в рамках конкретной задачи. Создается исключительно от ветки **main** и в нее же сливается посредством **MR**.
- **hotfix** — ветка для экстренных исправлений. Создается от ветки **main**, в нее же сливается, после чего изменения переносятся в ветки **release-candidate** и (или) **release**, посредством черри-пиков.
- **pick** — ветки переноса изменений между ветками. При необходимости переноса изменений между ветками, именно в данную ветку **черри-пикаются** изменения и от неё же создаётся **MR**.

Совет

Правила наименования веток

Тестирование веток

1. **main** — основная ветка разработки тестируется на астре **smtu-test**.
2. **release-candidate** — кандидат на релиз, тестируется на астре **pgDev**.
3. **release** — стабильная версия, не тестируется, так как с неё выпускаются релизы.

Процесс выпуска релизов

Выпуск релизов происходит еженедельно, каждый понедельник в 16:00. Сначала ветка **release-candidate** сливается в **release**, после чего актуализируется посредством слияния в неё ветки **main**. После слияния выпускаются релизы модулей по ветке **release**.

Организация разработки

Перед началом работы необходимо убедиться, что ваша локальная ветка **main** актуальна и синхронизирована с удалённым репозиторием. Только после этого можно создавать новую рабочую ветку.

Для новых функциональных изменений создаётся ветка типа **feature**, для критических исправлений — **hotfix**. Название ветки должно однозначно идентифицировать задачу, включая её идентификатор в системе учёта задач.

В процессе работы рекомендуется регулярно синхронизировать свою рабочую ветку с актуальным состоянием ветки **main**. Это помогает минимизировать потенциальные конфликты при последующем слиянии. Все коммиты должны содержать понятные сообщения, позволяющие однозначно определить суть изменений.

Изменения в рабочую ветку следует вносить атомарными коммитами, каждый из которых представляет логически завершённое изменение. Это упрощает процесс ревью и анализ истории изменений.

Совет

Правила наименования коммитов

После завершения всех изменений по задаче необходимо создать Merge Request (MR) в ветку `main`. MR должен содержать исчерпывающее описание внесённых изменений, включая:

- цель и мотивацию изменений;
- описание реализации;
- номер задачи;
- ссылки на MR.

В процессе **Code Review** могут возникнуть замечания, требующие внесения дополнительных изменений. Все правки вносятся в ту же рабочую ветку и автоматически появляются в существующем MR.

После успешного слияния MR с веткой `main` может потребоваться перенос изменений в другие ветки (например, в `release-candidate` или `release`). Для этого используются специальные **pick**-ветки, создаваемые на основе целевой ветки. Изменения переносятся **черри-пиками**, которые позволяют выборочно применять коммиты из одной ветки в другую.

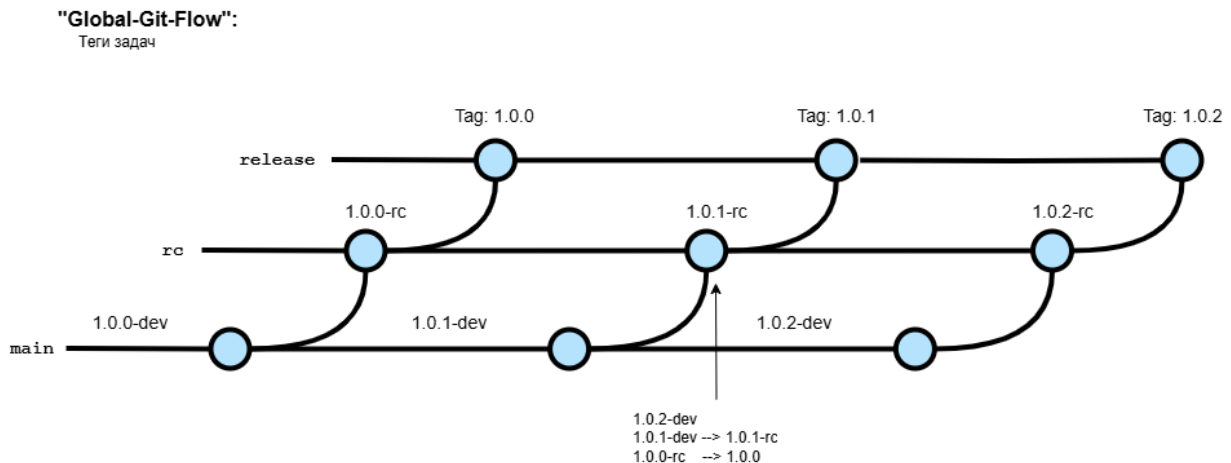
При переносе изменений в ветку `release` обязательно нужно также перенести их в соответствующую ветку `release-candidate`. Для каждого переноса создаётся отдельный MR с указанием исходного MR в `main`. Это обеспечивает прослеживаемость изменений между ветками.

Заключение

Соблюдение описанной модели ветвления обеспечивает:

- Стабильность production-окружения.
- Контролируемый процесс разработки.
- Эффективное управление релизами.
- Минимизацию конфликтов при слиянии кода.

Выпуск релизов ядровых модулей



Версионирование релизов

Версия состоит из пяти разрядов, где последний не является обязательным:

`generation.major.minor.build.patch`.

Версии формируются по следующим правилам:

- **Поколение** — Крупные изменения, которые ломают обратную совместимость. Старшая версия может означать полное переписывание кода. Так же в этом типе релиза происходит очистка миграционных скриптов предыдущей версии. Устанавливается только на предыдущее поколение.
- **Мажор** — Крупные изменения, которые могут ломать обратную совместимость. К примеру переключение сервера приложений или обновление SBT плагина.
- **Минор** — Плановые изменения в рамках развития модуля, а также изменение конфигурации модулей проекта (добавление, удаление модулей).
- **Билд** — Сквозная нумерация. Увеличивается при выпуске любой из версий, не считая патчей.
- **Патч** — Изменения критических ошибок на уже выпущенных версиях. От патчей нельзя создавать другие версии, кроме последующих патчей.

Пример обычной версии: 1.4.25.612.

Пример версии с патчем: 1.4.25.612.1.

Управление задачами через релизы в Support

Билд-версии не учитываются, так как являются сквозными и увеличиваются вместе с любой другой версией, не считая патча.

Для удобства наименования в примерах разряды будут помечаться:

- Поколение [Generation] - X.
- Мажор [Major] - Y.
- Минор [Minor] - Z.

Релизы в Support используются для отслеживания состояния функциональности:

- X.Y.Z-dev - задача находится в разработке (ветка `main`).
- X.Y.Z-rc - задача в кандидате на релиз (ветка `release-candidate`).
- X.Y.Z - задача выпущена в стабильной версии (ветка `release`).

Процесс выпуска релизов

Выпуск релизов происходит еженедельно, каждый понедельник в 16:00.

1. Слияние `release-candidate` в `release`. После слияния, релиз в Support X.Y.Z-rc переименовывается в X.Y.Z.
2. Слияние `main` в `release-candidate`. После слияния, релиз в Support X.Y.Z-dev переименовывается в X.Y.Z-rc. Также создаётся новый тег задач X.Y.(Z + 1)-dev.

После слияния веток выпускаются релизы модулей.

Внимание

Релизы в Support относятся к задачам в менеджере задач и не являются git-тегами, не считая `release` ветки.

Примеры жизненного цикла версии

1. Начало разработки версии 1.0.2:

- Ветка `main`: 1.0.2-dev.
- Ветка `release-candidate`: 1.0.1-rc.
- Ветка `release`: 1.0.0.

2. После слияния `release-candidate` в `release`:

- Ветка `main`: 1.0.2-dev.
- Ветка `release-candidate`: 1.0.2-rc.
- Ветка `release`: 1.0.1 (бывшая 1.0.2-rc).

3. После слияния `main` в `release-candidate`:

- Ветка `main`: 1.0.3-dev (новая версия для разработки).
- Ветка `release-candidate`: 1.0.2-rc (бывшая 1.0.2-dev, автоматически переименовывается в Support при выпуске релизов).
- Ветка `release`: 1.0.1.

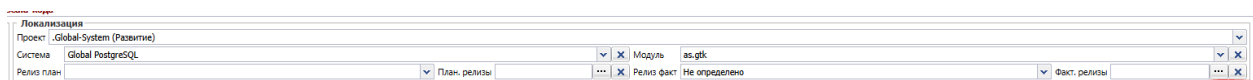
Организация разработки

После выполнения задачи и создания МР, необходимо установить тег к задаче, к примеру 1.0.3-dev. По мере слияния веток и выпуска релизов, он будет переименован и в конечном итоге станет 1.0.3. Таким образом все смогут отслеживать, когда и в каком релизе будет выпущена задача.

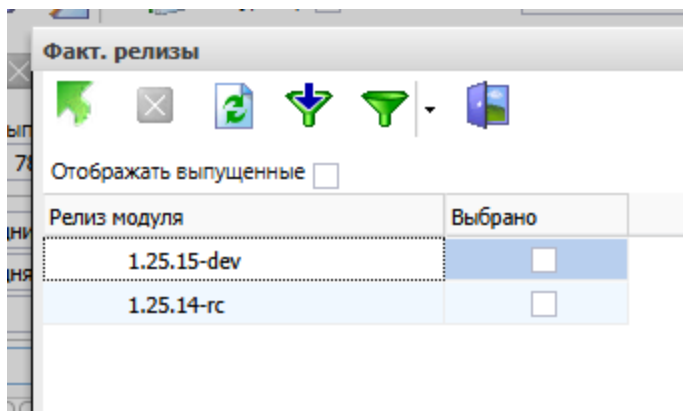
Внимание

При необходимости черри-пикать ваш код в ветки `release-candidate` или `release`, необходимо указывать корректные версии в релизах к задаче.

Релиз выставляется в поле «фактический релиз».



Локализация			
Проект	Global-System (Развитие)		
Система	Global PostgreSQL		
Модуль	as.git		
Релиз план	План, релизы	Релиз факт	Не определено



Если вы видите в **Support**, что на задаче стоит версия с постфиксом **dev**, значит она попадёт в **release** через один понедельник.

Иначе, если версия с постфиксом **rc**, то в **release** будет в следующий понедельник.

Если же постфиксов нет, то задача уже находится в **release** под соответствующим тегом.

Выпуск тегов ядровых модулей

Расписание

Каждый понедельник после 16:00 по МСК.

Подготовка

Сделать объявление в чат Kernel Bites Perm о времени выпуска, что бы мейнтейнеры не принимали MR.

Процедура выпуска

1. Актуализация веток

Для каждого модуля (gtk и btk):

```
cd <module>
git switch main && git fetch && git pull
git switch release-candidate && git fetch && git pull
git switch release && git fetch && git pull
```

2. Проверка сборки

Убедитесь, что проект собирается на всех ветках: **main**, **release-candidate**, **release**).

3. Слияние веток

Внимание

Обязательно актуализируйте ветки, перед этим шагом!

```
git switch release
git merge --no-ff release-candidate
```

При конфликтах - разрешить их вручную (через `terminal` или `IDEA`).

4. Публикация изменения

Проверьте сборку на релизных ветках и запустите изменения, после чего актуализируйте ветку `release-candidate`:

```
git switch main

git branch -d release-candidate
git push origin --delete release-candidate

git switch -c release-candidate
git push -u origin release-candidate
```

После слияния `rc` в `release` для каждого модуля, необходимо проверить сборку. Если проект собирается, можно пушить изменения в `release`.

После пуша, необходимо актуализировать ветку `release-candidate`, для этого надо её удалить и создать заново от `main`:

```
git switch main

git branch -d release-candidate
git push origin --delete release-candidate

git switch -c release-candidate
git push -u origin release-candidate
```

После актуализации кода необходимо выпустить минорные версии модулей (именно минорные) через `gsf-cli` или `IDEA`.

Завершение выпуска

Сообщить в чат `Kernel Bites Perm`, что выпуск завершён и укажите новые версии модулей.

После чего актуализируйте в саппорте фактические релизы по [документации](#).

Развёртывание рабочего места

Под Windows

Рабочее место под Windows с редактором исходного кода IntelliJ IDEA Community Edition.

Необходимые компоненты

- IntelliJ IDEA Community Edition;
- Scala Plugin;
- Global3FrameworkStarterKit.

Установка

1. Установите IntelliJ IDEA Community Edition.
2. Установите в IDE IntelliJ IDEA плагин **Scala**.
3. Запустите IntelliJ IDEA и установите плагин **Global3se-plugin-1.0-SNAPSHOT**, поставляемый с **Global3FrameworkStarterKit** (подробнее по установке плагина).
4. Перезапустите IntelliJ IDEA и на главной странице перейдите на вкладку **Global Framework**.
5. Установите JDK 1.8.
6. Установите шаблоны IntelliJ IDEA (.idea-файлы).
7. Установите **gsf-cli** в директорию `C:\programs\gsf-cli`.
8. Установите SBT версии 1.8.2 в директорию `C:\programs\sbt`.
9. Установите Git.
10. Установите дополнительные инструменты.

Настройка проекта на примере pgDev

Настраиваем проект с использованием **gsf-cli** (подробнее об утилите командной строки **Global System Framework**).

1. Для добавления нового проекта запустите скрипт `add_project.cmd`, расположенный по адресу: `C:\programs\gsf-cli\links\add_project.cmd`.
 - Имя проекта: **pgDev**.
 - Укажите путь к JDK (не JRE).
 - Путь к репозиторию: `https://extgit.global-system.ru/appdev/internal/pgdev.git` (вставка из буфера обмена — правой кнопкой мыши).
 - Остальные параметры оставьте по умолчанию.
 - Когда будет предложено **запустить IDEA**, выберите **Да**.
После завершения экспорта закройте IDEA и выберите **Сконфигурировать IDEA** → **Да**.
2. Создайте ярлык на рабочем столе `start_idea_pgDev`, ссылающийся на:
`C:\programs\gsf-cli\workspace\links\pgDev\start_idea.cmd`, и запускайте IDEA через него.
3. В настройках **File | Settings | Build, Execution, Deployment | Compiler | Scala Compiler | Scala Compile Server**:
 - Установите **Maximum heap size, MB**:
 - 16384, если объём ОЗУ => 32 ГБ;
 - 8192, если объём ОЗУ = 16 ГБ.

- Включите опцию **Compile independent modules in parallel, in up to** и укажите значение 6.
4. Аналогично задайте **Maximum heap size, MB** в настройках:
File | Settings | Build, Execution, Deployment | Build Tools | sbt.
 5. В файле **Help | Edit Custom VM Options** установите параметр **-Xmx**:
 - **-Xmx16384m**, если объём ОЗУ => 32 ГБ;
 - **-Xmx8192m**, если объём ОЗУ = 16 ГБ.
 6. Соберите проект (**Ctrl+F9**).
 7. На панели **sbt** для проекта PGDEV выполните задачу **publishDevDependencies**.
 8. Настройте схемы согласно инструкции:
Подключение XSD-схем к редактору кода.
 9. Настройте подключение к PostgreSQL в DBeaver:
 - Хост: **pgDevDB.gs.local**
 - Порт: **5432**
 - База данных: **pgdev**
 - Пользователь: **pgdev**
 - Пароль: **pgdev**

Совет

Описание настройки проекта с базой разработки

Под Astra Linux

Рабочее место под Astra Linux с редактором исходного кода GigaIDE.

Необходимые компоненты

- Astra Linux;
- GigaIDE 2024.3;
- Global-ERP-Toolkit-GigaIde;
- GSF-CLI for Linux;
- SBT 1.8.2;
- JDK 8.

Установка

1. Установите Astra Linux.
2. Обновите компоненты системы и установите Git. Для этого выполните в терминале следующие команды:

```
su -  
apt-get update  
apt-get install git
```

3. Создайте папку **Global** в домашней директории пользователя.
4. В созданной директории создайте поддиректорию **GigaIDE** и распакуйте в неё дистрибутив GigaIDE.
5. В директории из пункта 3 создайте поддиректорию **gsf-cli** и распакуйте в неё дистрибутив GSF-CLI.
6. В директорию из пункта 3 создайте поддиректорию **sbt** и распакуйте в неё дистрибутив SBT версии 1.8.2.
7. Установите дистрибутив JDK 8 (**bellsoft-jdk8u432+7-linux-amd64-full.deb**) по ссылке выше.
8. Чтобы Git не запрашивал пароль при каждом обращении, выполните следующую команду и перезапустите терминал:

```
git config --global credential.helper store
```

9. Запустите GigaIDE и установите плагин **Global-ERP-Toolkit-GigaIde**. Исполняемый файл для запуска GigaIDE расположен по адресу:
/home/name_user/Global/gigaide-CE-242.21829.142.2/bin/idea.
10. Перезапустите GigaIDE и на главной странице перейдите на вкладку **Global Framework**.
11. Установите шаблоны: в пункте **idea files** нажмите кнопку **Применить** и закройте GigaIDE.
12. Выполните следующие действия для установки необходимых зависимостей:

- Необходимо раскомментировать строки с репозиториями в файле, расположенном по адресу:
/etc/apt/sources.list.
- Выполняем команды в терминале:

```
sudo apt update  
sudo apt install python3.11-venv
```

- Запускаем скрипт, расположенный по адресу: **gsf-cli/bin/initenv.sh.**

13. Выкачать проект с помощью команд:

```
cd /home/admin/global/gsf-cli/links  
./add_project.sh
```

Внимание

Путь до jdk: **/usr/lib/jvm/bellsoft-java8-full-amd64.**
Путь до IDEA: **/home/admin/global/GigaIDE.**
Путь до sbt: **/home/admin/global/sbt.**

Под Alt Linux

Рабочее место под Alt Linux с редактором исходного кода GigaIDE.

Необходимые компоненты

- Alt Linux Workstation 10;
- GigaIDE 2024.2;
- GSF-CLI for Linux;
- SBT 1.8.2;
- JDK 8.

Установка

1. Установите Alt Linux.
2. Обновите компоненты Linux. Для этого необходимо. выполнить следующие команды в терминале:

```
su -  
apt-get update  
apt-get install git  
apt-get install subversion  
apt-get install python  
apt-get install pip
```

3. Создайте папку Global в директории пользователя.
4. В директории из предыдущего пункта необходимо создать поддиректорию GigaIDE и разархивировать в неё дистрибутив GigaIDE.
5. В директории из пункта три необходимо создать поддиректорию gsf-cli и разархивировать в неё дистрибутив GSF-CLI.
6. В директории gsf-cli необходимо создать папку python, а затем выполнить команду:

```
python3 -m venv /home/admin/global/gsf-cli/python  
source /home/admin/global/gsf-cli/python/bin/activate  
pip3 install -r /home/admin/global/gsf-cli/requirements.txt
```

Внимание

Вместо пути `~/home/admin/global` указать путь до вашей папки `global`.

7. В директорию из пункта три необходимо создать поддиректорию sbt и разархивировать в неё дистрибутив sbt версии 1.8.2.
8. Установите дистрибутив JDK 8 (bellsoft-jdk8u432+7-linux-amd64.tar.gz) по ссылке выше. И проведите установку дистрибутива с помощью следующих команд:

```
su -  
tar -zxvf bellsoft-jdk8u432+7-linux-amd64.tar.gz  
-C /usr/lib/jvm/
```

9. Что бы git каждый раз не спрашивал пароль заново, необходимо ввести следующую команду и перезапустить терминал:

```
git config --global credential.helper store
```

10. Выкачать проект с помощью команд:

```
cd /home/admin/global/gsf-cli/links  
./add_project.sh
```

Внимание

Путь до jdk: /usr/lib/jvm/jdk8u432.
Путь до IDEA: /home/admin/global/GigaIDE.

11. Добавить [GlobalERP Tool Kit](#) плагин для GigaIDE.

Ревью Кода Scala/SQL

Руководство по Ревью Кода Scala/SQL (GlobalFramework)

Это руководство служит рекомендацией для начинающих ревьюеров, указывая, на что необходимо обращать внимание при выполнении ревью кода. Оно содержит рекомендации, собранные разработчиками команды, и направлено на **повышение качества, производительности и стабильности кода**. Документ охватывает общие принципы, архитектурные и стилистические нормы, а также важные аспекты производительности и безопасности.

1. Метаописания и Структура Классов

Именованние классов

Имя класса должно быть уникальным, на латинице в формате <Модуль>_<Имя> (единственное число, именительный падеж, например, `Lbr_Book`). Имена сущностей регистрозависимы и должны точно совпадать с ожиданиями фреймворка (CamelCase).

Цель: Обеспечить ясность структуры, избежать конфликтов, гарантируя корректную работу фреймворка и улучшить читаемость кода.

Именованние атрибутов

Атрибуты должны иметь осознанные имена, без транслита, соответствующие содержанию поля (camelCase).

Цель: Улучшить читаемость и документировать структуру данных, соответствовать стилю Scala и GSF.

Типы атрибутов

Для каждого поля должно быть указано имя, тип данных и связь. Простые поля используют типы `NLong`, `NNumber`, `NString`, `NDate` и т.д.; ссылочные поля — `Reference`, `VariableReference` или `GID`. Проверьте, что все `Reference` указывают на существующий класс и тип правильный (например, `GID` для переменной ссылочности). Типы полей должны быть корректно указаны и соответствовать БД (например, `NLong`, `NNumber`, `NBigDecimal`, `NString`, `NGid`, `NDate` и т.п.).

Цель: Обеспечить корректную работу с `null`-значениями, безопасность типов, интеграцию с фреймворком и согласованность схемы БД и метаданных.

Соответствие БД и метаданных

Названия и типы полей в метаданных должны совпадать с колонками в БД (`NLong` → `bigint`, `NString` → `varchar/text`, `NDate` → `date`). При джоинах убедитесь в совместимости типов (избегайте неявных преобразований идентификаторов и используйте корректные приведения).

Цель: Обеспечить корректную работу ORM, избежать ошибок типов и обеспечить производительность запросов.

Связи

Проверить корректность внешних ключей/ссылок, отраженных в метаданных. Использовать `GID` для переменной ссылочности в одном поле. Убедиться, что все связи отражены в метаданных (поля помечены как `Reference` или `VariableReference`).

Цель: Обеспечить целостность данных, корректную навигацию по объектной модели и работоспособность связанных функций GSF.

JSON/Jsonb-поля

Если используется динамическое расширение (JSONB), проверьте настройку контейнера (тип `Json`) и работающую схему ключ–значение. Также проверьте необходимость индексов.

Цель: Обеспечить эффективное хранение и извлечение гибких структур данных, при необходимости — повысить производительность поиска.

Отсутствие дубликатов

Убедиться, что в метаописаниях нет дублирующих или неиспользуемых полей и неактуальных связей.

Цель: Улучшить читаемость метаданных, упростить поддержку, снизить вероятность логических ошибок и избыточной сложности модели.

2. Конфигурация Модулей GSF

`project.yaml`

Проверьте секции конфигурации проекта (версии Scala/Java, параметры сборки, список модулей). Например: `scalaFeatureRelease`, `sbtPlugin`, `modules` с описанием каждого модуля – имя, `source` (репозиторий), `branch`, флаг `isPublish`. Убедитесь, что указаны все нужные модули (например, `calculation`, `source`, `target`, `custom` при наличии) и их настройки верны.

Цель: Обеспечить корректную сборку проекта, управление зависимостями между модулями, использование нужных версий библиотек и плагинов.

Структура каталогов

Поддерживайте корректную структуру каталогов (как сгенерировано Configurator), включая папки `src`, `resources`, тесты и пр.

Цель: Облегчить навигацию по коду, упростить настройку IDE и сборочных скриптов, повысить согласованность между проектами.

Зависимости

Зависимости между модулями должны быть описаны явно в `build.sbt`, а так же продублированы в `module-info.xml` как метаданные

Цель: Обеспечить корректную сборку, изоляцию модулей, избежать проблем при сборке и развертывании.

Конфигурации сборки/публикации

Конфигурации сборки (например, `sbt`) и плагины должны быть корректны. Названия модулей и подпроектов должны соответствовать соглашениям.

Цель: Обеспечить стабильность и воспроизводимость процесса сборки и публикации, улучшить читаемость и поддержку проекта.

3. Использование стандартных компонентов GSF

Использование сервисов GSF

По возможности используйте готовые сервисы фреймворка: аудит изменений, автонумерацию, группировку, поиск по шаблону, прикрепленные файлы и т.д. Например, если требуется фильтрация классов, включите «Универсальный фильтр» – он позволит фильтровать по полям класса и его коллекций на уровне БД. Группировка объектов должна работать через механизмы группировки GSF.

Цель: Снизить вероятность ошибок, ускорить разработку, обеспечить единообразие функциональности, упростить поддержку и обновление.

Кэширование и запросы

Если используется кэширование GSF (Shared-cache), убедитесь, что для часто запрашиваемых сущностей настроены `cache-index` в ORM и при запросах используется `.unique()`. Используйте объектные запросы (`OQuery`) с `tryCacheQueryResults()` там, где это целесообразно. Проверьте стратегии инвалидирования кэша.

Цель: Повысить производительность за счёт снижения обращений к БД, обеспечить актуальность данных в кэше.

Логирование

Для записи логов в базу задействуйте встроенный `LogTransaction` (выделяет отдельную лог-сессию). При большом количестве записей выполняйте `commitByInterval()` (например, каждые 1000 записей). Такой подход изолирует логи от основной транзакции и снижает нагрузку на БД.

Цель: Изолировать логи от основной транзакции, избежать потери логов при откате, снизить вероятность конфликтов и нагрузку на БД.

4. Обработка Ошибок и Зависимости

Прикладные исключения

В коде обрабатывайте только «прикладные» исключения — наследники `AppException`. Системные исключения не перехватывайте, а выбрасывайте дальше, пропускайте (так фреймворк сможет корректно прервать сессию).

Цель: Чётко разделить бизнес-логику ошибок и системные сбои, предотвратить скрытие серьёзных проблем и нарушение целостности данных.

Объявление исключений

Создавайте свои исключения как подклассы `AppException` и заводите для них фабрику (например, `object ExceptionName extends ExceptionFactory(new ExceptionName(_))`). Выбрасывайте исключения через `throw AppException(...)` или `e.raise(...)` (сохранит стек вызовов). При необходимости логирования ошибки используйте отдельную лог-транзакцию.

Цель: Обеспечить единообразие, упростить обработку, легко идентифицировать тип ошибки, гарантировать запись лога при откате основной транзакции.

Логирование ошибок

При необходимости логирования ошибки используйте отдельную лог-транзакцию. В `try/catch` используйте минимальное информативное логирование; причина исключения не скрывается.

Цель: Изолировать запись лога от основного процесса, избежать засорения логов, сохранить ключевую информацию для диагностики, сохранить причину исключения.

Транзакции и зависимые задачи

Избегайте слишком длинных транзакций (несколько минут и более) – PostgreSQL плохо их переносит. Разбивайте большие операции на логические блоки и выполняйте `session.commitWorkAuto()` после каждой партии изменений. Например, при массовой загрузке в цикле вызывайте `commitWorkAuto()` через заданный интервал, чтобы регулярно сбрасывать пакет в БД. При этом используйте `flush()` / `commit()` осмотрительно: не делайте неожиданных коммитов внутри server-side методов, чтобы не прерывать ожидания отката.

Цель: Снизить вероятность конфликтов, рост WAL, улучшить общую производительность БД, уменьшить вероятность сбоев, облегчить откат, сохранить согласованность данных.

5. Особенности SQL в GSF (Валидации и Ограничения)

Оптимизация запросов

Всегда проверяйте план выполнения (`EXPLAIN`) для поиска «узких мест». Сложные запросы по возможности разбивайте на подзапросы, упрощая логику.

Цель: Выявить проблемы с производительностью, улучшить читаемость, потенциально повысить производительность.

Индексы и параметры

При фильтрации используйте индексируемые поля (например, не фильтруйтесь по служебным столбцам типа `gid` без индекса). Всегда передавайте значения в запрос через параметры (`?` или именованные), чтобы избежать SQL-инъекций и позволить СУБД кэшировать планы.

Цель: Обеспечить быстрое выполнение запросов, защитить от SQL-инъекций, ускорить выполнение за счёт переиспользования планов.

Соединения и типы

При JOIN таблиц по ключам используйте одинаковые типы полей (например, связывайте `bigint` с `NLong`); избегайте неявных преобразований (например, конвертации `varchar` и `text`). При необходимости приводите типы явно.

Цель: Обеспечить эффективное использование индексов, избежать дорогостоящих преобразований данных, улучшить производительность.

Рекурсивные запросы

Если пишете рекурсивные CTE, обязательно добавьте условие выхода (обычно это `WHERE id NOT IN (...)` с ограничением глубины) и защиту от заикливания. И используйте `NOT EXISTS` вместо `NOT IN`, чтобы учесть `NULL` и не терять результаты.

Цель: Предотвратить бесконечные циклы в запросе, обеспечить надёжность при наличии `NULL`-значений.

6. Требования к Структуре Scala-Кода

Соглашения по стилю

Соблюдайте стандарты именования Scala (классы – `CamelCase`, методы/переменные – `camelCase`) по руководству Scala. `Global3-FrameWork` регистрозависим, поэтому имена сущностей (классов, полей, методов) должны точно совпадать с тем, как они объявлены. Не используйте подчёркивания в именах (за исключением разделителя модуля и имени класса).

Цель: Обеспечить корректную работу фреймворка, повысить читаемость кода, облегчить его понимание и поддержку.

D/A-паттерн кода

При генерации классов учтите, что фреймворк генерирует две иерархии: `Domain`- и `Application`-части. Доменные классы (`ClassDpi`, `ClassDvi`) перезаписываются при генерации и содержат «скелет» логики, а прикладные классы (`ClassApi`, `ClassAvi`) – расширяют их и используются для ручной бизнес-логики. **Логика должна писаться только в прикладных классах (Api/Avi)**, доменные файлы не редактируйте.

Цель: Обеспечить стабильность при регенерации кода, изолировать бизнес-логику от автоматически генерируемого кода.

Null-типы GSF

Для работы со значениями из БД используйте расширенные типы GSF (`NLong`, `NNumber`, `NGid`, `NDate`, `NString`, `NBigDecimal` и т.д.), которые безопасно обрабатывают `null`. Например, идентификаторы – `NLong` (фабрика `nl`), числовые поля – `NNumber` или `NBigDecimal` (фабрика `nr`). Важное правило: **для финансовых расчётов используйте `NBigDecimal` (`BigDecimal`)**, чтобы избежать ошибок двоичной арифметики. При сравнении nullable-значений в скриптах GSF применяйте оператор `===` (предотвращает NPE) вместо стандартного `==`.

Цель: Защитить от `NullPointerException`, обеспечить корректную обработку `null`, гарантировать точность вычислений для финансовых данных, обеспечить безопасность при сравнениях.

7. Общие Принципы и Стиль Кода

Комментарии

Код должен сопровождаться понятными комментариями.

Цель: Помочь другим разработчикам понять назначение и логику кода, поддерживать профессиональный стиль документации.

Именованное

Переменным и атрибутам необходимо давать осознанные имена, транслит необходимо избегать. Следуйте `scala`-конвенциям наименования сущностей.

Цель: Улучшить читаемость, документировать структуру данных, соответствовать стилю `Scala` и `GSF`.

Форматирование

`sql` и `scala` код должны быть отформатированы и не превращаться в «кашу».

Цель: Улучшить читаемость, облегчить понимание и сопровождение кода.

Scaladoc

Писать `scaladoc` к методам, классам и переменным.

Цель: Предоставить официальную документацию к API, облегчить поддержку и использование кода другими разработчиками.

8. Безопасность и Надёжность

`null`

Использование `null` должно быть полностью исключено для всех ссылочных типов.

Цель: Предотвратить `NullPointerException`, использовать безопасные GSF-типы.

`null` для **AnyVal**

Для значимых типов (`AnyVal`), таких как `JsonObject`, запрещено использовать `null` или его приведение (`null.asInstanceOf[JsonObject]`). Вместо этого необходимо использовать пустые экземпляры (например, `JsonObject()`), а проверку на наличие данных выполнять через `.nonEmpty`.

Цель: Предотвратить `NullPointerException` при работе с такими типами.

`head` / `get`

Необходимо избегать использования `head` или `get`; вместо них следует использовать `headOption` или `getOption`, чтобы избежать NPE.

Цель: Обеспечить безопасную обработку отсутствия значения, сделать код более надёжным.

`if-else` / `pattern-matching`

При ветвлении кода `if-else` необходимо указывать ветку `else`; для `pattern-matching` обязателен случай `case _ =>`.

Цель: Гарантировать обработку всех возможных сценариев, предотвратить `MatchError`, сделать код более надёжным.

9. Производительность

`.view`

При использовании нескольких комбинаторов подряд (например, `arr.map.flatMap.filter`) необходимо добавлять `.view` в начало, чтобы не создавались временные коллекции.

Цель: Улучшить производительность за счёт снижения расхода памяти.

Pattern matching в комбинаторах

При использовании комбинаторов по карте необходимо применять pattern matching для деструктуризации кортежей. Плохо: `any_map.foreach { tpe => tpe._1 + tpe._2 }`. Хорошо: `any_map.foreach { case (key, value) => key + value }`.

Цель: Сделать код более выразительным, типобезопасным и читаемым.

Изменяемые коллекции

В параметрах методов и возвращаемых значениях не должны использоваться изменяемые коллекции `collection.mutable`.

Цель: Повысить безопасность и предсказуемость кода, уменьшить вероятность ошибок.

ArrayBuffer

Необходимо использовать `ArrayBuffer` вместо `ListBuffer`.

Цель: Обеспечить лучшую производительность для операций вставки и доступа по индексу.

Выбор коллекции

При выборе коллекции (`List`, `Array`, `Vector`, `ArrayBuffer`) необходимо учитывать асимптотическую сложность операций (вставка, доступ по индексу), чтобы обеспечить оптимальную производительность.

Цель: Использовать подходящие структуры данных для ускорения выполнения операций.

Большие реестры

Получение данных из больших реестров должно выполняться реляционными запросами в БД по максимальному количеству условий с предварительным `session.flush()`, чтобы не перегружать ОЗУ сервера.

Цель: Снизить нагрузку на память сервера приложений, повысить производительность за счёт фильтрации на уровне БД.

10. Работа с API и ROP

HTTP-пакеты

Прямое обращение к http пакетам запрещено; необходимо использовать `btk_httppkg`.

Цель: Обеспечить централизованное управление HTTP-вызовами, соблюдение стандартов безопасности и логирования.

Файлы

При работе с файлами необходимо использовать стандартное api — `btb_fileapi`.

Цель: Гарантировать корректную обработку файлов, безопасность и совместимость с инфраструктурой.

`copyAro`

Метод `copyAro` не должен использоваться, так как все данные провайдера строки копируются в оперативную память.

Цель: Предотвратить чрезмерный расход памяти, особенно при работе с большими объектами или в циклах.

Загрузка ROP по gid

При загрузке ропов по `gid` необходимо использовать `Btb_Pkg.loadByGid` (обёрнутый в `TryApp`). Полученный `rop` должен матчиться с нужным API, например: `case Btb_ClassApi(rop) =>`.

Цель: Обеспечить безопасную загрузку объектов и правильную их типизацию.

Поиск по коду

Для поиска по коду необходимо использовать методы `_____Api().findByMnemonicCode("")`, а не прямые запросы. Объекты API должны быть объявлены через `lazy val`.

Цель: Использовать встроенные механизмы GSF, повысить читаемость и поддерживаемость кода.

`getVar`

Вместо конструкции `api().load(getVar("idAnyClass").asNLong)` необходимо использовать метод `get`, так как нет гарантии, что `getVar` вернёт значение вместо `null`.

Цель: Обеспечить более безопасное и надёжное получение данных.

Использование source-generated конструкций

Для доступа к атрибутам необходимо использовать конструкции, сгенерированные генератором источников (например, `rop.get(_.attr)`), а не методы `getByAttrName` и `getAttrByName`, так как последние используют рефлекссию, ошибки которой могут быть обнаружены только во время выполнения.

Цель: Обеспечить безопасность на этапе компиляции, предотвратить ошибки, связанные с рефлексией.

11. Работа с Базой Данных и SQL (Дополнения)

`checkworkability`

Обращение к базе данных в `checkworkability` должно быть исключено; нужные данные необходимо выносить в `onRefresh` и делать доступными через `getVar`.

Цель: Критично для производительности UI, предотвратить неотзывчивость интерфейса.

Вложенные циклы

Запросы внутри вложенных циклов использовать нельзя.

Цель: Избежать экспоненциального роста числа запросов к БД, сохранить производительность.

`onRefresh`

В `onRefresh` необходимо использовать `refreshByKey(parent)` вместо `byKey(parent)`.

Цель: Обновить данные в кэше, повысить эффективность.

`@FlushBefore`

При необходимости использования `onRefresh` на основе `selectStatement` в детализации документа необходимо отключать `@FlushBefore`, чтобы избежать ошибок при создании документа с обязательными атрибутами.

Цель: Предотвратить конфликты сессии и ошибки при инициализации документа.

`0Query` и итерация

При загрузке в кеш через `0Query` необходимо использовать итерацию по списку или получение элемента в конце, что гарантирует выполнение запроса.

Цель: Обеспечить корректное поведение ленивых запросов.

`0Query` и условия

Условия в `0Query`, зависящие от полей, которые могут измениться в кэше, должны обрабатываться одним из способов: предварительный `session.flush()`; вынос условия в `.filter` результата; использование транзакционного индекса.

Цель: Гарантировать, что условия в запросе будут учитывать актуальное состояние данных.

Интерполяция строк

Вместо интерполяции строк необходимо использовать метод `on` в `sql` запросах.

Цель: Обеспечить безопасную параметризацию, защитить от SQL-инъекций.

`Symbol` в `on`

При передаче параметров в запросы на основе Anorm через метод `on` необходимо использовать `Symbol("idGds")`, а не символьные литералы через апостроф (`'idGds'`), так как последний устарел.

Цель: Соответствовать современным стандартам, обеспечить корректную работу.

12. Архитектура и Дизайн

`lazy`

При объявлении переменных в классе необходимо использовать ключевое слово `lazy`, чтобы они инициализировались только при первом использовании.

Цель: Улучшить производительность, избежать ненужных вычислений.

`dataInstall`

При регистрации типов, закладок, атрибутов, функциональных настроек, процедур в `dataInstall` необходимо добавлять `lazy val` с мнемокодами и `id` для зарегистрированных значений.

Цель: Улучшить читаемость и поддержку кода установки данных.

Изменение параметров

При редактировании метода типы параметров не должны изменяться, это сохраняет обратную совместимость.

Цель: Критично для стабильности API, предотвратить ошибки в использующем коде.

Новые параметры

Перед добавлением новых параметров в метод необходимо учитывать обратную совместимость. Новые необязательные параметры необходимо добавлять в конец сигнатуры метода.

Цель: Избежать необходимости изменять весь код, который вызывает этот метод.

Сложная логика в AVI

Сложную логику не нужно расписывать в операции `Avi`, если она может быть вынесена в `API/PKG`. AVI-операция должна сводиться к вызову метода из `API/PKG`, это упрощает поддержку и минимизирует дублирование кода.

Цель: Улучшить поддерживаемость кода, разделить ответственности, тестировать бизнес-логику отдельно от UI-логики.

Scala-классы с компаньоном

При создании структур Scala-классов с объектом-компаньоном необходимо использовать решение, позволяющее проектное переопределение: `def list(): List = { new List {...} }`.

Цель: Обеспечить более гибкую настройку поведения классов на уровне проекта.

13. Работа с Атрибутами и Полями (Дополнения)

Source-generated доступ

Для доступа к атрибутам необходимо использовать конструкции, сгенерированные генератором источников (например, `rop.get(_attr)`), а не методы `getByAttrName` и `getAttrByName`, так как последние используют рефлексия, ошибки которой могут быть обнаружены только во время выполнения.

Цель: Обеспечить безопасность на этапе компиляции, предотвратить ошибки, связанные с рефлексией.

`getSelfVar с A.idPerson.name`

В выборках при обращении к стандартным атрибутам необходимо использовать `getSelfVar(A.idPerson.name)` или просто `A.idPerson.asNLong` вместо `getSelfVar("idPerson")`, это позволяет поймать ошибки во время компиляции при удалении атрибута.

Цель: Обеспечить безопасность типов на этапе компиляции.

Хранимые поля в AVI

Хранимые поля класса в AVI должны получаться через объект `A`, сгенерированный генератором источников, чтобы избежать ошибок времени выполнения при переименовании атрибута в классе.

Цель: Сделать код устойчивым к переименованиям атрибутов.

```
var v case class v onRefresh
```

В `case class`'ах, которые возвращаются в `onRefresh`, атрибуты должны быть мутабельными (`var`).

Цель: Обеспечить корректную работу с UI и обновление данных.

14. Строки и JSON

Интерполяция строк

При сборке строк из статических и динамических элементов необходимо использовать интерполяцию `s"text${value}otherText"` или многострочный вариант: `s"""1-st line |2-nd line |... |last line"""`. Интерполяция должна использоваться вместо конкатенации, так как она оптимальнее и более читаема.

Цель: Повысить читаемость, обеспечить оптимизацию компилятором.

```
.stripMargin
```

Для многострочных строк необходимо использовать `|` и `.stripMargin`, чтобы сохранить форматирование и структуру файла.

Цель: Улучшить читаемость многострочных строк в коде.

```
StringBuilder
```

При сборке очень больших строк необходимо использовать `StringBuilder` вместо `pos.map(s"...").mkString()`.

Цель: Обеспечить более эффективную работу с большими строками.

```
scala.util.parsing.json
```

Использовать нельзя, предпочтительнее `JObject`.

Цель: Использовать интегрированный с GSF и более безопасный тип.

```
Btk_JsonPkg
```

Использование `Btk_JsonPkg` необходимо избегать из-за зафиксированного разного поведения при его использовании с `JObject`.

Цель: Предотвратить неожиданное поведение и ошибки.

15. Специфика UI: AVI и Списочные Отображения

`getVar / getSelfVar / A в checkWorkAbility`

В `Avi.checkWorkAbility` и других AVI-операциях списочных отображений значения полей необходимо получать через `getVar`, `getSelfVar` или `A`, а не через `thisRop()`, чтобы избежать ошибки `load id = null` в пустом списке.

Цель: Предотвратить ошибки при работе с пустыми списками.

Поля с большим объёмом данных

В списочные отображения не должны выводиться поля с большим объёмом данных, чтобы не перегружать сервер. Например, поле `sResponse` журнала сообщений должно быть вынесено на закладку детализации или храниться в файле.

Цель: Повысить производительность UI и улучшить пользовательский опыт.

@FlushBefore в LookUp

В `LookUp`-отображениях для `onRefresh` необходимо добавлять аннотацию `@FlushBefore(mode = FlushBeforeMode.Disabled)`.

Цель: Предотвратить нежелательные сбросы сессии при обновлении данных в `LookUp`.

`DefaultRep#deactivateOper`

Вместо `oper.isActive = someCondition` необходимо использовать `DefaultRep#deactivateOper` для деактивации операций при невыполнении условий, это обеспечивает корректную работу при различных перекрытиях.

Цель: Использовать встроенный механизм фреймворка для корректной деактивации операций.

`prepareSelectStatement`

Сортировка по умолчанию и передача макросов фильтрации должны быть указаны в `prepareSelectStatement`, а не в `selectStatement`.

Цель: Улучшить структуру и читаемость кода настройки выборки.

16. Специфика UI: Кэширование, Логика и Настройки

Переходы состояний

Переходы состояний должны определяться сравнением `nOrder` и формированием условия из двух частей (из какого в какое), например: `nvStateFrom < 100.nn && nvStateTo >= 100.nn`, это обеспечивает работоспособность логики даже при удалении перехода.

Цель: Сделать логику переходов устойчивой к изменениям структуры состояний.

```
@Setter(refreshAfter = true)
```

Аннотация `@Setter(refreshAfter = true)` должна выполнять `selection.refreshItem()` в конце сеттера.

Цель: Обеспечить обновление UI после изменения значения.

Кастомные выборки

Редактируемые кастомные выборки должны быть реализованы объектным запросом, чтобы собрать пул изменений и отправить их в БД одной транзакцией.

Цель: Повысить эффективность, обеспечить целостность данных.

```
onRefreshExt и id
```

При реализации кастомных выборок на объектном запросе результат выборки должен содержать поле `id` для работы `onRefreshExt`.

Цель: Обеспечить корректную работу механизма обновления расширенных данных.

```
AdditionalInfo
```

Использование `AdditionalInfo` необходимо дважды обдумывать. Предпочтительнее получать данные через `onRefreshExt`.

Цель: Улучшить структуру и производительность, избегать избыточного кода.

```
AdditionalInfo2
```

При переопределении отображения, в котором уже используется `AdditionalInfo`, новые вычисляемые атрибуты не должны добавляться в новый `AdditionalInfo2`. В `OnRefresh` необходимо использовать несколько case-классов: `thisApi().byParent(getIdMaster).map { rop => (rop, getAdditionalInfo(rop), getAdditionalInfo2(rop)) }`.

Цель: Улучшить структуру и читаемость кода обновления данных.

17. Отражение (Reflection)

Обоснование

Использование `reflection` должно быть оправдано; в общем случае оно запрещено.

Цель: Повысить безопасность типов, производительность, упростить сопровождение кода.

`scala.reflect.io._`

Использовать нельзя, предпочтительнее Java-классы.

Цель: Обеспечить лучшую стабильность и совместимость.

18. Специфика Модулей и Проектов

Модуль STK

Для получения остатков необходимо использовать `ru.bitec.app.stk.Stk_Pkg#getRemainsMulti`. Для получения цены необходимо использовать `ru.bitec.app.stk.Stk_Pkg#getnPrCostConsSum`.

Цель: Использовать стандартные, оптимизированные методы для получения данных из модуля STK.

Модуль АСТ

Запрос оборотов должен выполняться с учётом логовой таблицы изменений через `union all`, чтобы получить полную и актуальную информацию.

Цель: Гарантировать корректность и полноту данных об оборотах.

Проектные модули

При открытии MR необходимо указывать соответствующую ветку с учётом проекта, на котором разрабатывается. Например, для проекта СНГ разработка ведётся на `sng-internal-dev` → ветка проектного модуля `gs`; `pdev` → `dev`.

Цель: Обеспечить правильную интеграцию изменений в проектную ветку.

19. Безопасность и Целостность Данных

SQL-инъекции

Все входные данные параметризованы, нет риска SQL-инъекций.

Цель: Защитить от SQL-инъекций, ускорить выполнение за счёт переиспользования планов.

Ограничения целостности

Ограничения целостности (`foreign key`, `unique`) соответствуют метаданным.

Цель: Обеспечить корректность и согласованность данных на уровне БД.

Валидации

Валидации на уровне приложения дублируют, но не заменяют БД-ограничения.

Цель: Обеспечить дополнительный уровень защиты и пользовательскую обратную связь.

Чувствительные данные

Чувствительные данные маскируются в логах и не попадают в открытые сообщения об ошибках.

Цель: Защитить конфиденциальную информацию от утечки.

20. Соответствие Best Practices

Документация

Убедитесь, что код и конфигурации соответствуют рекомендациям из официального руководства (например, настройка логирования, работа с транзакциями, параллельные вычисления и т.д.). Всегда проверяйте разделы «Практика разработки» в документации GlobalFramework и сверяйтесь с примерами.

Цель: Повысить качество, поддерживаемость и безопасность кода, облегчить его понимание новыми разработчиками.

Чистота кода

Код должен быть читабельным и понятным (часто читают больше, чем пишут). Держите методы короткими, избегайте дублирования. Имена должны ясно отражать назначение сущности. Следуйте общим принципам GSF: не используйте служебные символы (например, «!», «?», «@») в идентификаторах, оформляйте документацию и комментарии по необходимости.

Цель: Улучшить читаемость, облегчить понимание и сопровождение кода.

21. Критические ошибки

DML через ASQL

DML (INSERT/UPDATE/DELETE) выполняется через ASQL. — **Блокирующее**

Цель: Предотвратить нарушение целостности данных из-за автокоммита ASQL.

Долгая транзакция

Долгая транзакция, захватывающая большое число строк без разбивки. — **Блокирующее**

Цель: Избежать проблем с производительностью и блокировками БД.

Функции GSF в массовых выборках

Массовое использование `getAttribute/getmemocode/getheadline` в больших выборках/отчётах. — **Блокирующее**

Цель: Избежать чрезмерной нагрузки на БД.

Арифметика с N-типами

Арифметика с N-типами без проверки nullable (`.isNotNull`). — **Блокирующее**

Цель: Предотвратить `NullPointerException` и ошибки времени выполнения.

Конкатенация строк SQL

SQL-строки формируются конкатенацией с пользовательскими данными. — **Блокирующее**

Цель: Предотвратить уязвимость к SQL-инъекциям.

Дублирование логики

Необъяснимое и несогласованное дублирование логики в разных модулях. — **Блокирующее**

Цель: Упростить сопровождение, снизить вероятность ошибок, упростить внесение изменений.

Чек-лист для ревью кода в GlobalFramework

Этот чек-лист служит практическим инструментом для начинающих и опытных ревьюверов. Он представляет собой **структурированный перечень контрольных точек** для систематической проверки кода. Документ направлен на **повышение качества, производительности и стабильности кода** и охватывает все ключевые аспекты разработки: от метаописаний и конфигурации до работы с API, SQL, безопасности и специфики модулей.

1. Метаописания и Структура Классов

- [] **Именованное:** Классы (`<Модуль>_<Имя>`, `CamelCase`) и атрибуты (`camelCase`) — уникальны, осознанны, без транслита.
- [] **Типы и связи:** GSF-типы (`NLong`, `NNumber`, `NString`, `NDate`, `NBigDecimal`, `NGid` и т.п.) корректны и соответствуют БД. Поля с `Reference`, `VariableReference`, `GID` настроены и отражают связи в метаданных.
- [] **JSON/Jsonb-поля:** При использовании динамического расширения (JSONB) проверена настройка контейнера (тип `Json`), схема ключ-значение и необходимость индексов.

- [] **Чистота метаданных:** Отсутствуют дублирующие или неиспользуемые поля и неактуальные связи.
-

2. Конфигурация Модулей GSF

- [] **project.yaml:** Проверены секции конфигурации проекта (версии Scala/Java, параметры сборки, список модулей). Указаны все нужные модули (`calculation`, `source`, `target`, `custom` при наличии) и их настройки (ветки, `isPublish`).
 - [] **Структура и зависимости:** Соответствие шаблонам проекта (`src`, `resources`, тесты). Зависимости между модулями описаны явно, циклические зависимости отсутствуют.
 - [] **Сборка и публикация:** Конфигурации сборки (например, `sbt`) и плагины корректны. Названия модулей и подпроектов соответствуют соглашениям.
-

3. Использование стандартных компонентов GSF

- [] **Встроенные сервисы:** По возможности используются готовые сервисы фреймворка (аудит, автонумерация, группировка, поиск по шаблону, прикрепленные файлы).
 - [] **Кэширование:** Для часто запрашиваемых сущностей настроены `cache-index` в ORM, используется `.unique()` и `tryCacheQueryResults()`. Проверены стратегии инвалидирования кэша.
 - [] **Логирование:** Для записи логов используется `LogTransaction`. При большом количестве записей применяется `commitByInterval()`.
-

4. Обработка Ошибок и Зависимости

- [] **Исключения:** Обрабатываются только `AppException`. Системные исключения выбрасываются дальше. Свои исключения создаются как подклассы `AppException` с фабрикой. Логирование ошибок — в отдельной транзакции.
 - [] **Транзакции:** Избегаются длинные транзакции. Большие операции разбиты на блоки с `session.commitWorkAuto()`. Неожиданные коммиты внутри `server-side` методов отсутствуют.
-

5. Особенности SQL в GSF

- [] **Оптимизация и безопасность:** Проверен план выполнения (`EXPLAIN`). Все запросы параметризованы. Используется `NOT EXISTS` вместо `NOT IN`.
- [] **ASQL vs ATSQL:** ASQL — только для чтения; ATSQL — для DML и блокировок.
- [] **JOIN и типы:** Выполняются по совместимым типам. Избегаются неявные преобразования.
- [] **Рекурсивные CTE:** Имеют условие выхода и защиту от заикливания.
- [] **Массовые операции:** Реализованы через `COPY` или батчи. Долгие транзакции избегаются.

- [] **Функции GSF:** `getAttribute`, `getmнемocode`, `getheadline` и т.п. не используются в больших выборках без кэширования.
-

6. Требования к Структуре Scala-Кода

- [] **D/A-паттерн:** Бизнес-логика в `ClassApi/ClassAvi`. Генерируемые `ClassDpi/ClassDvi` не изменяются.
 - [] **Null-типы GSF:** Используются `NLong`, `NNumber`, `NBigDecimal` (для финансов), `==`, `.isNotNull`.
 - [] **Стиль и читаемость:** Соблюдается `CamelCase/camelCase`, нет подчёркиваний. Методы короткие, используются `view`, `pattern matching`, `trait` для повторного использования.
 - [] **Производительность:** Избегаются лишние коллекции, используется `ArrayBuffer`. Циклы имеют верхний предел. Для больших наборов — стриминг.
 - [] **Безопасность:** `null` исключён. Используются `headOption/getOption`, а не `head/get`. Ветка `else` и `case _ =>` обязательны.
-

7. Производительность

- [] **.view:** Используется при нескольких комбинаторах подряд.
 - [] **Большие реестры:** Запросы выполняются с максимальным количеством условий и `session.flush()`.
 - [] **Циклы:** Максимальное число итераций проверено.
 - [] **Мониторинг:** Для массовых операций предусмотрен мониторинг и точки прерывания.
-

8. Работа с API и ROP

- [] **HTTP/файлы:** Используются `btk_httppkg` и `btk_fileapi`.
 - [] **Загрузка/поиск:** `Btk_Pkg.loadByGid`, `findByМнемoCode("")`. Объекты API объявлены через `lazy val`.
 - [] **getVar:** Используется `get`, а не `asNLong`.
 - [] **Source-generated конструкции:** Для доступа к атрибутам используется `rop.get(_._attr)`.
-

9. Работа с Базой Данных и SQL (Дополнения)

- [] **checkworkAbility:** Не содержит запросов к БД.
 - [] **Вложенные циклы:** Запросы вне вложенных циклов.
 - [] **onRefresh:** Используется `refreshByKey(parent)`.
 - [] **@FlushBefore:** Отключён в детализации документа.
 - [] **0Query:** Используется итерация по списку или получение элемента в конце. Условия обрабатываются корректно (flush, filter, транз. индекс).
 - [] **Параметры:** Используется `Symbol("param")`, а не символьные литералы.
-

10. Архитектура и Дизайн

- [] **lazy:** Используется при объявлении переменных и в `dataInstall`.
 - [] **API-методы:** Типы параметров не меняются. Новые необязательные параметры добавляются в конец.
 - [] **AVI:** Сложная логика вынесена в API/PKG. AVI-операция сводится к вызову метода из API/PKG.
 - [] **Scala-классы:** Используется решение, позволяющее проектное переопределение.
-

11. Работа с Атрибутами и Полями (Дополнения)

- [] **Source-generated доступ:** Используется `rop.get(_.attr)` вместо `getByAttrName`.
 - [] **getSelfVar:** Используется `A.idPerson.name` вместо строкового имени.
 - [] **Хранимые поля в AVI:** Получаются через объект `A`.
 - [] **var в case class:** Атрибуты в `case class` в `onRefresh` — мутабельны.
-

12. Строки и JSON

- [] **Интерполяция:** Используется `s"text${value}otherText"` или многострочный вариант.
 - [] **.stripMargin:** Применяется для многострочных строк.
 - [] **StringBuilder:** Используется при сборке очень больших строк.
 - [] **JSON:** Не используется `scala.util.parsing.json`. Использование `Btk_JsonPkg` избегается.
-

13. Специфика UI: AVI и Списочные Отображения

- [] `getVar / getSelfVar / A: B checkWorkAbility` значения получаются через `getVar`, `getSelfVar` или `A`.
 - [] **Большие поля:** Не выводятся в списочные отображения.
 - [] `@FlushBefore:` В `LookUp`-отображениях для `onRefresh` добавлена аннотация `@FlushBefore(mode = Disabled)`.
 - [] `DefaultRep#deactivateOper:` Используется для деактивации операций.
 - [] `prepareSelectStatement:` Сортировка по умолчанию и макросы фильтрации указаны в `prepareSelectStatement`.
-

14. Специфика UI: Кэширование, Логика и Настройки

- [] **Переходы состояний:** Определяются сравнением `nOrder` (условие из двух частей).
 - [] `@Setter(refreshAfter = true):` Выполняет `selection.refreshItem()` в конце сеттера.
 - [] **Кастомные выборки:** Реализованы объектным запросом, результат содержит `id`.
 - [] `AdditionalInfo:` Используется осторожно. Предпочтительнее `onRefreshExt`. Не создается `AdditionalInfo2` для новых атрибутов.
-

15. Отражение (Reflection)

- [] **Обоснование:** Использование `reflection` оправдано; в общем случае запрещено.
 - [] `scala.reflect.io._:` Не используется, предпочтительнее `Java`-классы.
-

16. Специфика Модулей и Проектов

- [] **Модуль STK:** Для остатков — `getRemainsMulti`, для цены — `getnPrCostConsSum`.
 - [] **Модуль АСТ:** Запрос оборотов выполняется с учётом логовой таблицы через `union all`.
 - [] **Проектные модули:** При открытии MR указывается соответствующая ветка (например, `gs` для `sng-internal-dev`).
-

17. Безопасность и Целостность Данных

- [] **SQL-инъекции:** Исключены за счёт параметризации.
 - [] **Целостность:** Ограничения (foreign key, unique) соответствуют метаданным. Валидации на уровне приложения дублируют БД-ограничения.
 - [] **Чувствительные данные:** Маскируются в логах и сообщениях об ошибках.
-

18. Соответствие Best Practices

- [] **Scaladoc:** Пишется scaladoc к методам, классам и переменным.
 - [] **Комментарии:** Код сопровождается понятными комментариями.
 - [] **Форматирование:** sql и scala код отформатированы и не превращаются в «кашу».
 - [] **Именованное:** Переменным и атрибутам даются осознанные имена, транслит избегается. Следуются scala-конвенции наименования сущностей.
 - [] **Чистота кода:** Код читабельный, без дублирования. Имена ясно отражают назначение. Не используются служебные символы в идентификаторах.
-

19. Красные Флаги (Блокирующие)

- [] **DML через ASQL:** DML (INSERT/UPDATE/DELETE) выполняется через ASQL. — Блокирующее
- [] **Долгая транзакция:** Долгая транзакция, захватывающая большое число строк без разбивки. — Блокирующее
- [] **Функции GSF в массовых выборках:** Массовое использование getattribute/getmemocode/getheadline в больших выборках/отчётах. — Блокирующее
- [] **Арифметика с N-типами:** Арифметика с N-типами без проверки nullable (.isNotNull). — Блокирующее
- [] **Конкатенация строк SQL:** SQL-строки формируются конкатенацией с пользовательскими данными. — Блокирующее
- [] **Дублирование логики:** Необъяснимое и несогласованное дублирование логики в разных модулях. — Блокирующее

Подключение сторонних библиотек к локальному globalserver

Прямое подключение

Для прямого подключения сторонних библиотек к серверу, необходимо скопировать .jar файлы в папку lib Примерный путь до директории

```
C:\programs\gsf-cli\workspace\dists\<project_name>\Global3se\server\lib
```

Пример для oracle драйверов

```
C:\programs\gsf-cli\workspace\dists\<project_name>\Global3se\server\lib
├─ ojdbc6.jar
├─ orai18n.jar
├─ ucp.jar
└─ ...
```

Пример содержимого project.yaml

```
scalaFeatureRelease: 2.13
javaFeatureRelease: 21
buildKit:
  name: ru.bitec.phosagro
  version: SNAPSHOT
applicationServer:
  source: <applicationServer_url>
  isPublish: true
sbtPlugin:
  source: <sbtPlugin_url>
  branch: 1.0.26
  isPublish: true
modules:
  - bs:
    source: <modules_url>
    branch: 1.85.42.1388
    isPublish: true
```

Детальное описание полей конфигурации:

scalaFeatureRelease: Версия Scala, которая будет использоваться при сборке проекта.

javaFeatureRelease: Версия Java, используемая для компиляции проекта.

buildKit: Блок с информацией о комплекте сборки:

- **name:** Полное имя артефакта (например, Maven ID).
- **version:** Версия сборки, например SNAPSHOT или конкретный релиз.

applicationServer: Конфигурация серверной платформы:

- **source:** URL или путь к комплекту приложения.
- **isPublish:** Логический флаг, указывающий, должен ли сервер публиковаться (например, в репозиторий).

sbtPlugin: Настройки подключаемого SBT-плаги́на:

- **source:** URL к репозиторию с исходным кодом плагина.
- **branch:** Ветка, используемая для сборки.
- **isPublish:** Определяет, будет ли плагин опубликован как часть сборки.

modules: Список модулей проекта. Каждый модуль описывается именованной секцией, где:

- **<module_name>** — имя модуля (например, bs, btk).

- **source**: URL к репозиторию модуля.
- **branch**: Ветка, используемая при сборке.
- **isPublish**: Флаг публикации данного модуля.

13.7 Руководство Jexl

Введение и основы JEXL

Введение в языки выражений

Языки выражений — специализированные языки для вычисления выражений во время работы программы. Они позволяют динамически выполнять математические, логические и строковые операции, обращаться к свойствам объектов и вызывать методы без компиляции кода.

Языки выражений работают в работающей программе, позволяя гибко менять логику — например, проверять условия `user.age > 18` или вычислять формулы `price * quantity` без перезапуска приложения.

Зачем они нужны:

- Гибкость — изменение логики без перекомпиляции приложения;
- Безопасность — ограниченный набор операций снижает риски выполнения вредоносного кода;
- Удобство — простой синтаксис для нетривиальных вычислений;
- Динамическая оценка — выражения могут загружаться из внешних источников.

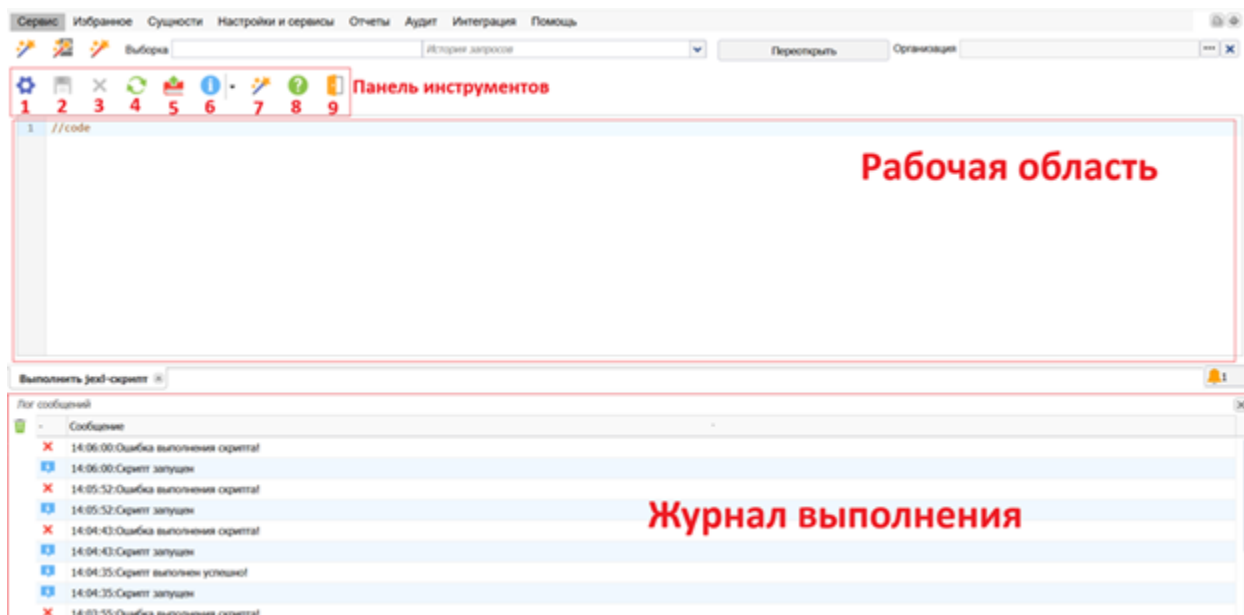
JEXL — библиотека, реализующая язык выражений для вычисления и выполнения скриптов в Java-приложениях.

Редактор JEXL

Путь доступа:

1. Откройте главное меню системы.
2. Перейдите в раздел Сервисы.
3. Выберите подраздел Инструменты.
4. Нажмите пункт Выполнить JEXL-скрипт.

При открытии редактора отображается рабочее пространство с тремя функциональными областями:



Панель инструментов — набор кнопок для управления скриптами:

1. **Выполнить** — запуск текущего скрипта.
2. **Сохранить**.
3. **Откатить**.
4. **Обновить**.
5. **Открыть скрипт из библиотеки JEXL** — выбор скрипта из общего/личного хранилища.
6. **Информация + Аудит JEXL** — просмотр метаданных и истории изменений

Сервис	Избранное	Справочники	Продажа	Банк	Склад	Закупка	Настройки	Отчеты	Помощь
Организация	АО "Предприятие"	Текущая дата	04.04.2025	Рабочий период с		по		Выбрать период	
с	04.04.2025 00:00:00	по	04.04.2025 23:59:59	Пользователь					
Пользователь	Время запуска	Время окончания	Скрипт	Результат	STACKTRACE	SAUDINFO			
admin	04.04.2025 12:50:36	04.04.2025 12:50:39	BBK_GroupApi.test()	Успех					
admin	04.04.2025 12:59:39	04.04.2025 14:11:27	BBK_GroupApi.test()	Успех					
admin	04.04.2025 14:03:13	04.04.2025 14:03:19	var nriblue = function(number) { ...	Успех					
admin	04.04.2025 14:03:26	04.04.2025 14:03:26	var nriblue = function(number) { ...	Ошибка	Caused by: org.apache.commons.jexl3.JexlException				
admin	04.04.2025 14:03:55	04.04.2025 14:03:55	var nriblue = function(number) { ...	Ошибка	Caused by: org.apache.commons.jexl3.JexlException				
admin	04.04.2025 14:04:35	04.04.2025 14:04:35	var nriblue = function(number) { ...	Успех					
admin	04.04.2025 14:04:43	04.04.2025 14:04:43	var nriblue = function(number) { ...	Ошибка	Caused by: org.apache.commons.jexl3.JexlException				
admin	04.04.2025 14:05:52	04.04.2025 14:05:52	var nriblue = function(number) { ...	Ошибка	Caused by: org.apache.commons.jexl3.JexlException				
admin	04.04.2025 14:06:00	04.04.2025 14:06:00	var nriblue = function(number) { ...	Ошибка	Caused by: org.apache.commons.jexl3.JexlException				

7. **Проверить правильность JEXL скрипта** — проверка синтаксиса написанного кода.
8. **Помощь** — краткая справка по существующим методам.
 - **Помощь** — вкладка с методами и их кратким описанием;
 - **Список объектов** — список объектов API всей системы;
 - **Контекст** — фильтр отображения методов по контексту, API, математическим функциям;
 - **К оглавлению** — возврат к редактору кода;
 - **Обновить**;
 - **Проверить правильность Jexl скрипта**;
 - **Официальная документация**.

9. Выход.

Рабочая область — написание и редактирование JEXL-кода.

Журнал выполнения — хронология выполнения скриптов, ошибки и предупреждения, системные сообщения.

- Очистить логи — удаление всей текущей истории сообщений из журнала.

Основы синтаксиса выражений

Простые выражения

JEXL поддерживает стандартные операторы, аналогичные Java:

Арифметика:

```
var sum = 5 + 3;           // 8
var total = 8 * 10;        // 80
var remainder = 10 % 3;    // 1
```

Логические операции:

```
var isActive = true;
var isEligible = (age >= 18) && isActive;
var isAdmin = (role == "admin") || (role == "superuser");
```

Сравнения:

```
price == 100
name != 'admin'
score > 90
```

Поддерживаемые типы данных

JEXL работает с основными типами:

- числа — целые (42), дробные (3.14);
- строки — в одинарных („text“), двойных («text») или обратных кавычках (test);
- булевы значения — true, false;
- null — отсутствие значения;
- коллекции — списки [1, 2, 3], мапы {„key“: „value“}, множества {1, 2, 3}.

Обертки для безопасной передачи данных

В JEXL для работы со Scala-методами, требующими строгой типизации и поддержки null, используются специальные обертки.

Scala — статически типизированный язык, где:

- null для примитивов (Long, Int) запрещен.

При вызове Scala-методов из JEXL:

- JEXL передает «сырые» значения (String, Number, null);
- Scala ожидает строгие типы (NLong, NGid).

Null-типы и их назначения:

Null-тип	Назначение
NLong	Работа с идентификаторами (ID)
NNumber	Работа с целыми/дробными числами
NGid	Работа с глобальными идентификаторами (GUID)
NDate	Работа с датами
NString	Работа со строками (с поддержкой null)
NBigDecimal	Точные расчеты (деньги, финансы)
NDuration	Арифметика временных промежутков

Создание Null-типа на примере NLong:

```
var NLong = function(number) {  
    return new ("ru.bitec.app.gtk.lang.NLong", number);  
};  
var nlValue = NLong(211);
```

Конкретное применение null-типов:

```
// Создаем Map с параметрами для заказа  
Map1.put(  
    new ("ru.bitec.app.gtk.lang.NLong", ropStageDet.id),  
    new ("ru.bitec.app.gtk.lang.NNumber", ropStageDet.nQty)  
);  
  
// Создаем заказ  
idvOrder = Stm_OrderOutApi.createOrderByContractStage(  
    ropStage.id,  
    idvDepOwner,  
    idvOrderType,  
    asScala(Map1), // Основные параметры  
    asScala(EmptyMap), // Доп. параметры 1  
    asScala(EmptyMap), // Доп. параметры 2  
    null  
);
```

Работа с переменными

Способы объявления переменных

JEXL предоставляет три способа объявления переменных: `let`, `const` и `var`, каждый с своей областью видимости и правилами изменяемости.

let — Локальная переменная

Ключевое слово `let` создаёт переменную, доступную только внутри блока `{ }`, где она объявлена, включая вложенные блоки. Переменная не может быть переопределена в той же области.

Пример:

```
if (56 < 100) {  
    let sText = "Example text";  
}  
dialogs.showMessage(sText) // ошибка: переменная объявлена в другом блоке
```

const — Неизменяемая переменная

Переменная, объявленная через `const`, имеет блочную область видимости, но её значение нельзя изменить после инициализации.

Пример:

```
if (56 < 100) {  
    const sText = "Example text";  
    sText = "Changed text"; // Ошибка: изменение константы  
}
```

Исключение: если `const` ссылается на объект, его внутренние свойства могут изменяться — неизменной остаётся только ссылка.

Пример:

```
const arrayEven = [2, 4, 6, 8];  
arrayEven[0] = 10;  
dialogs.showMessage(arrayEven[0]); // Вывод: 10
```

var — Глобальная переменная

По умолчанию `var` создаёт переменную, видимую во всём скрипте, и позволяет её переопределять:

```
if (56 < 100) {  
    var sText = "Example text";  
    sText = "Changed text";  
}  
sText = "Outer text"; // Изменение разрешено  
dialogs.showMessage(sText); // Вывод: Outer text
```

Когда что использовать:

- `const` — для значений, которые не должны меняться;
- `let` — для обычных переменных внутри блоков;
- `var` — для глобальных переменных (с осторожностью).

Объявления переменных поддерживаемых типов

Числовые типы:

```
var nValue = 42; // Целое число
var fValue = 42.0f; // Число с плавающей точкой
var lValue = 42L; // Длинное целое
var dValue = 42.0d; // Число двойной точности
var bigIntValue = 42H; // Большое целое
var bigDecValue = 42.0B; // Число высокой точности
var hexValue = 0x2A; // Шестнадцатеричное число
var octValue = 052; // Восьмеричное число
var sciValue = 4.2E+1D; // Число в научной нотации
```

Строковые типы:

```
var sStr1 = "Hello"; // Строка (двойные кавычки)
var sStr2 = 'World'; // Строка (одинарные кавычки)
var sStr3 = `line1 // Многострочный комментарий
line2
line3
`;
var intValue = 10
var interpolatedStr = `Value: ${intValue}`; // Строка с подстановкой
var regex = ~/pattern\d+/; // Регулярное выражение
```

Специальные значения:

```
var bVal = true; // Логическое значение
var nVal = null; // Пустое значение
var tuple2 = (v1, v2) -> { return new(`scala.Tuple2`, v1, v2); }; // Кортеж
```

Коллекции:

```
var array = [1, 2, 3]; // Массив
var objArray = [1, "two", 3.0]; // Массив объектов
var list = [1, 2, 3, ...]; // Список
var set = {"a", "b", "c"}; // Множество
var map = {"key1": 1, "key2": 2}; // Карта
var range = 1..10; // Диапазон чисел
var gor = Stm_OrderOutApi.load(id); // Строка роп соответствующая объекту БД
```

Пустые коллекции:

```
var emptyArray = []; // Пустой массив
var emptyList = [...]; // Пустой список
var emptySet = {}; // Пустое множество
var emptyMap = {:}; // Пустая карта
```

Доступ к свойствам объектов

Два варианта доступа к объекту:

```
// Через точку (автоматически вызывает getter)
var sClientName = order.client.name;

// Через метод
var nTotal = order.getTotal();
```

Запись JEXL-макросов

Сервис записи JEXL-макросов преобразует действия пользователя в готовые JEXL-скрипты. Инструмент создает сценарии автоматизации без ручного написания кода и глубокого изучения JEXL.

Основные сценарии использования:

- Массовое создание объектов — добавление позиций номенклатуры, контрагентов, других сущностей;
- Пакетное обновление данных — изменение реквизитов у группы объектов;
- Формирование отчетных форм — создание печатных форм с настройкой шаблонов;
- Импорт данных из файлов — обработка XLSX-файлов для переноса информации.

Примечание

Записываются только вызовы `Api` и `Pkg`. Макросы не содержат вызовы интерактивной логики.

Создание макросов

Начало записи

1. Перейдите в **Сервис** → **Инструменты** → **Начать запись JEXL-скрипта**.
2. В главном меню появится красный флаг — индикатор записи.

Примечание

Перед записью сохраните или откатите все несохраненные данные во всех формах.

При несохраненных данных система выдаст ошибку о невозможности изменения активности перехвата вызовов методов.

Запись действий

После начала записи система отслеживает все действия. Выполните операции с данными, которые нужно записать.

Завершение и сохранение

1. Остановите запись через:
 - Красный флаг в главном меню;
 - **Сервис → Инструменты → Закончить запись JEXL-скрипта.**
2. Просмотрите записанный скрипт в появившемся окне.
3. При закрытии окна подтвердите сохранение.
4. Выберите папку (по умолчанию — **Личные**).
5. Скрипт сохранится в Библиотеке скриптов.

Просмотр и редактирование

Для работы с сохраненными скриптами:

1. Перейдите в **Настройки и Сервисы**.
2. Выберите **Сервисы JEXL → Библиотеки JEXL**.
3. Найдите скрипт в нужной папке.

Доступны просмотр, редактирование и выполнение скриптов.

Примеры скриптов

Создание печатной формы:

```
var rop_Rpt_Report1 = Rpt_ReportApi.insert();
Btk_ObjectGroupApi.register(rop_Rpt_Report1, 27851L, 1B, 1B);
Rpt_ReportApi.setSystemName(rop_Rpt_Report1, 'SomeReportName');
Rpt_ReportApi.setCaption(rop_Rpt_Report1, 'Отчет');
Rpt_ReportApi.setidModule(rop_Rpt_Report1, 901L);
var rop_Rpt_ReportVersion1 = Rpt_ReportVersionApi.insertByParent(rop_Rpt_Report1);
Rpt_ReportVersionApi.setidReportType(rop_Rpt_ReportVersion1, 451L);
```

В сформированном скрипте все идентификаторы имеют исходный вид. Если этот скрипт планируется использовать на других базах данных, где идентификаторы будут отличаться, необходимо заменить их получение в скрипте - например, на метод `findByMnemonicCode`.

Видоизмененный скрипт:

```
var rop_Rpt_Report1 = Rpt_ReportApi.insert();
Btk_ObjectGroupApi.register(rop_Rpt_Report1, 27851L, 1B, 1B);
Rpt_ReportApi.setSystemName(rop_Rpt_Report1, 'SomeReportName');
Rpt_ReportApi.setCaption(rop_Rpt_Report1, 'Отчет');
Rpt_ReportApi.setidModule(rop_Rpt_Report1, Btk_ModuleApi.findByMnemonicCode('btk'));
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
var rop_Rpt_ReportVersion1 = Rpt_ReportVersionApi.insertByParent(rop_Rpt_Report1);
Rpt_ReportVersionApi.setidReportType(rop_Rpt_ReportVersion1, Rpt_ReportTypeApi.
↪findByMnemonicCode('jasper'));
```

Обработка данных Excel

Для работы с `xlsx` в `jexl` используется библиотека `poi.apache.org`.

Для удобства загрузки файлов написана *функциональная библиотека* `Bts_XlsxPkg`. Метод `uploadParseFiles` открывает диалог выбора файла и парсит его, предоставляя разработчику возможность его обработки. Создается объект класса `org.apache.poi.ss.usermodel.Workbook`.

Для выполнения массовой обработки данных из файлов Excel, вы можете использовать следующий метод:

```
lib("Btk_XlsxLib").uploadParseFiles(fun);
```

Где `fun` - это функция `Jexl`, которая будет выполняться для обработки данных из Excel.

Пример обработки `xlsx`:

```
var fun = x -> {
  var sheet = x.getSheet("Материалы");
  var lastRowNum = sheet.getLastRowNum();
  var i = 1;
  while (i <= lastRowNum ){
    var svNomName = sheet.getRow(i).getCell(3).getStringCellValue(); // Условное
↪наименование
    var idvMSRItem = sheet.getRow(i).getCell(4).getNumberCellValue().toBigDecimal(); //
↪ЕИ
    var idvGost = sheet.getRow(i).getCell(5).getStringCellValue(); // ГОСТ
    var idvSortamentType = sheet.getRow(i).getCell(6).getNumberCellValue().
↪toBigDecimal(); // Тип сортамента

    var rop_Bs_Goods1 = Bs_GoodsApi.insert();
    Bs_GoodsApi.setsNomName(rop_Bs_Goods1, svNomName); // установка условного наименования
    Bs_GoodsApi.setidMeasureItem(rop_Bs_Goods1, idvMSRItem); // установка ЕИ
    Bs_GoodsApi.setsGost(rop_Bs_Goods1, svNomName); // установка условного наименования
    Bs_GoodsApi.setsNomName(rop_Bs_Goods1, svNomName); // установка условного наименования

    commit();
    i = i+1;
  }
  true;
}

lib("Btk_XlsxLib").uploadParseFiles(fun);
```

Приведенный ниже пример функции `fun` иллюстрирует, как обработать данные из файла Excel и создать объекты в системе на их основе.

Расширенная обработка с разными типами данных:

```

var fun = x -> {
// Получение листа "Лист1" из файла Excel
var sheet = x.getSheet("Лист1");

// Определение количества заполненных строк в файле
var lastRowNum = sheet.getLastRowNum();
var i = 0;

while (i <= lastRowNum) {
// Класс для которого мы создаем объекты
var rop_RplTst_AllDbTypes1 = RplTst_AllDbTypesApi.insert();

// Получение значений из ячеек Excel и преобразование их к необходимым типам
// Для получения значения типа Long нужно дополнительно использовать метод toLong
var vjson = sheet.getRow(i).getCell(0).getStringCellValue();
var vsystemname = sheet.getRow(i).getCell(1).getStringCellValue();
var vcaption = sheet.getRow(i).getCell(2).getStringCellValue();
var vgidrefclassany = sheet.getRow(i).getCell(3).getStringCellValue();
var vidobjecttype = sheet.getRow(i).getCell(4).getNumericCellValue().toLong();
var vdate = toDate(sheet.getRow(i).getCell(5).getStringCellValue());

// Преобразование для NNumber
var vNumber = sheet.getRow(i).getCell(6).getNumericCellValue().toBigDecimal();

// Установка значений для созданного объекта
RplTst_AllDbTypesApi.setjJson(rop_RplTst_AllDbTypes1, vjson);
RplTst_AllDbTypesApi.setsSystemName(rop_RplTst_AllDbTypes1, vsystemname);
RplTst_AllDbTypesApi.setsCaption(rop_RplTst_AllDbTypes1, vcaption);
RplTst_AllDbTypesApi.setgidRefAny(rop_RplTst_AllDbTypes1, vgidrefclassany);
RplTst_AllDbTypesApi.setNumber(rop_RplTst_AllDbTypes1, vNumber);

// Проверка и установка idObjectType только если он не равен 0
if (vidobjecttype != 0) {
    RplTst_AllDbTypesApi.setidObjectType(rop_RplTst_AllDbTypes1, vidobjecttype);
}

// Установка даты
RplTst_AllDbTypesApi.setdDate(rop_RplTst_AllDbTypes1, vdate);

// Сохранение изменений
commit();

// Увеличение счетчика строк
i = i + 1;
}

// Возврат булевого значения, необходимого для выполнения скрипта
true;
}

```

Запуск скрипта

После написания функции `fun`, выполните скрипт, и появится модальное окно для выбора файлов Excel. Выберите необходимые файлы, и функция начнет обработку данных и создание объектов в

системе на основе данных из Excel.

С помощью этой библиотеки вы можете эффективно работать с данными Excel и использовать записанные Jexl-скрипты для массовой обработки и создания объектов по сложным сценариям.

Полный тестовый скрипт обработки Excel:

```
var fun = x -> {
  var sheet = x.getSheet("Лист1");
  var lastRowNum = sheet.getLastRowNum();
  var i = 0;
  while (i <= lastRowNum ){
    var rop_RplTst_AllDbTypes1 = RplTst_AllDbTypesApi.insert();
    var vjson = sheet.getRow(i).getCell(0).getStringCellValue();
    var vsystemname = sheet.getRow(i).getCell(1).getStringCellValue();
    var vcaption = sheet.getRow(i).getCell(2).getStringCellValue();
    var vgidrefclassany = sheet.getRow(i).getCell(3).getStringCellValue();
    var vidobjecttype = sheet.getRow(i).getCell(4).getNumericCellValue().toLong();
    var vdate = toDate(sheet.getRow(i).getCell(5).getStringCellValue());

    RplTst_AllDbTypesApi.setjJson(rop_RplTst_AllDbTypes1, vjson);
    RplTst_AllDbTypesApi.setsSystemName(rop_RplTst_AllDbTypes1, vsystemname);
    RplTst_AllDbTypesApi.setsCaption(rop_RplTst_AllDbTypes1, vcaption);
    RplTst_AllDbTypesApi.setgidRefAny(rop_RplTst_AllDbTypes1, vgidrefclassany);
    if (vidobjecttype != 0) RplTst_AllDbTypesApi.setidObjectType(rop_RplTst_AllDbTypes1,
↵vidobjecttype);
    RplTst_AllDbTypesApi.setdDate(rop_RplTst_AllDbTypes1, vdate);

    commit();
    i = i+1;
  }
  true;
}

lib("Btk_XlsxLib").uploadParseFiles(fun);
```

Внимание

- **Пустые ячейки:** могут возвращать 0 вместо null — добавляйте проверки.
- **Типы данных:** следите за соответствием типов Excel и системы. В приведенном примере, используется метод getStringCellValue() для получения текстовых значений, и getNumericCellValue().toLong() для числовых значений.
- **Возвращаемое булево значение:** функция должна возвращать true/false.

Работа с файлами

Загрузка во временный каталог

```
files.uploadFileToTemp(res -> {  
    var file = res.file(); //java.io.File  
    //чтение содержимого файла  
    var utils = new('org.apache.commons.io.FileUtils');  
    var content = utils.readFileToString(file, "UTF-8");  
  
    dialogs.showMessageDialog(content);  
    true;  
});
```

Примечание

Работает только для jexl-скриптов в контексте выборки.

Выгрузка строки как файла

```
var text = "Привет!"  
files.downloadBytes(text.getBytes("UTF-8"), "Имя_Файла.txt", true, false);
```

Примечание

Работает только для jexl-скриптов в контексте выборки.

Необходимые права

Для использования сервиса:

1. Стандартные права на приложение и доступ к Сервис → Инструменты → Закончить запись JEXL-скрипта;
2. Привилегия Доступ к инструментам меню Сервиса для объекта Btk_ManagementPkg.

Коллекции и структуры данных

В этом разделе описаны основные типы коллекций в JEXL: списки, массивы, множества, карты и кортежи. Вы узнаете, как объявлять коллекции, добавлять и удалять элементы, а также выполнять основные операции для работы с данными. Материал содержит практические примеры для каждого типа коллекций.

Список

Список является изменяемым типом данных (mutable). Это значит, что можно добавлять и удалять элементы, а также изменять существующие.

Исходный массив для примера:

```
var mvStock = ["P-1001", "P-1002", "P-1003", ...];
```

Основные операции

Объявление списков:

```
var mvStockApproachOne = [...];
```

Добавление элементов (только для mutable):

```
mvStock.add("P-1004");
```

Получение элементов:

```
mvStock[0]; // способ 1  
mvStock.get(0); // способ 2
```

Проверка наличия элемента:

```
mvStock.contains("P-1002"); // возвращает true/false
```

Удаление элементов (только для mutable):

```
mvStock.remove("P-1002");
```

Размер списка:

```
mvStock.size(); // возвращает количество элементов
```

Проверка на пустоту:

```
mvStock.isEmpty(); // возвращает true/false
```

Поиск индекса элемента:

```
mvStock.indexOf("P-1001"); // возвращает индекс или -1
```

Замена элемента (только для mutable):

```
mvStock.set(0, "newElement"); // способ 1  
mvStock[0] = "newElement"; // способ 2
```

Объединение списков (только для mutable):

```
var listUnion = [1,2,3, ...];  
mvStockApproachOne.addAll(listUnion);
```

Очистка списка (только для mutable):

```
mvStock.clear(); // удаляет все элементы
```

Массив

Массив — это неизменяемая (immutable) структура данных с фиксированным размером. Вы не можете добавлять и удалять элементы, но можно изменять значения существующих элементов (если они сами по себе изменяемы).

Исходный массив для примера:

```
var mvStock = ["P-1001", "P-1002", "P-1003"];
```

Основные операции

Объявление массива:

```
var mvStockApproachOne = [];  
var mvStockApproachTwo = [1, 2, 3];
```

Получение элементов:

```
mvStock[0]; // способ 1  
mvStock.get(0); // способ 2
```

Проверка наличия элемента:

```
mvStock.contains("P-1002"); // возвращает true/false
```

Размер списка:

```
mvStock.size(); // количество элементов
```

Поиск индекса элемента:

```
mvStock.indexOf("P-1001"); // возвращает индекс или -1
```

Множество

Доступны только mutable (изменяемые) множества. **Основные операции**

Создание множества:

```
var setData = {"P-1001", "P-1002", "P-1003"}; // Множество с элементами  
var setEmptyData = {}; // Пустое множество
```

Добавление элементов:

```
setData.add("P-1004"); // Добавление одного элемента  
setData.addAll({"P-1005", "P-1006"}); // Добавление другого множества  
setData.addAll(["P-1007", "P-1008"]); // Добавление элементов из списка
```

При добавлении списка порядок элементов множества сбивается.

Проверка и получение элементов:

```
var exists = setData.contains("P-1001"); // Проверка наличия → true/false
```

Удаление элементов:

```
setData.remove("P-1001"); // Удаление по значению  
setData.clear(); // Полная очистка множества
```

Работа с размерами:

```
var size = setData.size(); // Количество элементов  
var isEmpty = setData.isEmpty(); // Проверка на пустоту → true/false
```

Операции с множествами:

```
// Пересечение (оставляет только общие элементы)  
setData.retainAll({"P-1005", "P-1006"});  
  
// Разность (удаляет элементы другого множества)  
setData.removeAll({"P-1002", "P-1003"});
```

Карта

Далее в примерах будем работать с этой картой:

```
var mapData = {"bwhasError" : 11, "svKeyTransaction" : 21}; // объявление словаря
```

Внимание

Если в карте добавляется несколько элементов с одинаковыми ключами, то последнее значение перезапишет предыдущее. В словаре все ключи уникальны, и каждому ключу соответствует только одно значение.

```
var mapData = {  
    "bwhasError": 11,  
    "svKeyTransaction": 21, // Это значение будет перезаписано  
    "svKeyTransaction": 22 // Останется только это  
};  
// Итоговая карта: {"bwhasError": 11, "svKeyTransaction": 22}
```

Основные операции со словарями

Получение и добавление элемента:

```
mapData.e = 5; // добавление нового ключа (способ 1)  
mapData.put("c", 3); // добавление нового ключа (способ 2)  
mapData["d"] = 4; // добавление нового ключа (способ 3)  
  
var approachTwo = mapData.bwhasError; // получение значения по ключу (способ 1)  
var approachOne = mapData.get("bwhasError"); // получение значения по ключу (способ 2)  
var approachThree = mapData["bwhasError"]; // получение значения по ключу (способ 3)
```

Получить и добавить значению по ключу можно тремя способами: через точку, квадратные скобки, метод `get`.

- Точечная нотация работает только с ключами, которые являются валидными идентификаторами (без пробелов, дефисов, начинающиеся с буквы).
- Для ключей со спецсимволами (тире, пробелы, точки) используйте квадратные скобки.
- Методы `put()`/`get()` работают всегда, но требуют больше кода.

Пример с ограничениями:

```
var mvStock = {"P-1001" : 1, "P-1002" : 2, "P-1003" : 3};  
var nCode = mvStock.P-1001; // Дефис воспримется как оператор
```

Проверка наличия ключа:

```
mapData.containsKey("bwhasError"); // возвращает true/false
```

Удаление элемента:

```
mapData.remove("bwhasError"); // удаляет пару "ключ-значение" по ключу
```

Размер словаря:

```
mapData.size(); // количество пар "ключ-значение"
```

Получение всех ключей:

```
mapData.keySet(); // возвращает массив ключей (например, ["bwhasError", "svKeyTransaction"  
↪])
```

Получение всех значений:

```
mapData.values(); // возвращает массив значений (например, [11, 21])
```

Проверка на пустоту:

```
mapData.isEmpty(); // true, если карта пустая
```

Изменение значения по ключу:

```
mapData["bwhasError"] = 10; // присваивает новое значение ключу
```

Добавление новой пары «ключ-значение»:

```
mapData["newKey"] = 42; // добавляет новый ключ с значением
```

Очистка словаря:

```
mapData.clear(); // удаляет все элементы
```

Объединение словарей:

```
var newMap = {"key1": 1, "key2": 2};  
mapData.addAll(newMap); // добавляет все пары из newMap в mapData (существующие ключи ↪  
↪перезаписываются)
```

Операция entrySet:

```
var mapData = {"bwhasError" : 11, "svKeyTransaction" : 21};
var setMap = mapData.entrySet(); // Получаем множество пар ключ-значение

for (entry : setMap) { // Итерация по всем элементам (парам ключ-значение)
    var key = entry.getKey(); // Получить ключ текущей пары
    var value = entry.getValue(); // Получить значение текущей пары
}
```

Возвращает множество(Set) пар ключ-значение - объектов Map.Entry.

Каждый Entry содержит:

- getKey() - уникальный ключ;
- getValue() - соответствующее значение.

Перебор значений идет в обратном порядке.

Когда использовать entrySet():

- Когда нужны и ключи, и значения в цикле.
- Для модификации значений через entry.setValue().
- Для получения «снимка» всех данных Map

Получение ключа по значению

Нет встроенной функции для получения ключа по значению, но можно написать свою локальную функцию для этой цели:

```
// Получения ключа по значению
var findKeysByValue = function(map, targetValue) {
    var matchedKeys = [...]; // Сюда будем складывать подходящие ключи
    for (let key : map.keySet()) { // Перебираем каждый ключ
        if (map[key] == targetValue) { // Если значение по ключу совпадает с искомым
            return key; // Добавляем ключ в результат
        }
    }
    return matchedKeys; // Возвращаем найденные ключи в виде списка
};

var keyListForValue = findKeysByValue(mapData, 11);
dialogs.showMessageDialog(toString(keyListForValue));
```

По такому же принципу можно реализовать удаление ключа по значению.

Кортеж

Кортежи — это неизменяемые (immutable) структуры данных, которые позволяют хранить фиксированное количество элементов разных типов. Они полезны, когда нужно временно сгруппировать данные без создания отдельного класса.

В JEXL отсутствует встроенный тип данных Кортеж (Tuple), но его можно эмулировать с помощью интеграции с Scala. Кортежи Tuple2 — это естественный способ передать такие данные из JEXL в Scala-код.

```
var tuple2 = (v1, v2) -> { return new(`scala.Tuple2`, v1, v2); };
```

- new(scala.Tuple2, ...) — создает экземпляр Scala-кортежа, используя JVM-интеграцию.
- Функция принимает 2 аргумента и возвращает неизменяемую пару (v1, v2).

В Scala стандартные кортежи (Tuple) ограничены 22 элементами (до Tuple22):

```
var tuple22 = (v1, v2, v3,v4, v5, v6,v7, v8, v9,v10, v11, v12, v13, v14, v15,v16, v17, ↵  
↪ v18,v19, v20, v21,v22) -> {  
    return new(`scala.Tuple22`, v1, v2, v3,v4, v5, v6,v7, v8, v9,v10, v11, v12, v13, v14,  
↪ v15,v16, v17, v18,v19, v20, v21,v22);  
};
```

```
dialogs.showMessage(toString(tuple22(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ↵  
↪ 16, 17, 18, 19, 20, 21, 22)));
```

Чтобы передавать параметры в Scala-коллекции, используем asScala():

```
// Массив пар [Имя_параметра, Значение]  
var tuple2Params = asScala([  
    tuple2("npInParam1", 101), // Параметр 1 (Long)  
    tuple2("npInParam2", 1251), // Параметр 2 (Long)  
    tuple2("npInParam3", 10001), // Параметр 3 (Long)  
    ... // Остальные элементы (если есть)  
]);
```

Особенности:

- asScala() — преобразует JEXL-массив в Scala-коллекцию;
- каждый элемент коллекции — кортеж («имя», значение).

Кортежи неизменяемы, что гарантирует:

- параметры не поменяются случайно во время выполнения процедуры;
- нет риска коллизий в многопоточных сценариях.

Контроль выполнения: условия, функции и обработка ошибок

В этом разделе описаны основные конструкции управления выполнением кода в JEXL: условные операторы, циклы, методы и функции, обработка ошибок. Материал содержит синтаксис и практические примеры для каждого элемента.

Условные конструкции и циклы

Условия if-else

Синтаксис:

```
if (условие) {  
    // блок кода при true  
} else {  
    // блок кода при false  
}
```

Пример:

```
if (order.total > 10000) {  
    discount = 0.1;  
} else if (order.total > 5000) {  
    discount = 0.05;  
} else {  
    discount = 0;  
}
```

Тернарный оператор

Синтаксис:

```
условие ? значение_если_true : значение_если_false
```

Пример:

```
// Простой пример  
var access = (id >= 18) ? "Разрешено" : "Запрещено";
```

Циклы

While

Синтаксис:

```
while (условие) {  
    // действие  
}
```

Пример:


```
// Простой пример
var i = 0;
while (i < 10) {
    total++;
    i++;
}
```

For

При использовании циклов важно понимать область видимости переменных:

- Если итерируемая переменная объявлена с `let` или `const` - она доступна только внутри блока цикла.
- Если переменная объявлена без ключевого слова (неявно) - она ведёт себя как `var`:
 - Доступна после выполнения цикла.
 - Сохраняет последнее присвоенное значение.
 - Если переменная с таким именем уже существовала, то переменная созданная в цикле затрет значение ранее объявленной переменной.

Способ 1:

Классический C-подобный цикл с индексом.

Синтаксис:

```
for (инициализация; условие; шаг) {
    // тело цикла
}
```

Пример:

```
let x = 0;
for (let i = 0; i < 10; ++i) {
    x += i
}
dialogs.showMessage(toString(x));
```

Подходит также для массивоподобных структур, где есть доступ по индексу (`array[i]`).

```
let x = 0;
let array = [100, 200, 300];
for (let i = 0; i < size(array); ++i) {
    x += array[i];
}
```

Позволяет гибко управлять индексом коллекции (например, менять шаг `i += 2`).

Способ 2:

Синтаксис:

```
for (let элемент : коллекция) {
    // тело цикла
}
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
// элемент доступен только здесь  
}
```

Используется для перебора элементов итерируемой структуры (массива, коллекции и т. д.). Итерируемая переменная получает значение каждого элемента на каждой итерации.

```
let x = 0;  
let array = [1, 2, 3];  
for (let item : array) {  
    x += item;  
}
```

Переменная `item` доступна только внутри цикла (если используется `let`).

Способ 3:

Синтаксис:

```
for (элемент : коллекция) {  
    // тело цикла  
    // элемент доступен здесь  
}  
// элемент ДОСТУПЕН и здесь (ведёт себя как var)
```

Если итерируемую переменную не объявлять никаким ключевым словом (`let`, `const`, `var`), то она будет объявлена как `var`, и будет иметь последнее присвоенное значение.

Пример коллекции:

```
let x = 0;  
let array = [1, 2, 3];  
for (item : array) {  
    x += item;  
}  
dialogs.showMessageDialog(toString(item)); // Будет выведено значение '3'
```

- `item` не объявляется явно — она создаётся неявно и «живёт» после цикла.
- Может привести к неожиданным эффектам, если `item` уже была определена ранее.
- Обход результатов объектного запроса с преобразованием в список.

```
var l= toJRops(Btk_SomeEntityApi.byParent(rop)).asList()  
for(r:l){  
    logInfo(r.id);  
    logInfo(r.sSystemName)  
}
```

Пример множества:

```
var mvStock = {"P-1001", "P-1002", "P-1003"};  
for(item : mvStock) {  
    dialogs.showMessageDialog(item);  
}
```

Пример карты:

```
var mvStock = {"P-1001" : 1, "P-1002" : 2, "P-1003" : 3};
for(item : mvStock) {
    dialogs.showMessageDialog(item); // Здесь item это значения словаря по ключу
}
```

ForEach

Синтаксис:

```
коллекция.foreach(элемент -> {
    // действия с элементом
});
```

В JEXL оператор foreach используется исключительно для функционального подхода, когда данные поступают в виде потока (например, из SQL-запроса или коллекции).

Ключевые особенности:

- потоковая обработка;
- нет возвращаемого значения.

Пример:

```
sql("SELECT name FROM users").foreach(row -> {
    logInfo("User: " + row.name);
});
```

Вызов методов и функций в JEXL

Базовый вызов методов

Синтаксис:

```
объект.метод(параметры)
```

Пример:

```
dialogs.showMessageDialog(ropvInvoice);
dialogs.showMessageDialog(toString(ropvInvoice.nsum));
```

Поддерживается цепочка вызовов:

```
sheet.getRow(0).getCell(1)
```

Определение пользовательской функции

Синтаксис:

```
function имяФункции(параметры) {  
    // тело функции  
    результат; // опционально  
}
```

Пример:

```
function calculateSum(a, b) {  
    // Тело функции  
    a + b; // Возвращаемый результат (неявный return)  
}  
  
dialogs.showMessageDialog(toString(calculateSum(2, 3)));
```

Объявление локальных функций

Синтаксис:

```
var fun_LoadInvoiceTemplate = function(x) {  
    // Тело функции  
    return true;  
};
```

Пример:

```
function test(params) {  
    dialogs.showMessageDialog("hello world");  
}  
  
const FX_DeleteInvisibleSymbolsF = function(sText) {  
    var query = sql(`Select regexp_replace('${sText}', '[\\s\\u200B\\u200C\\u200D\\uFEFF]', ' ', 'g') as ShortText;`)  
    ↪.asSingle();  
    return query.shorttext;  
}  
  
//-----  
//main code block  
//-----  
{  
    var sCaptionLong = 'A 694 EM 147_2024 12 105'  
    var sCaptionFunc = FX_DeleteInvisibleSymbolsF(sCaptionLong);  
    dialogs.showMessageDialog('FUNC: ' + sCaptionFunc);  
}  
  
const FX_DeleteInvisibleSymbolsF = function(sText) {  
    var query = sql(`  
        SELECT regexp_replace('${sText}', '[\\s\\u200B\\u200C\\u200D\\uFEFF]', ' ', 'g')  
        ↪AS ShortText;
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
    `).asSingle();  
    return query.shorttext;  
}  
  
//-----  
// Main code block  
//-----  
{  
    var sCaptionLong = 'A 694 EM 147_2024 12 105';  
    var sCaptionFunc = FX_DeleteInvisibleSymbolsF(sCaptionLong);  
    dialogs.showMessageDialog('FUNC: ' + sCaptionFunc);  
}
```

Обработка ошибок

Jexl в ядре не поддерживает исключения, механизм исключений реализован через аннотации.

Синтаксис:

```
@begin{  
    //code  
}@exception function(e){  
    // code  
}
```

Для бросания исключения можно воспользоваться командой `raise: raise("Текст исключения");`

Пример:

```
const ErrorHandler = new java.util.concurrent.atomic.AtomicReference(null); // Объявим  
↪обработчик ошибок  
  
... // Произвольный блок кода  
  
var svLogList = ''; // Переменная для лога успешности и ошибок  
ErrorHandler.set(null); // Сбросим обработчик ошибок перед попыткой поймать ошибку  
  
@begin { // Блок кода, в котором попытаемся поймать ошибку  
    ... // Произвольный блок кода  
    raise("Текст ошибки"); // Событие, которое вызывает ошибку  
}  
@exception function(vEx1) { // Обработка исключения  
    ErrorHandler.set(vEx1); // Сохраняем ошибку в наш обработчик  
}; // Обязательно ставим точку с запятой  
  
const ErrorCatch = ErrorHandler.get(); // Запросим значение из обработчика  
  
if (isNotNull(ErrorCatch)) { // Проверяем, была ли ошибка  
    svLogList += `r\n - Ошибка! ${ErrorCatch.getCause().getMessage()}`;  
} else {  
    svLogList += `r\n - Успешно!`;  
}
```

(продолжается на следующей странице)

```
dialogs.showMessage(svLogList); // Отображение логов на экране
```

Особенности работы в системе Global

Система Global предоставляет расширенные возможности для работы с данными через JEXL-скрипты. В статье описаны основные методы работы с переменными, выборками, данными и внешними сервисами.

Наименование переменных для nullable типов и атрибутов

Правила именования переменных: [Variable][a][t][Scope][Name][Suffix].

Компоненты именования:

- **[Variable]** — определяет тип данных:
 - n — число;
 - s — строка;
 - j — строка в формате JSON;
 - d — дата;
 - r, x — запись;
 - u, cur — курсор;
 - l, blob — бинарные данные;
 - c — символьные данные;
 - b — булево значение;
 - id — идентификатор;
 - gid — глобальный идентификатор.
- **[a]** — определяет, является ли переменная последовательностью;
- **[t]** — определяет пользовательский тип;
- **[Scope]** — область действия:
 - v — переменная процедуры;
 - p — параметр процедуры;
- **[Name]** — имя переменной;
- **[Suffix]** — суффикс:
 - _dz — системные атрибуты;
 - _z — проектные атрибуты.

Примеры именования параметров:

- dStart — дата начала в таблице;
- dvStart — дата начала переменной процедуры;

- dgMaxDate — максимальная дата переменной пакета;
- tvdaDate — тип коллекции дат в процедуре;
- davDate — коллекция дат в процедуре;
- uvStudents — курсор в процедуре.

Основные методы и принципы работы

Поиск методов класса

Для просмотра доступных методов любого класса выполните следующие шаги:

1. Перейдите в **Настройки системы** → **Обозреватель проектов**.
2. Введите имя класса (например, `Stm_OrderOut`).
3. Откройте файл с суффиксом `Api` (например, `Stm_OrderOut_Api`).
4. Перейдите на вкладку **Методы**.

В этом разделе отображаются все доступные методы класса. Большинство классов содержат стандартный набор методов для работы.

Работа с текущей выборкой

При запуске JEXL-скрипта часто происходит работа внутри существующей выборки (например, при выделении строки в таблице).

С помощью `selection.getVar` можно получить атрибуты текущей выборки:

```
var id = selection.getVar("атрибут");
var idvOrder = selection.getVar("id"); // "id" - название атрибута
```

Загрузка строки через gor

gor — специальный тип данных для переменных, соответствующий строке объекта в таблице базы данных. Внутри данной строки можно обратиться к любому физическому столбцу. По сути это проводник к строке в базе данных.

Принцип работы:

1. Сначала проверяет наличие данных в кэше.
2. Если данных нет — загружает их из базы.
3. Предоставляет удобный доступ к полям строки.

Получить строку можно с помощью метода `load`. Полученное значение будет типа `gor`:

```
ropOrder = Stm_OrderOutApi.load(idvOrder);
```

Доступ к атрибутам строки

Обычные атрибуты (колонки таблицы) доступны напрямую через точку:

```
var svNumDoc = ropOrder.sNumDoc; // Номер документа
var idvClient = ropOrder.idClient; // ID клиента
```

Работа с JSON-атрибутами на примере Stm_OrderOut

Получение значений атрибутов:

1. Получение всего JSON объекта:

```
var jvParams = ropOrder.jParams_dz; // Получаем полный JSON-объект параметров
```

2. Получение конкретного значения по имени атрибута:

```
var svStation = Stm_OrderOutApi.getObjAttrValue(ropOrder, 'station'); // Получаем значение атрибута 'station'
```

3. Получение значения по ID атрибута:

```
// Получаем ID класса из текущей выборки
var idvClass = selection.getVar('idClass'); // Получим id класса
// Находим ID атрибута по мнемокоду 'station' для указанного класса
var idvAttrStation = Btk_AttributeApi.findByMnemonicCode(idvClass, 'station');
// Получаем значение атрибута для указанного заказа
var svStationById = Stm_OrderOutApi.getObjAttrValue(ropOrder, idvAttrStation);
```

Установка значений атрибутов:

1. Установка по имени атрибута — передать название класса и название атрибута:

```
Stm_OrderOutApi.setObjAttrValue(ropOrder, 'station', "Новая станция");
```

2. Установка по ID атрибута:

```
// Получаем ID класса из текущей выборки
var idvClass = selection.getVar('idClass');
// Находим ID атрибута по мнемокоду 'station' для указанного класса
var idvAttrStation = Btk_AttributeApi.findByMnemonicCode(idvClass, 'station');
// Получаем значение атрибута для указанного заказа
var svStationById = Stm_OrderOutApi.getObjAttrValue(ropOrder, svStationById);
Stm_OrderOutApi.setObjAttrValue(ropOrder, svStationById, "Новое значение"); // Установка значения для атрибута с указанным ID
```


Изменение значений

Чтобы изменить значение в строке, используйте специальные методы-сеттеры. У каждого класса свои сеттеры (смотрите в методах API):

```
// Меняем номер документа
Stm_OrderOutApi.setsNumDoc(ropOrder, "НОВЫЙ-001");

// Меняем клиента
Stm_OrderOutApi.setidClient(ropOrder, 12345L);
```

Поиск по мнемокоду

Часто нужно получить ID по коду (например, код способа доставки):

```
// Получаем ID способа доставки "ЖД"
var idvZHD = Bs_DeliveryMethodApi.findByMnemonicCode('ZHD');
```

Внимание

код чувствителен к регистру! „ZHD“ и „zhd“ — разные коды.

Пример полного скрипта:

```
// 1. Получаем ID из текущей выборки
var idvOrder = selection.getVar("id");

// 2. Загружаем строку заказа
var ropOrder = Stm_OrderOutApi.load(idvOrder);

// 3. Получаем атрибуты
var svNumDoc = ropOrder.sNumDoc; // Номер
var idvStationEndJson = Stm_OrderOutApi.getObjAttrValue(ropOrder, 'idStationEndJson');

// 4. Меняем номер
Stm_OrderOutApi.setsNumDoc(ropOrder, "НОВЫЙ-002");

// 5. Ищем ID способа доставки
var idvDelivery = Bs_DeliveryMethodApi.findByMnemonicCode('ZHD');
```

Преобразование типов данных

Метод	Пример использования
<code>.asJLong()</code>	<code>"123".asJLong()</code>
<code>.toLong()</code>	<code>"456".toLong()</code>
<code>.toInteger()</code>	<code>"789".toInteger()</code>
<code>.toDouble()</code>	<code>"12.34".toDouble()</code>
<code>.toBigDecimal()</code>	<code>"123.45".toBigDecimal()</code>
<code>.asDate()</code>	<code>"2024-01-01".asDate()</code>
<code>.toDate("формат")</code>	<code>toDate("01.01.24", "dd.MM.yy")</code>
<code>.toString()</code>	<code>123.toString()</code>
<code>.asString()</code>	<code>321.asString()</code>
<code>.asList()</code>	<code>sql("SELECT...").asList()</code>
<code>.asSingle()</code>	<code>sql("SELECT...").asSingle()</code>
<code>.asScala()</code>	<code>[1,2,3].asScala()</code>
<code>.asJava()</code>	<code>scalaSeq.asJava()</code>
<code>.toJObject()</code>	<code>toJObject('{ "id":1 }')</code>

Работа с датами

Метод	Описание	Пример
<code>getFirstDay</code>	Возвращает дату начала месяца от текущей даты	<code>var a = getFirstDayOfMonth();</code> <code>var b = getFirstDayOfMonth('27.01.2025 12:42:10');</code>
<code>getFirstDay</code>	Возвращает первый день указанного периода	<code>var a = getFirstDayOfPeriod(sysDate(), 'M');</code> <code>var b = getFirstDayOfPeriod('27.01.2025 12:42:10', 'Q');</code>
<code>getLastDayC</code>	Возвращает дату конца месяца от текущей даты	<code>var a = getLastDayOfMonth();</code> <code>var b = getLastDayOfMonth('27.01.2025 12:42:10');</code>
<code>getLastDayC</code>	Возвращает последний день указанного периода	<code>var a = getLastDayOfPeriod(sysDate(), 'M');</code> <code>var b = getLastDayOfPeriod('27.01.2025 12:42:10', 'Q');</code>
<code>nanoTime</code>	Получить значение <code>System.nanoTime()</code>	<code>var a = nanoTime();</code>
<code>sysDate</code>	Возвращает текущую дату	<code>var a = sysDate();</code>
<code>toDate</code>	Переводит строку в дату по указанному формату	<code>var a = toDate("27.02.2020 12:42:10");</code> <code>var b = toDate("27.02.2020 12:42:10", "dd.MM.yyyy HH:mm:ss");</code>
<code>truncDate</code>	Получение даты на начало дня от переданной	<code>var a = truncDate(sysDate());</code>
<code>truncYear</code>	Получение даты на начало года от переданной	<code>var a = truncYear(sysDate());</code>
-	Вычитает из даты указанное количество дней	<code>var dYesterday = sysDate() - 1;</code>
+	Добавляет к дате указанное количество дней	<code>var dTomorrow = sysDate() + 1;</code>

Работа в контексте выборки

Контекст выборки (JexlSelScript) расширяет контекст бизнес-логики возможностью работы с выборками и пользовательским интерфейсом. Используется в тех случаях, когда необходимо получить данные из полей для пользовательского ввода.

Метод	Описание	Пример использования
<code>varExists(name)</code>	Проверяет существование переменной в текущей и мастер-выборках	<code>if (varExists("client_id")) { ... }</code>
<code>selfVarExists(name)</code>	Проверяет существование переменной только в текущей выборке	<code>if (selection. selfVarExists("temp_value")) { ... }</code>
<code>getVar(name)</code>	Получает значение переменной из текущей или мастер-выборки	<code>var id = selection. getVar("doc_id").asLong();</code>
<code>getSelfVar(name)</code>	Получает значение только из текущей выборки	<code>var code = selection. getSelfVar("product_code");</code>
<code>setVar(name, value)</code>	Устанавливает значение переменной (ищет в иерархии выборок)	<code>selection.setVar("status", "approved");</code>
<code>setSelfVar(name, value)</code>	Устанавливает значение только в текущей выборке	<code>selection.setSelfVar("comment", "Не заполнено");</code>
<code>addVar(name, value)</code>	Добавляет новую переменную в текущую выборку	<code>selection. addVar("processed_at", sysDate());</code>
<code>newForm()</code>	Создает новую форму	<code>Act_DocEdit.newForm().params(..).open();</code>
<code>.form()</code>	Возвращает форму текущего контекста	<code>selection.form();</code>
<code>.</code>	Возвращает указанную операцию для дальнейшего выполнения	<code>selection. opers("CREATESTMORDERIN");</code>
<code>opers('Операция')</code>	Выполняет ранее полученную операцию	<code>selection. opers("CREATESTMORDERIN"). execute();</code>

Работа с формой выборки

Создание и настройка формы:

```
var res = Btk_ClassAvi // Класс, предоставляющий форму выбора
.list() // Инициализация формы списка (табличное отображение)
.newForm() // Создание новой формы
.params({ // Передача параметров в форму в виде карты
    "one": 1,
    "two": 2,
    "three": 3,
    "more": "many more"
})
.locates({"id": 146800}) // Выбор строки
.results(["id", "idClass"]) // Какие поля вернуть после выбора
.openLookup(); // Открытие окна
```

Метод	Описание
<code>.list()</code>	Форма будет отображаться как таблица (список записей). Возможные отображения можно посмотреть в обозревателе проекта в классе с расширением <code>Avi</code>
<code>.newForm()</code>	Создает новую формы
<code>.params()</code>	Передает параметры в форму (могут использоваться для фильтрации/логики)
<code>.locates()</code>	Установка строки на активную
<code>.results()</code>	Задаёт поля, которые вернутся после выбора
<code>.openLookup()</code>	Открывает окно выбора
<code>.findSelection('0</code>	Ищет и возвращает выборку , содержащую указанное отображение
<code>.baseRep()</code>	Возвращает ссылку на объект, к которому был вызван метод

Пример: Поиск и выполнение операций в связанной выборке

Демонстрирует, как найти выборку, связанную с конкретным документом, и выполнить в ней операцию.

```
var sel = selection.form() // получает форму по выделенному элементу
.findSelection(Stm_OrderOutAvi.Card() // Ищет и возвращает **выборку**, содержащую
↪ указанное отображение
.baseRep()); // возвращает базовое представление объекта (ссылку)

if (sel != null) { // если выборка найдена, то выполняет действие
    sel.opers("CREATESTMORDERIN").execute(); // выполняет переданную операцию
}
```

Для работы с текущей выборкой можно использовать сокращенный синтаксис:

```
selection.opers("CREATESTMORDERIN").execute();
```

Обработка результата:

```
if (res.getLookupResult() == LookupResult.ok()) {
    // Пользователь выбрал значение и нажал "OK"
    for(var i: 1 .. res.size()) { // Цикл по всем выбранным записям
        dialogs.showMessage("id " + res.getData(i, 0) + " idClass " + res.getData(i, 1));
    }
} else {
    // Пользователь закрыл форму или нажал "Отмена"
    dialogs.showMessage("Значение не было выбрано");
}
```

- `res.getLookupResult()` — проверяет, как закрыли форму:
 - `LookupResult.ok()` — нажали «Выбрать»;
 - `LookupResult.cancel()` — нажали «Отмена» или закрыли окно.
- `res.size()` — количество выбранных записей (если форма поддерживает множественный выбор).
- `res.getData(i, 0)` — доступ к данным:
 - `i` — номер строки (начинается с 1);
 - `0` — индекс поля (соответствует порядку в `.results([«id», «idClass»])`).

Пример работы с мастер-выборками:

```
// Получаем значение из атрибута "id" выборки, приводим его к типу NString и пишем в
↳ переменную idTree
var idTree = getVar("id").asNString();

// Получаем атрибут DGLOBALENDDATE из мастер-выборки
var dEndDate = getVar("super$DGLOBALENDDATE").asNDate();

// Вызываем метод из пакета с передачей параметров
// Обратите внимание, для обращения к Api или пакетам используются их короткие имена
↳ (без скобок)
var fltCond = Act_UniversalReportPkg.getUniFilterCondByIdTree(idTree, dEndDate);

// Вызывается ещё один метод, возвращающий объект scala-класса immutable.Map[NString,
↳ Any]
var filters = Act_UniversalReportPkg.getFilterValues(idTree);

/* Определение Map-ы внутри jexl-скрипта. Отметим, что Map внутри jexl и объект scala-
↳ класса
Map (неважно mutable или immutable) - это разные объекты. Наиболее важным
фактом является то, что передать scala-объект Map, полученный в предыдущей
строке,
в пакетный метод, принимающий scala-объект Map, в jexl-скрипте напрямую нельзя,
именно поэтому приходится получать значения из переменной filters,
перезаписывать
их в jexl-Map param и потом передавать param в scala-метод. */

var param = {
  "flt_idDepOwner" : filters['flt_idDepOwner'],
  "flt_idAcc" : filters['flt_idAcc'],
  "flt_dFrom" : filters['flt_dFrom'],
  "flt_dTo" : filters['flt_dTo'],
  "flt_idAdjustMethod" : filters['flt_idAdjustMethod'],
  "flt_idAccCor" : filters['flt_idAccCor'],
  "uniFilterCondition_dz": fltCond
};

// Открываем новую форму с переданными параметрами
Act_TransAvi.defList().newForm().params(param).open();
```

Мастер-выборки:

- Доступ через префикс super\$;
- Пример: getVar("super\$PARENT_ID").

Типизация:

- Используйте .asString(), .asDate() для явного преобразования;
- Для чисел: .asBigDecimal(), .asJLong().

Дополнительные методы работы с формой выборки

Открывает мастер пользовательского ввода по указанному идентификатору формы.

```
Btk_WizardLib().launch(getSelfVar("id").asNLong); // В качестве параметра передается id  
↪ формы.
```

Работа с SQL

Для работы с SQL используются команды:

1. Для запросов на чтение:

```
sql(sqlText:String)
```

2. Для запросов на запись:

```
tsql(sqlText:String).execute()
```

Методы обработки для sql:

Метод	Описание	Пример
.asList()	Вернуть список записей	sql("SELECT * FROM users").asList()
.asSingle()	Вернуть одну запись	sql("SELECT * FROM users WHERE id=1").asSingle()

Методы обработки для tSql:

Метод	Описание	Пример
.execute()	Выполнить запрос	tsql("DELETE FROM temp").execute()

Примеры:

Простые запросы:

```
// Чтение  
var activeUsers = sql("SELECT name FROM users WHERE is_active=true").asList();  
  
// Запись  
tsql("UPDATE orders SET status='done' WHERE id=42").execute();
```

Обработка результатов:

```
// Вывод имен пользователей  
sql("SELECT name FROM users").foreach(row -> {  
    logInfo("User: " + row.name);  
});  
  
// Подсчет суммы  
var total = 0;  
sql("SELECT amount FROM payments").foreach(p -> {
```

(продолжается на следующей странице)

```
total = total + p.amount;
});
```

Параметризация:

```
var productId = selection.getVar('id');
var product = sql("SELECT * FROM products WHERE id=${productId}").asSingle();

var productId = selection.getVar('id');
var sSqlText = ''; // объявляем переменную, в которой будем собирать sql-выражение
sSqlText = "SELECT * FROM products WHERE id="; //
sSqlText += toString(productId);
sSqlText += "\r\n order by id";
var product = sql(sSqlText).asSingle();

var productId = selection.getVar('id');
var product = sql(`
    SELECT *
    FROM products
    WHERE id=${productId}`)
.asSingle();
```

Дополнительные методы

Метод	Описание	Пример использования
<code>lpad(str, len, ch)</code>	Дополнение строки слева	<code>lpad("5", 3, "0") → "005"</code>
<code>rpadd(str, len, ch)</code>	Дополнение строки справа	<code>rpadd("5", 3, "0") → "500"</code>
<code>flush()</code>	Синхронизация с БД	<code>flush()</code>
<code>commit()</code>	Подтверждение транзакции	<code>commit()</code>
<code>nv1(a, b)</code>	Возвращает a, если не null, иначе b	<code>nv1(var, "default")</code>
<code>isNull(x)</code>	Проверка на null	<code>isNull(obj)</code>
<code>isNotNull(x)</code>	Проверяет, что переданное значение не null	<code>isNotNull(obj)</code>
<code>pgArrayToLong</code>	Конвертация массива в список	<code>pgArrayToLongList(arr)</code>
<code>parseId(gid: Object): Long</code>	Получение идентификатора объекта из NGid-a. Работает для строкового или NGid-значения	<code>var sGid = "14323/44334"; var id = parseId(sGid);</code>
<code>parseIdClass(g Object): Long</code>	Получение идентификатора класса из NGid-a. Работает для строкового или NGid-значения	<code>var sGid = "14323/44334"; var idClass = parseIdClass(sGid);</code>
<code>toJRops(rops: Traversable[Row], JexlToJRops)</code>	Обход записей, возвращенных объектным запросом, например <code>byParent</code> . Выполняет запрос на чтение к базе данных и позволяет обойти записи	<code>var l = toJRops(Btk_SomeEntityApi.byParent(rop)).asList(); for(r:l){ logInfo(r.id); logInfo(r.sSystemName); }</code>
<code>toJObject(json String): JexlToJObject</code>	Парсинг json-объекта	<code>var jData = toJObject('{ "id": 1 }'); logInfo(jData.getLong("id"));</code>

Работа с BTS процедурами

Переход в раздел процедур:

Для того чтобы открыть перечень BTS-процедур, перейдите в **Настройки системы** → **Настройки и сервисы** → **Процедуры**. Здесь можно просмотреть различные варианты существующих процедур. Можно перейти в карточку любой процедуры и посмотреть, какие действия там описаны.

Выберите в фильтре **Тип: Процедура для сбыта** и создайте новую тестовую процедуру, нажав кнопку **Создать**. Введите наименование и код процедуры.

Написание JEXL-кода:

Предположим, мы хотим рассчитать некоторую формулу на основе входящих параметров. Чтобы процедура вернула значение, объявим раздел с результатом и вернем соответствующее значение. Объявим входные параметры и расчетную формулу:

```
var nvParam1 = npInParam1;
var nvParam2 = npInParam2;
var nvParam3 = npInParam3;
var nvSum = (npInParam1 + npInParam2) * npInParam3;

// Для того чтобы процедура вернула значение, добавим возврат результата:
return nvSum;
```

Сохраните код.

Вызов процедуры через JEXL-редактор:

Перейдите в отдельный редактор JEXL. Объявите служебную функцию для типизации данных. Данная функция будет возвращать тип данных `scala.Tuple2`, состоящий из двух переменных.

```
var tuple2 = (v1, v2) -> { return new(`scala.Tuple2`, v1, v2); };
```

Заведите массив параметров, в котором запишите входящие значения. В нем будет объявлен массив Scala (это делается за счет указания последнего элемента с троеточием). Первые три элемента будут объявлены с помощью функции `tuple2`. В каждом таком элементе массива будет указано название параметра и его значение.

```
var tuple2Params = asScala([
    tuple2("npInParam1", 101),
    tuple2("npInParam2", 1251),
    tuple2("npInParam3", 10001),
    ...
]);
```

Объявите переменную, в которую при помощи API-функции `findByMnemonicCode` поместите идентификатор ранее созданной процедуры:

```
val idvBtsProcedure = Bts_ProcedureApi.findByMnemonicCode(`Sale_Test1`);
```

Далее вызовите конструкцию для расчета значения BTS-процедуры и запишите результат в переменную. Для этого используйте API-метод `Bts_ProcedureLib`, передав в него:

- идентификатор процедуры;
- массив параметров;
- дополнительный контекст выборки для расчета.


```
var nvSumCalc = Bts_ProcedureLib.execById(idvBtsProcedure, tuple2Params, selection.  
↳coreRep());
```

Метод `Bts_ProcedureLib.execById` ожидает параметры в формате, который понимает Scala, поэтому мы используем кортеж. Кортежи здесь — это «мост» между JEXL и Scala, обеспечивающий четкую, безопасную и удобную передачу параметров.

Верните результат процедуры и выведите его на экран.

```
dialogs.showMessage(toString(nvSumCalc));
```

Аналогичным образом можно создавать и более сложные BTS-процедуры.

Работа с мониторингом сессий сервера приложений

Система автоматически ведет метаинформацию о сессиях всех пользователей. Доступ к этой информации организован следующим образом:

Настройки системы → Сервис → Инструменты → Мониторинг сессий сервера приложений.

В списке отображаются все активные сессии с указанием последнего выполненного действия для каждого пользователя.

Для работы с сессионными данными текущего пользователя предусмотрены следующие методы:

Получение текущего действия:

```
session.applicationInfo().action();
```

Обновление информации о действии:

```
session.applicationInfo().action_$eq("Значение текущего действия");
```

Пример использования:

```
for (ropWW: ropaIntWar) {  
    session.applicationInfo().action_$eq(``Phosagro_SB0205_NEW - ВП - ${toString(ropWW.  
↳gid)}``);  
    Stk_InternalWarrantApi.setdExecDateOut(ropWW, cDate);  
    session.applicationInfo().action_$eq(``Phosagro_SB0205_NEW - выполняем ВП - $  
↳{toString(ropWW.gid)}``);  
    Stk_InternalWarrantApi.setidStateOut(ropWW, idvStateIntWar);  
    session.applicationInfo().action_$eq(``Phosagro_SB0205_NEW - Создаем PCO``);  
    var ropWarOut = Stk_WarrantOutApi.createWarrantOutByInternalWarrant(ropWW);  
    // Заполняем даты исполнения  
    Stk_WarrantOutApi.setdDocDate(ropWarOut, cDate);  
    Stk_WarrantOutApi.setdExecDate(ropWarOut, cDate);  
    // Выполняем PCO  
    session.applicationInfo().action_$eq(``Phosagro_SB0205_NEW - выполняем ВП - $  
↳{toString(ropWarOut.gid)}``);  
    Stk_WarrantOutApi.setidState(ropWarOut, idvStateWarr);  
};
```

Метод `setUniChar Value` используется для установки значения универсальной характеристики (универсального атрибута) объекта в системе.

```
setUniCharValue(rop: ApiRop, idpUniChar: Long, pValue: Any): Unit
```

Параметр	Тип (Scala)	Описание
rop	Rop	Контекст выполнения операции (объект, к которому применяется изменение)
idpUniChar	Long	Идентификатор универсальной характеристики
pValue	Any	Новое значение характеристики (может быть Long, String и др)

Пример использования:

```
// 1. Получаем GID значения "." из справочника
val gidChr = sql("""
    SELECT r.gid
    FROM Btk_UniversalReference r
    JOIN btk_objecttype b ON r.idobjecttype = b.id
    WHERE r.sSystemname = '.'
    AND b.scode = 'ur_02C_sort_type'
""").asSingle();

Stm_OrderOutApi.setUniCharValue(rop, '02C_sort_type', gidChr.gid);
```

Работа с печатью

Пользователь может:

- Сформировать отчёт по его системному имени;
- Запустить JEXL-скрипт печатной формы, привязанной к типу объекта.

Создание отчета происходит с помощью следующего метода:

```
Rpt_Lib.createReportExJexl(
    reportName: String,
    reportVersionDate: Date,
    postBuildAction: PostBuildAction,
    propertyMap: Map[String, Any],
    idpObjectTypePrintForm: NLong = None.n1
): Unit
```

Параметр	Тип (Scala)	Описание
idpObjectTypePrintFor	Long	id ПФ на типе объекта. Может быть null
reportName	String	Имя отчета
reportVersionDate	Date	Дата
postBuildAction	PostBuildAction	Действие, которое необходимо произвести после заполнения отчёта
propertyMap	Map[String, Any]	Карта входящих параметров

Пример использования:

```

var idvOrderClass = selection.getVar("idClass");
var idvOrder = selection.getVar("id");

var NLong = function(number) { // локальная функция для преобразования Long в NLong
    return new ("ru.bitec.app.gtk.lang.NLong", number);
};

var spReportName = "Stm_InvoiceOut";
var dpReportVersionDate = sysDate();
var vPostBuildAction = session
    .sbtClassLoader()
    .loadClass('ru.bitec.app.gtk.gl.postbuildaction.PostBuildAction')
    .design();

var propertyMap = {
    "idSrcObject" : NLong(idvOrder),
    "idSrcClass" : NLong(idvOrderClass)
};

var idpObjectTypePrintForm = null;

Rpt_Lib.createReportExJexl(
    spReportName,
    dpReportVersionDate,
    vPostBuildAction,
    asScala(propertyMap),
    idpObjectTypePrintForm
);

```

Тип PostBuildAction:

PostBuildAction — это перечисление (enum) из класса ru.bitec.app.gtk.gl.postbuildaction.PostBuildAction, определяющее действия после построения отчета.

Доступные значения:

- `.show()` — загружает отчет на клиент;
- `.save()` — сохраняет отчет;
- `.print()` — загружает отчет на клиент и, если в конфигурации системы включена опция печати (`Configuration.Printing.enableNetworkPrinting`), печатает отчет на локальном (для сервера) принтере.

Примечание: Для `.print()` поведение зависит от конфигурации сервера. Если печать не настроена, отчет просто загружается на клиент.

Особенности:

- Для работы с ID используется тип NLong (внутренний класс системы);
- `asScala()` преобразует Java-коллекции в Scala-совместимые;
- Если `idpObjectTypePrintForm = null` — отчет строится по `reportName`.

Практические примеры

В этом разделе рассмотрены два практических сценария: автоматическое создание пунктов маршрута для складов и обработка перехода состояний документов с валидацией бизнес-правил.

Создание пункта маршрута для каждого склада

1. SQL-запрос для получения списка складов. Получим ID и коды складов, у которых тип объекта «Склад» (stock) или «Склад в пути» (StockInWay). Полученные записи преобразуем в список с помощью метода `asList()`.

```
var sqlStock = sql(`
    SELECT stk.id as id, stk.sCode as sCode FROM stk_stock stk
    JOIN btk_objecttype ot ON ot.id = stk.idObjectType
    where ot.sCode = 'stock' OR ot.sCode = 'StockInWay'
`).asList()
```

2. Поиск пункта маршрута по коду склада. С помощью цикла `for` сделаем преобразование над полученными записями. По коду нашего склада получим `id` пункта маршрута.

```
var sqlStock = sql(`
    SELECT stk.id as id, stk.sCode as sCode FROM stk_stock stk
    JOIN btk_objecttype ot ON ot.id = stk.idObjectType
    where ot.sCode = 'stock' OR ot.sCode = 'StockInWay'
`).asList()

for(r: sqlStock) {
    var idvPoint = Tms_PointApi.findByMnemoCode(r.scode);
}
```

3. Создание нового пункта маршрута (если не найден). Если вернувшееся значение пункта маршрута равно `null`, то нужно создать его, заполнив код, название и привязку текущего склада.

```
var sqlStock = sql(`
    SELECT stk.id as id, stk.sCode as sCode FROM stk_stock stk
    JOIN btk_objecttype ot ON ot.id = stk.idObjectType
    where ot.sCode = 'stock' OR ot.sCode = 'StockInWay'
`).asList()

for(r: sqlStock) {
    var idvPoint = Tms_PointApi.findByMnemoCode(r.scode);
    if (isNull(idvPoint)) {
        var ropPoint = Tms_PointApi.insert();
        Tms_PointApi.setsCode(ropPoint, r.scode);
        Tms_PointApi.setsCaption(ropPoint, r.scode);
        Tms_PointApi.setidStock(ropPoint, r.id);
    }
}
```

4. Обновление существующего пункта маршрута (если не привязан к складу). Если полученный пункт маршрута существует, то загружаем его по полученному ранее `id`. Далее проверяем у полученного пункта маршрута привязку к складу, если значение равно `null`, то устанавливаем значение текущего склада.

```

var sqlStock = sql(`
    SELECT stk.id as id, stk.sCode as sCode FROM stk_stock stk
    JOIN btk_objecttype ot ON ot.id = stk.idObjectType
    where ot.sCode = 'stock' OR ot.sCode = 'StockInWay'
` ).asList()

for(r: sqlStock) {
    var idvPoint = Tms_PointApi.findByMnemoCode(r.scode);
    if (isNull(idvPoint)) {
        var ropPoint = Tms_PointApi.insert();
        Tms_PointApi.setsCode(ropPoint, r.scode);
        Tms_PointApi.setsCaption(ropPoint, r.scode);
        Tms_PointApi.setidStock(ropPoint, r.id);
    } else {
        var ropPointW = Tms_PointApi.load(idvPoint);
        if (isNull(ropPointW.idStock)) {
            Tms_PointApi.setidStock(ropPointW, r.id);
        }
    }
}

```

Пример расширения обработки состояния документа

1. Проверка на пустое старое значение:

```
if(isNull(oldValue)) return 0;
```

Если текущее состояние документа (oldValue) не задано, обработка прекращается (возвращается 0).

2. Получение порядковых номеров состояний:

```
var idStateFromOrder = Btk_ClassStateApi.getOrder(oldValue);
var idStateToOrder = Btk_ClassStateApi.getOrder(value);
```

- idStateFromOrder - порядковый номер **старого состояния** документа;
- idStateToOrder - порядковый номер **нового состояния**.

Получим текущее состояние документа, воспользовавшись API-методом класса состояния.

3. Проверка условий для обработки:

```
if(idStateFromOrder <= 100 && idStateToOrder > 100 && rop.idObjectType == Btk_
ObjectypeApi.findByMnemoCode("Tms_DeliveryAct"))
```

- idStateFromOrder <= 100 - проверяет, что текущее состояние документа имеет порядковый номер <= 100. Обычно это соответствует статусам «Аннулировано» или «Формируется».
- idStateToOrder > idStateFromOrder - гарантирует, что новое состояние действительно является переходом ВВЕРХ по workflow. Исключает случаи, когда статус меняется на равный или более ранний.
- rop.idObjectType == Btk_ObjectypeApi.findByMnemoCode(«Tms_DeliveryAct») - убеждается, что работаем именно с документом типа «Рейс» (Tms_DeliveryAct).

4. Получение заказа (Stm_OrderOut):

```
var rop00 = Stm_OrderOutApi.loadByGid(rop.gidSrc);
if(rop00.idObjectType != Btk_ObjectTypeApi.findByMnemonicCode("OrderOut_Service"))
```

Получаем ропу заказа с помощью Api-метода loadByGid по gidSrc(родитель) текущего документа. Если тип заказа **НЕ** «OrderOut_Service» (то есть это не сервисный заказ), выполняется дальнейшая проверка.

5. Определение договора:

```
var ropContract = null;
if(isNotNull(rop.idAddContract))
    ropContract = Cnt_ContractApi.load(rop.idAddContract);
else
    ropContract = Cnt_ContractApi.load(rop.idContract);
```

Если у заказа (rop) есть доп. договор (idAddContract), он загружается. Иначе загружается основной договор (idContract).

6. Проверка отложенного ценообразования:

```
var bDeferredPricing = Cnt_ContractApi.getObjAttrValue(ropContract, "bDeferredPricing");
if(bDeferredPricing == true)
```

Получаем значение атрибута bDeferredPricing (флаг «Отложенное ценообразование»). Если флаг true, выполняется проверка даты.

7. Проверка даты отложенного ценообразования:

```
var dDeferredPricing = Stm_ActOutApi.getObjAttrValue(rop, "dDeferredPricing");
if(isNull(dDeferredPricing) || dDeferredPricing < sysDate())
    raise("Требуется выполнить корректировку цены");
```

Получаем дату dDeferredPricing (когда должна быть выполнена корректировка цены). Если дата **не** задана или **уже** прошла, выводится ошибка.

Общий код:

```
if(isNull(oldValue))return 0;
var idStateFromOrder = Btk_ClassStateApi.getOrder(oldValue)
var idStateToOrder = Btk_ClassStateApi.getOrder(value);

if(idStateFromOrder <= 100 && idStateToOrder > 100 && rop.idObjectType == Btk_
↳ObjectTypeApi.findByMnemonicCode("Tms_DeliveryAct")){
    var rop00 = Stm_OrderOutApi.loadByGid(rop.gidSrc)
    if(rop00.idObjectType != Btk_ObjectTypeApi.findByMnemonicCode("OrderOut_Service")){
        var ropContract = null
        if(isNotNull(rop.idAddContract))
            ropContract = Cnt_ContractApi.load(rop.idAddContract)
        else
            ropContract = Cnt_ContractApi.load(rop.idContract)
        var bDeferredPricing = Cnt_ContractApi.getObjAttrValue(ropContract,
↳"bDeferredPricing")
        if(bDeferredPricing == true){
            var dDeferredPricing = Stm_ActOutApi.getObjAttrValue(rop, "dDeferredPricing")
            if(isNull(dDeferredPricing) || dDeferredPricing < sysDate()){
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        raise("Требуется выполнить корректировку цены")
    }
}
}
```

13.8 Дополнительно

Локализация приложений

Локализация приложений осуществляется средствами Java Resource Bundle и специальными тэгами, внедряемыми в код и метаданные приложений.

Словари

Для хранения строк на разных языках используются словари. Словарём является пакет ресурсов (Resource Bundle), содержащий properties-файлы со строками {ключ}={значение}:

- {имя_словаря}.properties
Словарь по умолчанию
- {имя_словаря}_{язык}.properties
Словарь для указанного языка.
- {имя_словаря}_{язык}_{локаль}.properties
Словарь для указанного языка с уточнением локали.

Создание словаря

Новый словарь создается через контекстное меню от каталога с ресурсами.

1. В idea перейдите в каталог в котором хотите создать словарь
2. Вызовите на каталоге контекстное меню
3. Выполните пункт New > Resource Bundle

Имя словаря должно совпадать с именем сущности или пакета.

В создавшиеся файлы, добавьте локализованные строки в формате {ключ}={значение}

```
Caption=Валюта
ID=Идентификатор
MultiLineName=Многострочный \
текст
```

Обратите внимание, что при создании многострочной ресурсной строки, каждую строку необходимо заканчивать символом \.

Использование локализованных строк

Локализация строк доступна в:

- Scala-коде `*Dvi/Avi` – классов
- Scala-коде `*Dpi/Api` – классов
- Scala-коде `*_Pkg` – классов
- Аннотациях `@Oper`, `@MainSelection`
- `*.avm.xml` – файлах.

Для свойств, хранящих «наименование»

Для получения значения локализованной строки из словаря, имя которого совпадает с текущей сущностью, необходимо указать ключ локализованной строки, обрамлённый тэгами: `[#Ключ_строки]`

Если необходимо получить значение строки из произвольного словаря, необходимо указать полный путь к словарю: `[#ru/bitec/app/{модуль}/Словарь.Ключ_строки]`

Локализованные строки в Scala-коде

Для подстановки локализованного значения строки в Scala-код используется макрос `ls"#Ключ_строки"` или `ls""#Ключ_строки""`.

```
def foo(): Unit = {  
    showMessage(ls"[#MyLocalizedMessage]")  
}
```

Локализованные строки в аннотациях

```
@MainSelection(caption = "[#Gs3_StockMainMenu.caption]")  
object Gs3_StockMainMenuAvi extends ProjectApplicationAvi {}
```

Локализованные строки в *.avm.xml

```
<representation caption="#caption">  
    <attributes>  
        <attr name="sSystemName"  
            caption="#sSystemName"  
            isVisible="true"/>  
    </attributes>  
</representation>
```


Ввод значений на другом языке

Операция **Ввести текст на другом языке** позволяет ввести значение на другом языке для выбранного редактируемого поля \

Операция доступна если:

1. Ввод включен в характеристиках класса или глобально в настройках модуля **btk**
2. Поле в выборке называется так же как и атрибут у редактируемого объекта класса
3. Поле редактируемо
4. Поле должно быть выбрано(нажали на поле ввода)

В списке выберете редактируемое поле и в контекстном меню выберете **Информация -> Ввести текст на другом языке** В карточке выберете редактируемое поле и на тулбаре **Информация -> Ввести текст на другом языке**

Включение

Для класса **Карточка** класса -> **Характеристики** -> **Доступен ввод текста на других языках**

Глобально **Настройки и сервисы** -> **Настройки модулей системы** -> **Общие настройки модулей** -> **btk** -> **bTextLangEnabledAll**

Пользовательские (динамические) приложения

Пользовательские (динамические) приложения позволяют создавать и настраивать новые приложения без изменения исходного кода. Такой «Low-code» подход ускоряет внедрение бизнес-логики.

Основные возможности

1. **Создание приложений**
добавление нового приложения и его настройка из интерфейса, без привлечения программиста.
2. **Гибкая конфигурация**
детальная настройка операций главного меню приложения.
3. **Управление правами доступа**
возможность настроить права для приложения и его операций.
4. **Миграция конфигураций**
созданные приложения можно переносить между базами с помощью менеджера конфигураций.

Создание нового приложения

1. Перейдите в раздел **Настройка системы > Сущности > Динамические приложения**
2. Нажмите на кнопку **Создать**
3. Заполните все обязательные поля:
4. В разделе **Дополнительно** карточки приложения выполните операцию **Синхронизировать административные объекты**

После этого пользовательское приложение будет доступно для выбора в стандартном списке приложений

Примечание

Мнемокод (имя) динамического приложения нельзя изменять после создания!

Пункты меню

Главное меню пользовательского приложения формируется на основе Default-отображения указанного Avi-контроллера (по умолчанию — Bs_DynamicMainMenuAvi, который наследует настройки от ApplicationAvi):

Пункт меню	№ п/п	И...	Активность
Обработка оповещений, созданных из RabbitMQ	9000		✓
Обработка сообщения RabbitMQ	9000		✓
Сервис	0		✓
Отчёты	99		✓
Помощь	999	✗	✓
Избранное	3	★	✓
Клонированная операция записи Jexl-скриптов	10000		✓
Обработка оповещения о просроченной лицензии	9000		✓
Обработка оповещения о получении входящей задачи	9000		✓

Верхняя синяя панель: пункты главного меню динамического приложения с настройками по умолчанию. Также включает в себя автоматически заполняемые фильтры

Структура главного меню: те же пункты, представленные в виде дерева в карточке приложения

Внимание

После каждого изменения структуры пунктов меню **НЕОБХОДИМО** создавать адм. объекты
Для этого можно использовать операцию "Синхронизировать административные объекты" в разделе "Дополнительно" карточки динамического приложения)

Логика работы пунктов меню задаётся Jexl-скриптом, который выполняется в контексте текущей выборки. Для пользовательских пунктов меню скрипт можно редактировать

Добавление пунктов меню

Пункты меню можно создавать двумя способами:

1. Из динамического меню

Импортирует выбранные элементы со всеми предками. Такие пункты меню используют для открытия отображения функцию `ru.bitec.app.btk.menu.Btk_MenuTreeLib#callMenuProcess` с соответствующим мнемокодом, что упрощает перенос с одной базы на другую

2. Вручную

Добавляет новую операцию с настройками по умолчанию, после чего её параметры можно откорректировать

Редактирование структуры и свойств

Для редактирования пунктов главного меню в детализации к карточке динамического приложения доступно дерево операций **Структура главного меню**. Она содержит как операции Avi-контроллера, так и добавленные пользовательские элементы.

- Изменение основных свойств (порядковый номер, иконка, активность) можно выполнять прямо в дереве **Структура главного меню**
- Дополнительные параметры редактируются на вкладке **Свойства**
- Jexl-скрипт операции задаётся на вкладке **Jexl-скрипт**

Примечание

Внутренняя логика работы с пунктами меню пользовательских приложений полностью повторяет пользовательские jexl-операций (см. раздел «Пользовательские jexl-операции»)

- В качестве мастер выборки всегда используется
`Btk_Sel0perOverride.sMasterSel = 'Btk_DynamicMainMenuAvi'`
- Мастер отображение совпадает с мнемокодом динамического приложения
`Btk_Sel0perOverride.sMasterRep = '<Имя динамического приложения>'`
- Все операции так же имеют префикс `jexl_`

Управление иерархией

Для задания вложенности пунктов меню используется атрибут `headOperation` — мнемокод родительской операции. Изменить его можно во вкладке **Структура главного меню > Свойства**: при клике на кнопку выбора откроется список доступных операций.

Логирование

При необходимости логирования в базу данных, необходимо обеспечить:

- независимость от основной ветки изменения данных
- высокую отказоустойчивость
- низкую нагрузку на сервер/базу данных

Для реализации такого логирования существует специальный инструмент – логирующая транзакция.

Данный инструмент позволяющий выделить запись логов в отдельную сессию. Для уменьшения количества обращений к базе производите сброс данных из логирующей транзакции в базу не чаще, чем раз в 1000 update/insert.

Пример:

```
val logTransaction = new LogTransaction() {  
  override protected def run(): Unit = {  
    Rpl_IntImportLogApi().setError(Rpl_IntImportLogApi().load(idp), value)  
    commitByInterval()  
  }  
}  
  
logTransaction.execute()
```

Более подробно примеры использования можно посмотреть в модуле интеграции Rpl ru.bitec.app.rpl.channel.intg.in.Rpl_IntImportLogApi – методы с постфиксом LT.

При выполнении `.execute()` сервер производит попытку найти логирующую сессию, если она уже была инициализирована. Если не была – выделяется новая. Внутри блока `LogTransaction()` `implicit session` является логирующая сессия и все действия производимые с использованием `logSession` относятся к ней.

В `LogSession` происходит `commit`:

- При освобождении `LogSession` `LogSession` освобождается на `endWork`.

Смотрите также методы основной сессии:

- `beginWork`
- `endWork`

Проектные расширения

Точка расширения

Аббревиатуры:

- `Ept` - Точка расширения
- `Ext` - Расширения

Точка расширения (ExtensionPoint) – специфический пакет, методы которого объявлены таким образом, что к ним можно подключить дополнительный исполняемый код из других пакетов (расширений/Extension). Код расширений будет выполнен при вызове метода точки расширения.

```
class Btk_ExampleEpt extends ExtensionPoint {
  val doSomething = declFunc("doSomething") {
    (sf: SuperFunc[Int], a: Int, b: Int) =>
      val superVal = sf().getOrElse(0)
      a + b + superVal
  }
}
object Btk_ExampleEpt extends PkgFactory[Btk_ExampleEpt]
```

Пример представляет объявление «Точки расширения» с методом doSomething

Правила наименования для точек расширения, которые используются для дополнения функционала сеттеров, вставки, удаления – Xxx_Ept, где Xxx – имя модуля, для которого точка расширения создается (например Bs_Ept, Btk_Ept).

Расширение (Extension) – специфический пакет, методы которого подключаются к методам точки расширения, и выполняются при вызове соответствующего метода точки расширения.

К методу точки расширения может быть подключено произвольное число методов расширений.

При первом обращении к классу точки расширения, производится поиск и сканирование файлов META-INF/extensions.xml. Расширения, у которых свойство targetEpt совпадает с именем точки расширения, подключаются к точке расширения.

```
class Gs3_Btk_ExampleExt extends Extension {
  subFunc("doSomething") {
    (sf: SuperFunc[Int], a: Int, b: Int) =>
      val superVal = sf(a - 2, b).getOrElse(0)
      superVal + 2
  }
}
```

Пример представляет объявление метода расширения. На вход метод получает ссылку на метод расширения, находящийся ниже в очереди вызовов.

Внимание

Если не вызвать sf(), методы расширений, находящиеся ниже в очереди не будут вызваны.

Правила наименования для расширений, которые используются для дополнения функционала других модулей для сеттеров, вставки, удаления – Xxx_YyyExt, где Xxx – имя модуля расширения, Yyy – имя модуля, в котором находится точка расширения (например Mct_BsExt).

Конфигурирование расширений

Конфигурация расширений находится в файле META-INF/extensions.xml.

Структура файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<exts>
  <ext class="ru.bitec.app.btk.ext.Btk_ExceptionHandlerExt"
        targetEpt="ExceptionHandlerEpt"/>
</exts>
```

Порядок вызовов

По умолчанию, метод точки расширения получает порядковый номер «0», а все методы расширений – номер «10». В результате, в первую очередь будут вызваны все методы расширений, в произвольном порядке, в конце будет вызван код точки расширения.

Для изменения порядка вызовов необходимо задать порядковые номера методам расширений.

```
subFunc("doSomething", 1) {
  (sf: SuperFunc[Int], a: Int, b: Int) =>
}
}
```

Проектное перекрытие кода Api, Avi, Lib, Pkg

Для перекрытия кода Api, Avi, Lib, Pkg необходимо:

1. Создать свой класс-наследник от перекрываемого класса.
2. Зарегистрировать перекрытие класса в файле META-INF/overrides.xml

Пример перекрытия Avi-класса

Код исходного Avi-класса Gs3_RootTestAvi.scala

```
object Gs3_RootTestAvi extends Gs3_RootTestAvi

@AvmFile(name = "Gs3_RootTest.avm.xml")
class Gs3_RootTestAvi extends Gs3_RootTestDvi {
  override def list(): List = {
    new List {
      override def meta = this
    }
  }
}
```

Код перекрытия Avi-класса Gs3_RootTestOverrideAvi.scala

```
object Gs3_RootTestOverrideAvi extends Gs3_RootTestOverrideAvi

@AvmFile(name = "Gs3_RootTestOverride.avm.xml")
class Gs3_RootTestOverrideAvi extends Gs3_RootTestAvi {
  override def list(): List = {
    new List {
      override def meta = this
    }
  }
  trait List extends super.List {
  }
}
```

Аннотация @AvmFile может быть не указана, в этом случае будет использоваться аннотация из класса-предка.

Текст META-INF/overrides.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<overrides>
  <replace-with
    class="ru.bitec.app.gs3.test.Gs3_RootTestOverrideAvi">
    <when-type-is class="ru.bitec.app.gs3.test.Gs3_RootTestAvi"/>
  </replace-with>
</overrides>
```

Пример перекрытия Api

Код перекрытия

```
class Gs3_RootTestOverrideApi extends Gs3_RootTestApi {
}

object Gs3_RootTestOverrideApi extends ApiFactory[
  java.lang.Long,
  Gs3_RootTestAro,
  Gs3_RootTestOverrideApi
]
```

Текст META-INF/overrides.xml

```
<replace-with
  class="ru.bitec.app.gs3.test.collections.Gs3_RootTestOverrideApi">
  <when-type-is
    class="ru.bitec.app.gs3.test.collections.Gs3_RootTestApi"/>
</replace-with>
```

Пример перекрытия Pkg

Код перекрытия

```
class Btk_OverridePkg extends Btk_Pkg {
}

object Btk_OverridePkg extends PkgFactory[Btk_OverridePkg]
```

Текст META-INF/overrides.xml

```
<replace-with class="ru.bitec.app.gs3.Btk_OverridePkg">
  <when-type-is class="ru.bitec.app.btk.Btk_Pkg"/>
</replace-with>
```

Пример перекрытия Lib

Код перекрытия

```
class Btk_OverrideLib extends Btk_Lib {
}

object Btk_OverrideLib extends AviLibFactory[Btk_Lib]
```

Текст META-INF/overrides.xml

```
<replace-with class="ru.bitec.app.gs3.Btk_OverrideLib">
  <when-type-is class="ru.bitec.app.btk.Btk_Lib"/>
</replace-with>
```


Запрет перекрытия

Для запрета проектного перекрытия кода Avi-класса, необходимо отметить класс аннотацией @Final.

```
@Final
class Gs3_RootTestAvi extends Gs3_RootTestDvi {}
```

Для запрета проектного перекрытия кода Api,Lib,Pkg классов, необходимо отметить класс аннотацией @Final или ключевым словом final.

```
@Final
class Gs3_RootTestApi extends Gs3_RootTestDpi[
  Gs3_RootTestAro,
  Gs3_RootTestApi,
  Gs3_RootTestAta
] {}
```

или

```
final class Gs3_RootTestApi extends
Gs3_RootTestDpi[
  Gs3_RootTestAro,
  Gs3_RootTestApi,
  Gs3_RootTestAta
] {}
```

Jexl расширения методов

С помощью расширений методов имеется возможность доработки и модификации стандартных серверных методов и операций пользовательских интерфейсов из приложения **Настройки системы**. Расширение представляет из себя jexl-скрипт, вызываемый перед или после выполнения стандартного метода/операции.

Расширения Api-методов

Добавление точек расширения в Api предусмотрено только в следующие методы:

- **Сеттеры атрибутов** – Dpi-сеттеры всех атрибутов выбранного класса;
- **AfterEdit** – Dpi-метод, вызываемый на окончание редактирования объекта класса, например при редактировании строки и перемещении курсора на другую строку;
- **BeforeEdit** – Dpi-метод, вызываемый в начале редактирования объекта класса;
- **FlushObject** – метод записи (сохранения) объекта в базу данных;
- **Delete** – Dpi-метод удаления объекта класса из базы данных;
- **Insert** – Dpi-метод создания нового объекта класса.

Примечание

Все расширения вызываются из „Dpi“. Т.е. срабатывание перед сеттером - это срабатывание перед вызовом Dpi-сеттера.

Примечание

Для методов «FlushObject» и «BeforeEdit» можно добавлять только точки расширения «До».

Выполнение этих точек будет производиться в контексте сессии, без использования интерактивной бизнес логики (т.е. нельзя использовать методы открытия всплывающих окон и диалогов, открытия каких либо интерфейсов системы).

Для подключения точек расширения Api-методов перейдите: Сущности - Обзорщик проекта. Выбрав Api нужного класса, перейдите на закладку Точки расширения. На закладке Расширения до или Расширения после создайте запись нового расширения и пропишите ему исполняемый скрипт.

В скрипт передаются следующие параметры:

- **rop** (роп объекта) - во всех случаях, кроме точек расширения «До» метода **Insert** и «После» метода **Delete**
- **idDeleted** (id удаленного объекта) - для точки расширения «После» метода **Delete**
- **value** (новое устанавливаемое значение) - для сеттеров
- **oldValue** (старое устанавливаемое значение) - для сеттеров

Пример скрипта с использованием вызова процедур из Bts:

```
Bts_ProcedureApi.execByMnemonicCode(  
    "ProcedureCode",  
    {"argName1": value1, "argName2": value2, ...}  
)
```

Пример выброса ошибки на FlushObject, если у объекта не заполнен атрибут sDescription:

```
var aro = rop.copyAro();  
if (aro.sDescription() == null){  
    raise("Не заполнено описание!")  
}
```

Пример выброса ошибки на Сеттере состояния, если у объекта не заполнен атрибуты:

```
var fromState = oldValue  
var toState = value  
if (fromState == 100 && toState == 300){  
    var aro = rop.copyAro();  
    if (aro.sAttr1() == null || aro.nAttr2 == null){  
        raise("Не заполнены все поля!")  
    }  
}
```

Расширения Avi-методов

Добавление точек расширения в Avi предусмотрено в сеттеры выборки, в ее операции или на события выборки, такие как `beforeEdit`, `afterEdit` и т.д.

Выполнение этих точек будет производиться в контексте выборки.

Для подключения точек расширения Avi-методов перейдите: **Настройки системы - Обозреватель проекта**. Выбрав нужную выборку, перейдите на закладку **Редактор операций**. На закладке **Расширения до** или **Расширения после** создайте запись нового расширения и пропишите ему исполняемый скрипт.

Пример подстановки даты детализации из мастера на операцию `insert`:

```
var ropParent = ParentApi.load(getVar("super$id").asNLong());
var ropDet = DetApi.load(getVar("id").asNLong());
DetApi.setdDate(ropDet, ropParent.dDate);
```

Пример подстановки значения фильтра на событие `beforeFirstOpen`:

```
setVar("flt_bNotActive", 1)
```

Перекрытие `acObject` и `acObjectItem`

Для перекрытия `acObject` и `acObjectItem` перейдите: **Настройки системы - Обозреватель проекта**. Выберите нужную выборку, перейдите на закладку **Настройки отображений**. Выберите нужное отображение и установите флаг переопределения `acObject/acObjectItem`. В закладках напишите JEXL-скрипт для получения нового значения. В JEXL-скрипте можно получить стандартное значение `acObject/acObjectItem` вызвав функцию `default()`;

Пользовательские события

Реализована возможность для `Api` добавлять собственные события, а так же для существующих стандартных событий добавлять новые типы срабатывания.

Добавление нового события

Новое событие будет отображено в списке событий `Api`. Оно создается без типов срабатываний, типы срабатывания привязываются к событию отдельным методом.

Пример добавления нового события:

```
Btk_MethodExtensionApi().registerClassEvent(
    idvClass,
    "SomeUserEventName",
    "Новое пользовательское событие"
)
```

Пример удаления пользовательского события:

```
Btk_MethodExtensionApi().unregisterClassEvent(
    idvClass,
    "SomeUserEventName"
)
```

Добавление типа срабатывания к событию

Стандартные события (сеттеры, вставка и тд.) имеют по умолчанию 2 типа срабатывания: До и После, а пользовательские события создаются без типа срабатывания. Реализована возможность добавления нового типа срабатывания к событию. Каждый тип срабатывания будет отображаться отдельной закладкой у события.

Создание нового типа срабатывания:

```
Btk_MethodExtensionTypeApi().register("SomeUserEventType", "На пересчет денормализации")
```

Создание связи между событием и типом срабатывания:

```
Btk_MethodExtensionApi().registerEventTypeToClassEvent(
    idvClass,
    "SomeUserEventName",
    Btk_MethodExtensionTypeApi().findByMnemonicCode("SomeUserEventType")
)
```

Удаление связи между событием и типом срабатывания:

```
Btk_MethodExtensionApi().unregisterEventTypeFromClassEvent(
    idvClass,
    "SomeUserEventName",
    Btk_MethodExtensionTypeApi().findByMnemonicCode("SomeUserEventType")
)
```

Вызов пользовательского события или пользовательского типа срабатывания из кода приложения

Разработчик в коде сам размещает вызов срабатывания события в нужном месте.

Пример вызова:

```
Btk_MethodExtensionApi().runMethodExtensionByClass(idvClass, "SomeUserEventType",
    ↪ "SomeUserEventName", "valueStr" -> "abc", "valueNum" -> 1.nn)
```

Переданные в метод значения параметров будут доступны в Jexl-скриптах этого события.

Пример обращения к переменной из Jexl-скрипта:

```
println("Значение переменной valueStr: " + valueStr + " Значение переменной valueNum: " +
    ↪ valueNum.toString());
```

Интеграция с сервером

Код инициализации модуля

На событие (пере-)загрузки прикладного кода, возможно выполнить свой код. Для этого, в каталоге со scala-кодом модуля, необходимо создать файл с именем {модуль}_ModuleInit.scala, содержащий

```
object {модуль}_ModuleInit {}
```

Пример:

```
Gs3_ModuleInit.scala
object Gs3_ModuleInit {
  // Размещённый тут код будет выполнен при запуске загрузчика классов
  SBT
}
```

Обработка системных событий сервера

В процессе работы, сервер генерирует ряд системных событий, который можно обработать в прикладном коде. Для этого, необходимо подписаться на события объекта `ServerEventSource` в код инициализации модуля.

```
object Gs3_ModuleInit {
  private val _logger: Logger =
    Logger.Factory.get("ru.bitec.app.gs3.Gs3_ModuleInit")
  ServerEventSource.subscribeOnInstanceStart(
    (conAccessor) => _logger.info("OnInstanceStart")
  )
  ServerEventSource.subscribeOnConnect(
    (conAccessor, btkUserInfo) =>
      _logger.info(s"OnConnect: ${btkUserInfo.name}")
  )
}
```

Перечень событий:

- **OnInstanceStart**
Событие возникает после подключения пула sql-соединений сервера приложений к базе данных
- **OnInstanceStop**
Событие возникает перед отключением пула sql-соединений сервера приложений к базе данных
- **OnConnect**
Событие возникает в процессе запуска пользовательской сессии, после успешной авторизации пользователя в системе. Для запрета подключения пользователя к системе, необходимо выбросить ошибку.
- **OnSessionStart**
Событие возникает после начала сеанса работы, сеанс работы начинается после авторизации, перед открытием выборки выбора приложения
- **OnSessionStop**
Событие возникает перед остановкой сеанса работы

Пример смотри: `ru.bitec.app.btk.Btk_ModuleInit`

Параллельные вычисления

Параллельные вычисления позволяют кратно ускорить расчет большого объема данных, за счет одновременного выполнения заданий по расчету.

Общий алгоритм параллельных вычислений в основной сессии:

1. Создать пул потоков
Основной характеристикой пула потоков является кол-во возможных одновременно исполняемых задач.
2. Запросить данные для расчетов
3. Разбить данные для расчета на пачки для параллельного вычисления
По каждой пачке необходимо сформировать задание на расчет
4. Направить задания для расчета в пул потоков
При этом задания на расчет выполняются параллельно в пуле потоков

Планирование допустимого количества параллельных задач

1. Пусть S - количество доступных активных сессий базы данных
 S можно приблизительно рассчитать как кол-во ядер доступных postgresql*2
2. Пусть C - количество ядер доступных серверу приложения
3. Рассчитать количество потоков как:

Наименьшее($C*2, S*2$)

Примечание

Данный алгоритм дает приблизительную оценку, так как оптимальное кол-во потоков зависит от соотношения нагрузки на диски и процессоры, а так же от административных квот на оборудование

Инструменты для параллельных вычислений

- `ru.bitec.app.gtk.eclipse.parallel.Parallel.withPool`
Метод получения пула для параллельных вычислений.
Доступен в контексте прикладной сессии(`api.pkg.avi`).
- `ASQL""select..."".withTempFileAs`
Запрос с сохранением результата в файл. Данный запрос позволяет минимизировать потребление памяти и сессий базы данных во время долгих вычислений.
Доступен в контексте прикладной сессии.
- `dialogs.withInfoForm`
Отображает сообщение с индикацией расчета.
Доступен в отображения
- `dialogs.showInfoForm`
Обновляет сообщение с индикацией расчета.
Доступен в отображении

Совет

Дополнительную информацию смотрите в документации методов.

Шаблон параллельных вычислений

```
dialogs.withInfoForm("Подготовка данных для расчета") {
  //Выполнить параллельные вычисления в 16 потоках
  Parallel.withPool(16) { pool =>
    //Размер пачки
    val batchSize = 500
    //Пачка для обработки
    val batch = ArrayBuffer.empty[NLong]
    //Количество обработанных записей
    var executedSize = 0

    //Отправка пачки на выполнение
    def submitBatch(): Unit= {
      val curSize = batch.size
      pool.submit(batch.toSet) { implicit session =>
        ids =>
          //ВНИМАНИЕ:
          //В процедуре вычисления не доступны данные из основной сессии
          //Поэтому работа может идти только с данными переданными
          //В процедуру submit, и полученными в текущие замыкание
          //в данном примере с именем ids
          //Передать можно только данные которые можно безопасно сериализовать
          //Попытка сослаться на другие данные приведет к ошибке в момент
          ↪ выполнения.

          //Массовая загрузка top
          for (r <- Btk_QueryPkg().largeInQuery(SomeApi(), "id", ids)) {
            //Выполнение вычислений
          }
        }.onSuccess { r =>
          //Обработка результатов выполнения идет в основном потоке
          //поэтому здесь доступно использование любых переменных

          //Расчет общего количества посчитанных данных
          executedSize = executedSize + curSize
          //Обновление диалога прогресса
          dialogs.showInfoForm(s"Выполнение расчетов ${executedSize}")
        }
      }
      batch.clear()
    }

    //В данном примере запрос с большими данными для вычисления сохраняется в файл
    //Это позволяет сократить затраты по оперативной памяти и используемым
    //сессиям базы данных в момент расчет
    //Возможно работа с несколькими полями for(id~name<-...withTempFileAs(nLong("id
    ↪")~nStr("id")) )
  }
}
```

(продолжается на следующей странице)

```

//Для этого необходимо сделать import anorm._
for (id <-
    ASQL"""
    SELECT t.id FROM SomeTable t
    """ .withTempFileAs(nLong("id"))
) {
    //Формирование пачек на параллельное выполнение
    batch += id
    if (batch.size >= batchSize) {
        submitBatch()
    }
}
//Вычисление оставшихся данных
if (batch.nonEmpty) {
    submitBatch()
}
}
}

```

Асинхронное обновление данных в связанных классах

Асинхронное обновление данных в связанных классах позволяет на событие в одном документе обновить атрибуты в другом, который может быть заблокирован в данное время.

Обновление происходит каждые пять минут.

Внимание

На выполнение операции обновления даётся десять попыток, то есть в течении пятидесяти минут атрибуты обновятся или нет, если документ всё ещё заблокирован.

Что бы добавить обновить данные в связанном классе, необходимо зарегистрировать её в очереди:

```

Btk_SyncDataQueueApi().register(
    sOpCode = "UpdateDocVerByDoc",
    sJexl =
        """var ropDocVer = Wf_DocVerApi.load(123L);
        |Wf_DocVerApi.setsCaption(ropDocVer, "new Caption");
        |""".stripMargin,
    gidSrc = parent_Rop.gid,
    gidPurpose = purpose_Rop.gid,
    isOnlyLastExecute = false,
    isExecuteAnotherBySrcOnError = true
)

```

Очередь группируется по „Коду операции“ и „Глобальному идентификатору объекта источника“ и образует подгруппу выполняемых скриптов. В свою очередь подгруппа сортируется по дате создания и внутри по идентификатору записи очереди.

Флаг `isOnlyLastExecute` означает, что в подгруппе будет выполнен только последний скрипт, остальные будут помечены как не обязательные. Если признак выставлен только на одной записи в подгруппе, он повлияет на всю группу.

Флаг `isExecuteAnotherBySrcOnError` означает, что если текущий скрипт был выполнен с ошибкой, то последующие скрипты всё равно будут выполняться. Если признак не стоит, то последующие скрипты не будут выполнены, но у них будет увеличен счётчик пропуска.

Журнал очереди обновления данных можно посмотреть в приложении «Настройка системы» -> «Настройки и сервисы» -> «Очередь синхронизации данных классов». В этом же отображении можно внепланово синхронизировать данные.

Все исполненные операции автоматически очищаются через 180 дней после создания. А все не исполненные через 365 дней.

Средства мониторинга работы системы.

Мониторинг системы реализован в двух независимых формах:

1. Средства мониторинга встроенные в сервер приложений
Реализованы по средствам обычных пользовательских интерфейсов системы, и расположены в меню инструментов.
2. Внешние средства мониторинга
Реализованы на основе протокола передачи телеметрии `OpenTelemetry`. Метрики передаются в инструмент визуализации `Grafana`

Мониторинг в приложениях GlobalERP

Для открытия внутренних средств мониторинга необходимо выполнить операцию главного меню Сервис - Инструменты - Монитор сессий сервера приложений

Монитор сессий сервера приложений

Пользовательский интерфейс, который содержит в себе набор инструментов, для просмотра состояния сервера приложений и базы данных.

Пользовательские сеансы

Закладка Пользовательские сеансы отображает список всех сессий сервера приложений. Если сервер приложений работает в режиме кластера, то будут отображены сессии других узлов.

Перечень основных столбцов:

- Системное имя пользователя и Пользователь - информация о пользователе
- Ip-адрес клиента - ip-адрес, с которого подключено клиентское приложение
- Системное имя приложения и Приложение - информация о запущенном приложении Global
- Системное имя активной формы и Активная форма - информация о текущей активной форме
- Действие - информация о выполняемых действиях сессии.
- Кластерный узел - Имя узла, к которому подключен клиент.
- Имя базы данных - имя БД, выбранное в окне логина при авторизации.
- блок квот - отображает включены ли квоты на сессию, и текущие использованные ресурсы.

Перечень закладок

- **Сессии базы данных** - отображает активные подключения к БД, выбранной пользовательской сессии. Позволяет просмотреть выполняемый запрос, и увидеть блокирующую сессию, если она есть
- **Расшифровка** - отображает информацию об открытых формах пользовательской сессии
- **Стек сеанса** - позволяет получить актуальный стек выполняемых действий пользовательской сессии.

Активные подключения к базе данных

Закладка **Активные подключения к базе данных** позволяет отобразить активные в данный момент запросы к БД, а так же информацию о пользовательских сессиях, выполняющих запросы.

Перечень основных столбцов:

- **Системное имя пользователя и Пользователь** - информация о пользователе
- **Ip-адрес клиента** - ip-адрес, с которого подключено клиентское приложение
- **Системное имя приложения и Приложение** - информация о запущенном приложении Global
- **Системное имя активной формы и Активная форма** - информация о текущей активной форме
- **Действие** - информация о выполняемых действиях сессии.
- **Кластерный узел** - Имя узла, к которому подключен клиент.
- **Имя базы данных** - имя БД, выбранное в окне логина при авторизации.

Инструменты анализа Базы данных

Так же расположены на форме **Монитор сессий сервера приложений** на закладке **Анализ базы данных**. Позволяют узнать состояние сессий БД, блокировки, выполняемые запросы и размер таблиц аудита по бизнес-объектам. [Подробнее о структурах аудита БД.](#)

Анализ размера таблиц

Инструмент, который позволяет увидеть текущий размер таблиц БД. Операция **История** позволяет отследить динамику увеличения таблиц

Работа автовакуума

Интерфейс, который отображает информацию о работе автоматической очистке таблиц.

Автовакуум - специальный процесс БД `postgresql`, который занимается очисткой устаревших данных в таблицах. Архитектура `postgresql` каждое изменение в таблице регистрирует новой записью, и требуется удалять устаревшие записи, которые не являются актуальными.

[Документация postgresql vacuum](#)

На этой закладке, основное внимание требуется уделять таблицам с выключенным автовакуумом. При нормальной работе системы, все таблицы должны автоматически очищаться.

Доступные операции:

- **Запуск vacuum full**

Команда на полную очистку таблицы с опцией **full**.

Выбирает режим «полной» очистки, который может освободить больше пространства, но выполняется гораздо дольше и запрашивает исключительную блокировку таблицы. Этот режим также требует дополнительное место на диске, так как он записывает новую копию таблицы и не освобождает старую до завершения операции. Обычно это следует использовать, только когда требуется высвободить значительный объем пространства, выделенного таблице.

- **Запуск vacuum analyze**

Обновляет статистику, которую использует планировщик для выбора наиболее эффективного способа выполнения запроса.

Операции следует запускать, только если автовакуум был отключен, или таблица часто изменяется, и автоматическая очистка не справляется.

Анализ активных запросов

Инструмент, который позволяет отобразить текущие активные запросы к БД, и информацию о вызвавших их пользовательских сессиях.

Перечень основных столбцов:

- **pid** - pid процесса БД, по которому можно идентифицировать сессию БД.
- **Время выполнения** - время выполнения запроса в минутах.
- **Состояние** - состояние сессии БД
Основные состояния:
 - **idle** - сессия БД находится в режиме ожидания. Т.е. подключение к БД есть, но в данный момент никаких запросов не выполняется. Стандартное ожидание для неактивных подключений, которые находятся в пуле подключений сервера приложений
 - **active** - сессия БД в данный момент выполняет запрос.
 - **idle in transaction** - сессия БД находится в режиме ожидания, но есть открытая транзакция. Опасное состояние сессии, т.к. длинные открытые транзакции отрицательно влияют на производительность **postgresql**. Наличие такой сессии сигнализирует о проблеме.
- **Системное имя пользователя и Пользователь** - информация о пользователе
- **Ip-адрес клиента** - ip-адрес, с которого подключено клиентское приложение
- **Системное имя приложения и Приложение** - информация о запущенном приложении **Global**
- **Форма** - информация о текущей активной форме
- **Кластерный узел** - Имя узла, к которому подключен клиент.
- **Действие** - информация о выполняемых действиях сессии. Эта колонка будет заполнена, если:
 - сессия принадлежит выполняемому фоновому заданию, в том числе интеграция
 - выполняется построение печатной формы

Операция История позволяет просмотреть исторические данные активных запросов, чья длительность превышала 15 минут.

Блокировки

Отображает дерево блокировок сессий БД.

Блокировка записи — метод предотвращения одновременного доступа к данным в базе данных, чтобы предотвратить противоречивые результаты. Чаще всего возникают, если несколько сессий БД пытаются изменить одну и ту же запись в таблице.

Корневыми записями этого дерева будут сессии, вызывающие какие-либо блокировки других сессий. Записи второго и следующих уровней показывают сессии, которые ожидают освобождения блокировок.

Таким образом можно проследить цепочки сессий, ожидающих освобождения блокировок.

Операция История позволяет просмотреть исторические данные по блокировкам БД.

Журнал алертов

Специальный журнал, в который периодически пишется информация о выявленных проблемах.

Журнал заполняется по средствам фоновых заданий:

1. Сохранить Долгий запрос - Btk_SlowQueryHistoryUpdate
2. Сохранить размер таблиц на дату - Btk_TableSizeHistoryUpdate

События записываемые в журнал:

1. Запрос БД выполняется более 15 минут
2. Резкое увеличение размеров таблиц
3. На таблице выключен автовакуум
4. Транзакция в БД выполняется более 15 минут
5. Фоновое задание с ночным расписанием выполняется в дневное время

Журнал размеров аудита

Доступен во вкладке **Настройка системы > Аудит \ Анализ размеров аудита**. Содержит в себе размеры таблиц аудита. К ним относятся системные таблицы: телеметрия сервера, блокировки БД, открытие выборок и другие, описанные выше; а также таблицы конкретных бизнес-объектов, по которым ведётся аудит. [Подробнее о структурах аудита БД](#).

Используется для поиска объектов системы, аудит которых занимает большое количество места на хранилище сервера. Позволяет:

1. Определить самые загруженные таблицы для проведения очистки
2. Разделять количество записей аудита по уровням: от бизнес-объекта, до класса и отдельного атрибута
3. Отключить аудит конкретного атрибута

Структура отчёта

- **Главная выборка** - размер таблиц аудита
Список всех таблиц, хранящихся в табличном пространстве аудита (по умолчанию AUD) с указанием размера и кол-ва записей
- **Детализация первого уровня** - классы наследники бизнес-объекта
Количество записей аудита дифференцированное по классам в рамках выбранного бизнес-объекта
- **Детализация второго уровня** - атрибуты класса
Отражает кол-во записанных действий (добавление \ изменение \ удаление) по каждому атрибуту выбранного класса

Примечание

Стоит учитывать, что одна строка в таблице аудита бизнес-объекта может содержать информацию о действиях над **любым количеством** атрибутов! По этой причине сумма кол-ва действий над отдельными атрибутами класса обычно больше чем кол-во записей в общем по этому же классу в детализации первого уровня

Доступные операции

- В группе операций **Дополнительно**:
 - **Синхронизировать историю аудита выделенных классов**
Инкрементальное обновление размера аудита атрибутов по выделенным классам
 - **Синхронизировать историю аудита всех классов**
Инкрементальное обновление размера аудита атрибутов всех классов. Работает аналогично ночной работе актуализации кэша
 - **Пересчитать историю аудита выделенных классов**
Полное обновление кэша. Полезно в том случае, если атрибуты класса были удалены \ изменены

Принципы заполнения

Таблицы аудита объектов представляют собой огромные массивы данных с сотнями тысяч записей, кроме того, операции над атрибутами хранятся в БД в виде JSONB-объектов, чей парсинг добавляет вычислительной сложности в и без того ресурсоёмкую задачу.

Поэтому данные о размерах аудита атрибутов **кэшируются**, при чём данные обновляются **инкрементально** (то есть обрабатываются только новые записи, добавленные после момента последнего кэширования). Это происходит каждую ночь в 5 утра.

Примечание

Если на таблицу аудита объекта установлено задание очистки, пересчёт кэша аудита атрибутов происходит автоматически. Подробнее о заданиях очистки

Предупреждение

При удалении и переименовании атрибутов и чистке таблиц аудита объектов для корректности отображаемых данных **ТРЕБУЕТСЯ** пересчитывать кэш аудита (операция **Пересчитать историю аудита выделенных классов**)

Мониторинг телеметрии в Grafana

Общая информация о внешних средствах мониторинга и телеметрии

Внешний мониторинг организован следующим образом:

1. Сервер приложений, используя стандарт **OpenTelemetry** отправляет метрики, трассировки и логи во внешний сервис (коллектор).
2. Коллектор перенаправляет метрики, трассировки и логи в инструмент визуализации данных **Grafana**
3. В **Grafana** реализованы дашборды визуализации метрик и трассировки.

Конфигурирование сервера приложений

В сервере приложений в каталоге с конфигурациями (`../application/config`) располагаются конфигурационные файлы телеметрии:

- `otel-globalserver.config.yaml`
Основной конфигурационный файл OpenTelemetry SDK. Управляет активностью телеметрии и настройками экспорта.
- `otel-sdk.config.yaml`
Дополнительный конфигурационный файл. Управляет настройками системной телеметрии, специфичной для сервера приложений.

Документация телеметрии сервера приложений

Основные моменты:

1. Для включения телеметрии сервера приложений необходимо настроить файл `otel-globalserver.config.yaml`
 - включить телеметрию общим флагом (`disabled: false`)
 - настроить `endpoint`-ы для коллекторов метрик, логов и трассировки
2. Изменения конфигурационных файлов применяются только после перезагрузки экземпляра сервера приложений.

Grafana как средство визуализации

Grafana — это платформа с открытым исходным кодом для визуализации, мониторинга и анализа данных.

Позволяет отображать данные в графическом виде из разных источников данных.

В нашем мониторинге используются следующие источники данных:

- **Prometheus** - база данных метрик
- **Loki** - база данных логов
- **Tempo** - система хранения трассировок

Основы стандарта OpenTelemetry

OpenTelemetry - стандарт сбора и передачи телеметрических данных. Стандарт описывает набор практик и инструментов, определяющих, какие сигналы может генерировать приложение.

Три основных типа сигналов:

- метрики
- трассировка
- логи

Каждый сигнал может обладать рядом дополнительных свойств, которые могут характеризовать событие, с которым связан сигнал. Например:

- Имя пользователя
- Кластерный узел
- и тд.

Метрики

Метрики - количественные данные о работе системы. Например:

- Время выполнение операции
- Количество успешно выполненных операций
- Количество ошибок при выполнении операций.
- Время выполнения запроса к БД
- и тд.

Метрики делятся на несколько типов:

- **Counter** (увеличивающийся счетчик)
Представляет из себя метрику, которая увеличивается с течением времени. Например, количество запросов к БД.
- **UpDownCounter** (увеличивающийся и уменьшающийся счетчик)
Представляет из себя метрику, которая может как увеличиваться так и уменьшаться с течением времени. Например, количество подключений к БД

- Gauge (измеритель, спидометр)

Представляет из себя метрику, которая используется для измерения чего-либо в конкретный момент времени. Например, время выполнения запроса к БД

- Histogram (гистограмма)

Позволяет накапливать исторические данные на клиентской стороне, и затем отправлять их сигналом во внешний мир.

Трассировка

В стандарте OpenTelemetry реализована через объекты с типом `span`.

`Span` - сигнал, который имеет начало и конец, а так же может иметь родительский `span`. С помощью этих сигналов можно построить дерево трассировки, вкладывая `span`-ы друг в друга, и отмечая начало и конец события.

Пример дерева `span`-ов:

```
+ Выполнение операции
  |_+ Выполнение запроса к серверу приложений
      |_+ Выполнение запроса к БД
          |_+ Обработка результатов запроса к БД
```

Логи

Сигнал, который из себя представляет логовую запись, сделанную системой.

Типы и виды метрик сервера приложений

Сервер приложений посылает различные виды сигналов. Часть из них отправляется сервером приложений, другая же часть прикладным решением.

Системные метрики

Метрики сервера приложений, настраиваемые через Администратор метрик:

- Время реакции на действие пользователя
 - `rpc.server.duration.nanos` - длительность обработки RPC сервером. Число наносекунд.
 - `rpc.client.duration.millis` - длительность выполнения RPC с точки зрения клиента, равная длительность передачи по сети + длительность обработки RPC сервером. Число миллисекунд

Полный перечень метрик сервера приложений

Прикладные метрики и трассировки

Метрики и трассировки, отправляемые из кода прикладного решения.

Метрики пользовательских операций

Специальные метрики, которые отслеживают выполнение пользователем операций в пользовательском интерфейсе. На каждое выполнение операции осуществляется отправка различных метрик:

- `btm_item_telemetry.user_operation_exec_time` - время выполнения операции
- `btm_item_telemetry.user_operation_calls_total` - количество вызовов операции
- `btm_item_telemetry.user_operation_calls_successful` - количество успешных вызовов операции
- `btm_item_telemetry.user_operation_calls_failed` - количество неуспешных вызовов операции

Атрибуты метрик:

- `solution` - Имя решения из конфигурации сервера приложений
- `userName` - Имя пользователя
- `session.sid` - Sid сессии сервера приложений
- `work.session.sid` - Sid рабочего сеанса сервера приложений
- `cluster.node` - имя кластерного узла
- `selection.name` - имя выборки
- `representation.name` - имя отображения
- `selection.caption` - наименование выборки
- `oper.name` - имя операции
- `oper.caption` - наименование операции
- `form.name` - имя главной выборки формы
- `form.caption` - наименование главной выборки формы
- `form.representation.name` - имя отображения главной выборки формы

Трассировки пользовательских операций

- `btm_item_telemetry.user_operation_trace` - трассировка выполнения операции.

Пишется при выполнении пользователем какой-либо операции или сеттера. На каждое пользовательское действие создается `span`, если это действие вызвало другие операции или события, они будут добавлены в дерево трассировки.

- `btm_item_telemetry.open_selection_trace` - трассировка открытия выборки.

Пишется, если на выборку целиком включена трассировка или для пользователя включена вся телеметрия. Представляет из себя дерево `span`-ов, в котором будут отображены все операции, события и сеттеры, выполняемые при открытии выборки.

Если выборка имеет дочерние выборки, то трассировка их открытия, так же будет добавлена в дерево `span`-ов.

Атрибуты трейсов:

- `solution` - Имя решения из конфигурации сервера приложений
- `userName` - Имя пользователя
- `session.sid` - Sid сессии сервера приложений
- `work.session.sid` - Sid рабочего сеанса сервера приложений
- `cluster.node` - имя кластерного узла
- `selection.name` - имя выборки
- `representation.name` - имя отображения
- `selection.caption` - наименование выборки
- `oper.name` - имя операции
- `oper.caption` - наименование операции
- `form.name` - имя главной выборки формы
- `form.caption` - наименование главной выборки формы
- `form.representation.name` - имя отображения главной выборки формы

Метрики и трассировки, объявленные в прикладных модулях

Разработчики в прикладном коде могут объявлять собственные метрики и трассировки для реализации телеметрии в каких-либо высоко-нагруженных местах.

Пример работы с объектной метрикой

1. Объявление метрики (вызов требуется добавить в `dataInstall`)

```
//Объявление метрики с именем Some_Metric_Name, которая относится к Адм. объекту
↪Some_ObjectName
Btk_AcObjectMetricApi().register(
    Btk_AcObjectApi().findByMnemonicCode("Some_ObjectName"),
    "Some_Metric_Name",
    "Пример метрики прикладного кода",
    "Демонстрация использования прикладной метрики "
)
```

2. Использование метрики в коде

```
//получение счетчика
Btk_TelemetryPkg().getAcObjectLongCounter("Some_Metric_Name", "Some_Metric_Name").
↪foreach{counter =>
    //увеличение счетчика, если эта метрика включена в администраторе метрик
    counter.add(42)
}
```

Пример работы с объектной трассировкой

1. Объявление трассировки

```
//Объявление трассировки с именем Some_Trace_Name, которая относится к Адм. объекту
↪Some_ObjectName
Btk_AcObjectTraceApi().register(
    Btk_AcObjectApi().findByMnemonicCode("Some_ObjectName"),
    "Some_Trace_Name",
    "Пример метрики прикладного кода",
    "Демонстрация использования прикладной метрики "
)
```

2. Использование трассировки

```
//получение счетчика
Btk_TelemetryPkg().getAcObjectTrace("Some_Trace_Name", "Some_Metric_Name").foreach
↪{spanBuilder =>
    //выполнение действий с созданием span-а. Перед началом действия создается span,
↪по окончании действия - span закрывается.
    spanBuilder.forSpan{span =>
        try {
            //код выполнения
            Thread.sleep(5000)
        } finally {
            //закрытие span-а
            span.end()
        }
    }
}
```

Ядровые метрики приложения

- `app.transaction.active.count` - Кол-во открытых транзакций БД
- `app.jexl.count` - Кол-во выполненных jexl скриптов
- `app.rest.count` - Кол-во полученных Rest-запросов
- `app.error.count` - Кол-во ошибок (бизнес-логика)
- `app.setting.change.count` - Число измененных системных настроек

Метрики использованных ячеек памяти

- `app.work_session.ui_cell.count` - Кол-во ui-ячеек
- `app.work_session.ui_row.count` - Кол-во ui-строк
- `app.work_session.read_cell.count` - Кол-во загруженных ячеек
- `app.work_session.insert_cell.count` - Кол-во созданных ячеек
- `app.work_session.update_cell.count` - Кол-во измененных ячеек

Атрибуты метрик:

- `solution` - Имя решения из конфигурации сервера приложений
- `user` - Имя пользователя
- `session_id` - Sid сессии сервера приложений
- `session_kind` - Тип сессии сервера приложений

Метрики фоновых заданий

- `app.job.active.count` - Количество активных джобов
- `app.job.run.failed` - Количество ошибок джобов

Включение и фильтрация прикладных метрик осуществляется через **Администратор метрик**

Администратор метрик

Инструмент, который позволяет управлять активностью и фильтрацией прикладных и серверных метрик.

Расположен **Приложение Администратор - Настройки - Телеметрия**.

В интерфейсе слева располагается список с возможными настройкам, справа - отображается выборка этих настроек.

Управление прикладными метриками

Адм. объекты

Эта настройка позволяет включить метрики и трассировку для операций пользовательского интерфейса. Представляет из себя дерево, где корневым узлом является **администрируемый объект**, на втором уровне расположены выборки, входящие в этот объект.

Варианты включения:

- Конкретная операция
- Выборка целиком
- Бизнес объект целиком
- для всех пользователей
- для конкретных пользователей

В детализации расположены закладки:

- **Настройка телеметрии выборки** - управляет включением телеметрии на выборку целиком.
 - **Все пользователи** - при установке признака для всех пользователей, выполняющих операции этой выборки, будет отправлять телеметрию
 - **Включена трассировка** - признак означает, что для всех операций выборки будет формироваться трассировка
 - **Включены метрики** - признак означает, что для всех операций выборки будут формироваться метрики

- **Настройка телеметрии элементарных привилегий** - управляет включением телеметрии на конкретные операции выборки. Отображает список элементарных привилегий выборки, и позволяет включать телеметрию точно. Детализация **Настройка телеметрии объекта** функционально повторяет закладку **Настройка телеметрии выборки**, но ее настройки относятся к конкретным операциям.

Пользователи

Настройка, которая позволяет включить все прикладные метрики и трассировки для конкретного пользователя. Если пользователь добавлен в этот список, и включен признак **Включена вся телеметрия**, то будет опрашиваться телеметрия по всем операциям и объектным метрикам и трассировкам.

Объектные метрики и трассировки

Инструмент, предназначенный для объявления метрик и трассировок прикладного кода. Позволяет разработчикам объявить свои собственные метрики и трассировки, и использовать как телеметрию для особо-нагруженных или проблемных участков кода приложения.

Основной принцип работы:

1. Разработчик регистрирует метрику или трассировку в специализированном справочнике, присвоив им уникальное системное имя.
2. В прикладном коде в необходимых местах отправляет значения метрик или трассировки, используя методы пакета `ru.bitec.app.btk.telemetry.Btk_TelemetryPkg`
3. В администраторе метрик администратор включает или выключает активность объектных метрик и трассировок.

Управление системными метриками

Управляется через закладку **Серверная телеметрия**. Позволяет настроить необходимость отправки метрик сервера приложений для всех пользователей и для конкретных.

Управляет метриками **Время реакции на действие пользователя сервера приложений**.

Grafana

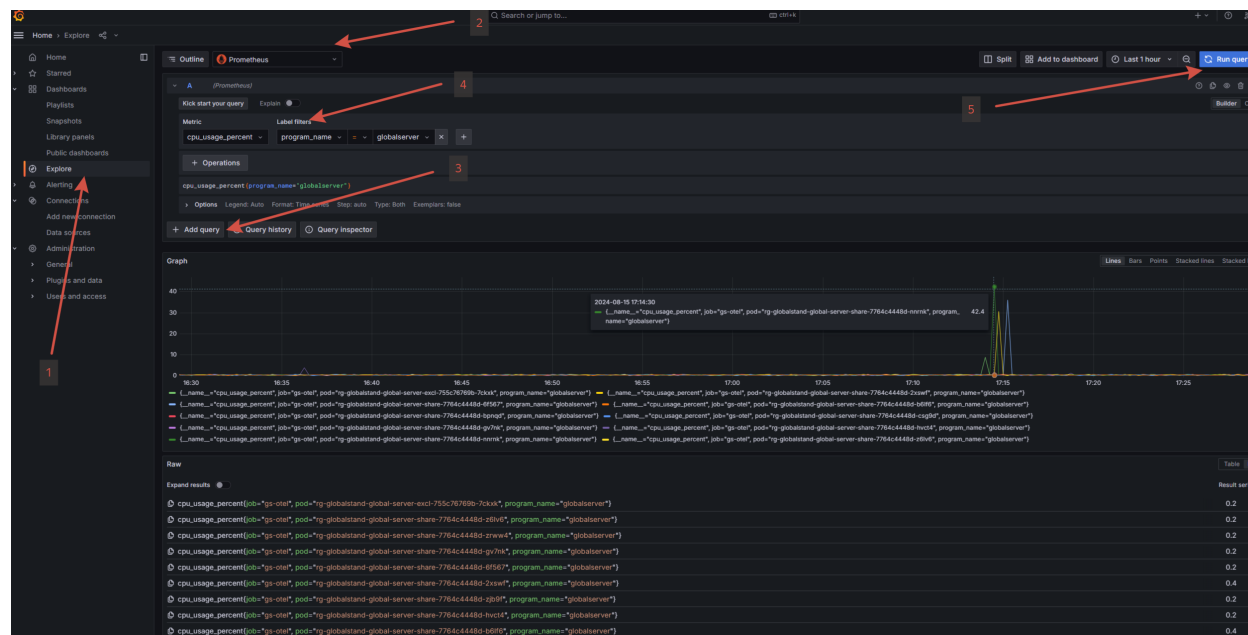
Графана поставляется как отдельный сервис. Если система развернута в виде кластера **Kubernetes**, то по умолчанию для подключения к веб-интерфейсу используется 3000 порт основного адреса. Т.е. если для входа в систему используется адрес `http://192.168.29.17`, то для подключения к веб-интерфейсу графаны используется адрес `http://192.168.29.17:3000/`. Это может быть изменено, и конфигурируется системным администратором.

Для проектов, где система развернута не через кластер **Kubernetes**, адрес подключения определяется системным администратором.

Запрос метрик

Для просмотра логов выполните следующие действия:

1. В меню выберите **Explore**
2. Выберите источник данных **Prometheus**
3. Добавьте новый запрос, если требуется
4. Укажите один из фильтров:
 - **metric** - имя конкретной метрики
 - **label** - дополнительные атрибуты метрики:
 - **program_name** - имя сервиса, поставщика метрик. (globalserver, globalscheduller и тд.)
 - **podname** - имя кластерного узла
 - и тд.
5. Выполните запрос, нажав **Run query**



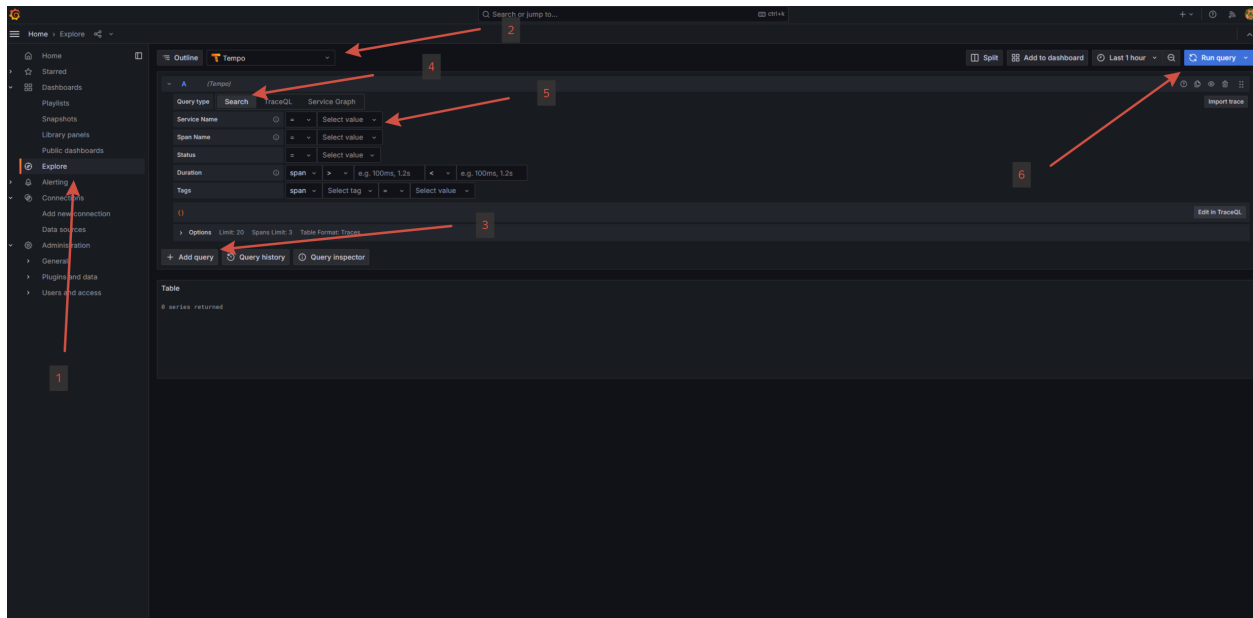
Запрос трассировки

Для просмотра трассировки выполните следующие действия:

1. В меню выберите **Explore**
2. Выберите источник данных **Темпо**
3. Добавьте новый запрос, если требуется
4. Укажите **QueryType = Search**
5. Укажите один из фильтров:
 - **Service Name** - имя сервиса, поставщика метрик. (globalserver, globalscheduller и тд.)

- Span Name - имя трассировки

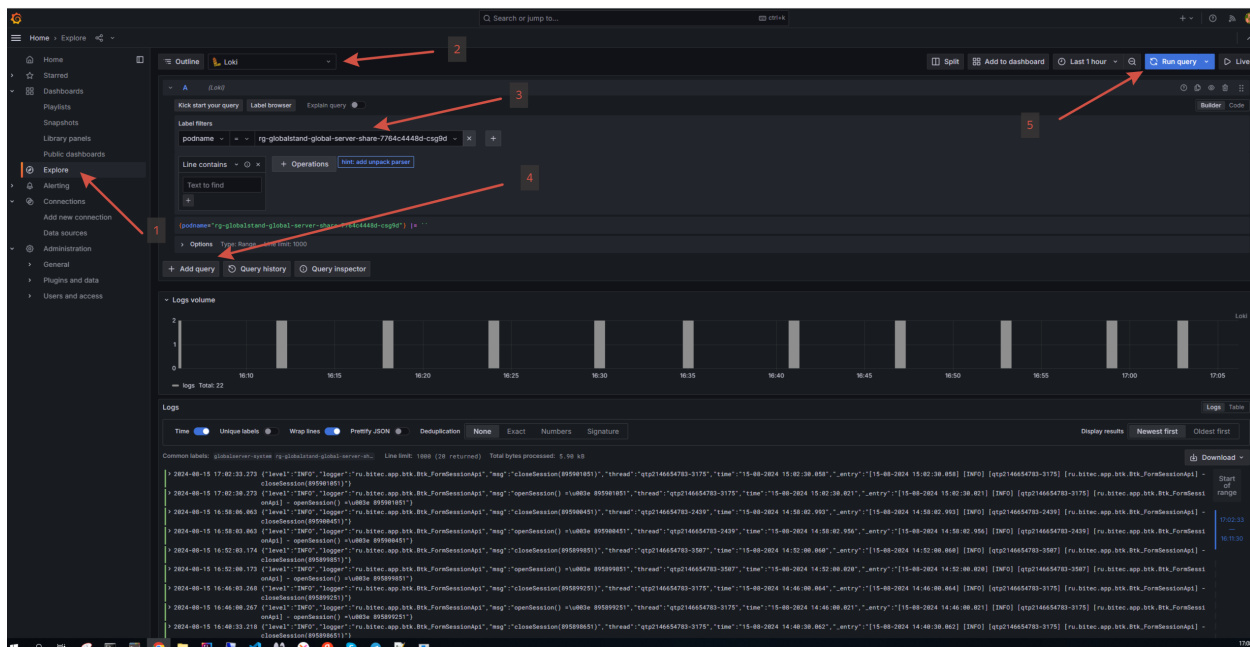
6. Выполните запрос, нажав Run query



Просмотр логов

Для просмотра логов выполните следующие действия:

1. В меню выберите **Explore**
2. Выберите источник данных **Loki**
3. Добавьте новый запрос, если требуется
4. Укажите один из фильтров:
 - filename - имя файла логов
 - log - имя сервиса, чьи логи требуется просмотреть (globalserver, globalscheduller и тд.)
 - podname - имя кластерного узла
5. Выполните запрос, нажав Run query

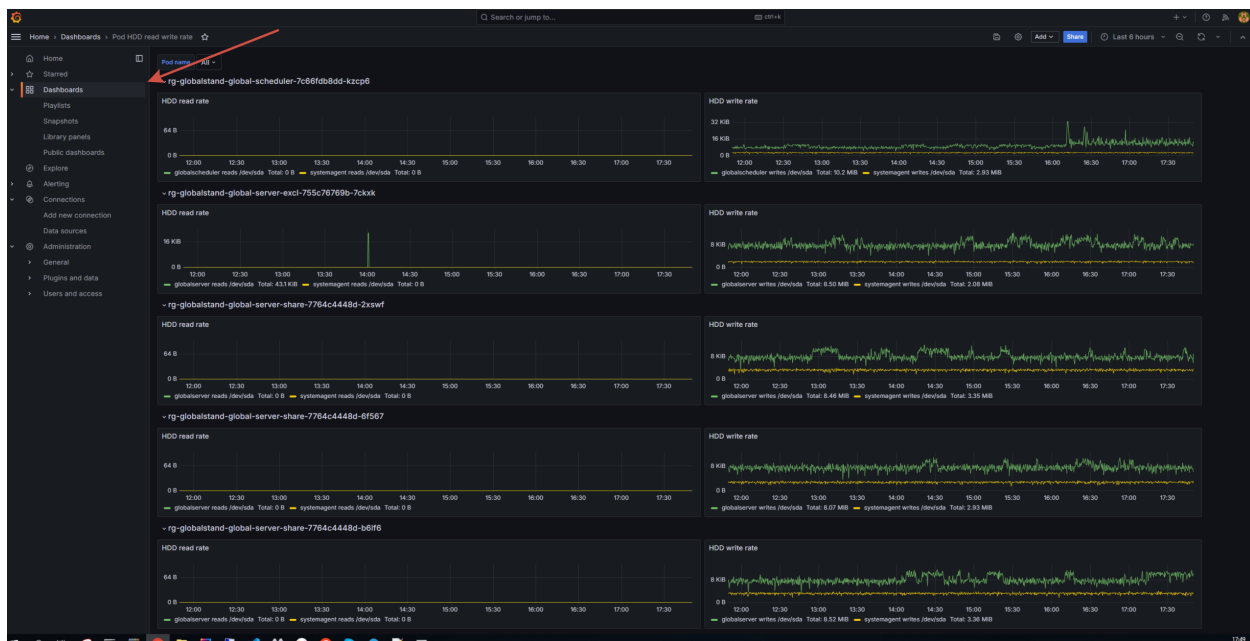


Дашборды

Графана поставляется с предустановленными дашбордами, которые позволяют отобразить ту или иную информацию.

Для просмотра логов выполните следующие действия:

1. В меню выберите Dashboards
2. В появившемся списке выберите нужный дашборд.



Структуры аудита

Система Global предоставляет множество инструментов по контролю за изменением данных, позволяющих отслеживать

- Добавление, изменение и удаление данных
- Блокировки и ошибки при работе БД
- Исполнение периодических работ
- Изменение размера таблиц
- Открытие выборок и выполнение операций
- Исполнение JEXL-скриптов

Все таблицы аудита хранятся в табличном пространстве, указанном в `Btk_AuditPkg().schema`. По умолчанию это `AUD`. В этом разделе документации представлена информация по большинству из них

Аудит классов

Содержит информацию об создании экземпляров класса, изменении атрибутов, выполнении операций. Позволяет отследить взаимодействия с данными вплоть до конкретного пользователя, секунды и рабочего места. Пользовательский интерфейс доступен в **Настройка системы > Аудит \ Аудит открытия форм и выполнения операций**.

Записи аудита хранятся в таблицах отдельных для каждого класса-шапки бизнес-объекта. Названия таблиц формируются следующим образом:

Формирование названий таблиц аудита классов

`${системное имя класса-шапки бизнес-объекта}_dzaud:`

- `Btk_User_dzaud`
- `Bs_Goods_dzaud`

Изменения обрабатываются в `SessionImpl#flush()`. Сбор информации о значении атрибутов происходит до и после применения изменений сессии к базе данных. Запись в аудит - после

[Подробнее об аудите классов](#)

Хранимые поля

- `idrootclass` - id класса-шапки бизнес-объекта
- `gidrootobj` - gid класса-шапки бизнес-объекта
- `srootobjhl` - наименование класса-шапки бизнес-объекта
- `idclass` - id класса
- `gidobj` -
- `sobjhl` -
- `idaction` - id действия пользователя
один из `Btk_AuditAction`: «Создание», «Изменение», «Удаление»

- **joldvalue** - **JSON**-объект
хранящий значения атрибутов объекта до изменений
- **jnewvalue** - **JSON**-объект
хранящий новые значения изменённых атрибутов
- **joldvaluehl** - **JSON**-объект
хранящий наименования ссылочных атрибутов до изменений
- **jnewvaluehl** - **JSON**-объект
хранящий новые наименования изменённых ссылочных атрибутов
- **ddate** - время и дата изменения
- **sremark** - примечание
- **suser** - имя пользователя
- **sclienthostip** - IP-адрес
- **joperinfo** -

Журнал блокировок базы данных `btblockshistory`

Отображает дерево блокировок сессий БД. Подробнее...

Журнал аудита сторонних систем `btidextaudit`

Содержит историю изменения идентификатора внешних систем. Выполняется на каждую установку значения в сеттере `Btk_ExtSystemAvi.ListMaster#setsKey`

Хранимые поля

- **gidObj** - объект, к которому подключена внешняя система
- **sOldValue** - предыдущее значение
- **sNewValue** - изменённое значение
- **idExtSystem** - идентификатор внешней системы
- **idExtSystemHL** - наименование внешней системы
- **dDate** - когда произошло изменение
- **sUser** - кем было произведено изменение
- **sClientHostIP** - IP-адрес
Адрес, с которого была инициирована процедура изменения

Скрипты взаимодействия с таблицей

Регистрация

```
Btk_JIdExtPkg.installTable();
```

Добавление записи

```
insertAudInfo(  
    gidObj: NGid,  
    idExtSystem: NLong,  
    sOldValue: NString,  
    sNewValue: NString  
);
```

Журнал выполнения работ btk_jobexechistory

Содержит историю исполнения регулярно выполняемых работ **Quartz Scheduler**. Позволяет анализировать исполнение работ: длительность, время запуска и его корректность. Обновляется каждые 15 минут работой **Btk_SlowQueryHistoryUpdate**

Хранимые поля

- **idJobEvent** - идентификатор события
- **sScript** - текст исполняемого JEXL-скрипта
- **nPeriod** - время исполнения [минуты]
- **bSlow** - длительная работа
Признак для работ, выполняющихся более 15-ти минут
- **bNight** - ночная работа
Признак для работ, выполняющихся с 22:00 до 08:00

Скрипты взаимодействия с таблицей

Регистрация

```
Btk_MonitorDBActivityPkg.dataInstall();
```

Обновление

```
Btk_MonitorDBActivityPkg.updateSlowQueryHistory();
```

Журнал информации о работе БД btk_monitordbactivityaud

Содержит в себе предупреждения и ошибки работы с базой данных. Обновляется каждые 15 минут с помощью работы Btk_SlowQueryHistoryUpdate. [Подробнее...](#)

Хранимые поля

- dcreate - дата и время события
- stype - тип предупреждения
из Btk_MonitorDBActivityPkg.DBAlertTypes:
«DBBloat», «AutoVacuumOff», «SlowTransaction», «SlowQuery», «FastGrowSize»,
«NotInScheduleJob»
- sfrom - источник
- stypefrom - тип источника
- sshortinfo - сообщение
- sdescription - описание
- squery - запрос

Скрипты взаимодействия с таблицей

Регистрация

```
Btk_MonitorDBActivityPkg.dataInstall();
```

Обновление

```
Btk_MonitorDBActivityPkg.updateSlowQueryHistory();
```

Журнал выполнения медленных задач btk_slowqueryhistory

Журнал SQL-запросов, выполняющихся дольше 15 минут (время, определяющее какой запрос будет считаться долго выполняющимся определяется в Btk_MonitorDBActivityPkg#getIntervalToAcceptAsSlowQuery). Обновляется каждые 15 минут с помощью работы Btk_SlowQueryHistoryUpdate.

Хранимые поля

- username - имя пользователя

- **dcreate** - дата создания записи аудита
- **query** - текст SQL-запроса
- **nduration** - длительность [минут]
- **state** - состояние выполнения \
 - **idle** - сессия БД находится в режиме ожидания. Т.е. подключение к БД есть, но в данный момент никаких запросов не выполняется. Стандартное ожидание для неактивных подключений, которые находятся в пуле подключений сервера приложений
 - **active** - сессия БД в данный момент выполняет запрос.
 - **idle in transaction** - сессия БД находится в режиме ожидания, но есть открытая транзакция. Опасное состояние сессии, т.к. длинные открытые транзакции отрицательно влияют на производительность **postgresql**. Наличие такой сессии сигнализирует о проблеме.
- **sclienthostip** - IP-адрес источника запроса
- **sappname** - название главной выборки
- **sform** - название формы
отображение, выборка, наименование
- **sclusternode** - кластерный узел
- **ssid** - sid сеанса
- **stack** - трассировка стека
- **saction** - действие
Эта колонка будет заполнена, если:
 - сессия принадлежит выполняемому фоновому заданию, в том числе интеграция
 - выполняется построение печатной формы

Скрипты взаимодействия с таблицей

Регистрация

```
Btk_MonitorDBActivityPkg.dataInstall();
```

Обновление

```
Btk_MonitorDBActivityPkg.updateSlowQueryHistory();
```

Журнал размера таблиц основных классов btk_tablesizehistory

История размера таблиц PostgreSQL и системных классов. Позволяет отслеживать изменения размеров таблиц и индексов к ним во времени. Обновляется каждые 20 часов с помощью работы Btk_TableSizeHistoryUpdate

Хранимые поля

- dcreate - дата и время
- sname - название таблицы
- stablespace - табличное пространство
- nsize - собственный размер таблицы [байт]
- nindexsize - размер индексов [байт]к
- ntotalsize - общий размер [байт]

Скрипты взаимодействия с таблицей

Регистрация

```
Btk_MonitorDBActivityPkg.dataInstall();
```

Обновление

```
Btk_MonitorDBActivityPkg.updateTableSizeHistory();
```

Журнал открытия выборок и выполнения операций selopenhistory

Это настраиваемая система, позволяющая вести расширенный аудит действий пользователя при открытии форм и элементов внутри формы. Пользовательский интерфейс доступен в Настройка системы > Аудит \ Аудит открытия форм и выполнения операций. Для каждой конкретной формы можно включить\отключить аудит, настроить какие взаимодействия будут сохраняться:

- Btk_SelSettingAudTypeApi#sFullAud - признак ведения полного аудита
- Btk_SelSettingAudTypeApi#sOnlySuccess - признак ведения аудита успешно выполненных действий

Существует несколько событий, создающих запись аудита:

- при открытии формы
 - EntityAvi#onShow() - успешное открытие формы

- в `EntityAvi#onLoadMeta()` подписка на события неудачных открытий форм
- `EntityAvi#beforeCloseForm()` - закрытие формы
- при вызове операции
 - в `EntityAvi#onLoadMeta()` подписка на события исполнения операций

Подробнее об аудите открытия форм и выполнения операция

Хранимые поля

- `id` - ИД записи в аудите для текущей выборки
- `suser` - имя пользователя
- `ssel` - выборка
- `srep` - отображение
- `scaption` - наименование отображения
- `dstart` - время открытия
- `dend` - время закрытия
- `gidobject` - глобальный идентификатор объекта взаимодействия
- `stype` - тип взаимодействия, `Btk_SelAuditPkg.SelAuditType`
одно из списка: «Главная выборка», «Операция», «Детальная выборка»
- `sparam` - параметры, с которыми открывалась выборка
- `sresult` - результат взаимодействия `SelAuditResult`
одно из списка: «Успешно», «Ошибка прикладного кода», «Ошибка доступа»
- `soper` - операция
- `sopercaption` - наименование операции
- `serrortext` - текст ошибки

Скрипты взаимодействия с таблицей

Регистрация

```
Btk_SelAuditPkg.dataInstall()
```

Добавление записи

```
Btk_SelAuditPkg.audInsert(
  idSession: NLong,
  id: NLong,
  idParent: NLong,
  sSel: NString,
  sRep: NString,
  sCaption: NString,
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
gidObject: NGid,  
sParam: NSString,  
sType: NSString,  
sResult: NSString = SelAuditResult.sSuccess,  
sOper: NSString,  
sOperCaption: NSString,  
sErrorText: NSString  
)
```

Журнал аудита JEXL jexlexechistory

Содержит информацию о запуске и результате выполнения скриптов на языке JEXL. Пользовательский интерфейс доступен в **Настройка системы > Аудит \ Аудит JEXL**.

Операции, которые вызывают созданием записи в таблице аудита:

- Btk_JexlScriptLibraryAvi.Card_Script#execScriptwithSettings()
- Btk_JexlScriptRunnerAvi.Card#execScript()

Хранимые поля

- id - уникальный ключ записи аудита
- suser - имя пользователя
- dstartexec - начало исполнения
- dendexec - конец исполнения
- sjexl - текст скрипта
- nresult - результат выполнения
1 - успех, 0 - ошибка
- sstacktrace - трассировка стека ошибки
- gidsrc - объект, из которого выполнялся вызов скрипта
- saudinfo -

Скрипты взаимодействия с таблицей

Регистрация

```
Btk_JexlScriptRunnerPkg.audDataInstal();
```


Добавление записи

```
Btk_JexlScriptRunnerPkg.audInsert(  
    dpStartExec: NDate  
    ,dEndExec: NDate  
    ,spJexl: NString  
    ,npResult: NNumber  
    ,spStackTrace: NString = None.ns  
    ,gidpSrc: NGid = None.ng  
);
```

Логирование событий ИБ в формате CEF

События информационной безопасности пишутся в логгер `ru.bitec.app.gtk.audit.InfoSecLogger`
Само сообщение отформатировано в формате Common Event Format

прим. CEF:Version|Device Vendor|Device Product|Device Version|Signature
ID|Name|Severity|[Extension]

События | Signature ID

Таблицы ключ=значение для [Extension] части CEF сообщения

1. **Общие** - Ключи есть в каждом сообщении где возможно

Ключ Extension key	Описание
suid	Имя пользователя клиента
suser	ФИО клиента
spid	Session Id. Идентификатор сессии клиента
src	IP-адрес клиента
shost	Имя кластерной ноды
cs6Label	«Superuser»
cs6	Наличие прав администратора - true/false
end	Дата и время в которое было зарегистрировано событие

2. **LoginSuccess** - Успешная авторизация

Ключ Extension key	Описание
suid	Имя пользователя клиента
suser	ФИО клиента
src	IP-адрес клиента
spt	Порт клиента
dst	IP-адрес сервера
dpt	Порт сервера
request	URL запроса
cs6Label	«Superuser»
cs6	Наличие прав администратора - true/false
end	Дата и время в которое было зарегистрировано событие

3. **LoginFailure** - Неуспешная авторизация

Ключ Extension key	Описание
suid	Имя пользователя клиента
reason	Причина неуспешного входа
src	IP-адрес клиента
spt	Порт клиента
dst	IP-адрес сервера
dpt	Порт сервера
request	URL запроса
end	Дата и время в которое было зарегистрировано событие

4. **LoginAttempt** - Превышение количества неудачных попыток ввода пароля

Ключ Extension key	Описание
suid	Имя пользователя клиента
src	IP-адрес клиента
spt	Порт клиента
dst	IP-адрес сервера
dpt	Порт сервера
request	URL запроса
end	Дата и время в которое было зарегистрировано событие

5. **Session** - Начало и остановка пользовательской сессии

Ключ Extension key	Описание
Общие	

6. **Application** - Открытие и закрытие выборки

Ключ Extension key	Описание
Общие	
cs1Label	«Application Name»
cs1	Имя приложения, выборка которого была открыта или закрыта

7. **PasswordChange** - Изменение пароля

Ключ Extension key	Описание
Общие	
duid	Имя пользователя которому сменили пароль
duser	ФИО пользователя которому сменили пароль

8. **OutdatedProdPassword** - Устарел продуктивный пароль (Сброс пароля)

Ключ Extension key	Описание
Общие	
duid	Имя пользователя которому нужно сменить пароль
duser	ФИО пользователя которому нужно сменить пароль

9. **OutdatedTempPassword** - Устарел временный пароль (Блокировка пользователя)

Ключ Extension key	Описание
Общие	
duid	Имя пользователя которому нужно запросить новый пароль
duser	ФИО пользователя которому нужно запросить новый пароль

10. **JexlExec** - Успешно/Неуспешно выполнен JEXL скрипт

Ключ Extension key	Описание
Общие	
cs1Label	«JEXL»
cs1	JEXL скрипт который выполняли
cs2Label	«Source»
cs2	GID источника выполнения
deviceCustomDate1Label	«ExecStart»
deviceCustomDate1	Дата и время начала выполнения
deviceCustomDate2Label	«ExecEnd»
deviceCustomDate2	Дата и время завершения выполнения

11. **AuditSettings** - Изменение настроек аудита

Ключ Extension key	Описание
Общие	
cs1Label	«Table Name»
cs1	Имя таблицы, настройки которой были изменены
cs3Label	«Old values»
cs3	Перечень старых атрибутов. Формат - «атрибут: новое значение»
cs4Label	«New values»
cs4	Перечень измененных атрибутов. Формат - «атрибут: новое значение»

12. **UserPriv** - Выдача и потеря привилегий пользователя

Ключ	Extension key	Описание
Общие		
spriv		Пользователь, который получил или потерял привилегию
cs1		ФИО клиента
dpriv		Системное имя полученной привилегии(Профиль)
cs2		Наименование полученной привилегии(Профиль)

13. ProfilePriv - Выдача и потеря привилегий профиля

Ключ	Extension key	Описание
Общие		
spriv		Профиль, который получил или потерял привилегию
cs1		Наименование полученной привилегии(Профиль)
dpriv		Системное имя полученной привилегии(Роль)
cs2		Наименование полученной привилегии(Роль)

14. UpdateRole - Изменения привилегий роли

Ключ	Extension key	Описание
Общие		
spriv		Роль у которой изменили привилегии
cs1		Наименование Роли
dpriv		Класс в котором произошли изменения см. список ниже
cs2		Наименование класса в котором произошли изменения
cs3Label		«Old values»
cs3		Перечень старых атрибутов. Формат - «атрибут: новое значение»
cs4Label		«New values»
cs4		Перечень измененных атрибутов. Формат - «атрибут: новое значение»

Классы по которым ведутся события ИБ

- Btk_AcRoleItemPrivGrant
- Btk_AcRoleItemPrivTypeGrant
- Btk_AcRoleObjPrivGrant
- Btk_AcRoleObjPrivTypeGrant
- Btk_AcRoleServerPriv
- Btk_AcRoleStateChangeGrant

Настройка

В директории Сервер\application\config добавить в logback-LoggerContext.xml и logback-LoggerContext-session.xml.

```
<appender name="SYSLOG" class="ch.qos.logback.classic.net.SyslogAppender">
  <!-- syslogHost, port - ссылка на Syslog сервер -->
  <syslogHost>localhost</syslogHost>
  <port>514</port>
  <facility>AUTHPRIV</facility>
  <!-- Паттерн Syslog сообщения. (Обязательные Syslog поля) + %msg(CEF Сообщение)-->
  <suffixPattern>%msg</suffixPattern>
  <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
    <evaluator class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
      <!--Здесь можно указать фильтрацию для событий
      "*" - все события
      прим.
      <marker>Session</marker>
      <marker>PasswordChange</marker>
      -->
      <marker>*</marker>
    </evaluator>
    <!--Правила фильтрации
    Логировать только указанные события:
    <OnMatch>NEUTRAL</OnMatch>
    <OnMismatch>DENY</OnMismatch>
    Логировать все кроме указанных событий:
    <OnMatch>DENY</OnMatch>
    <OnMismatch>NEUTRAL</OnMismatch>
    -->
    <OnMatch>NEUTRAL</OnMatch>
    <OnMismatch>DENY</OnMismatch>
  </filter>
</appender>
```

Добавите сам логгер

```
<!--Добавьте над root'ом если не хоте чтобы логгер не унаследовал от root'a level и
→ другие аппендеры-->
<logger name="InfoSecLogger" level="ALL" additivity="false">
  <appender-ref ref="SYSLOG"/>
</logger>
```

В результате получим SYSLOG CEF сообщение вида

```
Jan 31 16:09:10 DESKTOP-IPCDPC4 CEF:1|bitec|GlobalERP|0|LoginSuccess|Успешная
авторизация|Medium|suid=admin suser=Кузнецов В.М. src=127.0.0.1 spt=56453 dst=localhost
dpt=8080 request=http://localhost:8080/login/login.html cs6Label=Superuser cs6=true
end=Jan 31 2025 16:09:10
```

Логирование DDL-операций `btk_ddljournal`

Журнал операций изменения структур базы данных. В журнале содержатся записи о таких операциях как `CREATE`, `ALTER`, `DROP`, `SECURITY LABEL`, `COMMENT`, `GRANT` и `REVOKE`.

Для включения логирования операций требуется выполнить создание триггера. У пользователя, под которым подключается сервер при генерации таблиц, как правило прав на создание триггеров нет, поэтому включение нужно произвести вручную.

```
CREATE EVENT TRIGGER ddl_audit_trigger
ON ddl_command_end
EXECUTE FUNCTION log_ddl_operations()
```

Хранимые поля

- `dtimestamp` - Время окончания выполнения операции.
- `sdbusername` - имя пользователя в текущем контексте выполнения.
- `susername` - имя пользователя сеанса.
- `smachineame` - наименование хоста клиента.
- `sipaddress` - адрес клиента.
- `sschema` - имя схемы.
- `sentitity` - измененный объект.
- `sentititytype` - тип измененного объекта.
- `sddl` - операция изменения структуры данных.

Наименование хоста клиента не будет логироваться, если в базе данных параметр `log_hostname` включен. Включить его можно в конфиге БД, либо в консоли.

Сервисы сервера приложений

Введение

Сервисы реализованы в опциональном модуле `sys`, который входит в дистрибутив Global Server Scala Edition. Модуль также может быть подключён в конфигурационном файле сервера `global3.config.xml`:

```
<sbt name="sys"
  lazyLoad="false"
  source="C:\global3\sysappbin\"
  sourceMode="Jar"
  binaryFolder="C:\global3\sysappbin\"
/>
```

Ленивая загрузка должна быть отключена, чтобы сервисы могли всегда принимать запросы.

Более подробную информацию смотрите в руководстве по администрированию сервера.

SSH-консоль сервера

Global 3 SE Server включает в себя SSH-сервер (Secure Shell), к которому можно подключиться с помощью любого SSH-терминала. С помощью командной строки можно:

- Просматривать статистику, логи и управлять клиентскими сессиями.
- Перезагружать код прикладных приложений.
- Перезапускать веб-приложения.
- Выполнять SQL- и Jexl-скрипты.

Конфигурирование

Конфигурационный файл `global3.config.xml` может содержать секцию `<ssh/>`:

```
<ssh defaultDb="PostgreSql" port="22"/>
```

- **defaultDb** — алиас базы данных по умолчанию.
Используется как значение параметра по умолчанию для команды `login`. По имени базы определяются имя и пароль пользователя для подключения к базе.
- **port** — порт SSH-сервера.
По умолчанию: 22.
Значение может быть задано через JVM-опцию:

```
-Dglobal.ssh.port=xxxx
```

или через параметр командной строки при запуске сервера Global 3:

```
-global.ssh.port xxxx
```

Учётные данные пользователей, имеющих права на SSH-подключения, указываются в секции `<security>` конфигурационного файла `global3.config.xml`:

```
<security>
  <users>
    <user name="admin" password="admin" roles="ssh"/>
  </users>
</security>
```

- **user** — имя пользователя.
- **password | encryptedPassword** — пароль. Может быть указан в открытом или зашифрованном виде.
- **roles** — роли. Список ролей, доступных пользователю, через запятую.

Шифрование пароля

Для шифрования пароля необходимо запустить Global 3 Server с параметрами:

```
Start.bat -encryptPassword password -masterPassword masterpassword
```

Где:

- **-encryptPassword** — пароль, который необходимо зашифровать.
- **-masterPassword** — ключ шифрования. Если не указан, используется секретный ключ по умолчанию.

Результатом выполнения будет строка в консоли — зашифрованный пароль, переданный параметром **-encryptPassword**.

Подключение

Подключиться к SSH-серверу можно с помощью любого SSH-терминала. Рекомендуемый клиент — [PuTTY](#).

[Документация по PuTTY](#)

Основные параметры

Для подключения укажите:

- **Host Name** — `host` или `user@host`.
- **Port** — 22 или пользовательский порт.
- **Connection type** — SSH.

Чтобы не вводить параметры при каждом запуске, сохраните подключение в PuTTY, нажав кнопку **Save**.

Пример запуска из командной строки:

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -pw admin
```

Логирование

Для сохранения всего текстового вывода в файл на вкладке [Logging](#) укажите:

- **Session Logging** = All session output.
- **Log file name** = полный путь к файлу.

Внимание

В адресах пути:

- Символ `\` необходимо экранировать удвоением: `\\`.
- Символ `/` экранировать не нужно.

Подключение с использованием SSH-RSA ключа

При подключении из *.bat или *.sh можно использовать RSA-ключи вместо пароля:

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -i ssh_admin_private.ppk
```

Необходимые файлы размещены в подкаталоге .\ssh.

Ключи для подключения к пользователю admin выдаются по запросу.

Запуск терминала

При первом подключении к серверу появится сообщение с запросом разрешения на подключение. Нажмите: Да.

Команды

Для получения справки по доступным командам выполните:

```
help
```

Список команд:

- **alter server mode service|normal** — переключает сервер в сервисный режим и обратно. В сервисном режиме подключение пользовательских сессий возможно только от имени системного пользователя сервера приложений, указанного в конфигурационном файле.

```
<systemUsers>
  <user name="system" password="system"/>
</systemUsers>
```

- **attach session {sid}** — подключение к существующей пользовательской сессии.
- **attach db {dbAlias} [as sys]** — подключение к базе данных для выполнения сервисных операций и/или Jexl-скриптов.
- **clear** — очищает экран терминала.
- **clear persistence cache [{dbAlias}]** — очищает Shared-кэш объектов.
- **compare applib {path}** — сравнивает JAR-файлы в каталоге (или ZIP-архиве) {path} с JAR-файлами в каталоге SBT.jarFolder, соответствующем базе данных, к которой выполнено подключение командой attach db. Выполняется сравнение версий модулей и файлов *.odm.xml на наличие новых атрибутов.
- **copy applib [force] {path}** — копирует JAR-файлы из каталога {path} в каталог SBT.jarFolder, соответствующий подключённой базе данных. При указании ключевого слова force сравнение файлов *.odm.xml не выполняется. Сравнение версий модулей выполняется всегда.
- **Init schema** — выполняет создание или обновление объектов схемы БД.
- **execute {expression}** — выполняет однострочное Jexl-выражение.
- **exit** — закрывает SSH-подключение.
- **jexl [{file}]** — переключает терминал в режим ввода и выполнения скриптов.

Выполнение SQL

Для выполнения SQL-скрипта:

1. Подключитесь к сессии командами `login`, `attach` или `set sid`.
2. Перейдите в режим ввода скрипта командой `sql`.
3. Введите текст скрипта.
4. Выполните скрипт, введя символ `/` с новой строки.
5. Выйдите из режима ввода, введя символ `/` с новой строки.

Пример:

```
login admin/admin@postgres
sql
INSERT INTO gs3_roottest (
    id,
    idClass
) VALUES (
    (select
        nextval('GS3_ROOTTEST_SEQ'))
    , 12351
) ON CONFLICT DO NOTHING;
/
/
```

Выполнение SQL-скрипта из файла

Для выполнения SQL-скрипта из файла:

1. Подключитесь к сессии.
2. Выполните команду `sql {file}`, где `{file}` — путь к файлу на сервере.

Внимание

Файл должен находиться на локальном диске сервера.

Пример:

```
login admin/admin@postgres
sql
↪D:\svn\depot\ASSource\sysapplication\ssh\src\test\java\ru\bitec\app\ssh\ssh_
↪sql_exams.txt
```

Содержимое файла:

```
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),
↪12351) ON CONFLICT DO NOTHING;
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),
↪12351) ON CONFLICT DO NOTHING;
/
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),  
↪12351) ON CONFLICT DO NOTHING;  
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),  
↪12351) ON CONFLICT DO NOTHING;  
/  
/
```

Jexl-скрипты

Выполнение Jexl-скрипта

1. Подключитесь к сессии.
2. Переключитесь в режим ввода скрипта командой `jexl`.
3. Введите текст скрипта.
4. Выполните скрипт, введя символ `/` с новой строки.
5. Выйдите из режима ввода, введя символ `/` с новой строки.

Пример 1:

```
login admin/admin@postgres  
jexl  
var name = Btk_ClassApi.getCanonicalClassName("Btk_Object");  
Btk_ClassApi.getApiByCanonicalClassName(name);  
/  
/
```

Пример 2 (для `dataInstall`):

```
login admin/admin@postgres  
jexl  
Bbb_DBTypeApi.dataInstall();  
Btk_Pkg.commit();  
/  
/
```

Пример 3:

```
login admin/admin@postgres  
jexl  
Btk_Pkg.setRWSharedUOWEditType();  
Prs_EntTransApi.dataInstall();  
Btk_Pkg.commit();  
/  
/
```

Выполнение Jexl-скрипта из файла

1. Подключитесь к сессии.
2. Выполните команду `jexl {file}`, где `{file}` — путь к файлу **на сервере**.

Внимание

Файл должен находиться на локальном диске сервера.

Пример:

```
login
jexl
↪ D:\svn\depot\ASSource\sysapplication\ssh\src\test\java\ru\bitec\app\ssh\ssh_
↪ jexl_exams.txt
```

Содержимое файла:

```
var name = Btk_ClassApi.getCanonicalClassName("Btk_Object");
Btk_ClassApi.getApiByCanonicalClassName(name);
/
/
```

Контекст выполнения Jexl-скрипта

Возможны следующие контексты:

Пользовательская сессия

```
>login user/password@alias
>jexl
jexl>
/
/
>
```

Доступны все API-классы из SBT, соответствующего пользовательской сессии.

База данных

```
>attach db alias
>jexl
jexldb>
/
/
>
```

Доступны методы управления инстансом БД:

- `initschema()` — инициализация/обновление схемы БД.

Системный контекст

```
>attach db alias as sys
>jexl
jexlsys>
/
/
>
```

Административный контекст для управления сервером приложений. Доступен метод:

- `upgrade({release_path})` — инициализация/обновление схемы БД по пути к каталогу или ZIP-архиву с релизом.

Выполнение командного файла SSH

Для выполнения командного файла из командной строки:

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -pw admin -m ssh_logger_
↪test_script.txt
```

Внимание

В данном примере файл `ssh_logger_test_script.txt` размещён на диске клиентской машины, он считывается в момент подключения к SSH-серверу.

Пути к файлам в скрипте — локальные для сервера.

Содержимое файла:

```
login
log-info app el

execute Btk_ClassApi.getCanonicalClassName("Btk_Object")

jexl
var name = Btk_ClassApi.getCanonicalClassName("Btk_Object");
Btk_ClassApi.getApiByCanonicalClassName(name);
/
/

jexl D:/svn/depot/ASSource/sysapplication/ssh/src/test/java/ru/bitec/app/ssh/ssh_jexl_
↪exams.txt
jexl D:\\svn\\depot\\ASSource\\sysapplication\\ssh\\src\\test\\java\\ru\\bitec\\app\\ssh\\
↪ssh_jexl_exams.txt

sql
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),
↪12351) ON CONFLICT DO NOTHING;
/
/

sql D:\\svn\\depot\\ASSource\\sysapplication\\ssh\\src\\test\\java\\ru\\bitec\\app\\ssh\\
↪ssh_sql_exams.txt
```

(продолжается на следующей странице)

```
log-off all  
logout
```

Логирование

При выполнении скрипта, по умолчанию, в ssh-консоль выводится результат выполнения команды или сообщение об ошибке со стеком вызова.

Для вывода в ssh-консоль дополнительных логов, необходимо их включить.

В консоль можно вывести логи следующих типов:

- `oper` — логи из `ru.bitec.engine.model.operation`.
- `sql` — логи SQL-вызовов на уровне JDBC.
- `script` — логи скриптового языка Global 1 (неактуальны для Scala).
- `app` — логи из `ru.bitec.app.*`.

Пример:

```
Logger.Factory.get(Xxx_XxxApi.class).info("Текст сообщения")
```

- `el` — логи EclipseLink (более компактные SQL-вызовы).
- `all` — все выше перечисленные типы логов.

Для переключения уровней логирования используются команды (в порядке уменьшения детализации):

- `log-off;`
- `log-error;`
- `log-warn;`
- `log-info;`
- `log-debug;`
- `log-trace.`

Пример:

```
log-info app el
```

В ssh-лог будут попадать сообщения типов `app` и `el` для уровней: `error`, `warn`, `info`.

Для отключения вывода сообщений в ssh-лог:

```
log-off all
```

Перенаправление ssh-лога PuTTY в файл

При выполнении скриптов из командной строки вывод можно перенаправить в файл:

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -pw admin -m ssh_logger_  
↪test_script.txt -sessionlog sessionlog.txt
```

FAQ

Зависает ssh-терминал на этапе подключения к серверу

При запуске PuTTY из скрипта при первом подключении появляется окно **PuTTY Security Alert**, что может вызывать зависание.

Решение — автоматически подтвердить подключение:

```
echo yes | %plink% -ssh localhost -P 2222 -l admin -i ssh_admin_private.ppk
```

Примечание

Plink входит в состав дистрибутива PuTTY.

WebSocket консоль сервера

Global 3 SE Server предоставляет консоль управления, доступную через WebSocket-соединение. С помощью команд, передаваемых через WebSocket, возможно:

- Перегружать код прикладных приложений.
- Выполнять Jexl-скрипты.

Алгоритм работы с консолью

- Открытие WebSocket-соединения.
- Отправка команды аутентификации пользователя в системе.
- Отправка исполняемых команд.
- Закрытие WebSocket-соединения.

Открытие WebSocket-соединения

Для открытия WebSocket-соединения с консолью сервера необходимо выполнить HTTP-запрос по адресу:

```
ws[s]://{server[:port]}/app/sys/ws/console.
```

Формат команды

Командой является строка в формате:

```
{command}[\n{arguments}]
```

где:

- {command} — строка команды. Может состоять из одного или нескольких слов.
- \n — символ новой строки #10. Является разделителем команды и её аргументов. Не обязателен, если у команды нет аргументов.

- `{arguments}` — строка аргументов команды. Может содержать любые символы, включая переносы строк.

Формат результата выполнения команды

Результатом выполнения команды является строка в формате JSON:

```
{
  "success": true,
  "data": null,
  "exception": null,
  "exceptionStack": null
}
```

где:

- `success` — флаг успешности выполнения команды: `true` | `false`.
- `data` — результат выполнения команды: `null` | `"string"` | JSON.
- `exception` — сообщение возникшего исключения: `null` | `"string"`.
- `exceptionStack` — стек возникшего исключения: `null` | `"string"`.

Список команд

- `login` — выполняет аутентификацию пользователя `{user}` в базе данных `{database}` и запускает рабочий сеанс пользователя.
Использование команды `login`, в качестве способа аутентификации, было выбрано по причине невозможности использования HTTP-заголовков некоторыми WebSocket-клиентами. Например: [JMeter WebSocket Load Testing Sampler](#).

```
{user}/{password}@{database}
```

- `logout` — закрывает рабочий сеанс пользователя.
- `reload sbt` — выполняет перезагрузку прикладного кода текущего решения.
- `reload sbt force` — выполняет перезагрузку инфраструктуры EclipseLink, прикладного кода и общих библиотек текущего решения.
- `jexl` — любое значение, возвращённое из Jexl-скрипта, будет преобразовано в строку.

```
// Произвольный текст Jexl-скрипта
return 1 + 2;
```


Java пример взаимодействия с консолью

Библиотеки, используемые в примере клиента:

- «org.asynchttpclient» % «async-http-client» % «2.12.3».
- «com.fasterxml.jackson.core» % «jackson-databind» % «2.8.9».

```
package ru.bitec.app.examples.ws.console;

import org.asynchttpclient.AsyncHttpClient;
import org.asynchttpclient.Dsl;
import org.asynchttpclient.ws.WebSocket;
import org.asynchttpclient.ws.WebSocketListener;
import org.asynchttpclient.ws.WebSocketUpgradeHandler;

import com.fasterxml.jackson.databind.ObjectMapper;

import javax.websocket.CloseReason;
import java.io.Serializable;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;

public class WsConsoleClientTest {

    public static void main(String[] args) throws Exception {
        try (WsConsoleClient client = WsConsoleClient.open("ws://localhost:8080/app/sys/ws/
↵console")) {
            client.login("admin", "admin", "pgtest");
            System.out.println("1 + 2 = " + client.evaluateJexl("return 1 + 2;"));
            client.reloadSbt(false);
            client.logout();
        }
    }

    public class WsConsoleClient implements AutoCloseable {

        public static WsConsoleClient open(String url) throws Exception {
            return open(url, 3000);
        }

        public static WsConsoleClient open(String url, int timeout) throws Exception {
            return new WsConsoleClient().connect(url, timeout);
        }

        private final ObjectMapper objectMapper_ = new ObjectMapper();
        private int commandTimeout_ = 60000;
        private AsyncHttpClient asyncHttpClient_;
        private WebSocket websocketClient_;
        private volatile CompletableFuture<String> websocketResultFuture_;
        private final WebSocketUpgradeHandler wsHandler = new WebSocketUpgradeHandler.
↵Builder().addWebSocketListener(new WebSocketListener() {
            @Override
            public void onOpen(WebSocket websocket) {
```

(продолжается на следующей странице)

```

        // WebSocket connection opened
    }

    @Override
    public void onClose(WebSocket websocket, int code, String reason) {
        if (code != CloseReason.CloseCodes.NORMAL_CLOSURE.getCode()) {
            websocketResultFuture_.completeExceptionally(new Exception(reason));
        }
    }

    @Override
    public void onError(Throwable t) {
        websocketResultFuture_.completeExceptionally(t);
    }

    @Override
    public void onTextFrame(String payload, boolean finalFragment, int rsv) {
        websocketResultFuture_.complete(payload);
    }
}).build();

public WsConsoleClient connect(String url, int timeout) throws Exception {
    AsyncHttpClient syncHttpClient = Dsl.asyncHttpClient();
    try {
        websocketClient_ = syncHttpClient
            .prepareGet(url)
            .setRequestTimeout(timeout)
            .execute(wsHandler)
            .get();
    } catch (Exception e) {
        syncHttpClient.close();
        throw e;
    }
    asyncHttpClient_ = syncHttpClient;
    return this;
}

public void close() throws Exception {
    if (asyncHttpClient_ != null) {
        try {
            if (websocketClient_ != null && websocketClient_.isOpen()) {
                try {
                    websocketClient_.sendCloseFrame().get();
                } finally {
                    websocketClient_ = null;
                }
            }
        } finally {
            asyncHttpClient_.close();
        }
    }
}
}

```

```

public int getCommandTimeout() {
    return commandTimeout_;
}

public void setCommandTimeout(int commandTimeout) {
    this.commandTimeout_ = commandTimeout;
}

public void login(String user, String password, String database) throws Exception {
    sendCommand("login", String.format("%s/%s@s%s", user, password, database));
}

public void logout() throws Exception {
    sendCommand("logout");
}

public String evaluateJexl(String script) throws Exception {
    return sendCommand("jexl", script);
}

public void reloadSbt(boolean force) throws Exception {
    sendCommand("reload sbt" + (force ? " force" : ""));
}

String sendCommand(String command) throws Exception {
    return sendCommand(command, null);
}

String sendCommand(String command, String args) throws Exception {
    validateConnection();
    if (command == null || command.trim().isEmpty()) {
        throw new Exception("Command can not be null or empty.");
    }
    websocketResultFuture_ = new CompletableFuture<>();
    try {
        String payload = command + "\n" + (args != null ? args : "");
        websocketClient_.sendTextFrame(payload).get();
    } catch (Exception e) {
        websocketResultFuture_.completeExceptionally(e);
    }
    return parseResponse(websocketResultFuture_.get(commandTimeout_, TimeUnit.
↪MILLISECONDS));
}

private String parseResponse(String response) throws Exception {
    ConsoleResponse consoleResponse = objectMapper_.readValue(response, ↪
↪ConsoleResponse.class);
    if (consoleResponse.success) {
        return consoleResponse.data;
    } else {
        throw new Exception(consoleResponse.exception);
    }
}

```

```

    }
}

private void validateConnection() throws Exception {
    if (websocketClient_ == null) {
        throw new Exception("WebSocket connection is not connected.");
    }
    if (!websocketClient_.isOpen()) {
        throw new Exception("WebSocket connection is closed");
    }
    CompletableFuture<String> websocketResultFuture = websocketResultFuture_;
    if (websocketResultFuture != null && !websocketResultFuture.isDone()) {
        throw new Exception("Prior console command is not completed. Wait for the
↪result of the previous command.");
    }
}

private final static class ConsoleResponse implements Serializable {

    private static final long serialVersionUID = -5577579081118070434L;
    /**
     * Флаг успешного выполнения запроса. false, если при обработке запроса возникло
↪исключение.
     */
    private boolean success = false;
    /**
     * Строковые данные
     */
    private String data;
    /**
     * Текст возникшего исключения
     */
    private String exception;
    /**
     * Стек возникшего исключения
     */
    private String exceptionStack;

    public boolean getSuccess() {
        return success;
    }

    public void setSuccess(boolean success) {
        this.success = success;
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}

```

(продолжение с предыдущей страницы)

```
    }

    public String getException() {
        return exception;
    }

    public void setException(String exception) {
        this.exception = exception;
    }

    public String getExceptionStack() {
        return exceptionStack;
    }

    public void setExceptionStack(String exceptionStack) {
        this.exceptionStack = exceptionStack;
    }
}
}
```

Jexl через SOAP и REST

Jexl через SOAP XML

В Global 3 SE Server реализован SOAP XML-сервис, позволяющий выполнять Jexl-скрипты в контексте пользовательской сессии.

Аутентификация

Читайте в разделе Аутентификация в REST/SOAP-сервисах.

Схемы

WSDL-описание сервиса доступно по адресу:

```
http://{server:port}/app/sys/soap/sys-service-1.0.0?wsdl
```

XSD-схемы POJO запроса и ответа доступны по адресу:

```
http://{server:port}/app/sys/soap/sys-service-1.0.0?xsd=1
```

Передача бинарных данных с использованием MTOM

Сервис поддерживает оптимизированную передачу бинарных данных в сервис и обратно через поля `attachment` объектов `JexlSoapRequest` и `JexlSoapResponse`.

Доступ к данным SOAP-запроса из прикладного кода

В прикладном коде `Api-/Pkg`-классов, выполняющемся в результате вызова Jexl-скрипта из SOAP-сервиса, доступен объект `SoapContext`. `SoapContext` предоставляет доступ к данным SOAP-запроса и SOAP-ответа.

Для получения контекста выполните:

```
val soapContextOpt = SoapContext()
```

Методы, доступные в `SoapContext`:

- `hasInputAttachment: Boolean` — указывает на наличие у SOAP-запроса прикрепленных с использованием MTOM бинарных данных.
- `getInputData: String` — возвращает строковые данные SOAP-запроса.
- `forInputStream(foo: InputStream => Unit): Unit` — предоставляет доступ к прикрепленным бинарным данным.
- `setOutputData(data: String): Unit` — устанавливает строковые данные в SOAP-ответ.
- `forOutputStream(foo: OutputStream => Unit): Unit` — предоставляет доступ к выходному потоку бинарных данных, передаваемых средствами MTOM.

Методы обработки данных SOAP-запроса из Jexl-скрипта

В пакете `Gtk_SoapPkg` хранятся методы, использующие `SoapContext`:

- `setOutputData(value: AnyRef): Unit` — устанавливает вывод результата выполнения SOAP-запроса.
- `inputStreamToTemp: File` — создаёт временный файл с данными из прикрепленных данных SOAP-запроса.
- `attachFile(file: File): Unit` — принимает файл, закидывает его данные в выходной поток бинарных данных SOAP-запроса.

Пример использования из Jexl-скрипта:

```
var f = new("java.io.File", 'C:\users\userName\data.txt');
Gtk_SoapPkg.setOutputData('Данные в прикрепленном файле');
Gtk_SoapPkg.attachFile(f);
```

Данный скрипт, отправленный через SOAP-запрос, в ответ запишет:

- в `data` — строку 'Данные в прикрепленном файле';
- в `attachment` — закодированный base64 файл.

Јех1 через REST API сервис с использованием прикладных пакетов

В Global 3 SE Server реализован REST API-сервис, позволяющий выполнять Јех1-скрипты в контексте пользовательской сессии. В данной главе описывается выполнение Јех1-скриптов с использованием прикладных пакетов, наследуемых от **RestPkg**.

Выполнение Јех1 в контексте главной выборки приложения описано в *отдельной главе*.

Аутентификация

Читайте в разделе Аутентификация в REST/SOAP-сервисах.

Адреса

Доступно с AS 1.14 RC7:

- `http://{server:port}/app/sys/rest/es/pkg/Btk_JexlGatePkg/execute`
- `http://{server:port}/app/sys/rest/ss/pkg/Btk_JexlGatePkg/execute`

где:

- `{server:port}` — адрес подключения к серверу;
- `app/sys` — системный прикладной модуль;
- `rest` — шлюз для REST-запросов;
- `es / ss` — группировка по времени жизни GTK-сессии (*Exclusive Session*) / *Shared Session*;
- `pkg` — узел для доступа к пакетам;
- `Btk_JexlGatePkg` — имя прикладного пакета, предназначенного для выполнения Јех1-скриптов (*подробнее о REST-пакетах*).

Администрирование выполнения Јех1-скриптов

Администрирование выполнения Јех1-скриптов осуществляется через право роли на «Функции Јех1». Также можно отдельно *ограничить доступ роли к Btk_JexlGatePkg*.

Структура запроса

Тело POST-запроса имеет вид:

```
{
  "jexl": "jexl-скрипт"
}
```

Формат результата выполнения команды

Результатом выполнения команды является строка в формате JSON:

```
{
  "exception": null,
  "exceptionStack": null,
  "data": null,
  "success": true,
  "exceptionName": null
}
```

где:

- `exception` — сообщение возникшего исключения: `null` | `"string"`;
- `exceptionStack` — стек возникшего исключения: `null` | `"string"`;
- `success` — флаг успешности выполнения команды: `true` | `false`;
- `data` — результат выполнения команды: `null` | `"string"` | `JSON` | `Boolean`;
- `exceptionName` — имя класса исключения: `null` | `"string"`.

Пример использования

Для выполнения Jexl-скрипта будем использовать пакет `Btk_JexlGatePkg` и `Shared Session`. Тогда адрес запроса будет выглядеть так:

```
http://localhost:8080/app/sys/rest/ss/pkg/Btk_JexlGatePkg/execute
```

В качестве примера Jexl-скрипт будет выполнять проверку, является ли пользователь супер-пользователем:

```
{
  "jexl": "Btk_UserApi.isSuperUser()"
}
```

После выполнения POST-запроса сервер вернёт ответ:

```
{
  "exception": null,
  "exceptionStack": null,
  "data": false,
  "success": true,
  "exceptionName": null
}
```

Запрос выполнялся без ошибок, и Jexl-скрипт вернул значение `false`.

Ограничения

Время выполнения Jexl-скрипта ограничено *временем жизни сессии*.
Размер Jexl-скрипта не ограничен.

REST-сервис

REST-сервис с обработкой HTTP-запроса в прикладном пакете

В Global 3 SE Server реализован REST-сервис, позволяющий выполнять обработку HTTP-запроса в прикладном пакете в контексте прикладной сессии.

Возможны два режима обработки запросов:

- **Exclusive Session** — с сохранением состояния между запросами. Rest- и Gtk-сессии создаются при первом обращении клиента к сервису и закрываются по таймауту или при явном указании на необходимость закрытия сессий по завершению запроса.
- **Shared Session** — без сохранения состояния между запросами. Каждый запрос обрабатывается в новых Rest- и Gtk-сессиях. Доступно с AS 1.14 RC7.

Пакет должен быть унаследован от одного из трейтов RestPkg, RestESPkg, RestSSPkg.

```
class Xxx_XxxPkg extends Pkg with RestPkg {  
}  
object Xxx_XxxPkg extends PkgFactory[Xxx_XxxPkg] {  
}
```

Аутентификация

Читайте в разделе Аутентификация в REST/SOAP-сервисах.

Методы RestPkg, RestESPkg, RestSSPkg

- `get(relativePath: String): AnyRef` — вызывается при поступлении GET-запроса. Допускается возврат: `null`, `None`, `String`, `ResponseBuilder`.
- `post(relativePath: String): AnyRef` — вызывается при поступлении POST-запроса. Допускается возврат: `null`, `None`, `String`, `ResponseBuilder`.
- `onActivate()` — вызывается при подключении новой Gtk-сессии.
- `onDeactivate()` — вызывается при закрытии Gtk-сессии.
- `beforeReload(keyBundle: KeyBundle)` — вызывается перед перезагрузкой SBT.
- `afterReload(keyBundle: KeyBundle)` — вызывается после перезагрузки SBT и пересоздании Gtk-сессии.
- `isSupportsSharedSession(): Boolean` — метод указывает, что пакет может обрабатывать запросы в режиме разделяемой Gtk-сессии.
- `isSupportsExclusiveSession(): Boolean` — метод указывает, что пакет может обрабатывать запросы в режиме эксклюзивной Gtk-сессии.

Допустимые результаты методов `get()` и `post()`

От типа возвращённого значения будет зависеть способ формирования ответа на HTTP-запрос.

- `null`, `None` — ответ будет сформирован через `RestfulContext().get.responseBuilder`.
- `String` — в ответ будет записана возвращённая строка.
- `ResponseBuilder` — ответ будет сформирован через возвращённый `ResponseBuilder`. Он может быть равен `RestfulContext().get.responseBuilder` либо сформирован произвольным образом.
- `Response` — ответом будет возвращённое значение.

Адреса

Доступно с AS 1.14 RC7:

- `http://{server:port}/app/sys/rest/es/pkg/{Xxx_XxxxPkg}/{relativePath}`
- `http://{server:port}/app/sys/rest/ss/pkg/{Xxx_XxxxPkg}/{relativePath}`

где:

- `{Xxx_XxxxPkg}` — имя прикладного пакета, унаследованного от `RestfulPkg`;
- `{relativePath}` — произвольный путь, который будет передан в методы `get/post` прикладного пакета;
- `http://{server:port}` — адрес подключения к серверу;
- `sys` — системный прикладной модуль, в будущем будет `app/sys`;
- `rest` — шлюз для REST-запросов;
- `es / ss` — группировка по времени жизни Gtk-сессии (`Exclusive Session / Shared Session`);
- `pkg` — узел для доступа к пакетам.

Рабочее пространство

`Workspace` — рабочее пространство.

Используется для возможности параллельной работы нескольких Gtk-сессий в рамках одного пользователя в режиме `Exclusive Session`.

Для предотвращения неконтролируемого разрастания сессий количество сессий на пользователя в эксклюзивном режиме ограничено. На одного пользователя и один `workspace` может существовать только одна сессия.

Рабочее пространство задаётся в HTTP-заголовке:

`Workspace` — имя рабочего пространства пользователя.

Exclusive Session

Принцип работы сервиса

1. Получение HTTP-запроса (GET или POST).
2. Проверка авторизационных данных пользователя.
3. Получение, захват существующего рабочего сеанса или создание нового.
4. Вызов метода прикладного пакета `get(...)` или `post(...)`, соответственно.
В данном методе производится формирование тела HTTP-ответа.
5. Отправка ответа.

Жизненный цикл HTTP-сессий

При обращении к REST-сервису:

1. Если cookie `JSESSIONID` не задан, создаётся новая HTTP-сессия.
2. Иначе используется сессия с переданным идентификатором.
Если переданный ID некорректен, создаётся новая HTTP-сессия.
3. Обработка запроса.
4. Возврат результата.
При этом cookie `JSESSIONID` будет содержать идентификатор HTTP-сессии.

Время жизни HTTP-сессии — 15 минут. При отсутствии обращений к HTTP-сессии в течение этого интервала сессия уничтожается, и установленные в её атрибуты значения становятся недоступны.

Жизненный цикл Gtk-сессий

Gtk-сессия существует в разрезе четырёх параметров:

1. **Алиас базы данных** — получается из HTTP-заголовка `Database`. Если заголовок не передан, используется алиас из конфигурационного файла сервера.
2. **Имя пользователя** — получается из авторизационных данных, передаётся в HTTP-заголовке `Authorization`.
3. **Имя прикладного пакета** — получается из строки адреса.
4. **Рабочее пространство** — получается из HTTP-заголовка `Workspace`. Если не задано, используется `Default`.

Одной HTTP-сессии (одному `JSESSIONID`) может соответствовать только одна Gtk-сессия. При обращении к другому пакету из одной HTTP-сессии предыдущая Gtk-сессия будет деактивирована и закрыта.

Таймаут Gtk-сессии по умолчанию — 15 минут. Неиспользуемая Gtk-сессия будет закрыта через 15 минут. Изменить таймаут или закрыть сессию по завершению обработки запроса возможно через установку свойств `RestEXContext()`.

Создание новой сессии

Происходит в случае, если:

- в запрос не передан cookie HTTP-сессии;
- или Gtk-сессия не существует в уникальном разрезе:
 - Имя пользователя;
 - База данных;
 - Workspace;
 - Пакет.

При этом:

- Создаётся Gtk-сессия.
- Вызываются методы прикладного пакета:
 - `onActivate()`;
 - в соответствии с типом HTTP-запроса — один из методов обработки запроса:
 - * `get()`;
 - * `post()`.
- В ответ добавляется cookie HTTP-сессии.

Работа в текущей сессии

Происходит в случае, если:

- в запрос передан cookie HTTP-сессии;
- и Gtk-сессия существует в уникальном разрезе, и её cookie совпадает с переданным в запросе.

Таймаут сессии

Происходит в случае, если:

- в запрос передан cookie несуществующей HTTP-сессии;
- Gtk-сессия не существует.

При этом:

- Происходит создание новой сессии и вызов метода обработки (см. «Создание»).
- Возвращается новый cookie.

Конфликт сессии

Происходит в случае, если:

- в запрос передан cookie HTTP-сессии;
- Gtk-сессия существует в уникальном разрезе, и её текущий cookie не совпадает с cookie HTTP-сессии.

При этом:

- Генерируется ошибка захвата сессии.

Захват сессии

Происходит при условии:

- в запрос не передан cookie HTTP-сессии;
- или передан устаревший cookie, который совпадает с cookie в Gtk-сессии;
- Gtk-сессия существует в уникальном разрезе.

При этом:

- Вызываются методы прикладного пакета:
 - `onDeactivate()`;
 - `onActivate()`.
- В соответствии с типом HTTP-запроса вызывается один из методов обработки.
- Возвращается cookie новой сессии.

Доступ к данным REST-запроса из прикладного кода

В прикладном коде Pkg-класса, выполняемом при REST-запросе, доступен объект `RestESContext`, предоставляющий доступ к данным REST-запроса и REST-ответа.

Для получения контекста выполните:

```
val restContextOpt = RestESContext()
```

Методы контекста:

- `sessionTimeout: Long` — время жизни Gtk-сессии после последнего запроса.
- `request: HttpServletRequest` — объект-запрос, предоставляет доступ к параметрам, кукам и т.д.
- `responseBuilder: Response.ResponseBuilder` — билдер ответа.
- `markSessionClose(): Unit` — устанавливает флаг закрытия Gtk-сессии по завершении обработки запроса.
- `isSessionClose: Boolean` — возвращает значение флага закрытия Gtk-сессии.

Обработка ошибок

Коды ошибок

- 500 — server error. Прикладное исключение.
- 409 — conflict. Конфликт сессий.

Формат ошибок

При возникновении ошибки по умолчанию ответ будет возвращён в формате XML.

Для изменения формата ответа передайте HTTP-параметр **format**.

Доступные значения:

- xml;
- json.

Пример:

```
http://localhost:8080/app/sys/rest/es/pkg/Gs3_RestfulTestPkg/anypath?format=json
```

XML-ответ с ошибкой

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <status>%s</status>
  <error>
    <type>%s</type>
    <message>%s</message>
    <stacktrace><![CDATA[%s]]></stacktrace>
  </error>
</response>
```

JSON-ответ с ошибкой

```
{
  "response": {
    "status": 0,
    "data": {},
    "error": {
      "type": "",
      "message": "",
      "stacktrace": ""
    }
  }
}
```

Пример обращения к сервису

Java

```
package ru.bitec.app.sys.rest;

import org.junit.Test;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Invocation;
import javax.ws.rs.core.Cookie;
import javax.ws.rs.core.Response;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class PkgRestServiceTest0 {

    private static String rootAddress = "http://localhost:8080/sys/rest/es";
    private static Client client = ClientBuilder.newClient();
    private Path testPath = Paths.get(System.getProperty("user.dir"), "src\\test\\java\\
↪ru\\bitec\\app\\sys\\rest");
    private Path cookiePath = testPath.resolve("JSESSIONID.cookie");

    @Test
    public void get_2_Cookie_test() throws Exception {
        String JSESSIONID_Cookie = load_JSESSIONID_Cookie();
        Invocation.Builder builder = client.target(rootAddress + "/pkg/Gs3_
↪RestfulTestPkg/anypath").request()
            .header("Authorization", "Basic " + java.util.Base64.getEncoder().
↪encodeToString("admin:admin".getBytes("UTF-8")))
            .cookie(Cookie.valueOf(JSESSIONID_Cookie));
        Response response = builder.get();
        try {
            save_JSESSIONID_Cookie(response);
            String s = response.readEntity(String.class);
            System.out.println("response: " + s);
        } finally {
            response.close();
        }
    }

    private void save_JSESSIONID_Cookie(Response response) throws IOException {
        if (response.getCookies().containsKey("JSESSIONID")) {
            String cookie = response.getCookies().get("JSESSIONID").toString();
            Files.write(cookiePath, cookie.getBytes(Charset.defaultCharset()));
        }
    }

    private String load_JSESSIONID_Cookie() throws IOException {
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
    if (Files.exists(cookiePath))
        return new String(Files.readAllBytes(cookiePath), Charset.defaultCharset());
    else
        return null;
}
}
```

REST-сервис для взаимодействия с пользовательскими сессиями

Сервис доступен только в режиме разработки. В проектных решениях недоступен.

Функциональность сервиса:

1. Получение списка сессий, удовлетворяющих условиям поиска.
2. Выполнение Jexl-скриптов в контексте главной выборки приложения.

Получение списка сессий

При отправке HTTP GET на адрес сервиса будет возвращён JSON со списком сессий, удовлетворяющих условиям поиска. HTTP-запрос может содержать следующие параметры:

- **sbt** — фильтр по имени SBT;
- **clientId** — фильтр по идентификатору клиента (этот идентификатор присваивается каждому экземпляру браузера и содержится в куках);
- **user** — фильтр по имени пользователя;
- **app** — фильтр по имени главной выборки приложения (поиск осуществляется по вхождению переданного значения в полное имя главной выборки).

Пример HTTP GET:

```
http://localhost:8080/app/sys/rest/sessions?sbt=test&user=admin&clientId=123456&app=Xxx_
→xxxxxx
```

Пример ответа:

```
{
  "sessions": [
    {
      "sid": "E1",
      "id": "02b5f602-ac2c-4259-9452-18840a1cd124",
      "user": "admin",
      "clientId": "B54E2C1F-04E5-4BB2-9372-A9AB8E9C6676",
      "database": "PGTEST",
      "app": "gtk-ru.bitec.app.gs3.Gs3_TestXmlApplication"
    }
  ]
}
```


Выполнение Jexl в контексте главной выборки приложения

Простой HTTP POST

При отправке HTTP POST на адрес сервиса в контексте главной выборки приложения сессии с переданным идентификатором будет выполнен Jexl-скрипт, переданный в теле HTTP POST.

Адреса сервисов:

- `http://{server:port}/app/sys/rest/sessions/{id}/jexl/mainssel`.
Где {id} — идентификатор сессии. Значение {id} можно получить из результата HTTP GET к сервису.
- `http://{server:port}/app/sys/rest/sessions/new/jexl/mainssel?appname={имя_гл_выборки}`.
При этом будет запущена новая сессия.
Для корректного запуска новой сессии и открытия приложения необходимо передать имя главной выборки через HTTP-параметр `appname`.

Пример HTTP POST:

```
http://localhost:8080/app/sys/rest/sessions/949d77e9-80bf-4b93-bd3b-b424f8887783/jexl/  
↪mainssel  
http://localhost:8080/app/sys/rest/sessions/new/jexl/mainssel?appname=Gс3_  
↪TestXmlApplication
```

В ответе вернётся JSON с результатом выполнения:

```
{  
  "sessionId": "14849330-3196-4b33-80c5-caa370186720",  
  "result": "[результат выполнения Jexl]"  
}
```

Пример на Java

```
@Test  
public void post_test_1() throws Exception {  
    Map<String, Object> rootMap = (Map<String, Object>) Json.parse(doTestGET(rootAddress,   
↪+ "?sbt=test"));  
    List<Map<String, Object>> sessions = (List<Map<String, Object>>) rootMap.get(  
↪"sessions");  
    if (sessions.isEmpty())  
        throw new Exception("No one ESession opened.");  
    Map<String, Object> session = sessions.get(0);  
    String id = (String) session.get("id");  
    String rootAddress = "http://localhost:8080/app/sys/rest/sessions";  
    doTestPOST(rootAddress + "/" + id + "/jexl/mainssel", "Some Jexl Script");  
}  
  
public String doTestGET(String uri, String database) throws Exception {  
    System.out.println("HTTP GET: " + uri);  
    Invocation.Builder builder = client.target(uri).request()  
        .header("Authorization", "Basic " + java.util.Base64.getEncoder().  
↪encodeToString("admin:admin".getBytes("UTF-8")));  
    if (database != null) {  
        builder.header("Database", database);  
    }  
}
```

(продолжается на следующей странице)

```

    }
    String result = "";
    Response response = builder.get();
    try {
        System.out.println("response status = " + response.getStatus());
        System.out.println("response MediaType = " + response.getMediaType());
        if (response.getStatus() == 200) {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            try (InputStream inputStream = response.readEntity(InputStream.class)) {
                StreamHelper.copyStream(inputStream, out);
            }
            byte[] bytes = out.toByteArray();
            System.out.println("response: " + bytes.length + " bytes");
            result = new String(bytes);
        } else {
            result = response.readEntity(String.class);
        }
    } finally {
        response.close();
    }
    System.out.println(result);
    return result;
}

public String doTestPOST(String uri, String database, String body) throws Exception {
    System.out.println("HTTP POST: " + uri);
    Invocation.Builder builder = client.target(uri).request()
        .header("Authorization", "Basic " + java.util.Base64.getEncoder().
        ↪ encodeToString("admin:admin".getBytes("UTF-8")));
    if (database != null) {
        builder.header("Database", database);
    }
    String result = "";
    Entity entity = Entity.entity(body, MediaType.TEXT_PLAIN_TYPE);
    Response response = builder.post(entity);
    try {
        System.out.println("response status = " + response.getStatus());
        System.out.println("response MediaType = " + response.getMediaType());
        if (response.getStatus() == 200) {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            try (InputStream inputStream = response.readEntity(InputStream.class)) {
                StreamHelper.copyStream(inputStream, out);
            }
            byte[] bytes = out.toByteArray();
            System.out.println("response: " + bytes.length + " bytes");
            result = new String(bytes);
        } else {
            result = response.readEntity(String.class);
        }
    } finally {
        response.close();
    }
}

```

```

System.out.println(result);
return result;
}

```

Form HTTP POST

Альтернативным вариантом передачи Jexl-скрипта является отправка на адрес сервиса HTTP-формы, поле которой содержит Jexl-скрипт.

Адреса сервисов:

- `http://{server:port}/app/sys/rest/form/sessions/{id}/jexl/mainse1.`
- `http://{server:port}/app/sys/rest/form/sessions/new/jexl/mainse1?appname={имя_гл_выборки}.`

В ответе вернётся команда перенаправления на страницу входа с идентификатором сессии в качестве HTTP-параметра.

Пример HTML-страницы

```

<html>
  <body onload="onLoadFunc()">
    <script>
function onLoadFunc() {
  const form = document.createElement('form');
  form.method = 'POST';
  form.id = 'MultilinePostForm'
  form.action = 'http://localhost:8080/app/sys/rest/form/sessions/new/jexl/mainse1?
→ appname=G33_TestXmlApplication';
  const hiddenField = document.createElement('input');
  hiddenField.type = 'hidden';
  hiddenField.name = 'script';
  hiddenField.value = `Btk_ClassAvi.list().newForm().open();
G33_RootTestAvi.list().newForm().open();`;
  form.appendChild(hiddenField);
  document.body.appendChild(form);
  form.submit();
}
    </script>
  </body>
</html>

```

Сервис отчётов

Сервис предназначен для формирования печатных документов средствами сервера Global 3 по запросу из внешних систем.

Для получения файла со сформированным отчётом необходимо отправить HTTP GET по адресу сервиса.

Адреса сервисов:

- `http://{server:port}/app/sys/rest/report/{name};`
- `http://{server:port}/app/sys/rest/report/{name}/{date};`

где:

- `{name}` — системное имя печатной формы;
- `{date}` — дата версии отчёта.

Пример:

```
http://localhost:8080/app/sys/rest/report/Rpt_JasperSimpleQuery/15.02.2019
```

Аутентификация

Читайте в разделе Аутентификация в REST/SOAP-сервисах.

Параметризация отчёта

Для передачи параметров в формируемый отчёт передавайте значения через HTTP-параметры. Служебные символы и пробелы должны быть заменены escape-символами.

Пример:

```
http://localhost:8080/app/sys/rest/report/Rpt_JasperSimpleQuery?param1=value1&
↪param2=value2
```

Аутентификация в REST/SOAP-сервисах

Сервисы используют HTTP-аутентификацию двух типов: **Basic** и **Bearer**.

Basic — с использованием логина, пароля и имени базы

При Basic-аутентификации клиент должен передать в запросе:

- Имя пользователя;
- Пароль;
- Имя базы.

Имя пользователя и пароль передаются через HTTP-заголовок **Authorization**.

Значение: **Basic {Base64Cred}**, где **{Base64Cred}** — строка **user:password**, закодированная в Base64.

Имя базы может быть передано (в порядке приоритета):

- Через сегмент строки адреса. Пример: `http://server/{DATABASE}/;`
- Через HTTP-заголовок `Database` со значением `{DATABASE}`;
- Через HTTP-параметр `Database` со значением `{DATABASE}`. Пример: `http://server/?Database={DATABASE}`.

Где:

`{DATABASE}` — имя базы данных.

Если имя базы не передано ни одним из описанных способов, будет произведена попытка аутентификации с использованием имени базы по умолчанию. База по умолчанию определяется из конфигурационного файла `global3.config.xml` (в порядке приоритета):

- Значение атрибута `<databases defaultDb="{DATABASE}"/>`;
- Значение атрибута `<database alias="{DATABASE}"/>` первой базы в списке конфигураций `<databases/>`.

Bearer — с использованием токена аутентификации

При Bearer-аутентификации клиент должен передать в запросе:

- токен аутентификации;
- имя базы.

До версии AS 1.20 RC 15 включительно, доступны только токены, сформированные сервером приложений на основе: имени базы, имени пользователя и пароля.

До версии AS 1.20 RC 15 включительно, указания имени базы при Bearer-аутентификации не требуется.

Запрос должен содержать HTTP-заголовок `Authorization`.

Значение `Bearer {Token}`, где `{Token}` — ключ авторизации, присвоенный пользователю можно получить в прикладном коде от сущности:

```
session.user.token: String
```

Имя базы может быть передано (в порядке приоритета):

- Через сегмент строки адреса. Пример: `http://server/{DATABASE}/;`
- Через HTTP-заголовок `Database` со значением `{DATABASE}`;
- Через HTTP-параметр `Database` со значением `{DATABASE}`. Пример: `http://server/?Database={DATABASE}`.

Где:

`{DATABASE}` — имя базы данных.

Если имя базы не передано ни одним из описанных способов, будет произведена попытка аутентификации с использованием имени базы по умолчанию. База по умолчанию определяется из конфигурационного файла `global3.config.xml` (в порядке приоритета):

- Значение атрибута `<databases defaultDb="{DATABASE}"/>`;
- Значение атрибута `<database alias="{DATABASE}"/>` первой базы в списке конфигураций `<databases/>`.

Типы токенов аутентификации

- **ast** — *Application Server Token*.
- **gjwt** — *Gtk Json Web Token*.

Application Server Token

Токен формируется сервером приложений после входа в приложение Global с использованием имени пользователя и пароля. Токен возвращается клиенту в cookie **access_token**, привязанной к подмножеству адресов `http[s]://server/{DATABASE}/`. Эта cookie автоматически присоединяется ко всем запросам по адресам `http[s]://server/{DATABASE}/`, выполняемым из браузера, аутентифицированного в системе Global.

Токен устаревает после перезапуска сервера приложений.

Gtk Json Web Token

Токен по стандарту JWT формируется в модуле GTK (или на внешнем сервисе) и валидируется в модуле GTK.

GJWT разделяются на подтипы:

- **UserHash** — *Долгоживущий токен пользователя*.
- **UserCrt** — *Подписанный пользователем токен*.
- **ProxyCrt** — *Подписанный прокси-пользователем токен*.

Долгоживущий токен пользователя

Формируется администратором системы Global и сообщается пользователю любым удобным способом, исключаяющим утечку токена.

Токен, сопоставленный с учётной записью пользователя, хранится в БД решения.

Срок годности определяется администратором.

Тело JWT (payload) должно содержать следующие поля:

- **typ** — "UserHash".
- **sub** — имя пользователя, под которым необходимо аутентифицироваться.
- **exp** — дата устаревания.

При проверке валидности токена проверяется существование токена в списке сопоставленных с учётной записью токенов и дата устаревания.

Подписанный пользователем токен

Формируется пользователем и подписывается закрытым RSA-ключом пользователя. Открытый RSA-ключ, сопоставленный с учётной записью пользователя, хранится в БД решения. Срок годности определяется пользователем.

Тело JWT (payload) должно содержать следующие поля:

- **typ** — "UserCrt".
- **sub** — имя пользователя, под которым необходимо аутентифицироваться.
- **cid** — идентификатор открытого ключа в системе Global, сопоставленного с пользователем.
- **exp** — дата устаревания.

При проверке валидности токена на основе полей **sub** и **cid** из БД решения получается открытый RSA-ключ и проверяется валидность подписи JWT.

Подписанный прокси-пользователем токен

Прокси-аутентификация - это аутентификация под именем пользователя №2 от имени ↪ пользователя №1, именуемого прокси-пользователем.

Данный вид аутентификации может быть использован внешними планировщиками задач, которые ↪ необходимо выполнять под разными пользователями. При этом планировщику нет ↪ необходимости хранить и передавать секретные учётные данные пользователей.

Формируется прокси-пользователем и подписывается закрытым RSA-ключом прокси-пользователя. Открытый RSA-ключ, сопоставленный с учётной записью прокси-пользователя, хранится в БД решения.

Срок годности определяется прокси-пользователем.

Тело JWT (payload) должно содержать следующие поля:

- **typ** — "ProxyCrt".
- **sub** — имя пользователя, под которым необходимо аутентифицироваться.
- **psub** — имя прокси-пользователя, которому принадлежат закрытый и открытый ключи.
- **cid** — идентификатор открытого ключа в системе Global, сопоставленного с прокси-пользователем.
- **exp** — дата устаревания.

При проверке валидности токена на основе полей **psub** и **cid** из БД решения получается открытый RSA-ключ прокси-пользователя и проверяется валидность подписи JWT.

Если токен валиден и пользователь **psub** имеет права на выполнение кода от имени других пользователей, запрос аутентифицируется под именем пользователя **sub**.

Пример запроса с аутентификацией по токену

В примере используются библиотеки:

- "io.jsonwebtoken" % "jjwt-api" % "0.10.8".
- "io.jsonwebtoken" % "jjwt-impl" % "0.10.8".
- "io.jsonwebtoken" % "jjwt-jackson" % "0.10.8".
- "org.apache.httpcomponents" % "httpclient" % "4.5.8".

```
import io.jsonwebtoken.Jwts;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.junit.Assert;
import org.junit.Test;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.util.Calendar;
import java.util.Date;
import java.util.stream.Collectors;

public class AuthProviderTest {

    private PrivateKey getPrivateKey(String proxyUser) throws NoSuchAlgorithmException {
        // Формируется рандомный закрытый ключ.
        // В рабочем коде необходимо использовать реальный закрытый ключ, принадлежащий
        ↪ пользователю.
        return KeyPairGenerator.getInstance("RSA").generateKeyPair().getPrivate();
    }

    @Test
    public void authenticate_with_gjwt_proxy_cert() throws Exception {
        Calendar calendar = Calendar.getInstance();
        calendar.add(Calendar.DAY_OF_MONTH, 7);
        String proxyUser = "proxy_user";
        String gjwt = "gjwt_" + Jwts.builder()
            .setAudience("GS") // Кому предназначен токен.
            ↪ Необязательный параметр.
            .setIssuer("Scheduler") // Кем сформирован токен.
            ↪ Необязательный параметр.
            .setSubject("real_user") // Имя пользователя, под которым
            ↪ необходимо аутентифицироваться.
            .claim("typ", "ProxyCrt") // Подтип токена. В данном случае,
```

(продолжается на следующей странице)


```

→ токен для прокси-аутентификации.
        .claim("psub", proxyUser)           // Имя прокси-пользователя.
        .claim("cid", "123456789")          // Идентификатор открытого RSA-ключа
→ прокси-пользователя в системе Глобал.
        .setExpiration(calendar.getTime())  // Дата устаревания токена.
        .signWith(getPrivateKey(proxyUser)) // Закрытый ключ прокси-пользователя,
→ которым будет подписан токен.
        .compact();

    HttpGet httpGet = new HttpGet("http://localhost:8080/PGTEST/app/sys/rest/ss/pkg/
→ Gs3_RestfulTestPkg/anypath");
    httpGet.addHeader(HttpHeaders.AUTHORIZATION, "Bearer " + jwt);
    try (CloseableHttpClient client = HttpClients.createDefault()) {
        try (CloseableHttpResponse response = client.execute(httpGet)) {
            try (InputStream in = response.getEntity().getContent()) {
                String s = new BufferedReader(new InputStreamReader(in,
→ StandardCharsets.UTF_8))
                    .lines()
                    .collect(Collectors.joining(System.lineSeparator()));
                System.out.println(s);
            }
        }
    }
}

```

Администрирование REST-сервисов

Новое расположение документации

Администрирование SOAP-вызовов

Новое расположение документации

Работа с HTTP-запросами

Для отправки HTTP-запроса и обработки полученного ответа предусмотрен класс `Btk_HttpPkg`.

Доступна настройка конфигурации запроса `RequestConfig`, создание запроса `HttpRequest` с необходимым методом, заполнение тела запроса, обработка полученного ответа `HttpResult`.

Конфигурация и отправка запроса, обработка ответа

Построение запроса

Пример Get-запроса:

```
Btk_HttpPkg().get("https://httpbin.org/status/200".ns)
    .configAppend(_ .setConnectTimeout(10000))
    .configAppend(_ .setMaxRedirects(3))
    .headers(("key".ns, "value".ns))
    .basicAuth("login".ns, "password".ns)
    .build()
    .process(response => {
        print(response.getStatusLine.getStatusCode)
        EntityUtils.consume(response.getEntity)
    })
    .get
```

- `get()`, `post()` и прочие — создание запроса с нужным методом
- `configAppend()` — задаёт свойства конфигурации запроса
- `headers()` — для установки заголовков
- `basicAuth()` или `bearerAuth()` — для установки данных аутентификации
- `build()` — переход к обработчикам результата

Установка тела запроса. Отправка и получение JSON

Для Post-запроса есть возможность установки тела запроса с нужным типом содержимого с помощью методов:

- `setEntity(entity: HttpEntity)`
- `setStringEntity(body: NSString)`
- `setJSONEntity(body: NSString)`
- `setXMLEntity(body: NSString)`

Пример запроса, передающего и получающего JSON-объект:

```
val svRequestBody = "{ \"data\": \"Some Data Sample\" }"
val jvResponseBody = Btk_HttpPkg().post("https://httpbin.org/anything".ns)
    .setJSONEntity(svRequestBody)
    .build()
    .getJsonObject()
    .get
    print(jvResponseBody)
```

Обработка результата

Метод process

- С помощью лямбды `HttpResponse -> T`
- Реализовав интерфейс `org.apache.http.client.ResponseHandler`
- Реализовав интерфейс `org.apache.http.impl.client.AbstractResponseHandler` (Автоматическое формирование ошибок при http статусе ≥ 300)

Внимание

При обработке ответа с помощью пользовательского `ResponseHandler`, необходимо освобождать ресурсы и соединение, связанные с ответом.

1. Ручным закрытием `response.getEntity().getContent().close()`
2. Передачей `response.getEntity().getContent` в методы которые сами закрывают `InputStream`
3. Вызовом `EntityUtils.consume(response.getEntity)`

Готовые обработчики

Также, помимо `process()`, к использованию доступны следующие методы для получения ответа:

- `execute()` — выполнить запрос и вернуть `Try[Unit]`
- `getText()` — тело ответа в виде строки. `Try[NString]`
- `getJsonObject()` — ответ в виде `Try[JsonObject]`
- `getJSONArray()` — ответ в виде `Try[JArray]`
- `forStream[T](f: InputStream => T)` — обработать `InputStream`, полученный из содержимого ответа

Обработка ошибок

Пример обработки ответа с ошибкой:

```
try app {
  val jvResponseBody = Btk_HttpPkg().post("https://httpbin.org/json".ns)
    .basicAuth("login".ns, "password".ns)
    .setStringEntity("data")
    .build()
    .getJsonObject() match {
      // HttpResponseException если были использованы готовые обработчики или основанный
      ↪ на `HttpResponseException`.
      case Failure(ex: HttpResponseException) if ex.getStatusCode == 401 =>
        throw AppException(s"Ошибка ответа на запрос. Код: ${ex.getStatusCode}").
        ↪ Причина: Неверный логин или пароль.")
      case Failure(ex: HttpResponseException) if ex.getStatusCode == 405 =>
        throw AppException(s"Ошибка ответа на запрос. Код: ${ex.getStatusCode}").
```

(продолжается на следующей странице)

```

↪Причина: Использование недопустимого метода.")
    // Ошибки обработки, socket timeout и mn
    case Failure(ex) => throw ex
    case Success(jObj) => jObj
  }
  print(jvResponseBody)
}

```

Принципы работы commit и flush

Применение Session.commit, Session.commitWork, flush и flush(true)

session.commit - применяет изменения сессии к базе данных с успешным завершением транзакции, очищает транзакционный кэш после операции, обнуляет счетчик загруженных ячеек.

session.commitWork - применяет изменения сессии к базе данных без завершения транзакции, очищает транзакционный кэш после операции, обнуляет счетчик загруженных ячеек.

session.commitWorkAuto - используется при обработке больших объемов данных, чтобы избежать превышения лимита загружаемых ячеек. Работает аналогично commitWork, но имеет параметр nPercentTx и обрабатывает только, когда количество используемых ячеек достигает указанного в параметре процента от лимита. (По умолчанию 50% от лимита ячеек)

session.flush - применяет изменения сессии к базе данных без завершения транзакции. Есть параметр cleanTransactCache, если true очищает транзакционный кэш (загруженные объекты) после операции, обнуляет счетчик загруженных ячеек. Если false (по умолчанию), то происходит только синхронизация данных в БД, загруженные объекты остаются в кеше.

Внимание

Рекомендуется использовать session.commitWork вместо session.flush(true)

session.rollback() - откатывает текущую транзакцию и полностью очищает память.

Оптимальная стратегия коммитов:

PostgreSQL плохо работает с длинными транзакциями, так как:

- Долгие транзакции удерживают соединения и могут привести к нехватке пула соединений
- Долгая незавершенная транзакция может блокировать работу других пользователей

Поэтому рекомендуется использовать коммиты:

- по окончании логически завершенного блока операций с изменениями в данных
- перед открытием модального окна
- избегать коммитов в серверных методах, так как кто-то может вызвать метод в другом контексте и ожидать полный откат при ошибке, а получить лишь частичный, что касается изменения после коммита на методе.

При обработке больших объемов данных, чтобы избежать превышения лимита загружаемых ячеек, надо сбрасывать пачками данные в БД с очисткой транзакционного кэша при помощи commitWorkAuto

Пример

```
for (rv <- new ASelect {
  val id: NLongColumn = asNLong("id")
  val idBalAcc: NLongColumn = asNLong("idBalAcc")
  val nSum: NNumberColumn = asNNumber("nSum")
  SQL(sQuery)
  on("dExecDate" -> dExecDate, "idaDep" -> LongPgArray(idaDep), "idaEmployee" ->
  ↪ LongPgArray(idaEmployee), "idaAcc"-> LongPgArray(idaAcc.toList), "idBisObj" ->
  ↪ idBisObj, "idDepOwner" -> idDepOwner)
}) {
  insertByParent(Asf_InventoryApi().load(idpInventory)) :/ { rop =>
    setnOrder(rop, counter.nn)
    setIdInvNumb(rop, rv.id())
    setIdAcc(rop, rv.idBalAcc())
    setnSum(rop, rv.nSum())
    setnQtyRemain(rop, 1.nn)
    rop
  }
  session.commitWorkAuto()
}

session.commitWork()
```

Отличия commit и flush

Действие	<i>flush()</i>	<i>flush(true)</i>	<i>commit()</i>	<i>commitWork()</i>
Запись в БД	Да	Да	Да	Да
Фиксация транзакции	Нет	Нет	Да	Нет
Очистка памяти	Нет	Частично	Да	Да
Возможность отката	Да	Да	Нет	Да

Очистка кэша без коммита

Для очистки кэша и синхронизации данных с БД рекомендуется использовать метод `commitWork()`. Он полезен в случаях, когда:

- Требуется освободить ресурсы без изменения данных в БД.
- Обрабатывается большой объем данных и важно избежать накопления записей в памяти.
- Необходимо минимизировать риск потери транзакционной целостности из-за раннего коммита.

Пример использования

Загрузка объектов из БД частями. Используя реляционный запрос получает значения первичных ключей для массовой прогрузки объектов в память. Количество одновременно загружаемых объектов указывает в параметре `chunkSize`. Разработчик должен сам очищать память (вызывать `commitWorkAuto()`, `commitWork()` или `commit()`) внутри переданного обработчика.

```

{{{
val ids: List[NLong] = ...

Btk_BulkProcessPkg().chunkedQuery(
    SomeClassApi(),
    500,
    "select id from SomeClass where id = Any({ids})",
    Seq("ids" -> LongPgArray(ids)),
    Seq(Btk_ObjectAta) // прогрузка коллекций
) { rops =>
    rops.foreach { rop =>
        //обработка объекта
        SomeClassApi().setText(rop, svSomeText)
        //очистка памяти
        session.commitWorkAuto()
    }
    //очистка памяти
    session.commitWork()
}
}}}
```

Автокоммиты

Автокоммит может происходить в нескольких случаях:

- После выполнения пользовательских операций, если они вызвали изменения в базе данных. Операция выполняется, вызывается `flush()` для записи изменений в БД, а затем происходит `commit()`.
- При обновлении данных `refresh()`. Если в памяти есть несохраненные изменения и вызывается `refresh()`, система сначала выполняет `flush()`, фиксирует изменения в памяти и затем отправляет запрос в БД, что приводит к коммиту.
- На `onRefresh()` или `onRefreshExt()` с реляционным запросом. На открытии выборки происходит коммит, если используется реляционный запрос на `onRefresh` без аннотации `@FlushBefore(mode = FlushBeforeMode.Disabled)`. Из-за чего могут возникать ситуации, когда происходит неочевидный коммит при открытии отображения с закладкой или выпадающим списком, на которых не указана аннотация.

Примечание

Автокоммит для реляционных запросов на `onRefresh()` или `onRefreshExt()` происходит, чтобы в БД отображались внесенный пользователем изменения и выполняемый запрос мог их считать.

RabbitMQ

Общее

RabbitMQ — это брокер сообщений, обеспечивающий обмен данными между различными частями системы по принципу «издатель-подписчик». Он позволяет приложениям отправлять и получать сообщения асинхронно.

В приложении RabbitMQ используется для обмена сообщениями между модулями и уведомлений пользователя о событиях, таких как:

- Создание новых задач;
- Результаты фоновых процессов;
- Трассировка операций;
- Перехваты вызовов серверных методов;
- Общие уведомления.

Как реализовано взаимодействие с RabbitMQ

Обмен сообщениями

- Создаются exchange'ы (точки обмена) по пользователям: `btk.service.user.<userID>`.

Примечание

При запуске приложения по умолчанию создается точка обмена на пользователя

- Сообщения публикуются в exchange с указанием типа (type) и содержимого (body).

Получение сообщений

- Подписка на exchange выполняется методом `startConsume(exchangeName)` в (`Btk_RabbitLib`).
- Все входящие сообщения на текущий момент обрабатываются через `onReceiveMessage(...)`, где по type определяется нужный обработчик:
 - `Btk_AcTrace`
 - `Btk_BackgroundRun`
 - `Btk_ServerMethodIntercept`
 - Прочие — обрабатываются через `Btk_Ept`.

Удаление exchange

Метод `deleteRabbitExchange` в (`Btk_RabbitPkg`) позволяет удалить ненужные точки обмена.

Жизненный цикл подписки на сообщения

- При запуске приложения:
 - В `ApplicationAvi.onShow()` вызывается `Btk_ApplicationLib().onStartApplication()`.
 - Это инициирует `Btk_RabbitLib().onStartApplication()`, которая автоматически подписывает пользователя на его exchange.
- При завершении приложения:
 - Вызывается `Btk_RabbitLib().onCloseApplication()`, отписывая пользователя от exchange.

Пример отправки сообщения по RabbitMQ

Пример: Отправка сообщения от трассировки прав (`Btk_AcTracePkg.createAcTraceMessageHandler`)

```
def createAcTraceMessageHandler(
    sid: NString,
    sUserCluster: NString,
    sFlag: NString,
    sUser: NString,
    sSidUser: NString,
    sUserClusterStarted: NString
): Unit = {
    try app {
        // Формируем JSON-объект
        val jsonObj = createAcTraceJsonObject(sid, sUserCluster, sFlag, sUser, sSidUser,
        ↪sUserClusterStarted)
        val exchangeName = Btk_RabbitPkg().getServiceExchangeNameByUser(Btk_UserApi().
        ↪getByMnemonicCode(sUser).get.get(_id))

        // Отправляем сообщение в exchange с указанием типа "Btk_AcTrace"
        rabbitPkg.sendMessageWithType(jsonObj.toPrettyNString, TYPE_AC_TRACE, exchangeName)
    } catch {
        case e: AppException =>
            throw e
    }
}
```


Обобщенный способ отправки сообщения

```
Btk_RabbitPkg().sendMessageWithType(messageText, messageType, exchangeName)
```

- `messageText` – JSON-строка с содержимым;
- `messageType` – строка, идентифицирующая тип сообщения;
- `exchangeName` – имя точки обмена, куда отправляется сообщение.

StyleBuilder

StyleBuilder - Инструмент, предназначенный для настройки и управления стилями, предоставляя разработчикам необходимые инструменты для работы с цветами, шрифтами и парсингом стилей.

Инструменты, предназначенные для работы с цветом

1. **setBackgroundColor** - Устанавливает цвет заднего фона, подавать на вход можно hex код, GS код или GS константы. При передаче пустой строки происходит обнуление параметра.
2. **getBackgroundColor** - Возвращает цвет заднего фона.

Инструменты, предназначенные для работы с шрифтом

1. **setFont** - Устанавливает шрифт по названию. При передаче пустой строки происходит обнуление параметра.
2. **getFont** - Возвращает название шрифта.
3. **setFontSize** - Устанавливает размер шрифта. При передаче null-значения происходит обнуление параметра.
4. **getFontSize** - Возвращает размер шрифта.
5. **setFontColor** - Устанавливает цвет шрифта. При передаче пустой строки происходит обнуление параметра.
6. **getFontColor** - Возвращает цвет шрифта.
7. **setFontBold** - Устанавливает значение, указывающее, используется ли жирный шрифт. При передаче null-значения происходит обнуление параметра.
8. **isFontBold** - Возвращает значение, указывающее, используется ли жирный шрифт.
9. **setFontItalic** - Устанавливает значение, указывающее, используется ли курсив. При передаче null-значения происходит обнуление параметра.
10. **isFontItalic** - Возвращает значение, указывающее, используется ли курсив.
11. **setFontUnderLine** - Устанавливает значение, указывающее, задает ли этот стиль горизонтальную линию под шрифтом. При передаче null-значения происходит обнуление параметра.
12. **isFontUnderLine** - Возвращает значение, указывающее, задает ли этот стиль горизонтальную линию под шрифтом.
13. **setFontStrikeOut** - Устанавливает значение, указывающее, задает ли этот стиль горизонтальную линию через шрифт. При передаче null-значения происходит обнуление параметра.

14. `isFontStrikeOut` - Возвращает значение, указывающее, задает ли этот стиль горизонтальную линию через шрифт.

Использование

Примечание

```
val style = StyleBuilder().setBackgroundColor(Color.Blue).setFontColor(«#FFAAFF»)  
.setFont(Font.Impact).setFontUnderLine(1.nn).build()
```

Color

`Color` - Объект, предназначенный для парсинга цвета, поскольку цвет, выбранный в редакторе(`editorType="colorPick"`), сохраняется в десятичном формате.

Есть список предопределённых цветов для удобства использования `ru.bitec.app.btk.sel.utils.StyleBuilder.Color#Black`

Использование

Внимание

Необходимо переопределить `onRefresh` если цвет сохраняется не в десятичном формате, а  как Hex

Примечание

```
override def onRefresh: Recs = {  
  ConfigBase.fromJson(vShapeList) match {  
    case image: ImageConfig => image  
    case line: LineConfig => line.copy(  
      fill = line.fill.optionMap(fill => Color.clToDec(Color(fill.getOrElse().  
→getGsHex()).ns).getOrElse(NString()),  
      stroke = line.stroke.optionMap(stroke => Color.clToDec(Color(stroke.getOrElse().  
→getGsHex()).ns).getOrElse(NString()))  
    )  
    case _ => null  
  }  
}  
  
@Setter  
def setfill(event: SetterEvent): Unit = {  
  val color = Color(Color.decToCl(event.attr.asNNumber).getOrElse("")).getHex()  
  processSetter {  
    case image: ImageConfig => ???  
    case line: LineConfig => line.copy(fill = color)  
  }  
}
```

```
}  
}
```

DynamicEditorBuilder

DynamicEditorBuilder - Используется в случаях, когда необходимо сделать атрибут динамическим, то есть когда editType заранее неизвестен.

Методы для работы с DynamicEditorBuilder

1. **lookup** - Создать пустой Lookup билдер.
2. **lookup(lookupEditorString)** - Создать Lookup билдер заполненный от строки редактора. Принимает строку редактора.
3. **editButton** - Создать пустой EditButton билдер.
4. **editButton(editButtonEditorString)** - Создать EditButton билдер заполненный от строки редактора. Принимает строку редактора.
5. **buttonsEdit** - Создать пустой ButtonsEdit билдер.
6. **buttonsEdit(ButtonsEditEditorString)** - Создать ButtonsEdit билдер заполненный от строки редактора. Принимает строку редактора.
7. **button** - Создать пустой Button билдер.
8. **check** - Создать пустой Check билдер.
9. **calendar** - Создать пустой Calendar билдер.
10. **colorPick** - Создать пустой ColorPick билдер.
11. **combo** - Создать пустой Combo билдер.
12. **currency** - Создать пустой Currency билдер.
13. **datePick** - Создать пустой DatePick билдер.
14. **dateTimePick** - Создать пустой DateTimePick билдер.
15. **edit** - Создать пустой Edit билдер.
16. **hotKey** - Создать пустой HotKey билдер.
17. **hyperLink** - Создать пустой HyperLink билдер.
18. **icon** - Создать пустой Icon билдер.
19. **memo** - Создать пустой Memo билдер.
20. **memoInPlaced** - Создать пустой MemoInPlaced билдер.
21. **tagLookup** - Создать пустой TagLookup билдер.
22. **text** - Создать пустой Text билдер.
23. **timePick** - Создать пустой TimePick билдер.
24. **editPassword** - Создать пустой EditPassword билдер.
25. **imageCollection** - Создать пустой ImageCollection билдер.

Список параметров	LookupBuilder	ru.bitec.app.btk.sel.attr.editor.
LookupBuilder#editorParamSet		
Список параметров	EditButtonBuilder	ru.bitec.app.btk.sel.attr.editor.
EditButtonBuilder#editorParamSet		
Список параметров	ButtonsEditBuilder	ru.bitec.app.btk.sel.attr.editor.
ButtonsEditBuilder#editorParamSet		

Примеры использования

onRefresh

```
val svEditorType: NString = if (idvRefTable.isNotNull) {
    //для параметров, ссылочных на таблицу свой редактор -
    if (Prm_DimSetApi().byParent(idvRefTable).size == 1) {
        //для таблиц с одним измерением - выпадающий список
        DynamicEditorBuilder.lookup()
        .setLookupQuery("gtk-Prm_RefTabValAvi#Lookup_OneDim")
        .setLookupListAttribute("sHeadLine")
        .setLookupKeyAttribute("nValue")
        .setChangedAttribute("sCurValue")
        .build()
    } else {
        //для остальных таблиц - по трем точкам открываем дерево
        DynamicEditorBuilder.editButton().setCanEdit(0.nn).build()
    }
} else {
    viewValue.sEditorType
}
```

lookup

```
DynamicEditorBuilder
    .lookup()
    .setLookupQuery("gtk-Prm_RefTabValAvi#Lookup_OneDim")
    .setLookupListAttribute("sHeadLine")
    .setLookupKeyAttribute("sCurValue")
    .build()
```

button

```
DynamicEditorBuilder
    .button()
    .build()
```

buttonsEdit

```
DynamicEditorBuilder
    .buttonsEdit()
    .setCanEdit(0.nn)
    .build()
```

editButton

```
DynamicEditorBuilder  
.editButton()  
.setResetButtonVisible(true)  
.build()
```

check

```
DynamicEditorBuilder  
.check()  
.build()
```

calendar

```
DynamicEditorBuilder  
.calendar()  
.build()
```

colorPick

```
DynamicEditorBuilder  
.colorPick()  
.build()
```

combo

```
DynamicEditorBuilder  
.combo()  
.build()
```

currency

```
DynamicEditorBuilder  
.currency()  
.build()
```

datePick

```
DynamicEditorBuilder  
.datePick()  
.build()
```

dateTimePick

```
DynamicEditorBuilder  
.dateTimePick()  
.build()
```

edit

```
DynamicEditorBuilder  
.edit()  
.build()
```

hotKey

```
DynamicEditorBuilder  
.hotKey()  
.build()
```

hyperLink

```
DynamicEditorBuilder  
.hyperLink()  
.build()
```

icon

```
DynamicEditorBuilder  
.icon()  
.build()
```

memo

```
DynamicEditorBuilder  
.memo()  
.build()
```

memoInPlaced

```
DynamicEditorBuilder  
.memoInPlaced()  
.build()
```

tagLookup

```
DynamicEditorBuilder  
.tagLookup()  
.build()
```

text

```
DynamicEditorBuilder  
.text()  
.build()
```

timePick

```
DynamicEditorBuilder  
.timePick()  
.build()
```

editPassword

```
DynamicEditorBuilder  
.editPassword()  
.build()
```

imageCollection

```
DynamicEditorBuilder
.imageCollection()
.build()
```

SQL-инъекции

SQL-инъекция (sqli) — это уязвимость безопасности, которая позволяет злоумышленнику вмешиваться в запросы к базе данных приложения, внедряя злонамеренный SQL-код через входные данные, меняя исходный SQL-запрос.

Проблема возникает, когда приложение «слепо» объединяет пользовательские данные с кодом SQL-запроса, не проверяя и не обрабатывая их.

Пример инъекции

Исходный запрос:

```
SELECT * FROM users WHERE username = '[user_input_1]' AND password = '[user_input_2]'
```

Злоумышленник вводит в поле `username` специальную строку: `OR 1=1 --`. Поле `password` он может оставить пустым или заполнить чем угодно.

Злонамеренный ввод:

user_input_1: `admin“ or 1=1 –`

user_input_2: `[любой]`

Результат:

```
SELECT * FROM users WHERE username = 'admin' or 1=1 --' AND password = 'text'
```

Чем опасны SQL-инъекции?

Злоумышленник может:

1. Обойти аутентификацию.
2. Получить несанкционированный доступ к данным: читать, копировать, скачивать всю базу данных.
3. Изменять и удалять данные: удалять, изменять, добавлять записи.
4. Выполнять команды на сервере БД: в некоторых случаях атака может привести к полному захвату сервера.

Методы защиты

1. **Подготовленные выражения (Prepared Statement)** - наиболее эффективный метод защиты. Это механизм выполнения SQL-запросов, при котором шаблон SQL-запроса и данные передаются отдельно. Это полностью исключает возможность SQL-инъекций. **Предпочтительнее использовать именно этот метод защиты.**

```
val svName = "admin"
val svPass = "qwerty1234"
SQL(s"SELECT * FROM users WHERE username = {sUserName} AND password = {sPassHash}")
.on("sUserName" -> svName, "sPassHash" -> svPass)
```

2. **Использование функций экранирования.** Если использовать подготовленные выражения не представляется возможным, необходимо воспользоваться функциями экранирования, они «обезвреживают» специальные символы:

- `escapeString(input: String, quoteValue: Boolean): String` - для экранирования строки. Экранирует символы одинарной кавычки (‘) в строке. Если `quoteValue = true`, заключит значение в одинарные кавычки.

```
val sUserInput = SqlEscape.escapeString("' union select id from Btk_User --", false)
SQL(s"SELECT id FROM Bs_Goods WHERE sName = '$sUserInput' ")
```

- `escapeNumber(input: String): String` - для экранирования числа. Удаляет спецсимволы (‘;’, ‘--’, ‘/*’, ‘*/’, ‘(’, ‘)’, пробелы).

```
val sIdGood = SqlEscape.escapeNumber("1 OR 1=1 --")
SQL(s""""SELECT id FROM Bs_Goods WHERE id = $sIdGood""")
```

- `escapeName(input: String): String` - для экранирования имен сущностей, например имена таблиц и колонок. Экранирует одинарные кавычки (‘), удаляет спецсимволы ‘;’, ‘--’, ‘/*’, ‘*/’, ‘(’, ‘)’.
- `escapeDate(input: String): String` - для экранирования даты. Экранирует символы одинарной кавычки (‘).
- `escapeJson(input: String): String` - для экранирования json. Экранирует символы одинарной кавычки (‘).
- `escapeBoolean(input: String): String` - для экранирования булевого значения. Если строка не равна «true», вернет «false».

Наборы значений

Наборы значений находятся в Приложение "Настройка системы" -> Настройки и сервисы -> Наборы

Сервис Избранное Сущности Настройки и сервисы Отчеты Аудит Интеграция Помощь

Выборка История запросов Переоткрыть Организация

Отображать неиспользуемые

Код Import_Goods Наименование Import_Goods

Класс Bs_Goods ТМЦ

Значения набора Структура

Отображать неиспользуемые

Наименование	Вид сравнения	Значение фильтра	Включено
Импортный ТМЦ	Равно	ИМПОРТНЫЙ ТМЦ	☒

значений

На вкладке Структура набора

Настраивается условие универсального фильтра которое задает какие значения попадают в текущий набор значений

На вкладке Структура

Можно задать подчиненные наборы значений значения которых будут добавлены к текущему в запросах

Работа с наборами значений

Универсальный фильтр

Набор значений можно задать как значение для ссылочных атрибутов в УФ

Наименование	Вид сравнения	Значение фильтра	Включено
Группа Отбор И			☒
Модуль	Набор значений	moduleStartsWithC	☒

Код

Методы подбора значений находятся в `ru.bitec.app.btk.valueset.Btk_ValueSetApi`

`ru.bitec.app.btk.valueset.Btk_ValueSetApi#getValueSetSQL(id, bWithStructure)`

Возвращает SQL строку типа

```
SELECT t.id FROM Bs_Goods t
WHERE upper(t.sName) = upper('ИМПОРТНЫЙ ТМЦ')
UNION
SELECT t.id FROM Bs_Goods t
WHERE upper(t.sName) like upper('t' || '%' )
UNION
... -- И так все наборы значений входящие в структуру переданного
```

Параметры: id - Идентификатор набора значений bWithStructure - Добавлять вложенные наборы значений

`ru.bitec.app.btk.valueset.Btk_ValueSetApi#getValueSetGidSQL(id, bWithStructure)`

Тоже что и `ru.bitec.app.btk.valueset.Btk_ValueSetApi#getValueSetSQL` только gid вместо id

`ru.bitec.app.btk.valueset.Btk_ValueSetApi#getWhereCondition(id, tableName)`

Возвращает WHERE условие для переданного набора значений Параметры: id - Идентификатор набора значений tableName - Алиас для фильтруемой таблицы

Печатные формы

Нужно добавить условие набора в параметры ПФ

Через процедуру дополнительных параметров

Подробнее см. *Процедура получения дополнительных параметров печати для ПФ*

```
propertyMap.update("myCondition", Btk_ValueSetApi.getValueSetStructureSQL(Btk_
↪ValueSetApi.findByMnemonicCode("Import_Goods")))
```

Jasper

Добавить параметр в отчет

```
<parameter name="myCondition" class="java.lang.String"/>
```

```
SELECT *
FROM Bs_Goods t
WHERE t.id IN ($P!{myCondition})
```

Шаблоны отчетов(DOCX, XLXS, TXT)

```
SELECT *  
FROM Bs_Goods t  
WHERE t.id IN (&myCondition)
```

Работа механизма DataInstall

DataInstall — это механизм преднастройки данных, который позволяет разработчику задать стартовые значения для системы прямо в метаданных класса или пакета. Его основа — скрипты, содержащие код на JEXL. Такой скрипт может выполнять практически любые операции, однако на практике DataInstall чаще всего используют для:

- регистрации состояний и связей между ними;
- создания дефолтных записей;
- добавления типов объектов;
- заполнения справочников;
- создания служебных данных, необходимых для корректной работы модуля.

Запуск DataInstall является частью процесса генерации БД. Он происходит **при выполнении генератора таблиц**, который можно запустить:

- через инструмент разработки с поддержкой генерации метаданных;
- или через команду `upgrade` по SSH.

DataInstall — естественная часть жизненного цикла развертывания: когда метаданные обновлены, система сравнивает версии и решает, какие скрипты требуется выполнить.

Ключевые особенности

- Автоматическое выполнение при генерации базы данных — запуск происходит через генератор таблиц (в среде разработки или через команду `upgrade`), как часть жизненного цикла развертывания.
- Управление версиями скриптов — система выполняет скрипт только при первом появлении или изменении версии, сравнивая её с записями в таблице `Btk_DataInstallScript`.
- Поддержка зависимостей между скриптами — перед выполнением проверяется наличие всех зависимостей; отсутствие требуемого скрипта — ошибка.
- Отложенное выполнение (`delayed`) — скрипты с флагом `delayed` не выполняются в момент генерации, а регистрируются для запуска фоновой задачей в нерабочее время.
- Топологическая сортировка очереди выполнения — зависимости преобразуются в граф, из которого строится строгая последовательность запуска, гарантирующая корректный порядок.

Общая схема процесса

Механизм DataInstall выполняется как часть обновления системы и проходит через семь последовательных этапов, обеспечивающих корректную и предсказуемую установку начальных данных.

1. Загрузка информации о ранее выполненных скриптах

Система загружает данные о всех скриптах, которые уже были запущены ранее, включая их текущие версии и статус (обычный или отложенный). Это позволяет избежать повторного выполнения одних и тех же скриптов.

2. Сбор всех доступных скриптов из метаданных

Из всех классов и пакетов системы собираются все объявленные DataInstall-скрипты. Каждый скрипт получает уникальное имя, чтобы система могла однозначно его идентифицировать.

3. Определение скриптов, требующих выполнения

Сравниваются скрипты из метаданных с историей выполнения. Запуск необходим только для новых скриптов или тех, чья версия была увеличена. Отложенные скрипты выделяются отдельно.

4. Проверка зависимостей между скриптами

Проверяется, что все зависимости каждого скрипта будут выполнены до него. Также контролируется, что обычный скрипт не зависит от отложенного — такая связь считается некорректной.

5. Построение очереди выполнения

На основе проверенных зависимостей формируется упорядоченная последовательность скриптов. Это гарантирует, что каждый скрипт будет запущен только после всех своих зависимостей.

6. Выполнение скриптов по очереди

Скрипты запускаются в установленном порядке. Если скрипт обычный — он выполняется немедленно. Если отложенный — он регистрируется для фонового запуска. Ошибки одного скрипта не останавливают выполнение остальных.

7. Фиксация результатов в базе данных

После завершения обновления система фиксирует информацию о всех успешно выполненных скриптах.

Хранение истории выполнения скриптов

Информация о выполненных DataInstall-скриптах хранится в таблице `btm_datainstallscript`.

Она используется системой для определения, какие скрипты уже были запущены, и требуется ли их повторное выполнение при обновлении.

Таблица содержит следующие ключевые поля:

- **Имя** — системное имя скрипта;
- **Версия** — текущая версия скрипта;
- **Дата прогона** — дата и время последнего успешного выполнения скрипта;
- **Пользователь** — учётная запись, под которой было установлено подключение к базе данных при запуске скрипта.

Примечание

Подключение происходит **только на уровне БД**, пользователь не входит в саму ERP-систему.

- Если указано `sys` — скрипт запускался через SSH с правами системного пользователя.
- Если указан IP-адрес — подключение было выполнено напрямую с локального компьютера.

Создание DataInstall-скриптов

DataInstall полностью определяется **в коде**, а именно:

- в `odm.xml` для классов;
- в `pkg.xml` для пакетов.

Разработчик сам пишет скрипт прямо в XML-метаданных, где указывает:

- имя скрипта;
- его версию;
- зависимости;
- флаг отложенного выполнения (если требуется);
- текст JEXL.

Разработчик вручную добавляет в `odm.xml` или `pkg.xml` секцию `<dbData>` и прописывает в ней скрипт:

```
<dbData>
  <script name="dataInstall" version="1">
    <install>SomeApi.dataInstall();</install>
  </script>
</dbData>
```

Скрипт может содержать:

- прямой JEXL-код;
- вызов API-классов (частый вариант),

После этого JEXL-вызов будет выполнен генератором данных при обновлении версии скрипта.

Использование

Ниже приведен минимальный пример «от и до»: начиная с объявления класса и заканчивая Scala-API, который вызывается DataInstall-скриптом.

1. Фрагмент `Rpltst_ForDeleteTest.odm.xml`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
  name="Rpltst_ForDeleteTest"
  caption="Тест удаления"
  supertype="reference"
  cardEditor.representation="Card"
  listEditor.representation="List"
  viewOptions.openCardType="mdi">

  <attributes>
    <attr name="id" attribute-type="Long" caption="Идентификатор" order="-1" type=
↪ "basic" isVisible="false"/>
    <attr name="idClass" attribute-type="Long" caption="idClass" order="-2" type=
↪ "basic" isVisible="false"/>
    <attr name="sSystemName" attribute-type="Varchar" caption="Системное имя" order=
↪ "10" isMnemonicCode="true"/>
  </attributes>
</class>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
<attr name="sCaption" attribute-type="Varchar" caption="Наименование" order="20"
↪isHeadLine="true"/>
<attr name="sDescription" attribute-type="Varchar" caption="Описание" order="30"/
↪>
<attr name="idState" attribute-type="Long" caption="Состояние" order="70"
    type="refObject" ref.class="Btk_ClassState"/>
</attributes>

<dbData>
  <script name="dataInstall" version="1">
    <install>Rpltst_ForDeleteTestApi.dataInstall();</install>
  </script>
</dbData>
</class>
```

Здесь определяется один скрипт с именем `dataInstall`, который вызывает метод `dataInstall()` в соответствующем API-классе.

2. Код API, вызываемый скриптом `Rpltst_ForDeleteTestApi.scala`

```
class Rpltst_ForDeleteTestApi
  extends Rpltst_ForDeleteTestDpi[Rpltst_ForDeleteTestAro, Rpltst_ForDeleteTestApi,
↪Rpltst_ForDeleteTestAta] {

  override protected def entityAta: Rpltst_ForDeleteTestAta = Rpltst_ForDeleteTestAta

  def dataInstall(): Unit = {

    session.commit()
    Btk_Pkg().setRWSharedUOWEditType()

    // регистрация состояния "Формируется"
    Btk_ClassStateApi().register(
      idpMasterClass = idClass,
      spSystemName = "Rpltst_ForDeleteTestForming",
      spCaption = "Формируется",
      bpStartState = 1.nn,
      npOrer = 100.nn)
    session.flush()

    // регистрация состояния "Сформирован"
    Btk_ClassStateApi().register(
      idpMasterClass = idClass,
      spSystemName = "Rpltst_ForDeleteTestFormed",
      spCaption = "Сформирован",
      bpStartState = 0.nn,
      npOrer = 200.nn)
    session.flush()
  }
}

object Rpltst_ForDeleteTestApi extends ApiFactory[Long, Rpltst_ForDeleteTestAro, Rpltst_
↪ForDeleteTestApi]
```

(продолжается на следующей странице)

```
class Rpltst_ForDeleteTestAro extends Rpltst_ForDeleteTestDro
class Rpltst_ForDeleteTestAta extends Rpltst_ForDeleteTestDta[Rpltst_ForDeleteTestAro, Rpltst_ForDeleteTestApi, Rpltst_ForDeleteTestAta]
object Rpltst_ForDeleteTestAta extends Rpltst_ForDeleteTestAta
```

Этот метод создает два состояния класса. Именно такие операции и составляют типичные DataInstall-скрипты.

Ограничения

- Нельзя понижать версию скрипта — система выполнит его повторно только при увеличении версии.
- Изменение имени скрипта (**name**) приводит к созданию нового скрипта. Система определяет уникальность по полному системному имени (**sSystemName**). Старый скрипт остаётся в истории, новый добавляется отдельно — это может вызвать дублирование данных.
- Отсутствие или ошибка в зависимом скрипте блокирует выполнение зависящего от него. При этом общий процесс не останавливается.
- Обычный (не отложенный) скрипт не может зависеть от отложенного — такая связь считается некорректной и фиксируется как ошибка.
- Порядок выполнения скриптов определяется исключительно через зависимости. Прямое управление очередностью невозможно.
- Не рекомендуется выполнять тяжёлые операции, внешние вызовы или сетевые запросы. Скрипты работают в контексте обновления схемы и не предназначены для длительных или блокирующих действий.
- Отложенные скрипты не выполняются сразу — они регистрируются в системе и запускаются фоновой задачей в нерабочее время.

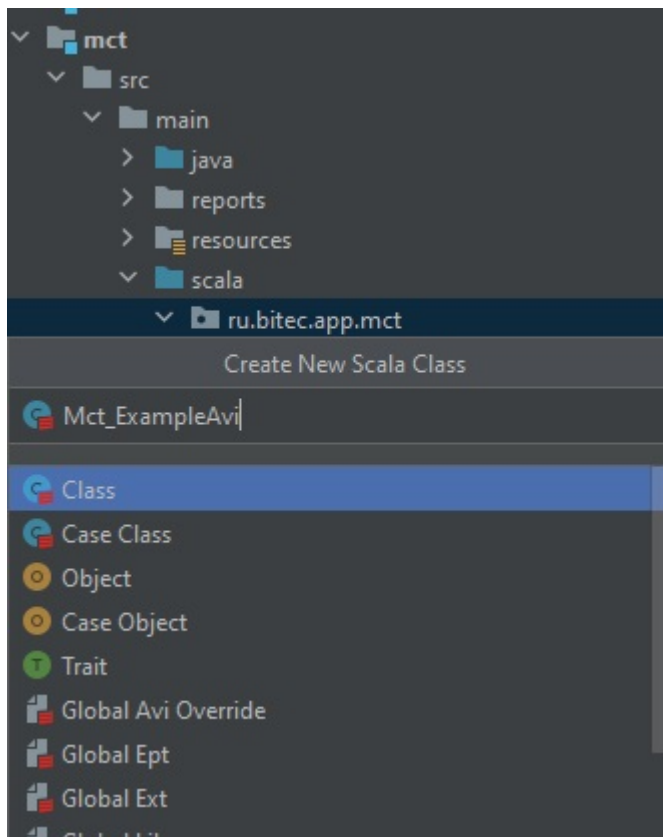
13.9 Приложение

Практические советы

Создание выборки без класса

Первичное создание файла выборки

1. В нужной директории проекта (**имя_модуля/src/main/scala/ru/bitec/app/имя_модуля/...**) создайте Scala класс, задайте ему имя в формате **<имя модуля>_<имя выборки>Avi**, например **Mct_ExampleAvi**.



- Класс должен наследоваться от `EntityAvi`.
- В файле должен присутствовать объект-компаньон, с его помощью в дальнейшем будут создаваться объекты целевых выборов в системе.
- У класса должна присутствовать аннотация `avm-файла`, где указывается файл с разметкой выборов. *Как создать avm-файл.*

Общий вид файла должен быть следующим:

```
package ...  
  
import ...  
  
object Mct_ExampleAvi extends Mct_ExampleAvi  
  
@AvmFile(name = "Mct_Example.avm.xml")  
class Mct_ExampleAvi extends EntityAvi{  
    Код выборки  
}
```


Создание трейта и методы получения данных, выводимых трейтом

1. В классе создается трейт, который в дальнейшем будет использован для открытия экземпляра выборки.

Этот трейт должен наследоваться от одного или более базовых трейтов из `EntityAvi`, или от созданных вами, например:

```
trait List extends Default with super.List

trait MyList extends List
```

Для каждого трейта должна быть создана функция-компаньон, причем если имя вашего трейта совпадает с именем базового трейта из `BaseAvi` (наследуется в `EntityAvi`) - `Card`, `List`, `Lookup`, то функцию компаньон надо переопределять, а если имя иное, то функцию нужно создать:

```
override def list(): List = {
  new List {
    override def meta = this
  }
}

def myList(): MyList = {
  new MyList {
    override def meta = this
  }
}
```

- Имя функции-компаньона начинается с прописной буквы, имя трейта с заглавной.
2. Если трейт создается для отображения данных из базы, тогда необходимо определить метод получения данных для выборки. Для этого существует два подхода:

Реляционный запрос:

1. Базово определяется в методе `selectStatement`.
2. Тело метода является обычным sql запросом, помещенным в scala-строку.

```
override protected def selectStatement: String = {
  s"""
    |select
    |  st.id
    |  ,st.sCaption
    |  ,st.sCode
    |  ,ot.sDescription as otherDescription
    |from someTable st
    |join otherTable ot on st.someParam = ot.otherParam
    |""".stripMargin
}
```

3. Если итоговый набор данных требуется отфильтровать при помощи данных, *переданных извне* (обычно из фильтров и параметров выборок), и запрос имеет линейную структуру (нет вложенных запросов, временных таблиц и т.д., в которых требуется фильтрация *по внешним данным*), тогда:

- Переопределяем метод `onRefresh` следующим образом:

```
override protected def onRefresh: Recs = {
  prepareSelectStatement(s"&DefFltReferenceMacro and someStatement and $
↪{someScalaCode}")
}
```

- `DefFltReferenceMacro` - стандартное *имя макроса соответствующего трейта из разметки авт-файла*. Вы можете задать любое другое удобное вам имя макроса.
- `someStatement` - дополнительные условия в формате sql.
- `someScalaCode` - любой Scala-код, результатом работы которого является *булево значение* или некие данные, которые подставляются в логическое выражение sql с одной из сторон:

```
s"....
... t.idState = ${getSelfVar("idState").asNLong}
...."
```

Объектный запрос:

1. Базово определяется в методе `onRefresh`.

Примечание

Метод `onRefresh` возвращает как строку в виде sql-запроса, так и набор Scala-объектов, определяющих набор данных выборки

2. В теле метода создается некоторая коллекция однотипных объектов:

```
override protected def onRefresh: Recs = {
  /***_SomeClassApi().load(id)
  /***_SomeClassApi().byParent(parentRop[или parentId]).toList[.filter[Not][.
↪orderBy]]]
  /**_someTXIndex().byKey(TXIndexKey).toList[.filter[Not][.orderBy]]]
  new OQuery(***_SomeClassAta.Type) {
    where(t.someClassAttr == someScalaValue [and[or] ....])
  }
}
```

- При помощи метода `load` - если требуется получить единственный объект (обычно используется в карточных выборках).
- При помощи метода `byParent` - если класс является коллекцией, с передачей в него `id` или гор родительского объекта.
- При помощи транзакционного индекса по переданному ключу - если класс не является коллекцией, или если нужно получить элементы коллекции не в разрезе предка, а в разрезе иного(-ых) атрибута(-ов) класса.
- При помощи запроса `OQuery`.

Последние три подхода выбирайте на ваше усмотрение, в зависимости от обстоятельств. Полученный в итоге набор гор класса можно финально отфильтровать при помощи методов `filter[Not]`, отсортировать при помощи `orderBy`.

В отличие от реляционного запроса, предыдущие четыре подхода формируют набор данных в формате списка гор определенного класса и содержат только данные этого класса. Если требуется добавить данные, не принадлежащие классу, то:

1. Если нужно получить данные без обработки методами api, тогда можете переопределить метод `onRefreshExt`:

```
override protected def onRefreshExt: String = {
  s"""
  with t as ( select
    :idParent as idParent
    :gid/*@NString*/ as gid
    ....
  )
  SELECT
    t1.sHeadline as idParentHL
    ,.....
  FROM t
  join parent_table t1 on t.idParent = t1.id....
  """
}
```

- Во временной таблице `t` указываем, какие данные забираем из основного набора данных. Этот запрос выполняется для каждой отдельной гор-ы. Далее при помощи обычного sql запроса вытягиваем дополнительные данные.
 - Для `gid` атрибутов указываем аннотацию `/*@NString*/`, что бы парсер корректно передал данные с типом `NGid`.
2. Если нужно получить данные обработанные методами api:
 1. В текущем трейте или его предке создается case class с атрибутами, которые вам требуется добавить на выборки:

```
case class AdditionalInfo(
  var someIdParameter: NLong,
  var someIdParameterHL: NString,
  var someStrParameter: NString,
  var someDateParameter: NDate,
  var someBoolParameter: NNumber,
  var someNumParameter: NNumber
)
```

- Обычно класс называется `AdditionalInfo`, но вы можете выбрать и иное имя.
- `someIdParameter` - ссылочный атрибут, добавляется для полноты данных выборки, *обычно является скрытым* (смотри пояснение свойства `editorType`, настройки `changeableAttr`), пользователю не виден.
- `someIdParameterHL`[или `MC`] - заголовок ссылочного атрибута (`MC` для кодов/мнемокодов), данные ссылочного атрибута, отображаемые пользователю.
- `someStrParameter` - иные строковые данные.
- `someDateParameter` - дата.
- `someBoolParameter` - булевы данные, обычно на выборку передаются в виде данных в формате `NNumber`: 0 - ложное значение, остальные - истина. Используются в атрибутах с редактором *check* (смотри пояснение свойства `editorType`).

- `someNumParameter` - иные числовые данные.

2. После создается метод `getAdditionalInfo(rop)`:

```
def getAdditionalInfo(rop: ***_SomeClassApi#ApiRop): AdditionalInfo = {
  //Получение и обработка данных
  AdditionalInfo(
    someIdParameter = ...
    someIdParameterHL = ...
    .....
    someNumParameter = ...
  )
}
```

3. Дополнительные данные добавляются в кортеж к изначальным данным выборки:

```
override protected def onRefresh: Recs = {
  val rop = ***_SomeClassApi().load(id)
  (rop, getAdditionalInfo(rop))
}
или
override protected def onRefresh: Recs = {
  ***_SomeClassApi().byParent(parentRop)//Или любые другие из трех
  ↪перечисленных методов получения набора данных класса
  .map(rop => (rop, getAdditionalInfo(rop)))
}
```

Создание трейта для получения данных от пользователя

Иногда требуется создать выборку, единственной целью которой является получение данных от пользователя. В таком случае:

1. Метод `onRefresh` выглядит следующим образом:

```
override protected def onRefresh: Recs = {
  "select 1 as id"
}
```

Тк у выборки всегда должен присутствовать атрибут `id` для корректной работы

2. У выборки в авт-разметке *создаем фильтры*, которые будет заполнять пользователи.

После введения данных пользователем существует несколько вариантов их обработки:

- Пользователь закрывает выборку

1. Если введенные пользователем данные не требуется возвращать в метод, вызвавший открытие текущей формы, то на закрытие выборки по согласию пользователя срабатывает метод `closeFormOk()`, соответственно в его переопределении в вашем трейте можно обработать введенные данные.
2. Если данные требуется вернуть в метод, вызвавший открытие текущей формы, то на закрытие выборки переопределяем метод `beforeCloseForm`:

```

override def beforeCloseForm(event: BeforeCloseFormEvent): Unit = {
    event.lookupResultObject = someObjectWithUserData
    super.beforeCloseForm(event)
}

```

- Где `someObjectWithUserData` - некая структура, хранящая пользовательские данные. Может быть List-ом, кортежем, case class-ом и т.д. на ваш выбор.
- Получить такие данные можно в методе, вызвавшем открытие выборки, определив тип возвращаемых данных:

```

val data = Mct_ExampleAvi.myList().newForm().params(Map(...)).
    results(List(attributesName)).openLookup()
val res = data.getResultObject.asInstanceOf[someObjectWithUserData]

```

Примечание

Здесь `someObjectWithUserData` - либо тот же самый тип, если есть возможность типизировать возвращаемое значение тем же классом, либо новая структура, схожая по строению, если, например, в `lookupResultObject` сохранен объект case class-a, недоступного в месте вызова выборки, тогда достаточно типизировать этот объект кортежем(или, например, листом, если в `lookupResultObject` записан лист), у которого вложенные типы следуют в том же порядке, что и у case class-a:

```

saved case class example(NLong, Boolean, List[NString]) -> Tuple3(NLong, Boolean, List[NString])

```

```

saved List[(case class example, NLong, ...)] -> List[(Tuple3(NLong, Boolean, List[NString]), NLong, ...)]

```

- Пользователь вызывает кастомную операцию на текущей выборке:
 - В этом случае требуется создать данную операцию, если ее еще нет, в текущем трейте или его предках, либо же переопределить такую операцию и забирать введенные пользователем данные напрямую из выборки при помощи методов `getVar("...")`/`getSelfVar("...")` и обрабатывать любым удобным вам способом.

Комбинированный подход

- Вы можете использовать комбинацию предыдущих пунктов в любом удобном вам виде - *выводить* какие-либо требуемые вам данные и *получать* от пользователя введенные им данные.

Создание авт-файла для выборки без класса

После того, как вами была *определена* логика работы вашей выборки, определены данные, с которыми выборка будет работать, требуется произвести разметку отображения.

Первичное создание авт-файла

- После того, как вы добавили Avi-файл выборки в определенное место проекта: `имя_модуля/src/main/scala/ru/bitec/app/имя_модуля/...`

Файл с разметкой необходимо добавить в каталог имеющий идентичный путь:

`имя_модуля/src/main/resources/ru/bitec/app/имя_модуля/...`

Внимание

Структуры каталогов, после `../ru/bitec/app/имя_модуля/` должны полностью совпадать

- Имя авт-файла обычно выбирают такое, же как и у файла выборки. Вместо суффикса `Avi` используем суффикс `.avm`, например `Mct_Example.avm.xml`
- После создания файла добавляем туда начальный код:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0" name="Mct_Example">

</view>
```

Свойство `name` тега `view` - имя выборки без суффиксов.

Создание разметки для соответствующего отображения

После создания трейта в Avi-файле выборки создаем тег `representation` с таким же именем:

```
trait MyList extends List
```

```
<representation name="representationName" caption="representationCaption">
  <filter name="globalFilterName">
    <macros name="macrosName">
      <condition id="condName" logicalOperator="logicalOperator" attr="condAttr"
↪operator="condOperator" isExpression="true\false" expression="someSqlExpression">
        <filterAttr attribute-type="fltAttrType" name="fltName" editorType=
↪"fltEditorType" order="fltAttrOrder" caption="fltCaption" defaultValue="defVal"
↪isLastInLine="true\false">
          <editor>
            <...>
          </editor>
        </filterAttr>
      </condition>
    </macros>
    <layout>
      <hBox>
        <vBox>
          <attr name="fltName"/>
          <vGroup>
            <hBox>
              <attr name="fltName"/>
            </hBox>
          </vGroup>
        </vBox>
      </hBox>
    </layout>
  </filter>
</representation>
```

(продолжается на следующей странице)

```

        <vBox>
            <attr name="fltName"/>
            <attr name="fltName"/>
        </vBox>
    </hBox>
</vGroup>
</vBox>
</hBox>
</layout>
</filter>
<layout>
    <composerType>
        <frame filter.isVisible="true\false" toolBar.isVisible="true\false" caption=
↪ "caption" description="description">
            <frameType>
                <layout>
                    <...>
                </layout>
            </frameType>
        </frame>
        <composerSettings/>
    </composerType>
</layout>
<bandGroups>
    <bandGroup name="bandGroupName" caption="Группа"/>
</bandGroups>
<attributes>
    <attr name="attrName" order="attrOrder" caption="attrCaption" editorType=
↪ "attrEditorType" isVisible="true\false" bandGroup="attrBandGroupName" isReadOnly="true\
↪ false" isRequired="true\false" isLastInLine="true\false"/>
</attributes>
<operations>
    <oper name="operName" order="operOrder" caption="operCaption" isActive="true\
↪ false"/>
</operations>
</representation>

```

Разберем код разметки:

Основной тег representation

- Содержит два основных свойства:

`name`

`name` = «`representationName`» - где `representationName` - имя отображения, такое же как у трейта созданного вами ранее, например *MyList*

`caption`

`caption` = «`representationCaption`» - где `representationCaption` описание отображения, которое увидит пользователь в шапке отображения.

- Внутри тега **representation** содержатся несколько тегов -
 - Тег *filter* .
 - Тег *layout* .

Тег filter

- Содержит одно основное свойство **name** = «`globalFilterName`» - Системное имя фильтра.
- Внутри тега **filter** присутствуют два тега разметки -
 - Тег *macros*
 - Тег *layout*

Тег macros

Обязательный

- Содержит одно свойство: **name** = «`macrosName`» , например `DefFltReferenceMacro`, нужен системе, для дальнейшей подстановки данных из фильтра при обработке *selectStatement-запроса*
- Внутри можно определить несколько тегов **condition**

Тег condition

- Содержит несколько свойств:
 1. **id** - системное имя тега, используется в обработке метаданных выборки. Обязателен для заполнения. Должен быть уникальным в разрезе макроса.

Обязателен

2. **logicalOperator** - логический оператор(`and\or`). С помощью указанного оператора текущий **condition** будет добавлен к остальным условиям в **selectStatement-запроса**.

Обязателен

3. **isExpression** = «`true\false`» - свойство тега, определяющее каким образом будет обработан фильтр.

Обязателен

Так же от его значения зависит обязательность указания дальнейших свойств фильтра -

- Если `true`:

* **expression** - условие. **Становится обязательным.**

По сути представляет из себя фрагмент sql условия, по которому происходит фильтрация:

```
(:filter$flt_bShowNotUsed = 1 or  
  (coalesce(t.bnotactive,0) = 0  
    and (t.dexpirydate is null or t.dexpirydate > current_date)  
  )  
)
```

```
1 = :filter$flt_bShowNotUsed
```

Внимание

Условие обязано быть заключено в скобки, если оно состоит из более чем одного логического элемента, иначе фильтрация итогового набора данных будет некорректна!

– Если false:

Тогда заполняем следующие, **ставшие обязательными**, атрибуты -

1. **attr** = «condAttr» - атрибут, по которому происходит фильтрация.

Если в *selectStatement-запросе* присутствуют несколько таблиц, тогда в качестве condAttr можно указать любой атрибут этих таблиц, используя соответствующий псевдоним. Например:

```
attr="ot.someAttribute"
```

2. **operator** = «condOperator» - оператор условия стандартного фильтра. Такой как:

```
operator`="equals"/"not equals"/"less"/"greater"/"greater or equals"/  
↪ "less or equals"/"in"/"between"/"like"/"is null"
```

По этому значению соответствующим образом будет проведено сравнение **attr** и значения, переданного из фильтра.

- Внутри тега может содержаться несколько тегов **filterAttr**

Ter filterAttr

- Содержит несколько свойств:

1. **name** - системное имя фильтра. Позволяет получить значение фильтра как в scala-коде выборки, так и внутри sql-запросов, вызываемых из scala-кода выборок. Например:

```
...  
val bVal = getVar("flt_bShowNotUsed").asNNumber.toBoolean  
...
```

```
s"""
    select
        ...
    from ...
    where ...
    logicalOperator (:filter$flt_bShowNotUsed = 1 or
        (coalesce(t.bnotactive,0) = 0
            and (t.dexpirydate is null or t.dexpirydate > current_date)
        )
    )
"""
```

2. **attribute-type** - тип атрибута стандартного фильтра:

- Varchar
- Number
- Long
- Date

3. **editorType** - определяет тип редактора, который будет отображаться при редактировании ячейки списка, дерева или отображаться в карточке:

- edit - Редактор в строке
- editButton - Редактор в строке с кнопкой
- currency - Денежный редактор
- lookup - Выпадающий список по запросу
- combo - Фиксированный выпадающий список
- check - Чекбокс
- datePicker - Редактор даты
- dateTimePick - Редактор даты и времени
- timePick - Редактор времени
- memo - Мемо - поле
- icon - Изображение
- calendar - Календарь
- buttonsEdit - Редактор в строке с произвольными кнопками
- imageCollection - Список картинок
- tagLookup - Выпадающий список с мультिवыбором
- editPassword - Редактор пароля в строке
- button - Кнопка
- Если **editorType** = **buttonsEdit**, тогда внутри этого тега надо указать еще два тега **editor** и **ref**:

```

<editor>
  <buttonsEdit canEditText="true\false" changeableAttr="someAttr">
    <buttons lookup="true\false" openCard="true\false" reset="true\false
↵"/>
  </buttonsEdit>
</editor>
<ref class="someClass"/>

```

Где:

- * **canEditText** - по умолчанию, в поля ввода типа «Редактор кнопкой» и «Редактор с произвольными кнопками» вводить текст запрещено. Установив данное свойство в True, возможно разрешить ввод текста в поле ввода.
 - * **changeableAttr** - свойство содержит имя атрибута, в который будет записан идентификатор ссылочного объекта, при редактировании ссылочного поля. Каждому ссылочному атрибуту класса соответствуют 2 атрибута выборки: значимый и отображаемый. Первый содержит идентификатор, а второй заголовок ссылочного объекта, который понятен пользователю. Значимый атрибут обычно является скрытым, а пользователь редактирует отображаемый атрибут, при этом необходимо, что бы редактор отображаемого атрибута знал имя значимого атрибута.
 - * **class** - свойство содержит системное имя класса, на который ссылается данный атрибут. Это свойство используется фильтром, для открытия формы выбора значения ссылочного атрибута, при формировании условия фильтрации.
4. **order** - порядковый номер. С его помощью можно настроить положение фильтров относительно друг друга.
 5. **isLastInLine** - свойство, отвечающее за перенос последующих фильтров на новую строку.
 6. **caption** - видимое имя фильтра.
 7. **defaultValue** - умолчательное значение фильтра.

Ter filter layout

Необязательный тег. Служит для более детальной настройки расположения фильтров на выборке.

- На первом уровне, внутри тега **layout** должны располагаться следующие теги:
 - **vSection** - секция фильтров
 - **vGroup** - группа фильтров
 - **vBox** - вертикальный контейнер
 - **hBox** - горизонтальный контейнер
- На втором и последующих уровнях вложенности могут быть расположены те же теги, а так же тег **attr**, у которого есть два базовых свойства:
 - **name** - системное имя *фильтра*.
 - **caption** - отображаемое имя фильтра, перекрывает видимое имя фильтра, указанное в теге *filterAttr*.
- Комбинируя данные теги, вы можете гибко настроить фильтры выборки.

Ter representation layout

- Содержит внутри себя один тег:
 - `simpleComposer` - отображается только эта выборка, без подчиненных.
 - `tabComposer` - подчиненные выборки будут отображаться закладками.

Вложенный тег:

```
<tabItems isVisible="false">
  <tabItem selection="gtk-Mct_ExampleAvi" representation="MyList" caption=
↪ "Имя закладки, которое увидит пользователь"/>
  <tabItem.../>
  ...
</tabItems>
```

- `dynamicComposer` - подчиненные выборки будут отображаться как отдельная часть формы.

Вложенный тег:

```
<dynamicItems masterAlign="top/bottom/left/right/client">
  <dynamicItem selection="gtk-Mct_ExampleAvi" representation="MyList" align="
↪ .."/>
</dynamicItems>
```

- * `masterAlign` - выравнивание главного фрейма. `client` - фрейм занимает всё доступное пространство компоновщика, не занятое фреймами с остальными типами выравнивания.
- * `align` - выравнивание детальной выборки.
- `tabDynamicComposer` - сочетает в себе функционал `tabComposer` и `dynamicComposer`.
- `tabDynDetComposer` - подчиненная выборка будет определяться автоматически, в зависимости от параметров этой выборки. Так же включает в себя функционал `tabComposer`.
- Внутри любого из перечисленных тегов находится один обязательный тег:

* `frame`

Основные свойства:

- `filter.isVisible` - будет ли видна панель фильтров на выборке.
- `toolBar.isVisible` - будет ли виден тулбар с операциями на выборке.
- `caption` - наименование фрейма. Отображается в первой строке заголовка фрейма или в заголовке формы, если фрейм является главным на форме
- `description` - описание. Отображается второй строкой на расширенном заголовке фрейма.

В этом теге, в свою очередь, устанавливает тег с указанием типа фрейма:

- `grid` - данные отображаются списком.
- `tree` - данные отображаются в виде дерева.
- `card` - данные отображаются карточкой.

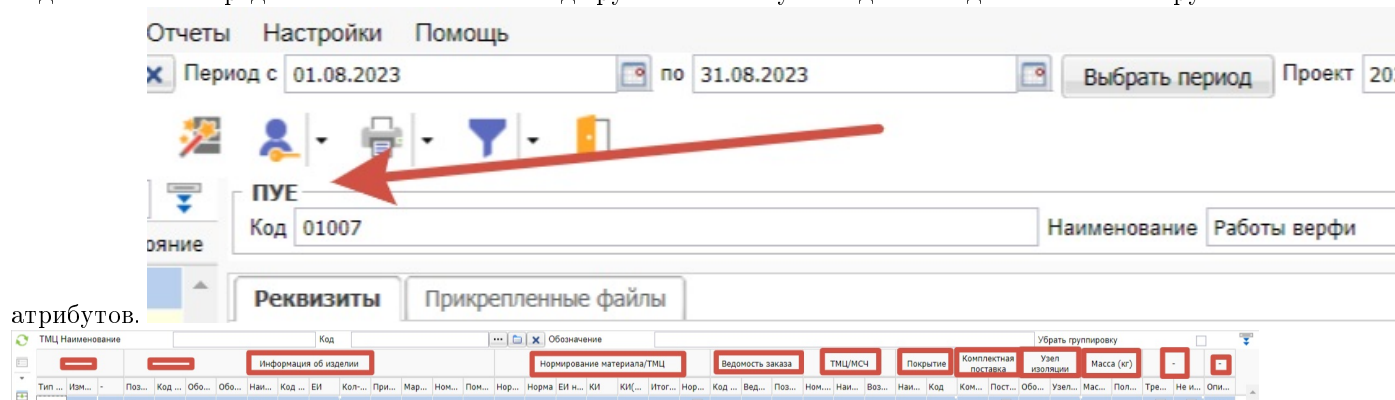
В случае выбора карточного типа, внутри текущего тега можно установить тег `layout`, полностью аналогичный тегу `layout` фильтра.

Ter representation bandGroups

Нужен для определения банд-групп. Содержит в себе набор тегов `bandGroup`:

```
<bandGroup name="bandName" caption="Имя банд-группы"/>
```

В дальнейшем определенные в этом теге банд-группы используются для объединения в этих группах



Ter representation attributes

Содержит в себе набор тегов `attr` со следующими свойствами:

- **name** - системное имя атрибута.
- **order** - свойство управляет порядковым номером колонки в списке/дереве и порядковым номером поля ввода в карточке.
- **caption** - Наименование атрибута.
- **editorType** - аналогичен *editorType* фильтра.
- **bandGroup** - свойство определяет группу объединения, в которую входит атрибут.
- **isVisible** - свойство видимости атрибута.
- **isReadOnly** - свойство редактируемости атрибута.
- **isRequired** - свойство обязательности заполнения атрибута.
- **isLastInLine** - аналогичен *isLastInLine* фильтра.

Ter representation operations

Нужен для первичной настройки операций, отображаемых в виде кнопок на тулбаре и в контекстном меню. В основном используется для того, что бы скрыть или отключить нужную операцию.

Содержит в себе набор тегов `oper` со следующими базовыми свойствами:

- **name** - системное имя операции. Имя должно совпадать с именем scala-метода (без учёта регистра) в Rep-классе, соответствующем отображению выборки.
- **order** - порядковый номер операции. Свойство управляет положением операции на панели управления, в меню, и прочих контролах управления. По умолчанию равно 0. Если значения совпадают, порядок операций будет соответствовать порядку в XML.
- **caption** - наименование операции. Отображается в меню и подменю.

- `isActive` - флаг управляет активностью операции. Если флаг снят, операция не загружается в клиент и, соответственно, не отображается на панели управления, в меню и т.д.

Проектное переопределение

Предисловие

Стоит отдельно указать, что помимо проектного переопределения существует так же опция наследования базовых файлов в проектных модулях. Для того что бы выбрать нужный подход, следует понимать что требуется сделать:

- Переопределить базовую логику так, что бы изменилось поведение этой логики в любом месте системы.

Например, если вы хотите во всей системе изменить выборку `Bs_GoodsAvi#Card`, добавить туда проектные атрибуты, изменить разметку выборки итд, и при этом вы хотите видеть эти изменения абсолютно во всех местах где вызывается эта выборка(аналогично с переопределением `api` и иных файлов).

Проектное переопределение

- Дополнить базовую логику работы и использовать её локально, в определенных ситуациях.

Например, если вы хотите дополнить вкладку `Bs_GoodsAvi#Card` проектными атрибутами, изменить её разметку итд, при этом измененную вкладку вы хотите видеть в каком то конкретном отображении, а в остальной системе оставить вид этой выборки базовым(аналогично с переопределением `api` и иных файлов).

Наследование базовой логики

Наследование базовой логики

Обычное наследование функционала. Реализуется с помощью ключевого слова `extends`. Определимся только с установленными практиками наименования и расположения наследующих файлов. В остальном работают обычные правила наследования. Рассмотрим пример наследования от `Rpl_QueryAvi`, и создание нового трейта с измененной логикой.

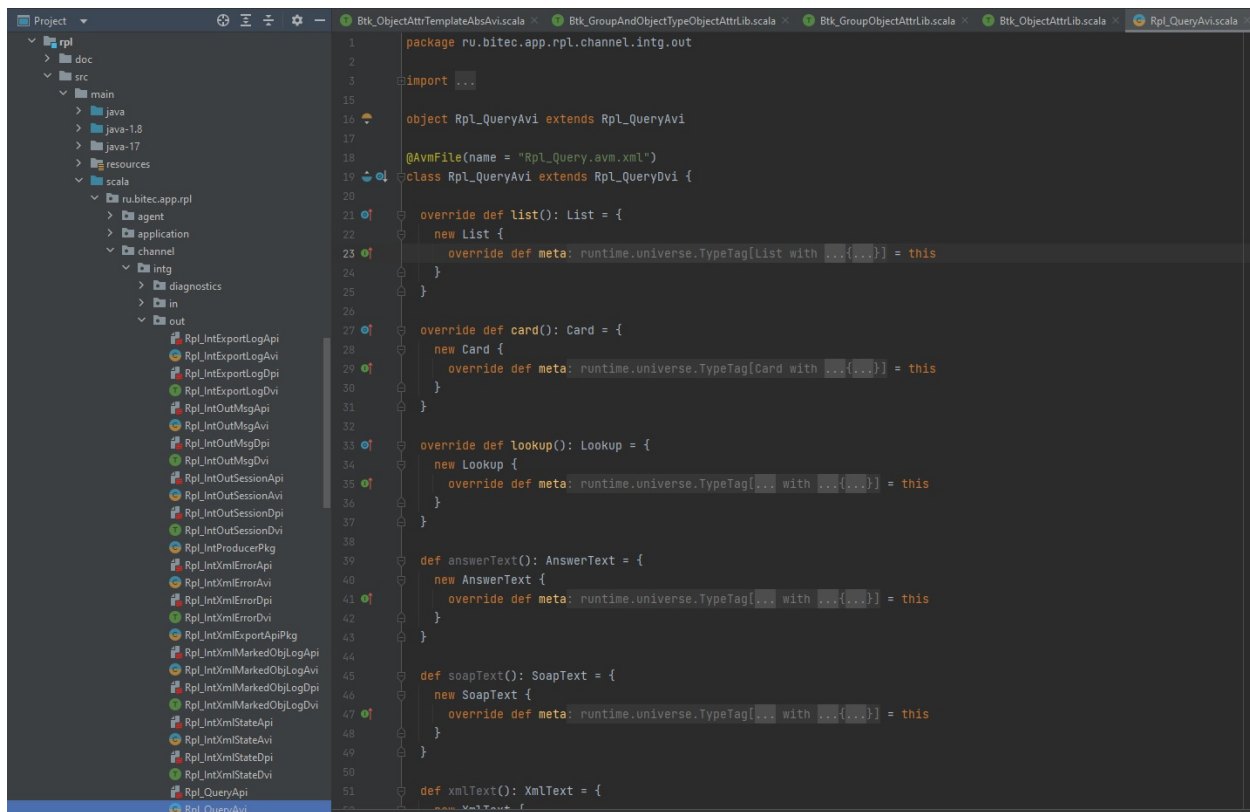
Расположение наследующего файла

Наследующий файл в вашем проекте должен располагаться в соответствующем модуле для проектных переопределений. Так же желательно повторять структуру директорий базового модуля. Так, если `Rpl_QueryAvi` расположен по следующему пути:

```
ru.bitec.app.rpl.channel.intg.out.Rpl_QueryAvi,
```

то и наследующий его файл рекомендуется располагать в

```
ru.bitec.app.<Имя проектного модуля переопределений>.channel.intg.out.<Имя наследующего
↪файла>
```



Наименование наследующего файла

При наследовании базового файла в ваш имя наследующего файла рекомендуется формировать следующим образом:

<Имя проектного модуля переопределений>_<Имя базового класса БЕЗ имени базового модуля>
 Так например, для Rpl_QueryAvi имя наследующего файла будет выглядеть так - <Имя проектного модуля переопределений>_QueryAvi.

Пример наследующего файла

```
package ru.bitec.app.<Имя проектного модуля переопределений>.channel.intg.out

import ru.bitec.app.rpl.channel.intg.out.Rpl_QueryAvi
import ru.bitec.gtk.core.AvmFile

@AvmFile(name = "Файл разметки.avm.xml")
class <Имя проектного модуля переопределений>_QueryAvi extends Rpl_QueryAvi{

  override def card(): Card = {
    new Card {
      override def meta = this
    }
  }
}
```

(продолжается на следующей странице)

```
trait Card extends Default with super.Card {
    ...
}

}
```

В случае наследования вы получаете возможность дополнить базовую логику, при этом не изменив поведение соответствующих выборов ранее созданных и выведенных в вашей системе.

Проектные переопределения

Рассмотрим проектное переопределение на примере класса. Проектное переопределение *выборки без класса*, pkg и lib файлов происходит аналогично.

Проектное переопределение файлов окружения класса

- Часто возникает необходимость переопределить или дополнить функционал определенного класса, в таком случае переопределяются нужные элементы *окружения класса*. Переопределяются обычно Avi, Api и avm.xml файлы окружения класса.

Переопределение Api

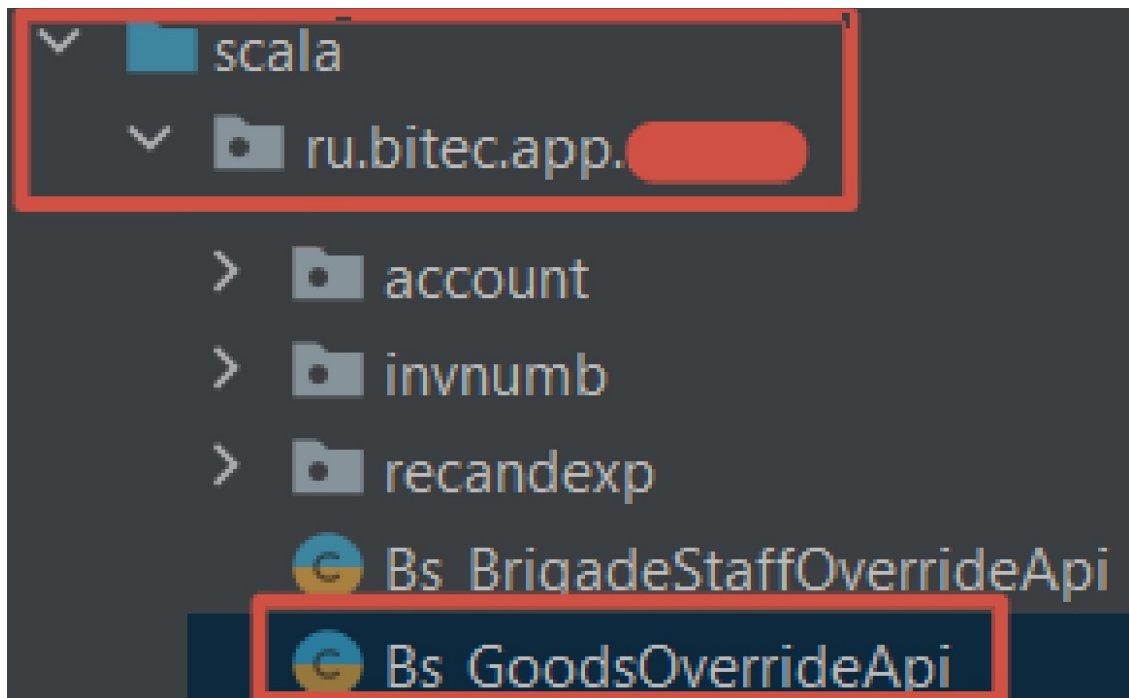
1. Внутри проектного модуля `.../src/main/scala/ru.bites.app.проектный_модуль`. <Если файл вложен в поддиректории, то нужно повторить их структуру> создайте api-файл.
 - Имя файла задавайте по следующему шаблону <Имя Класа С ИМЕНЕМ БАЗОВОГО МОДУЛЯ ЭТОГО ДЛЯ КЛАССА>OverrideApi.

Примечание

Обычно имена классов и файлов получаются с использованием имени модуля и имени класса `***_ИмяКласа`, однако для переопределяемых файлов не рекомендуется использовать такой подход. Лучше модифицировать имя базового файла. Например: `Bs_GoodsApi` -> `Bs_GoodsOverrideApi`.

Внимание

ВАЖНО! При составлении имени файла НЕ используйте имя текущего модуля! В имя файла добавляется только суффикс `Override!!!`



2. Содержание файла должно быть следующим:

```
package ru.bitec.app.<имя проектного модуля>.bs

import ...

class Bs_GoodsOverrideApi extends Bs_GoodsApi {
    Код класса
}

object Bs_GoodsOverrideApi extends ApiFactory[
    java.lang.Long,
    Bs_GoodsAro,
    Bs_GoodsOverrideApi
]
```

3. Внутри тела класса можете переопределять существующие методы и добавлять новые.

Переопределение Avi

1. В том же пакете что и *api*: `.../src/main/scala/ru.bitec.app.проектный_модуль/bs`. <Если файл вложен в поддиректории, то нужно повторить их структуру>, создайте Avi-файл:

- Имя файла получаем аналогично имени api-файла.

Например: `Bs_GoodsApi` -> `Bs_GoodsOverrideApi`.

```
package ru.bitec.app.<имя проектного модуля>.bs

import ...
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
object Bs_GoodsOverrideAvi extends Bs_GoodsOverrideAvi

@AvmFile(name = "Bs_GoodsOverride.avm.xml")
class Bs_GoodsOverrideAvi extends Bs_GoodsAvi {
    Код выборок
}
```

Внимание

ВАЖНО! При составлении имени файла НЕ используйте имя текущего модуля! В имя файла добавляется только суффикс `Override!!!`

- Общая структура файла аналогично обычному Avi-файлу.
 - Объект-компаньон должен иметь имя как у переопределяемого файла.
 - Аннотация `@AvmFile(name = "Bs_GoodsOverride.avm.xml")` не обязательна. Если ее не указывать, то будет взята настройка базового файла.
2. Внутри тела выборки можете переопределять существующие трейты и их методы, а так же создавать новые трейты и методы (в т.ч. внутри переопределенных трейтов).

Переопределение Avm

1. Рекомендуется создавать avm-файл в директории `.../src/main/resources/ru.bitec.app.проектный_модуль`. <Если файл вложен в поддиректории, то нужно повторить их структуру>, и уже в нем создавайте файл с разметкой.

- Таким образом структуры каталогов в разных ветках будут идентичны.

Например:

```
.../src/main/scala/ru.bitec.app.проектный_модуль/bs для проектной логики и
.../src/main/resources/ru.bitec.app.проектный_модуль/bs для проектной разметки.
```

1. Базовая структура полностью аналогична обычным avm-файлам.

- В теге `view` указываем имя класса и дополнительное свойство `th:extends`:

```
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.global-system.ru/xsd/global3-view-1.0"
xmlns:th="http://www.global-system.ru/xsd/global3-view-template-1.0"
name="Bs_GoodsOverride" th:extends="Bs_Goods.avm.xml">
```

- `name` - имя переопределенного файла, получаем аналогично прошлым пунктам.

Внимание

ВАЖНО! При составлении имени файла НЕ используйте имя текущего модуля! В имя файла добавляется только суффикс `Override!!!`

- `th:extends` - базовый `avm`-файл, нужен для того, что бы в переопределенном файле указывать только проектную разметку. Остальные настройки будут вытягиваться из базового файла.
2. Если вы переопределили `Avi`-файл и создали там новый трейт, или переопределили существующий трейт `Avi`-файла, изменив возвращаемые данные(добавили новые атрибуты выборки, переименовали старые итд) тогда в `avm` нужно создать тег `representation` с именем, соответствующим имени этого трейта, и настроить разметку.
 3. Так же, если вам требуется проектно переопределить только разметку, нужно помимо этого переопределить `Avi`-файл и указать переопределенный файл разметки в аннотации.

Внимание

Включение переопределенных файлов разметки отличается от включения переопределенной логики `Api` и `Avi` !!! Что бы подключить переопределенную разметку, нужно в соответствующем `Avi`-файле для класса использовать аннотацию `@AvmFile(name = «Bs_GoodsOverride.avm.xml»)`

Включение проектной логики

- После того, как вы создали необходимые вам переопределения проектной логики, требуется связать переопределяемый и переопределяющий файлы.
- Для этого в проектном модуле, в каталоге `/src/main/resources/META-INF` откройте или создайте файл `overrides.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<overrides>
  <replace-with class="ru.bitec.app.<Имя проектного модуля>.<Имя модуля, которому
  ↳ принадлежит переопределяемый файл>.<Имя переопределенного файла>">
    <when-type-is class="ru.bitec.app.<Имя модуля, которому принадлежит
  ↳ переопределяемый файл>.<Имя переопределяемого файла>" />
  </replace-with>
</overrides>
```

- В теге `replace-with` укажите свойство `class` - относительный путь до переопределенного файла начиная с каталога `ru.bitec.app`.
- В теге `when-type-is`, вложенном в предыдущий тег, укажите относительный путь до базового файла, так же начиная с каталога `ru.bitec.app`.
- Так например, если вы переопределили `Api` и `Avi`:

```
<?xml version="1.0" encoding="UTF-8"?>
<overrides>
  ....
  <replace-with class="ru.bitec.app.<ProjectModule>.bs.Bs_GoodsOverrideAvi">
    <when-type-is class="ru.bitec.app.gds.Bs_GoodsAvi" />
  </replace-with>
  <replace-with class="ru.bitec.app.<ProjectModule>.bs.Bs_GoodsOverrideApi">
    <when-type-is class="ru.bitec.app.gds.Bs_GoodsApi" />
  </replace-with>
```

(продолжается на следующей странице)

```
....
</overrides>
```

Примечание

Так же стоит заметить, что если вы переопределили какой-либо api/avi-файл, и в нем **переопределили** метод(ы) базового api/avi, тогда в другой части кода вы можете вызывать эти методы используя имя базового файла, ваша проектная логика будет подхватываться системой и вызываться от имени базового файла. Однако, если вы добавили новый метод, который отсутствует в базовом файле, вызывать его придется, используя имя переопределенного файла.

Работа с данными, хранящимися в jsonb контейнере

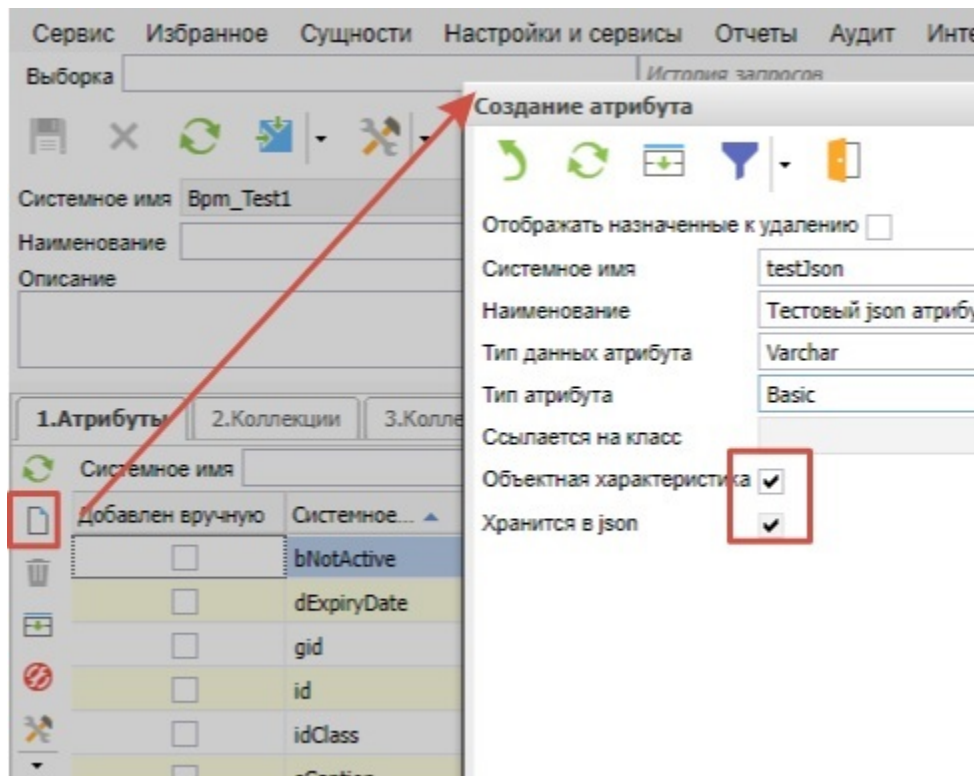
Атрибуты, хранящиеся в jsonb контейнере

Предисловие

В нашей системе есть два типа атрибутов:

- Базовые, определяемые в odm-файле
- Json-хранимые - определяемые как через интерфейс приложения(в том числе универсальные характеристики), так и через исходный код

В самом простом случае при создании json-атрибута в интерфейсе приложения он будет отмечен как объектная характеристика, хранящаяся в json.



Редактировать такие атрибуты можно на вкладке объектных характеристик соответствующего класса. Обработка этих атрибутов уже реализована средствами фреймворка.

Внимание

В базе для хранения данных используется контейнер типа jsonb

Получение и установка атрибутов, хранимых в json

Получение и обработка данных методами api текущего класса для объектных характеристик

Примечание

Все json-хранимые объектные характеристики и универсальные характеристики хранятся в контейнере с именем `jObjAttrs_dz`. И все перечисленные далее методы чтения и записи данных для атрибутов-характеристик работают именно с этим контейнером.

Для атрибута, хранимого в json и настроенного как объектная характеристика в `JObjectAttrApi.scala`, реализован набор методов для записи и чтения данных из контейнера:

Запись по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setObjAttrValue(rop: ApiRop,   
↳ idpAttr: NLong, pValue: Any)
```

- Метод установки значения json атрибута по его id
- Вызвать этот метод можно от api вашего класса
- `rop` - rop-a, чей атрибут вы хотите изменить
- `idpAttr` - id атрибута
- `pValue` - значение, которое хотите установить атрибуту

Метод проверяет, что переданное значение совпадает с типом данных, настроенным на атрибуте.

Запись по имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setObjAttrValue(rop: ApiRop,   
↳ spAttrName: NString, pValue: Any)
```

- Метод установки значения json атрибута по его системному имени
- Вызвать этот метод можно от api вашего класса
- `rop` - rop-a, чей атрибут вы хотите изменить
- `spAttrName` - системное имя атрибута

- `pValue` - значение, которое хотите установить атрибуту

Метод проверяет наличие настроенного атрибута в классе `Btk_Attribute`, если атрибут не найден, то кидает ошибку.

После успешного прохождения проверки вызывается метод `def setObjAttrValue(rop: ApiRop, idpAttr: NLong, pValue: Any)`.

Внимание

При передаче значения для устанавливаемого атрибута необходимо передавать данные, приведенные к соответствующему типу!

Чтение по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getObjAttrValue(rop: ApiRop, idpAttr: NLong)
```

- Метод получения значения json атрибута по его id
- Так же, как и предыдущие, проверяет наличие атрибута в `Btk_Attribute`
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-a, чей атрибут вы хотите получить
- `idpAttr` - id атрибута

Чтение по системному имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getObjAttrValue(rop: ApiRop, spAttrName: NString)
```

- Метод получения значения json атрибута по его системному имени
- Так же, как и предыдущие, проверяет наличие атрибута в `Btk_Attribute`
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-a, чей атрибут вы хотите получить
- `idpAttr` - id атрибута

Метод проверяет наличие настроенного атрибута в классе `Btk_Attribute`, если атрибут не найден, то кидает ошибку.

После успешного прохождения проверки вызывается метод `def getObjAttrValue(rop: ApiRop, idpAttr: NLong)`.

Универсальные характеристики(UC)

Универсальные характеристики являются отдельным сервисом, позволяющим подключить к объектам класса дополнительные атрибуты и не требующим компиляции исходного кода.

Универсальные характеристики хранятся в справочнике `Btk_UniversalCharacteristic`.

При подключении универсальной характеристики в качестве атрибута к целевому классу итоговые данные по этому атрибуту-характеристике записываются в контейнер `jObjAttrs_dz`, туда же, где хранятся значения json объектных характеристик.

Получение и обработка данных методами api текущего класса для универсальных характеристик

Запись UC по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setUniCharValue(rop: ApiRop, ┐  
↳ idpUniChar: NLong, pValue: Any)
```

- Метод установки значения UC json атрибута по его id
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-a, чей атрибут вы хотите изменить
- `idpUniChar` - id UC атрибута
- `pValue` - значение, которое хотите установить атрибуту. Возможна множественная установка значений

Метод проверяет, что переданное значение совпадает с типом данных, настроенным на атрибуте.

Запись по имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setUniCharValue(rop: ApiRop, ┐  
↳ spUniCharName: NString, pValue: Any)
```

Аналогичен предыдущему. Предварительно ищет id универсальной характеристики по системному имени. После вызывает метод вызывается метод `def setUniCharValue(rop: ApiRop, idpUniChar: NLong, pValue: Any)`. Возможна множественная установка значений.

Чтение UC по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getUniCharValue(rop: ApiRop, ┐  
↳ idpUniChar: NLong)
```

- Метод получения значения json атрибута по его id
- Так же как и предыдущие проверяет наличие атрибута в `Btk_Attribute`
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-a, чей атрибут вы хотите получить
- `idpUniChar` - id UC атрибута

Чтение UC по системному имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getUniCharValue(rop: ApiRop,   
↳ spUniCharName: NString)
```

Проверяет наличие универсальной характеристики по системному имени. После вызывает метод *def getUniCharValue(rop: ApiRop, idpUniChar: NLong)*.

Чтение и запись иных данных в json контейнер

Если вам нужно хранить иные структуры данных, не подходящие под прошлые пункты, тогда:

- Для получения доступа к данным в контейнере используйте метод подключения json документа `ru.bitec.app.gtk.eclipse.json.JEmbeddedDoc#parseProperty` с последующей конвертацией к `JObject` и метод `ru.bitec.app.gtk.eclipse.json.JEObject#set`
- Для добавления или перезаписи значения по ключу:

```
JEmbeddedDoc.parseProperty(rop, "Имя json контейнера").asJObject.set(key: NString,   
↳ value: NString)
```

Внимание

Если попытаться напрямую установить значение в атрибут, можно затереть данные из контейнера, которые там присутствовали ранее!

Примечания по работе с jsonb контейнерами

- Все строковые значения необходимо заворачивать в двойные кавычки "text"
- Если вы пытаетесь поставить в нестроковое поле пустое значение, то используйте значение „null“
- Оператор `->` возвращает элемент типа `jsonb`. Оператор `->>` возвращает элемент типа `text`. Для промежуточных манипуляций с объектами контейнера используйте `->`, а для получения конечного значения `->>`
- При добавлении значений при помощи `||` или `jsonb_set()` к уже существующему набору (например `jtypesizeattrs`) пар ключ-значение необходимо оборачивать его в `coalesce(jtypesizeattrs, '{}':: jsonb)`, так как если `jtypesizeattrs` до этого был `null`, то прибавляя к нему любое значение, мы получим `null`
- При добавлении значений при помощи `||` или `jsonb_set()` к уже существующему массиву (например `jSign_dz`) необходимо оборачивать его в `coalesce(jSign_dz, '[]':: jsonb)` так как если `jSign_dz` до этого был `null`, то, прибавляя к нему любое значение, мы получим `null`

Работа с контейнером jsonObjAttrs_dz

В контейнере jsonObjAttrs_dz данные хранятся в виде карты, соответственно сразу можно использовать метод -> для получения конечных данных из этой карты.

Например:

- Получение json-атрибута из контейнера:

```
select
  w.jsonObjAttrs_dz -> 'idJsonAttr' as idJsonAttr,
  cast(w.jsonObjAttrs_dz -> 'bJsonAttr' as numeric(1)) as bJsonAttr,
  cast(w.jsonObjAttrs_dz -> 'nJsonAttr' as numeric(38, 18)) as nJsonAttr,
  w.jsonObjAttrs_dz -> 'sJsonAttr' as sJsonAttr,
  cast(w.jsonObjAttrs_dz -> 'dJsonAttr' as date) as dJsonAttr,
  cast(w.jsonObjAttrs_dz -> 'dJsonAttr' as timestamp) as dJsonAttrDateTime
from (
  select '{"idJsonAttr": 123456, "bJsonAttr": 0, "nJsonAttr": 0.01, "sJsonAttr":
↪ "Some test string", "dJsonAttr": "01.01.2101 10:10:10"}' :: jsonb as jsonObjAttrs_
↪ dz) w
return: |idJsonAttr|bJsonAttr|          nJsonAttr|          sJsonAttr|  dJsonAttr|
↪         dJsonAttrDateTime|
↪         |      123456|          0|0.010000000000000000|"Some test string"|"2101-01-01"|
↪         |"2101-01-01 10:10:10.000"|
```

Работа с обобщенными json контейнерами в Postgres

Например:

- Получение поля из массива

```
select (w.jsign_dz -> 1) -> 'sFIO' -- 1 - индекс массива. Отсчет ведётся с 0. sFIO
↪ - имя искомого поля
from (select
  '['
    {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":
↪ null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
    {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪ "sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪ ": 95402}
  ]' :: jsonb as jsign_dz) w
return: "Линчук Владимир Владимирович"
```

- Аудејтим поле в массиве записей. В этом случае используем метод jsonb_set. Аргументы - первоначальное значение, путь к изменяемому значению, устанавливаемое значение

```
update stk_warrantin w
set jsign_dz = jsonb_set(coalesce(w.jsign_dz, '[]':: jsonb), '{1,sFIO}', '"Иванов
↪ Ива Иванович"') -- {1,sFIO} - путь к интересующему полю 1 - индекс в массиве.
↪ sFIO - поле в массиве.
where id = someId
```

```

select jsonb_set(coalesce(w.jsign_dz, '[]':: jsonb), '{1,sFIO}', '"Иванов Ива
↪Иванович"')
from (select
    '[
        {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
        {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
    ]' :: jsonb as jsign_dz) w

return: [
    {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
    {"sFIO": "Иванов Ива Иванович", "dDate": "04.12.2019", "sPosition":
↪null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType": 95402}
]

```

- Удаляем запись из результирующего массива (пару ключ - значение из объекта)

```

select w.jsign_dz - 1
from (select
    '[
        {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
        {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
    ]' :: jsonb as jsign_dz) w

return: [
    {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401}
]

```

- Удаляем поле из карты

```

select (w.jsign_dz -> 0) - 'sFIO'
from (select
    '[
        {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
        {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
    ]' :: jsonb as jsign_dz) w

return: {"sPosition": "Нач. отдела", "idEmployee": null, "idDepartment": 102696,
↪"sBasisDocument": null, "idBlankSignType": 95401}

```

- Добавляем запись в массив. В этом случае работаем с объединением двух массивов. Необходимо задать строку с новым объектом массива и добавить её с помощью операции || к старому значению

```

select coalesce(w.jSign_dz, '[]'::jsonb) ||
    '[
        {"sFI0": "Петров Е.В.", "sPosition": "тест", "idEmployee": null,
↪ "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401}
    ]'::jsonb
from (select
    '[
        {"sFI0": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee": ↪
↪ null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
        {"sFI0": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪ "sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪ ": 95402}
    ]'::jsonb as jsign_dz) w

return:
[
    {"sFI0": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee": null,
↪ "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
    {"sFI0": "Линчук Владимир Владимирович", "dDate": "04.12.2019", "sPosition
↪ ": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType": 95402},
    {"sFI0": "Петров Е.В.", "sPosition": "тест", "idEmployee": null,
↪ "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401}
]

```

- Можно полностью перезаписать контейнер jSign_dz. Можно сформировать строку при помощи обычных строковых методов и полностью перезаписать атрибут jSign_dz, используя приведения строки к типу jsonb и соблюдая синтаксис jsonb

```

update stk_warrantin w
set jsign_dz = ('[{"sFI0": '' || 'тест' || '', "sPosition": "тест", "idEmployee": ↪
↪ null, "idDepartment": ' || '1' || ', "sBasisDocument": null, "idBlankSignType": ↪
↪ 95401}]')::jsonb
where id = someId

```

- Разворачиваем массив подписей в записи

```

select value ->> 'sFI0' as sFI0
from jsonb_array_elements(
    '[
        {"sFI0": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee": null,
↪ "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
        {"sFI0": "Линчук Владимир Владимирович", "dDate": "04.12.2019", "sPosition
↪ ": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType": 95402}
    ]'::jsonb) v

return:
1: `Степанова Е.В.`
2: `Линчук Владимир Владимирович`

```

Индексный доступ в jsonb объектах

Postgresql позволяет индексировать поля с типом `jsonb` в двух вариантах:

1. Индексация существования ключа на верхнем уровне объекта. Поддерживает операторы `?`, `?&` и `?|` Пример индекса:

```
CREATE INDEX idxSomeName ON SomeTable USING GIN (jsonData)
```

2. Индексация существования пары путь/значение. Поддерживает оператор `@>`

```
CREATE INDEX idxSomeName ON SomeTable USING GIN (jsonData jsonb_path_ops)
```

Поле `jObjectAttrs_dz`, в котором хранятся значения `json`-атрибутов и универсальных характеристик, по умолчанию использует второй тип индекса.

Исследование производительности запросов

Характеристики теста

1. База данных:
 - PostgreSQL 16.5 (Debian 16.5-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
 - 64 Gb Ram, CPU: 3.5Ghz x 8, SSD
2. Таблица содержит индексируемое поле с типом `jsonB`. Тип индекса `gin (jsonb_path_ops)`
3. Количество строк в таблице: 80 млн.
4. Каждая строка содержит `jsonb`-объект, в котором 200 ключей:
 - 50 ключей, которые имеют 10 возможных значений (имена начинаются на `attr_10_vals_`)
 - 50 ключей, которые имеют 2 возможных значения (0 или 1) (имена начинаются на `attr_bool_vals_`)
 - 50 ключей, которые имеют 100 возможных значений (имена начинаются на `attr_100_vals_`)
 - 50 ключей, которые имеют уникальные значения (имена начинаются на `attr_1_vals_`)
5. Данные формировались со сдвигом на каждой формируемой строке, для получения разреженных данных.
 - Всего возможных ключей: 1600
 - 400 ключей, которые имеют 10 возможных значений (имена начинаются на `attr_10_vals_`)
 - 400 ключей, которые имеют 2 возможных значения (0 или 1) (имена начинаются на `attr_bool_vals_`)
 - 400 ключей, которые имеют 100 возможных значений (имена начинаются на `attr_100_vals_`)
 - 400 ключей, которые имеют уникальные значения (имена начинаются на `attr_1_vals_`)
6. `ddl-выражение` для создания таблицы:

```
CREATE TABLE public.tst_testobjattrs (
  scaption text NULL,
  jobj jsonb NULL
);
CREATE INDEX idx_tst_testobjattrs_jidext_dz ON public.tst_testobjattrs USING gin
↳(jobj jsonb_path_ops);
```

7. pg/sql-выражение для генерации тестовых данных:

```
DO $$
begin
FOR t IN 1 .. 800 loop
  EXECUTE (
    with attrs as materialized (
      select 'attr_10_vals_' || cast(t as text) as sName
        ,(select array_agg(cast(v % 10 + t as text) || '/' || cast(v % 10 + t
↳as text)) from generate_series(t*10, t*10 + 99) v ) as vals
        ,t as nNum
      from generate_series(1, 400) t
    union all
      select 'attr_bool_vals_' || cast(t as text) as sName
        ,(select array_agg(cast((v + t) % 2 as text)) from generate_series(t*10,
↳t*10 + 99) v ) as vals
        ,t as nNum
      from generate_series(401, 800) t
    union all
      select 'attr_100_vals_' || cast(t as text) as sName
        ,(select array_agg(cast(v as text) || '/' || cast(v as text)) from
↳generate_series(t*10, t*10 + 99) v ) as vals
        ,t as nNum
      from generate_series(801, 1200) t
    union all
      select 'attr_1_vals_' || cast(t as text) as sName
        ,(select array_agg(case when v > 0 then cast(t as text) || '/' ||
↳cast(t as text) end) from generate_series(1, 100) v ) as vals
        ,t as nNum
      from generate_series(1201, 1600) t
    )
    insert into tst_testobjattrs
    select 'capt' || t::text,
      (
        select jsonb_object(array_agg(a.sName), array_agg(a.vals[t % 100]))
          from generate_series(t % 400, t % 400 + 39) v
          cross join generate_series(1, 1201, 400) attrStep
          join attrs a on a.nNum = v + attrStep
        ) as jObj
    from generate_series(1, 100000) t;
  ');
  commit;
end loop;
END $$;
```

8. Размер таблицы:

- данные 208 Гб
- индекс 56 Гб

Запрос на точное совпадения значения

В этом запросе точно известны все значения атрибутов, по которым требуется найти строки

Выполняемый запрос:

```
select *
from tst_testobjattrs t
where t.jobj @> '{"attr_10_vals_102": "103/103"}'
and t.jobj @> '{"attr_10_vals_94": "95/95"}'
and t.jobj @> '{"attr_10_vals_73": "74/74"}'
and t.jobj @> '{"attr_10_vals_63": "64/64"}'
and t.jobj @> '{"attr_1_vals_1300": "1300/1300"}'
and t.jobj @> '{"attr_100_vals_888": "8941/8941"}'
and t.jobj @> '{"attr_bool_vals_467": "1"}'
and t.jobj @> '{"attr_bool_vals_494": "1"}'
```

План выполнения запроса. Используется индекс:

Step	Operation	Costs	Actual Time	Actual Rows	Loops
1	Bitmap Heap Scan on tst_testobjattrs t	(costs=1930.24..1931.37 rows=1 width=1464)	69.172..89.726	rows=500	loops=1
2	Recheck Cond: ((jobj @> '{"attr_10_vals_102": "103/103"}')::jsonb) AND (jobj @> '{"attr_10_vals_94": "95/95"}')::jsonb) AND (jobj @> '{"attr_10_vals_73": "74/74"}')::jsonb) AND (jobj @> '{"attr_10_vals_63": "64/64"}')::jsonb) AND (jobj @> '{"attr_1_vals_1300": "1300/1300"}')::jsonb) AND (jobj @> '{"attr_100_vals_888": "8941/8941"}')::jsonb) AND (jobj @> '{"attr_bool_vals_467": "1"}')::jsonb) AND (jobj @> '{"attr_bool_vals_494": "1"}')::jsonb)				
3	Heap Blocks: exact=500				
4	--> Bitmap Index Scan on idx_tst_testobjattrs_idx_idx_dz	(cost=0.00..1930.24 rows=1 width=0)	69.107..69.107	rows=500	loops=1
5	Index Cond: ((jobj @> '{"attr_10_vals_102": "103/103"}')::jsonb) AND (jobj @> '{"attr_10_vals_94": "95/95"}')::jsonb) AND (jobj @> '{"attr_10_vals_73": "74/74"}')::jsonb) AND (jobj @> '{"attr_10_vals_63": "64/64"}')::jsonb) AND (jobj @> '{"attr_1_vals_1300": "1300/1300"}')::jsonb) AND (jobj @> '{"attr_100_vals_888": "8941/8941"}')::jsonb) AND (jobj @> '{"attr_bool_vals_467": "1"}')::jsonb) AND (jobj @> '{"attr_bool_vals_494": "1"}')::jsonb)				
6	Planning Time: 0.059 ms				
7	Execution Time: 89.761 ms				

Запросы на неточное совпадение значения

Запросы like

Существующий индекс

В этой главе описываются возможности индексного доступа, при использовании оператора like.

Индекс gin (jsonb_path_ops) не обеспечивает возможность индексного доступа при использовании оператора like.

Выполняемый запрос:

```
select *
from tst_testobjattrs t
where (t.jobj ->> 'attr_10_vals_4') not like '1%'
```

Тип узла	Сущность
Parallel Seq Scan	tst_testobjattr

План выполнения. Полный просмотр таблицы:

Если выполнить запрос с условием, в котором будет точное совпадение значения и условие по like, то будет использоваться индекс с дополнительной фильтрацией.

Выполняемый запрос:

```
select *
  from tst_testobjattr t
 where t.jobj @> '{"attr_10_vals_10": "10/10"}'
        and t.jobj ->> 'attr_10_vals_4' like '1%'
```

Тип узла	Сущность	Стоимость
Bitmap Index Scan	idx_tst_testobjattr_idx_text	0.00 - 6904.50
Bitmap Heap Scan	tst_testobjattr	6905.50 - 2772946.05

План выполнения. Используется индекс:

Дополнительный функциональный индекс

Для решения задачи поиска значения, используя оператор like, возможно создать функциональный индекс.

Поиск по началу строки.

1. Создаем индекс с классом операторов text_pattern_ops

```
create index tst_testobjattr_idx_text on tst_testobjattr ((jobj ->> 'attr_10_vals_
↪3') text_pattern_ops)
```

2. Время создания: 5 мин 20 сек. размер: 528 МБ
3. Выполняем запрос

```
select *
  from tst_testobjattrs t
 where (t.jobj ->> 'attr_10_vals_3') like '100%'
```

Тим узла	Сущность	Стоимость
Bitmap Heap Scan	tst_testobjattrs	5456.57 - 1456225.33
Bitmap Index Scan	tst_testobjattrs_idx_text	0.00 - 5356.57

--like без доп условий (индексированный ключ)
select *
from tst_testobjattrs t
where (t.jobj ->> 'attr_10_vals_3') like '100%'

4. План запроса, используется индекс:

Поиск по началу строки.

1. Создаем индекс с классом операторов `gin_trgm_ops`. Потребуется подключенный модуль `pg_trgm` (поставляется по умолчанию с нашей системой). Модуль который разбивает слова на триграммы и выполняет поиск по ним. Отсюда следует, что искать строки следует от 3-х символов и больше

```
ccreate index tst_testobjattrs_idx_text_trg on tst_testobjattrs using gin ((jobj ->>
  ↳ 'attr_10_vals_2') gin_trgm_ops)
```

2. Время создания: 7 мин 8 сек. размер: 98 МБ
3. Выполняем запрос

```
select *
  from tst_testobjattrs t
 where (t.jobj ->> 'attr_10_vals_2') like '%10/%'
```

Тим узла	Сущность	Стоимость
Parallel Bitmap Heap Scan	tst_testobjattrs	21660.03 - 9200388.32
Bitmap Index Scan	tst_testobjattrs_idx_text_trg	0.00 - 20860.03

--like без доп условий (индексированный ключ gin)
select *
from tst_testobjattrs t
where (t.jobj ->> 'attr_10_vals_2') like '%10/%'

4. План запроса, используется индекс:

Запрос диапазона по числам

Для решения задачи поиска по диапазону чисел (<, <=, >, >=) возможно создать функциональный индекс.

1. Создаем индекс

```
create index tst_testobjattrs_idx_number on tst_testobjattrs (((jobj -> 'nNumber')  
--:: numeric))
```

2. Время создания: 4 мин 58 сек. размер: 1.7 ГБ
3. Выполняем запрос

```
select *  
  from tst_testobjattrs t  
 where (jobj -> 'nNumber') :: numeric >= 10  
       and (jobj -> 'nNumber') :: numeric <= 20
```

The screenshot shows a query execution plan for the provided SQL query. The plan consists of a single step: a Bitmap Index Scan on the index tst_testobjattrs_idx_number. The cost is 0.00 - 8396.57. The plan is shown in a window titled 'План выполнения запроса - 4'. On the right, a summary of the query is displayed, including the query text and various statistics.

Имя	Значение
Общее	
Тип узла	Bitmap Index Scan
Сущность	tst_testobjattrs
Стоимость	8496.57 - 1463265.33
Условие	
Details	
Parallel-Av	false
Async-Cap	false
Relation-N	tst_testobjattrs
Alias	t
Startup-Cost	8496.57
Total-Cost	1463265.33
Plan-Rows	400000
Plan-Width	1505
Recheck-C	(((jobj -> 'nNumber') :: numeric >= 10 and (jobj -> 'nNumber') :: numeric <= 20))

4. План запроса, используется индекс:

Запрос диапазона по датам

В json нет такого типа данных как даты, потому система хранит даты в виде строк в формате DD.MM.YYYY HH24:MI:SS. По этому для поиска по дате необходимо создать функциональный индекс, который бы переводил строку в дату.

Давайте попробуем создать индекс:

```
create index tst_testobjattrs_idx_json_date2 on tst_testobjattrs (to_date(jobj --> 'dDate  
--', 'DD.MM.YYYY HH24:MI:SS'))
```

Получим ошибку:

```
SQL Error [42P17]: ОШИБКА: функции в индексном выражении должны быть помечены как  
--IMMUTABLE
```

Для создания функциональный индексов требуется использовать только функции, помеченные как IMMUTABLE. Т.е. при одних и тех же входящих параметрах результат функции не должен изменяться. Функция to_date зависит от локали базы и не может использоваться при индексации.

По этому алгоритм действий следующий:

1. Создаем функцию конвертации строки в дату. Т.к. формат у нас жестко фиксирован, то результат функции всегда будет одинаковым, и мы можем пометить ее как IMMUTABLE:

```
create function strToDate_tst(stringDate text)
RETURNS timestamp
LANGUAGE sql
AS $function$
    select to_date(stringDate, 'DD.MM.YYYY HH24:MI:SS')
$function$
IMMUTABLE;
```

2. Создаем индекс

```
create index tst_testobjattrs_idx_json_date on tst_testobjattrs (strToDate_tst(jobj
↩->> 'dDate'))
```

3. Время создания: 10 мин 7 сек. размер: 528 МБ

4. Выполняем запрос

```
select *
from tst_testobjattrs t
where strToDate_tst((t.jobj ->> 'dDate')) between current_date - interval '100 DAY
↩' and current_date
```

Тип узла	Сущность	Стоимость
Index Scan	tst_testobjattrs	0.57 - 1597732.57

Имя: tst_testobjattrs_idx_json_date

Общие:

- Тип узла: Index Scan
- Сущность: tst_testobjattrs
- Стоимость: 0.57 - 1597732.57
- Строки: 400000
- Время: 10 мин 7 сек
- Условие: ((strToDate_tst((t.jobj ->> 'dDate')) between current_date - interval '100 DAY' and current_date))

Details:

- Parallel-Av: false
- Async-Cap: false
- Scan-Direc: Forward
- Index-Name: tst_testobjattrs_idx_json_date
- Relation-Name: tst_testobjattrs
- Alias: t
- Startup-Cost: 0.57
- Total-Cost: 1597732.57
- Plan-Rows: 400000
- Plan-Width: 1505
- Index-Conn: (tst_testobjattrs_idx_json_date)

SQL-запрос:

```
--сравнение без доп условий (индексированный ключ)
select *
from tst_testobjattrs t
where strToDate_tst((t.jobj ->> 'dDate')) between current_date - interval '100 DAY' and current_date
```

5. План запроса, используется индекс:

Выводы

1. При запросе строк по точному совпадению значения необходимо использовать оператор @>

Пример запроса:

```
select *
from tst_testobjattrs t
where t.jobj @> '{"attr_10_vals_102": "103/103", "attr_10_vals_94": "95/95"}'
```

2. Для поиска по началу строки (like с процентом справа) требуется создать дополнительный функциональный индекс с классом операторов text_pattern_ops.

Пример индекса:

```
create index tst_testobjattrs_idx_text on tst_testobjattrs ((jobj ->> 'attr_10_
↪vals_3') text_pattern_ops)
```

3. Для поиска по вхождению строки (like с процентом справа и слева) требуется наличие модуля `pg_trgm` (поставляется по умолчанию с нашей системой) создать дополнительный функциональный gin-индекс с классом операторов `gin_trgm_ops`.

Пример индекса:

```
create index tst_testobjattrs_idx_text_trg on tst_testobjattrs using gin ((jobj ->>
↪ 'attr_10_vals_2') gin_trgm_ops)
```

4. Для поиска по интервалу числовых значений требуется создать доп. функциональный индекс.

Пример индекса:

```
create index tst_testobjattrs_idx_number on tst_testobjattrs (((jobj -> 'nNumber')
↪:: numeric))
```

5. Для поиска по интервалу дат требуется создать доп. функциональный индекс и использовать `immutable`-функцию конвертации строки в дату.

Пример функции:

```
create function strToDate_tst(stringDate text)
RETURNS timestamp
LANGUAGE sql
AS $function$
    select to_date(stringDate, 'DD.MM.YYYY HH24:MI:SS')
$function$
IMMUTABLE;
```

Пример индекса:

```
create index tst_testobjattrs_idx_json_date on tst_testobjattrs (strToDate_
↪tst(jobj ->> 'dDate'))
```

6. Недостатки функциональных индексов в том, что на каждый ключ нужен отдельный индекс, что вызывает увеличение времени вставки и изменения записей таблицы, и увеличивает размер, занимаемый на диске.

Работа с сериализованными данными

JSON

JsonObject

Представляет собой обертку над мапой ключ-значение. Отлично подходит для небольших объектов ключ-значение и для больших объектов без конкретной структуры.

Способы получения

```
val fromStr = "".ns.toJValue.asJObject
val fromRop = JEmbeddedDoc.parseProperty(rop, "Имя json контейнера").asJObject
```

Геттеры

- Для получения данных из контейнера по ключу используйте один из типизированных геттеров `ru.bitec.app.gtk.lang.json.JObjectAbs`:

```
val jObj = JEmbeddedDoc.parseProperty(rop, "Имя json контейнера").asJObject
jObj.getNString(key)
jObj.getBoolean(key)
jObj.getNLong(key)
jObj.getNDate(key)
jObj.getNGid(key)
jObj.childJObject(key)
jObj.childJArray(key)

/** Общий геттер
 * ВАЖНО: JArray и JObject полученные через этот метод являются копиями т.е.
 * →изменения в них не отобразятся в ориг. объекте
 */
jObj.getJValue(key: NString)
```

- Для записи в контейнер используйте типизированные сеттеры `ru.bitec.app.gtk.lang.json.JObjectAbs`:

```
val jObj = "{}".ns.toJValue.asJObject
jObj.set("key".ns, value)
val innerJArray = jObj.createJArray("array".ns)
val innerJObject = jObj.createJObject("object".ns)
```

Object Mapper

Конвертирует JSON строку в конкретные объекты

Сериализация и десериализация

```
val jsonString = """"{"key":{"id":10,"value":"someValue"}}""""
/** ВАЖНО все классы используемые для десериализации должны
 * быть объявлены вне других классов
 */
case class Example(id: NLong, value: NString)

val map = GtkObjectMapper().readValue[Map[NString, Example]](jsonString)

val newJsonString = GtkObjectMapper().writeValueAsString(map)
```

Сериализация и десериализация абстрактных типов

```
// При сериализации наследников им добавится ключ "type"
@JsonTypeInfo(
  use = JsonTypeInfo.Id.NAME,
  include = As.PROPERTY,
  property = "type")
@JsonSubTypes(
  value = Array(
    new JsonSubTypes.Type(value = classOf[Type1], name = "type1"),
    new JsonSubTypes.Type(value = classOf[Type2], name = "type2")
  )
)
trait Base {
  def id: NLong

  def sValue: NString
}

@JsonTypeName("type1")
case class Type1(id: NLong, sValue: NString, nValue: NNumber) extends Base

@JsonTypeName("type2")
case class Type2(id: NLong, sValue: NString, gidValue: NGid) extends Base

val sJSON = """{
  "id": 10,
  "sValue": "Hello world",
  "type": "type2",
  "gidValue": "0/0"
}"""

val result = GtkObjectMapper().readValue[Base](sJSON) match {
  case Type1(id, sValue, nValue) => nValue.toString()
  case Type2(id, sValue, gidValue) => gidValue.toString()
}
```

XML

scala.XML

Сериализация и десериализация

```
// Получить их строки
val xmlText = """*XML строка*"
val xml = XML.loadString(xmlText)

// задать напрямую в коде, в '{}' можно указывать выражения которые возвращают текст или
↳ элементы
val xml = <Node>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
{Seq(1,2,3).map(num => <Node>{num}</Node>)}  
</Node>  
  
// задать напрямую в коде (scala 3)  
val xml = xml"""<Node></Node>"""  
  
// Превратить обратно в строку  
val xmlText = xml.toString
```

Получение значений

```
val xml =  
<example>  
  <tag name="Name"/>  
  <children>  
    <child name="1"/>  
    <child name="2"/>  
    <child name="3"/>  
    <child name="3">Hello</child>  
  </children>  
</example>  
  
// собирает все ноды в "example" с тегом "children", потом собирает все ноды в "children"  
→ " с тегом "child"  
val children = xml \ "children" \ "child"  
// собирает все ноды "child" во всем документе  
val children2 = xml \\ "child"  
  
// '@' берет атрибут у элемента и возвращает его строковое значение  
// ВАЖНО если элементов несколько то сложит их значения в одну строку, используйте если  
→ уверены что работаете с 1 элементом  
val firstChildrenName = children.headOption.toSeq \@ "name"  
  
// если нужно просто получить значения в виде списка то используйте '\' и добавьте "@" к  
→ названию атрибута  
val childrenNames = (children \ "@name").map(_.text)
```

Сериализация и десериализация в объекты

Автоматической конвертации в объекты в библиотеке нет, и лучший вариант писать самому

```
case class Child(sName: NString, sText: NString) {  
  def toXML(): Node = {  
    // Если в значение атрибута передать null, то атрибут будет удален из элемента  
    <child name={sName.get}>  
      {sText.get}  
    </child>  
    .copy(minimizeEmpty = true) // складывает Node типа <child></child> в <child/>  
  }  
}
```

(продолжается на следующей странице)

```

}

object Child {
  def fromXML(node: Node): Child = {
    Child(
      /*
       * Осторожнее с .text
       * 1) Он складывает все текстовые ноды в одну строку
       * 2) '\'' и '\\\'' возвращают пустые NodeSeq если ничего не нашли,
       * и .text вернет пустую строку, что не всегда желательно
       */
      sName = (node \ "@name").text.nullif(""),
      sText = node.text.nullif("")
    )
  }
}

// <child name="Joe">Hello world</child>
val exampleTo1 = Child(sName = "Joe", "Hello world").toXML()
// <child/>
val exampleTo2 = Child(sName = None.ns, None.ns).toXML()

```

Обработка XML

```

val xml =
<Отчет>
  <Документы>
    <Документ имя="Дог №1">
      <Сумма тип="Поступление">5000</Сумма>
      <Сумма тип="Расходы">100</Сумма>
    </Документ>
    <Документ имя="Дог №4">
      <Сумма тип="Поступление">200</Сумма>
      <Сумма тип="Расходы">30</Сумма>
    </Документ>
    <Документ имя="Дог №2">
      <Сумма тип="Расходы">100</Сумма>
    </Документ>
  </Документы>
</Отчет>

// Правило которое для элемента "Сумма" с типом "Поступление" разделит сумму указанную в
→ нем на 2
object subtractTAX extends RewriteRule {
  override def transform(n: Node): Seq[Node] = n match {
    case elem: Elem if elem.label == "Сумма" && (elem \ "@тип").text == "Поступление" =>
      val nCurrSum = NNumber.fromAny(elem.text)
      val nSumAfterTAX = (nCurrSum * 50.nn / 100.nn).round(2)
      elem.copy(child = Text(nSumAfterTAX.toString))
    case rest => rest
  }
}

```

```

    }
  }

  // Правило которое для элемента "Документ" посчитает сумму указанную в элементах "Сумма"
  object finalDocSum extends RewriteRule {
    override def transform(n: Node): Seq[Node] = n match {
      case elem: Elem if elem.label == "Документ" =>
        val sum = elem.descendant.collect {
          case elem: Elem if elem.label == "Сумма" && (elem \ "@тип").text == "Поступление"
            =>
              NNumber.fromAny(elem.text).nvl(0.nn)
        }.fold(0.nn)(_ + _)

        elem % Attribute(None, "сумма", Text(sum.toString), Null)
      case rest => rest
    }
  }

  // Правило которое для элемента "Отчет" посчитает сумму указанную в атрибутах для
  // элементов "Документ"
  object finalReportSum extends RewriteRule {
    override def transform(n: Node): Seq[Node] = n match {
      case elem: Elem if elem.label == "Отчет" =>
        val nFinalSum = (elem \ "Документ")
          .map(n => (n \ "@сумма").text)
          .map(sSum => NNumber.fromAny(sSum).nvl(0.nn))
          .fold(0.nn)(_ + _)

        elem % Attribute(None, "сумма", Text(nFinalSum.toString), Null)
      case rest => rest
    }
  }

  // собираем общее правило в порядке в котором мы хотим чтобы документ обработался
  val ruleTransformer = new RuleTransformer(subtractTAX, finalDocSum, finalReportSum)

  /*
  <Отчет сумма="2600">
  <Документы>
  <Документ сумма="2500" имя="Дог №1">
    <Сумма тип="Поступление">2500</Сумма>
    <Сумма тип="Расходы">100</Сумма>
  </Документ>
  <Документ сумма="100" имя="Дог №4">
    <Сумма тип="Поступление">100</Сумма>
    <Сумма тип="Расходы">30</Сумма>
  </Документ>
  <Документ сумма="0" имя="Дог №2">
    <Сумма тип="Расходы">100</Сумма>
  </Документ>
  </Документы>
  </Отчет>
  */

```

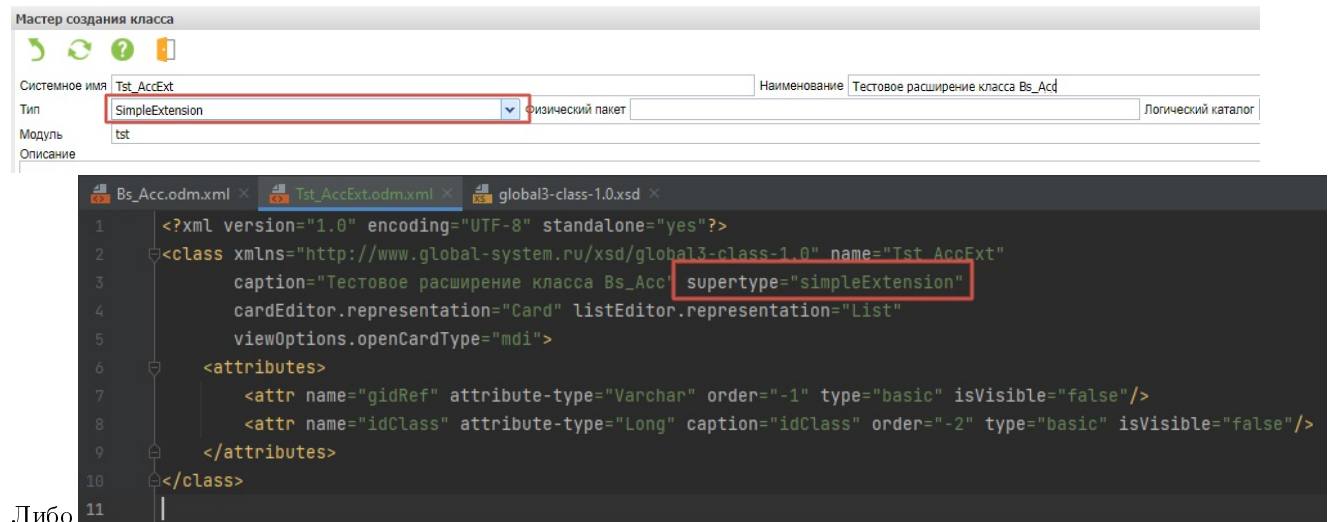


```
*/
val xmlResult = ruleTransformer(xml)
```

Классы-расширения. Simple Extensions

Создание класса-расширения

Класс-расширение создается полностью аналогично любому другому классу. При создании необходимо указать супертип `simpleExtension`.



Либо

Внимание

Если к классу-расширению предполагается добавлять коллекции, то он не может иметь супертип `simpleExtension`! В таком случае можно установить ему тип `Journal` или иной, наиболее подходящий в контексте задачи!

Внимание

Как и у миксинов, у классов-расширений не должно быть собственных `id` и `gid`, они ссылаются на `gid` мастер-объекта при помощи поля `gidRef`!

Внимание

В отличие от миксинов, при создании расширения, в мастер-классе не делается никаких записей, напрямую связанных с классом-расширением. В первую очередь потому, что большая часть расширений обычно создается в модулях, прямой доступ к которым из мастер-объекта невозможен!

Создание и удаление объекта класса-расширения

- Объект-расширение тесно связан с мастер-объектом и его жизненным циклом. Жизненный цикл объекта расширения не может выходить за пределы такового у его мастер-объекта, но может быть меньше. Для создания и удаления объекта-расширения требуется создавать *точки расширения* и добавлять их вызов в нужный момент жизненного цикла мастер-объекта.
- Объект-расширение может быть создан не раньше создания его мастер-объекта. Самый ранний этап жизненного цикла мастера, в который может быть создано расширение - момент создания мастера. Для этого можно использовать уже существующую точку расширения `Btk_Ept#dpiInjectionAfterInsert`. Код этой точки расширения будет вызван сразу же после создания мастер-объекта. Вызов этой точки расширения добавлен во все dpi по умолчанию, поэтому требуется создавать только вторую часть точкирасширения - файл `Ext.scala`.
- Аналогично производится удаление объекта `Btk_Ept#dpiInjectionBeforeDelete` - эта точка расширения вызывается перед удалением мастер-объекта. Её вызов так же доавлен во все dpi по умолчанию.

Внимание

Необходимо учитывать, что эти точки расширения вызываются при создании объектов ЛЮБЫХ классов! Поэтому во второй части точки расширения `dpiInjectionAfterInsert` или `dpiInjectionBeforeDelete` **СТРОГО НЕОБХОДИМО** проверять к какому классу принадлежит объект мастер-класса, который обрабатывается точкой расширения!

- Инициализация/удаление объекта-расширения происходит только в случае, если класс мастера соответствует классу мастер-объекта, для которого создается расширение.

```
7
8 class Tst_BtkExt extends Extension{
9
10  /**
11   * Вызывается из Dpi при вставке объекта.
12   */
13  subFunc( name = "dpiInjectionAfterInsert") { (sf: SuperFunc[Unit], rop: Rop[_]) =>
14    sf(rop).getOrElse()
15    val x = if (rop.gid.parseIdClass() === Bs_AccApi().idClass) {
16      //Инициализация объекта-расширения от мастер-объекта
17      Tst_AccExtApi().insertByOwner(rop)
18      //Дополнительный код
19      //...
20      //...
21    }
22  }
23
24
25  subFunc( name = "dpiInjectionBeforeDelete") { (sf: SuperFunc[Unit], rop: Rop[_]) =>
26    sf(rop).getOrElse()
27    val x = if (rop.gid.parseIdClass() === Bs_AccApi().idClass) {
28      //Удаление объекта-расширения от мастер-объекта
29      Tst_AccExtApi().deleteByKey(rop.gid.get)
30      //Дополнительный код
31      //...
32      //...
33    }
34  }
35
36 }
```

- Если вам необходимо укоротить жизненный цикл объекта-расширения, например если такой объект может существовать только пока мастер-объекту присвоен определенный тип объекта, тогда необходимо создать свою точку расширения для мастер-объекта, либо воспользоваться уже существующей точкой, предназначенной для создания объектов-расширений, однако с существующими точками стоит быть предельно внимательными и убедиться, что такая точка подходит для вашей задачи!

```

22  class Bs_AccApi extends Bs_AccDpi[Bs_AccAro, Bs_AccApi, Bs_AccAta] {
356  } {
357      val Array(idvA, idvB) = rvx.idaPair().get
358      val ropA = load(idvA)
359      setidAccLink(ropA, idvB)
360  }
361  }
362
363  override def setidAccType(rop: ApiRop, value: NLong): Unit = {
364      value match {
365          case sv if sv === someValue => Bs_Ept().insertExtBySomeValue(rop)
366          case ov if ov === otherValue => Bs_Ept().deleteExtBySomeValue(rop)
367      }
368      super.setidAccType(rop, value)
369  }
370  }
371  }
372
373
374  object Bs_AccApi extends ApiFactory[java.lang.Long, Bs_AccAro, Bs_AccApi]
375
376  class Bs_AccAro extends Bs_AccDro
377
378  class Bs_AccAta extends Bs_AccDta[Bs_AccAro, Bs_AccApi, Bs_AccAta]
379
380  object Bs_AccAta extends Bs_AccAta
381

```

Вызов такой точки расширения происходит из api-сеттера соответствующего атрибута *МАСТЕР-ОБЪЕКТА*, на который завязан жизненный цикл расширения. Вторая часть точки расширения объявляется соответствующим образом из парного для этой *точки* файла.

- В случае использования специализированной точки расширения для создания и удаления файлов допускается отсутствие проверки на совпадение классов родительских объектов, в отличие от использования общих точек `Btk_Ept#dpiInjectionAfterInsert` и `Btk_Ept#dpiInjectionBeforeDelete`.

Примечание

В SimpleExtentionApi определены базовые методы для работы с объектами-расширениями и доступные для работы из вашего класса-расширения по умолчанию. Для вывода данных объекта-расширения совместно с мастер-объектом рекомендуется создавать отдельную выборку в классе-расширении, в которую выводить все требуемые атрибуты (так же, при необходимости, на эту выборку можно добавлять атрибуты мастер-объекта), после чего прикреплять эту выборку к выборке мастер-объекта, используя настройку на типе объекта, регистрируя из модуля в котором создано расширение, закладку на тип(ы) мастер-объекта, либо используя схожий подход для возможности динамической настройки детализации выборок мастер-объекта без использования настроек на типе объекта (например при отсутствии такого атрибута у мастера).

Универсальный фильтр

Создание дополнительной группы фильтрации по классу

1. На операции lazyInitFilter инициализируем дополнительную группу.

```
override def lazyInitFilter(): Unit = {  
  if (!fltManager.isPopulatedRootGroup) {  
    Btk_FltPkg().createRootGroupByClass(fltManager, "Bs_Goods", "ТМЦ")  
  }  
  super.lazyInitFilter()  
}
```

2. На onApplyFilter получаем текст фильтрации по группе.

```
override def onApplyFilter(): Unit = {  
  super.onApplyFilter()  
  //проверяем, что группы инициализированы и фильтр активен в выборке  
  if (fltManager.isPopulatedRootGroup && fltManager.isActive) {  
    val alias = "tt"  
    val macros = Btk_FltPkg().generateMacroByGroup(fltManager, "Bs_Goods", alias)  
    selection.setMacro("GdsMacro", macros.where)  
  }  
}
```

3. В операции onRefresh используем установленный в onApplyFilter макрос:

- Через метод prepareSelectStatement.

```
override protected def onRefresh: Recs = {  
  prepareSelectStatement("&GdsMacro")  
}
```

- В тексте sql-запроса.

```
override protected def onRefresh: Recs = {  
  ""  
  select .....  
  from .....  
  where ....  
    and &GdsMacro  
  ""  
}
```

Создание дополнительной группы фильтрации с произвольными атрибутами

1. На операции lazyInitFilter инициализируем дополнительную группу.

```
override def lazyInitFilter(): Unit = {
  if (!fltManager.isPopulatedRootGroup) {
    Btk_FltPkg().createCustomRootGroup(fltManager, "Some_CustomGroup",
    ↪ "Произвольная группа", (builder) => {
      builder.addRefObjectAttr("idGds", "Тмд", "Bs_Goods")
      builder.addBasicAttr("sCode", "Код", AttributeTypes.Varchar, false)
    })
  }
  super.lazyInitFilter()
}
```

2. На onApplyFilter получаем текст фильтрации по группе.

```
override def onApplyFilter(): Unit = {
  super.onApplyFilter()
  //проверяем, что группы инициализированы и фильтр активен в выборке
  if (fltManager.isPopulatedRootGroup && fltManager.isActive) {
    val macros = Btk_FltPkg().generateMacroByGroup(fltManager, "Some_CustomGroup",
    ↪ "tt")
    selection.setMacro("SomeMacro", macros.where)
  }
}
```

3. В операции onRefresh используем установленный в onApplyFilter макрос:

- Через метод prepareSelectStatement.

```
override protected def onRefresh: Recs = {
  prepareSelectStatement("&SomeMacro")
}
```

- В тексте sql-запроса.

```
override protected def onRefresh: Recs = {
  """
  select ....
  from ....
  where ....
    and &SomeMacro
  """
}
```

Добавление JSON-атрибуту признака раскрываемости в дереве атрибутов и формирование атрибутов, отображающихся при его раскрытии

```
def afterBuildFltEntityMeta(fltMetaEntity: FltMetaEntity, fltManager: FltManager): Unit = {
  // проверка, что переданный класс - класс, для которого нужно добавить логику
  if (fltMetaEntity.name == "Bs_Goods") {
    // определяем, что json-атрибут доступен по умолчанию.
    val typeSizeAttrOpt = fltMetaEntity.attrMap.get("jTypeSizeAttrs".toLowerCase)
    if (typeSizeAttrOpt.isDefined) {
      val typeSizeAttr = typeSizeAttrOpt.get
      // ставим признак, что он может быть раскрыт в дереве
      Btk_FltMetaAttributePkg().setCanExpand(typeSizeAttr, true)

      // формируем атрибуты-потомки
      for (rvx <- new OQuery(Gds_TypeSizeCharacteristicAta.Type) {
      }) {
        // создаем значимые атрибуты, которые хранятся в json-контейнере
        val fltAttr = Btk_FltMetaAttributePkg().buildBasicJObjectAttr(
          systemName = rvx.get(_.sCode),
          caption = rvx.get(_.sCaption),
          dbType = AttributeTypes.Number,
          isBoolean = false,
          jsonColumnName = "jTypeSizeAttrs"
        )

        // устанавливаем созданному атрибуту потомка.
        Btk_FltMetaAttributePkg().setParentTree(fltAttr, typeSizeAttr)

        // добавляем атрибут в общий перечень атрибутов.
        fltMetaEntity.attrMap(fltAttr.systemName.toLowerCase) = fltAttr
      }
    }
  }
}
```

Примечание

Каждый из таких атрибутов формирует условие, которое получает значение из json-контейнера.

Добавление значимому атрибуту признака ссылочности и признака раскрываемости в дереве атрибутов

```
def afterBuildFltEntityMeta(fltMetaEntity: FltMetaEntity, fltManager: FltManager): Unit = {
  // Проверка, что переданный класс - класс, для которого нужно добавить логику.
  if (fltMetaEntity.name == "Bs_Goods") {
    // Вызываем из пакета `Btk_FltMetaAttributePkg` метод для создания значимого
    // атрибута `buildBasicAttr`.
    val fltAttr = Btk_FltMetaAttributePkg().buildBasicAttr(
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
systemName = "screateuser_dz",           // systemName - системное имя
↳ атрибута. Уникально в пределах одного класса,
caption = "Создавший пользователь",      // caption - отображаемое
↳ наименование,
dbType = AttributeTypes.Varchar,         // dbType - тип данных,
isBoolean = false)                       // isBoolean - булевый атрибут.

// Вызываем из пакета `Btk_FltMetaAttributePkg` метод для установки параметров
↳ преобразования атрибута в ссылочный `setExpandableParams`.
Btk_FltMetaAttributePkg().setExpandableParams(
    fltAttr = fltAttr,                    // fltAttr - атрибут фильтра,
↳ который преобразуется в ссылочный,
    selfAttrName = "screateuser_dz",      // selfAttrName - имя поля в текущем
↳ объекте (self), используемое в выражении selfJoinExpression,
    selfJoinExpression = "lower(${column})", // selfJoinExpression - выражение,
↳ которое формирует правую часть условия JOIN, используя поле текущего объекта,
    childAttrName = "susername",          // childAttrName - имя поля в
↳ дочернем объекте (child), используемое в выражении childJoinExpression,
    childJoinExpression = "lower(${column})", // childJoinExpression - выражение,
↳ которое формирует левую часть условия JOIN, используя поле дочернего объекта,
    refClass = "Btk_User")                // refClass - имя класса (таблицы),
↳ к которой делается JOIN.
// Здесь формируется join по атрибутам с приведением значений к нижнему регистру
↳ (через lower).
// ${column} - подставляет алиас таблицы и имя поля, которое указываем в
↳ `selfAttrName` или `childAttrName`
// в зависимости от выражения `selfJoinExpression` подставляется `selfAttrName`, а
↳ к `childJoinExpression` подставляется `childAttrName`.
// В результате получается запрос вида:
// join Btk_User t94 on lower(t94.susername) = lower(t93.screateuser_dz)

// Ставим признак, что атрибут может быть раскрыт в дереве.
Btk_FltMetaAttributePkg().setCanExpand(fltAttr, true)

// Добавляем атрибут в общий перечень атрибутов.
fltMetaEntity.attrMap(fltAttr.systemName.toLowerCase) = fltAttr
}
}
```

Скрытие группы «Отбор»

Для скрытия группы «Отбор» необходимо в операции инициализации фильтра lazyInitFilter установить флаг ru.bitec.app.btk.flt.FltAviManager#isAvailableMainGroup в значение False.

Пример кода:

```
override def lazyInitFilter(): Unit = {
    if (!fltManager.isPopulatedRootGroup) {
        fltManager.isAvailableMainGroup = false
    }
    super.lazyInitFilter()
}
```

(продолжается на следующей странице)

}

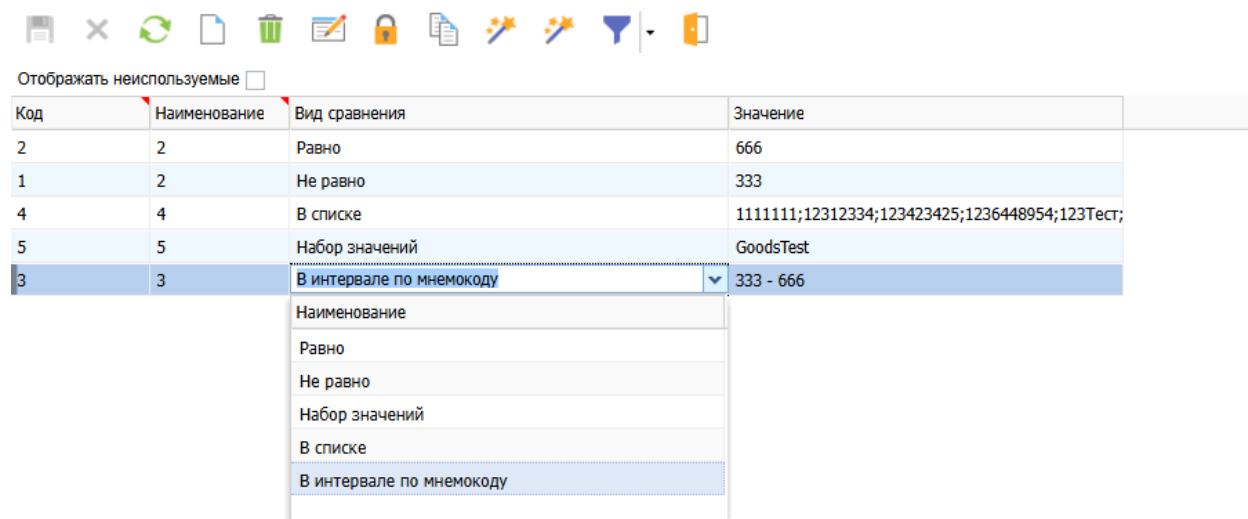
Как переопределить параметры события

Для передачи параметров в выборку сеттера ссылочного атрибута необходимо переопределить обработку событий в прикладной выборке, а именно функцию `EntityAviDefault#handleFltEvents`. Параметры, которые могут быть переопределены, представлены изменяемыми коллекциями для удобства работы:

- `setterParams: mutable.Map[String, Any]`: Список параметров, которые передаются в выборку клиентского сеттера без изменений.
- `availableValues: mutable.ArrayBuffer[Any]`: Список допустимых `id` или `gid`, определяющий возможные значения атрибута фильтра. Ограничивает список, открывающийся «по трём точкам».

```
override def handleFltEvents(event: FltEvent): Unit = {
  event match {
    // событие перед установкой значения ссылочного атрибута из интерфейса
    case event: FltBeforeClientSetRefObjectEvent =>
      event.setterParams.update("FLT_IDDEPOWNER", 47606.nl)
      // добавление к полученному от атрибута списку разрешённых идентификаторов
      ➔ дополнительных значений
      event.availableValues += 132021.nl
      // очистка и составление НОВОГО списка идентификаторов
      event.availableValues.clear()
      event.availableValues += Array(1.nl, 2.nl)
    case _ =>
  }
}
```

Настройка поля выбора в прикладной выборке с типом сравнения «Как в универсальном фильтре»



Отображать неиспользуемые ☐

Код	Наименование	Вид сравнения	Значение
2	2	Равно	666
1	2	Не равно	333
4	4	В списке	1111111;12312334;123423425;1236448954;123Тест;
5	5	Набор значений	GoodsTest
3	3	В интервале по мнемокоду	333 - 666

Наименование

- Равно
- Не равно
- Набор значений
- В списке
- В интервале по мнемокоду

В описанном примере будет создан выбор из справочника ТМЦ, и настройки фильтрации будут храниться в поле `jData`.

Настройка выборки

1. Создайте в классе поле с типом `jsonb` для хранения настроек.
2. В `Avi` объявите структуру, которая будет дополнять запрос в `onRefresh` дополнительными полями.

```
/**
 * Класс с дополнительными полями
 * @param sCompareKindName Имя типа сравнения. Невидимый
 * @param sCompareKindCaption Наименование типа сравнения
 * @param sFilterValue Значение фильтра
 * @param sFilterValueEditor Тип редактора для значения фильтра. Невидимый.
 * @param bFilterValueRo Признак, что значение фильтра не доступно для
 * редактирования. Невидимый.
 */
case class Row(
    var sCompareKindName: NString,
    var sCompareKindCaption: NString,
    var sFilterValue: NString,
    var sFilterValueEditor: NString,
    var bFilterValueRo: Boolean
)
```

3. В `Avi` в операции `onRefresh` добавьте вывод дополнительных полей, необходимых для работы фильтра. Воспользуйтесь методом `ru.bitec.app.btk.flt.selected.attribute.standalone`.

Btk_FltStandaloneLib#getDisplaySettings.

```
override def onRefresh: Recs = {
  val arrBuffer: ArrayBuffer[(Btkst_FltStandaloneApi#ApiRop, Row)] = ArrayBuffer()

  //делаем запрос объектов из БД
  for (rop <- new OQuery(Btkst_FltStandaloneAta.Type) {
  }) {
    //создаем атрибут, ссылочный на ТМЦ, и загружаем в него текущие настройки из БД
    val standaloneAttribute = FltStandaloneAttribute.refAttr(
      sRefClass = "Bs_Goods",
      sData = rop.get(_.jData)
    )

    //получаем настройки отображения атрибута в выборке
    val displaySettings = Btk_FltStandaloneLib().
    →getDisplaySettings(standaloneAttribute)

    //добавляем дополнительные поля в результат onRefresh
    arrBuffer += ((
      rop,
      Row(
        sCompareKindName = displaySettings.sCompareKindName,
        sCompareKindCaption = displaySettings.sCompareKindCaption,
        sFilterValue = displaySettings.sFilterValue,
        sFilterValueEditor = displaySettings.sFilterValueEditor,
        bFilterValueRo = displaySettings.bFilterValueRo
      )
    ))
  }

  arrBuffer.toVector
}
```

4. В Avi в операции onRefreshItem также добавьте вывод дополнительных полей.

```
override def onRefreshItem: Recs = {
  val rop = thisRop()

  //создаем атрибут, ссылочный на ТМЦ, и загружаем в него текущие настройки из БД
  val standaloneAttribute = FltStandaloneAttribute.refAttr(
    sRefClass = "Bs_Goods",
    sData = rop.get(_.jData)
  )

  //получаем настройки отображения атрибута в выборке
  val displaySettings = Btk_FltStandaloneLib().
  →getDisplaySettings(standaloneAttribute)
  (
    rop,
    Row(
      sCompareKindName = displaySettings.sCompareKindName,
      sCompareKindCaption = displaySettings.sCompareKindCaption,
      sFilterValue = displaySettings.sFilterValue,

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
sFilterValueEditor = displaySettings.sFilterValueEditor,  
bFilterValueRo = displaySettings.bFilterValueRo  
)  
)  
}
```

5. В Avi в операции checkWorkability добавьте блокировку поля ввода значения.

```
override def checkWorkability(): Unit = {  
  super.checkWorkability()  
  selection.attrs("sFilterValue").isReadOnly = selection.getSelfVar("bFilterValueRo"  
→").asBoolean()  
}
```

6. Добавьте сеттеры. Важно добавить их с признаками вызова refreshAfter и cwaAfter.

```
@Setter(refreshAfter = true, cwaAfter = true)  
def setsCompareKindName(event: SetterEvent): Unit = {  
  val res = Btk_FltStandaloneLib().setCompareKindName(  
    FltStandaloneAttribute.refAttr("Bs_Goods", thisRop().get(_).jData)),  
    event  
  )  
  //если были внесены изменения в настройку, то сохраняем ее в БД  
  if (res.hasChange) {  
    thisApi().setjData(thisRop(), res.attributeValue.buildJson().toNString)  
  }  
}  
  
@Setter(refreshAfter = true, cwaAfter = true)  
def setsCompareKindCaption(event: SetterEvent): Unit = {  
  
}  
  
@Setter(refreshAfter = true, cwaAfter = true)  
def setsFilterValue(event: SetterEvent): Unit = {  
  val res = Btk_FltStandaloneLib().setFilterValue(  
    FltStandaloneAttribute.refAttr("Bs_Goods", thisRop().get(_).jData)),  
    event  
  )  
  //если были внесены изменения в настройку, то сохраняем ее в БД  
  if (res.hasChange) {  
    thisApi().setjData(thisRop(), res.attributeValue.buildJson().toNString)  
  }  
}
```

7. В avm.xml добавьте настройку атрибутов.

```
<attributes>  
  <attr name="sCompareKindName" isVisible="false"/>  
  <attr name="sCompareKindCaption" isVisible="true" order="100" caption="Вид  
→сравнения">  
    <editor>  
      <lookup changeableAttr="sCompareKindName" isLookupLazyLoad="true"  
→
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
→lookupKeyAttr="sName"
        lookupListAttr="sCaption" lookupQuery="gtk-Btk_
→FltAttributeCompareKindAvi#Lookup_ByParams"
        isResetButtonVisible="true">
    <params>
        <param name="names" value="Equal;InEqual;InValueSet;InList;
→InterValMnemonicCode" dataType="String"/>
    </params>
</lookup>
</editor>
</attr>
<attr name="sFilterValue" isVisible="true" order="110" caption="Значение">
    <editor editorTypeAttr="sFilterValueEditor"/>
</attr>
<attr name="sFilterValueEditor" isVisible="false"/>
<attr name="bFilterValueRo" isVisible="false"/>
</attributes>
```

Примечание

Доступные типы сравнения указываются в теге `<params>` через символ `;`. Перечень системных имен указан в `ru.bitec.app.btk.flt.selected.attribute.FltAttributeCompareKind`.

Использование в бизнес-логике

1. Пример получения `where`-условия по сохраненной настройке:

```
val rop = thisRop()

//создаем атрибут, ссылающийся на ТМЦ, и загружаем в него текущие настройки из БД
val standaloneAttribute = FltStandaloneAttribute.refAttr(
    sRefClass = "Bs_Goods",
    sData = rop.get(_jData)
)

//сформировать макрос, где все условия будут накладываться на колонку t.id
val macros = Btk_FltStandalonePkg().generateMacro(standaloneAttribute, "t.id")

if (macros.hasFilter) {
    dialogs.showMessage(macros.where)
} else {
    dialogs.showMessage("Условий не наложено")
}
```

2. Пример получения перечня значений по сохраненной настройке:

```
val rop = thisRop()

//создаем атрибут, ссылающийся на ТМЦ, и загружаем в него текущие настройки из БД
val standaloneAttribute = FltStandaloneAttribute.refAttr(
```

(продолжается на следующей странице)

```

sRefClass = "Bs_Goods",
sData = rop.get(_jData)
)

//получить перечень Gid-ов, удовлетворяющих условию
val valuesOpt = Btk_FltStandalonePkg().getRefAttrValues(standaloneAttribute)

if (valuesOpt.isDefined) {
  dialogs.showMessage(
    s"""Кол-во: ${valuesOpt.get.size}
      |Первые 20 значений: ${valuesOpt.get.take(20).mkString(";")}
      |""".stripMargin
  )
} else {
  dialogs.showMessage("Условий не наложено")
}

```

Запуск отладки/теста

1. Запустите idea

Если вы запускаете idea от ярлыка `gsf-cli`, idea запустится с открытым проектом. Иначе может потребоваться открыть проект вручную.

2. При необходимости обновите sbt(отметка №2)

Обновление требуется, в случае если

- Произошли изменения в структуре проекта
Файлы `*.sbt` , `project.yaml`
- Обновились внешние библиотеки

3. • Для того, чтобы запустить отладку:

Для этого выполните пункт меню `Run > Debug 'Project_Name'`

- Для того, чтобы запустить тест:

Для этого нажмите на зелёную кнопку play напротив строку с объявлением теста в коде, далее `Debug 'Test_Name'`

Как вносить изменения

Данный совет напоминает какие действие надо совершать в среде разработки IntelliJ Idea при изменении тех или иных элементов в проекте решения.

Изменение разметки класса

1. Перейдите в проект IntelliJ Idea
2. Создайте или отредактируйте разметку класса
Файл с расширением `.odm.xml`
3. Запустите генерацию кода
Выберете в проекте файл с измененной разметкой и выполните пункт меню **Tools > External Tools > Generate Sources**
Генерация кода файлы окружения класса(класс для хранения данных в кэше, бизнес логика по умолчанию).
4. Запустите сборку проекта
Пункт меню **Build > Build Project**
5. Обновите метаданные в базе данных
Выберете в проекте файл с измененной разметкой и выполните пункт меню **Tools > External Tools > Generate Tables**
6. Перейдите в приложение для которого внесли изменения
По умолчанию для локального хоста приложение запускается по адресу `http://localhost:8080/`
7. Обновите код и библиотеки orm
Пункт меню **Сервис > Управление решением > Обновить код, БиблиотекиOrm**

Изменение разметки выборки или исходного кода

1. Перейдите в проект IntelliJ Idea
2. При необходимости отредактируйте разметку выборки
Файл с расширением `.avm.xml`
3. При необходимости измените программный код
Файлы с расширением `.scala, .java`
4. Запустите сборку проекта
Пункт меню **Build > Build Project**
5. Перейдите в приложение для которого внесли изменения
По умолчанию для локального хоста приложение запускается по адресу `http://localhost:8080/`
6. Обновите код
Пункт меню **Сервис > Управление решением > Обновить код**

Внимание

Убедитесь в приложении сброшен флаг Использовать кэш метаданных выборов, пункт меню **Сервис > Управление решением > Использовать кэш метаданных выборов**

Как обновить внешние зависимости

Данный совет предполагает что для работы с проектом вы используете `gsf-cli`

1. Закройте idea

Внимание

Должны быть закрыты все экземпляры IntelliJ Idea

2. Запустите ярлык `refresh.cmd` Ярлык находится в каталоге `[programs]\gsf-cli\workspace\links\[project_name]\refresh.cmd`, где
 - `programs`
Каталог куда установлен `gsf-cli`
 - `project_name`
Имя проекта для обновления
3. Откройте idea
Для этого запустите ярлык `[programs]\gsf-cli\workspace\links\[project_name]\start_idea.cmd`
4. Обновите sbt
Смотрите [документацию idea](#)
5. Соберите проект
Для этого выполните пункт меню `Build > Build Project`

Как переопределить методы API

1. Перейдите на необходимый класс API в среде разработки
2. Внутри класса пропишите `override def`
3. Из выпадающего списка выберите необходимый метод для переопределения.

Как переопределить методы AVI

1. Перейдите на необходимую выборку AVI в среде разработки
2. Перейдите на трейт отображения
3. Внутри трейта пропишите `override def`
4. Из выпадающего списка выберите необходимый метод для переопределения.

Как переопределить сеттеры API

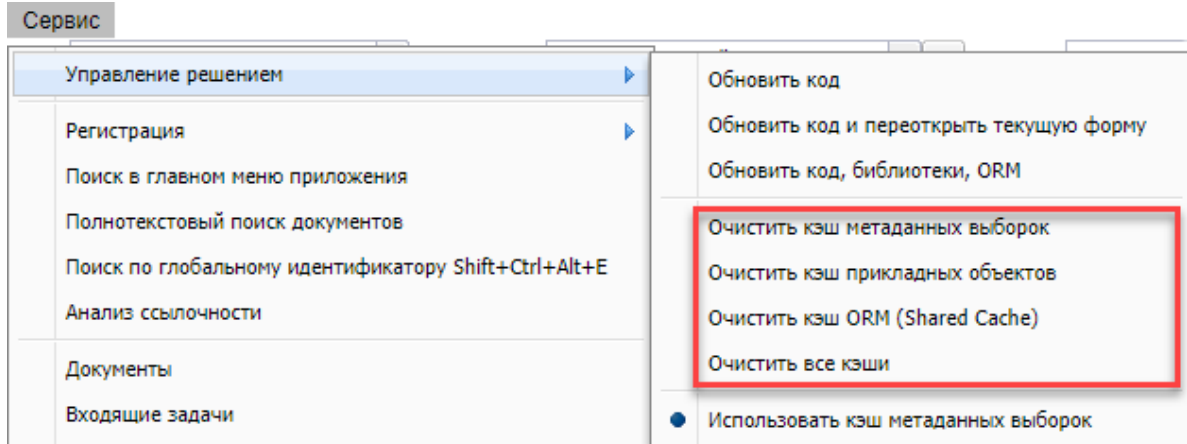
1. Перейдите на необходимый класс API в среде разработки
2. Внутри класса пропишите `override def set`
3. Из выпадающего списка выберите необходимый сеттер для переопределения и нажмите ввод.

Как переопределить сеттеры AVI

1. Перейдите на необходимую выборку AVI в среде разработки
2. Перейдите на трейт отображения
3. Внутри трейта пропишите `override def set`
4. Из выпадающего списка выберите необходимый сеттер для переопределения.

Как сбросить кэш

1. В любом приложении при наличии прав в пункте меню Сервисс - > Управление решением будет блок операций по сбросу кэша



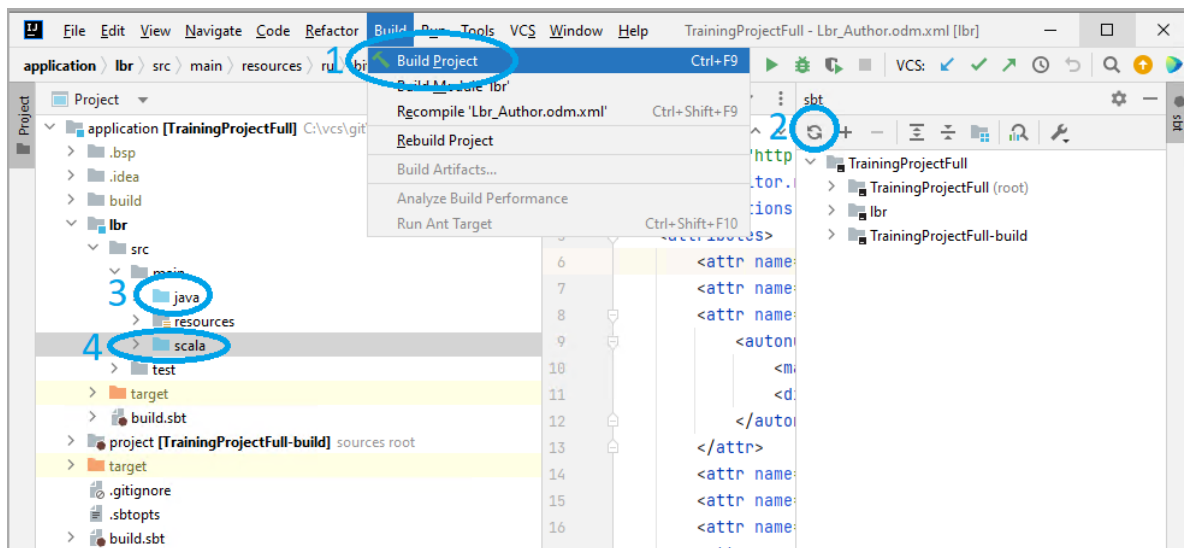
2. Для сброса кэша метаданных выборок необходимо снять признак **Использовать кэш метаданных выборок**

Как собрать проект

1. Запустите `idea`
Если вы запускаете `idea` от ярлыка `gsf-cli`, `idea` запустится с открытым проектом. Иначе может потребоваться открыть проект вручную.

Совет

Более подробно по работе с проектом смотрите [документацию idea](#)



2. При необходимости обновите sbt(отметка №2)

Обновление требуется, в случае если

- Произошли изменения в структуре проекта
Файлы *.sbt , project.yaml
- Обновились внешние библиотеки

Внимание

Папки исходного кода должны быть отмечены цветом, смотрите отметки №3 и №4.

3. Соберите проект

Для этого выполните пункт меню Build > Build Project(отметка №1)

Как создать класс

1. В директории с ресурсами (Наименование модуля/src/main/resources/ru/bitec/app) создать файл odm(Object domain model/Доменная модель сущности) с наименованием Наименование модуля/Наименование модуля_Наименование класса.odm.xml
2. В odm определить необходимые атрибуты, подключить коллекции и миксины.
3. Сгенерировать код по созданному odm файлу
 1. Кликнуть правой кнопкой мыши по файлу
 2. В подменю External Tools выбрать Generate Sources
4. Собрать проект
5. Сгенерировать таблицы по odm файлу
 1. Кликнуть правой кнопкой мыши по файлу
 2. В подменю External Tools выбрать Generate Tables

Как создать коллекцию

1. В директории с ресурсами (Наименование модуля/src/main/resources/ru/bitec/app) создать файл odm(Object domain model/Доменная модель сущности) с наименованием Наименование модуля/Наименование модуля_Наименование коллекции.odm.xml
2. Указать тип класса: `supertype="collection"`
3. В odm определить ссылочный атрибут на класс родитель у данного атрибута указать `genListCollectionRep="true"`
4. В odm определить необходимые атрибуты
5. Подключить коллекцию к классу родителю
 - Для этого в odm класса родителя необходимо указать

```
<collections>
  <collection name="Наименование коллекции" cascadeOnDelete="true" ref.attr=
  ↪ "Наименование ссылочного атрибута на класс родитель в коллекции"/>
</collections>
```

6. Сгенерировать код по odm класса родителя
 1. Кликнуть правой кнопкой мыши по файлу
 2. В подменю External Tools выбрать Generate Sources
7. Собрать проект
8. Сгенерировать таблицы по odm файлу коллекции
 1. Кликнуть правой кнопкой мыши по файлу
 2. В подменю External Tools выбрать Generate Tables

Как создать новое отображение

1. В Avi файле в классе объявите новый `trait`, унаследовав его от нужных отображений
2. В Avi файле в классе добавьте метод для отображения - например

```
def list(): List = {
  new List {
    override def meta = this
  }
}
```

В случае, если в переопределяемом отображении уже есть такой метод, нужно добавить `override`

3. В avm файле добавьте разметку для нового отображения

Как создать точку расширения

Модуль в котором необходима точка расширения - Модуль1. Модуль расширения - Модуль2.

1. В модуле в котором необходима точка расширения создаем класс с объектом компаньоном `Модуль1_Ept.scala`.
2. Класс унаследовать от класса `ExtensionPoint`.
3. Объект компаньон унаследовать от `PkgFactory[Модуль1_Ept]`.
4. Создать метод точки расширения:

```
val extentionMethodExamlpe = declFunc("extentionMethodExamlpe") {  
  (sf: SuperFunc[NString], arg1: NString, arg2: NString) =>  
    sf().getOrElse(None.ns)  
}
```

5. В модуле расширении создать класс `Модуль2_Модуль1Ext.scala`.
6. Унаследовать класс от класса `Extension`.
7. Создать метод расширения:

```
subFunc("extentionMethodExamlpe") { (sf: SuperFunc[NString], arg1: NString, arg2: NString) =>  
  ...  
}
```

8. В файле `src/main/resources/META-INF/extensions.xml` модуля расширения зарегистрировать новую точку расширения:

```
<exts>  
  <ext class="ru.bitec.app.Модуль2.Модуль2_Модуль1Ext" targetEpt="`Модуль1`_Ept"/>  
</exts>
```

Найти и открыть класс из настройки системы

1. Откройте приложение «Настройка системы»
2. Выберите пункт меню «Сущности» -> «Классы»
3. В фильтре введите системное имя необходимого класса
4. На панели операций нажмите на операцию «Редактировать объекты класса в списке».

Настройка автонумерации

1. Откройте `odm` класса;
2. Укажите для нужного атрибута свойство `type = autonum`;
3. Создайте внутри тега `attr` вложенный тег `autonum`. Укажите в нем следующие свойства:
 - `id` - уникальный в рамках класса идентификатор автонумерации.

- **isHoleFill** - заполнение пропусков (опционально). Если в настройках автонумерации указан разрез, то автоматически включено. Чтобы выключить нужно прописать **isHoleFill = false**.

4. Внутри тега **autonum** создайте вложенные теги:

- **mask** - маска автонумерации. Маска должна собой представлять jexl-скрипт, с экранированием спец. символов. Для каждого автонумерующегося атрибута может быть настроена своя маска автонумерации. В маске доступен для использования параметр **counter**, который будет заменен на значение счетчика.
- **dimension** - разрез автонумерации:
 - Установите у тега следующие свойства:
 - * **name** - название разреза. Задается всегда, даже если используется автонумерация без разреза;
 - * **attr** - атрибут, в рамках которого рассматривается разрез. Разрезы указываются из атрибутов класса. Используется только для автонумерации с разрезом.
 - * **order** - порядковый номер. Если его не задать, то разрезы будут обрабатываться в алфавитном порядке. Используется для автонумерации с разрезом по нескольким атрибутам.
 - Дополнительные возможности:
 - * При необходимости можно добавить более одного тега **dimension**. В этом случае разрез будет реализован по нескольким атрибутам.
 - * К значениям атрибутов разреза возможно применение выражений, они указываются во вложенном теге **expression** для **dimension**. На каждый разрез может быть не больше одного выражения. Значение, указанное в элементе **expression**, будет передано в обработчик Jexl-скриптов, параметр, заменяемый на значение атрибута разреза, называется **dimValue**.

Пример автонумерации без разреза:

```
<attr name="sNoDep"
  caption="Автонумерация без разреза"
  attribute-type="Varchar"
  type="autoNum"
  isHeadLine="true">
  <autonum id="1">
    <mask>lpad(counter,3,"0")</mask>
    <dimension name="dim1"/>
  </autonum>
</attr>
```

Пример указания нескольких разрезов:

```
<attr name="sTwoDepExpr"
  caption="Автонумерация с 2 разрезами"
  attribute-type="Varchar"
  type="autoNum">
  <autonum id="3">
    <mask>lpad(counter,3,"0")</mask>
    <dimension name="nNumber"
      attr="nNumber"
```

(продолжается на следующей странице)

```

        order="10" />
        <dimension name="dDate_dim"
            attr="dDate"
            order="20">
            <expression>truncDate(dimValue)</expression>
        </dimension>
    </autonum>
</attr>

```

5. Сформируйте окружение класса:

- Сформируйте исходный код для класса (`generate sources`);
- Выполните сборку проекта (`build`);
- Сформируйте таблицы класса (`generate tables`)

Настройка группировки класса

1. Откройте odm класса

2. У тега `class` укажите свойства:

1. `group.type`

- `single`
Единичная группировка, объект может входить только в одну группу
- `multi`
Множественная группировка, объект может входить в несколько групп

2. `group.root`

Системное имя корневой группы класса

3. `objectAttrCardType`

- `group`
Настройка объектных характеристик будет регулироваться группами
- `groupAndObjectType`
Настройка объектных характеристик будет регулироваться группами и дополняться настройками с типа объекта

4. `group.isPanelVisible`

Опционально. Если указать `true`, то панель групп будет отображаться при открытии списка объектов класса.

Пример тега `class`:

```

<class name="Some_Class" group.root="Profiles" group.type="single"
  <objectAttrCardType="group" group.isPanelVisible="true">

```

3. Под тегом `class` создайте тег `collections`, если он не создан

4. Под тегом `collections` создайте тег `var-collection`

5. Укажите свойства:

- `name = Btk_ObjectGroup`

- `ref.attr = gidSrcObject`

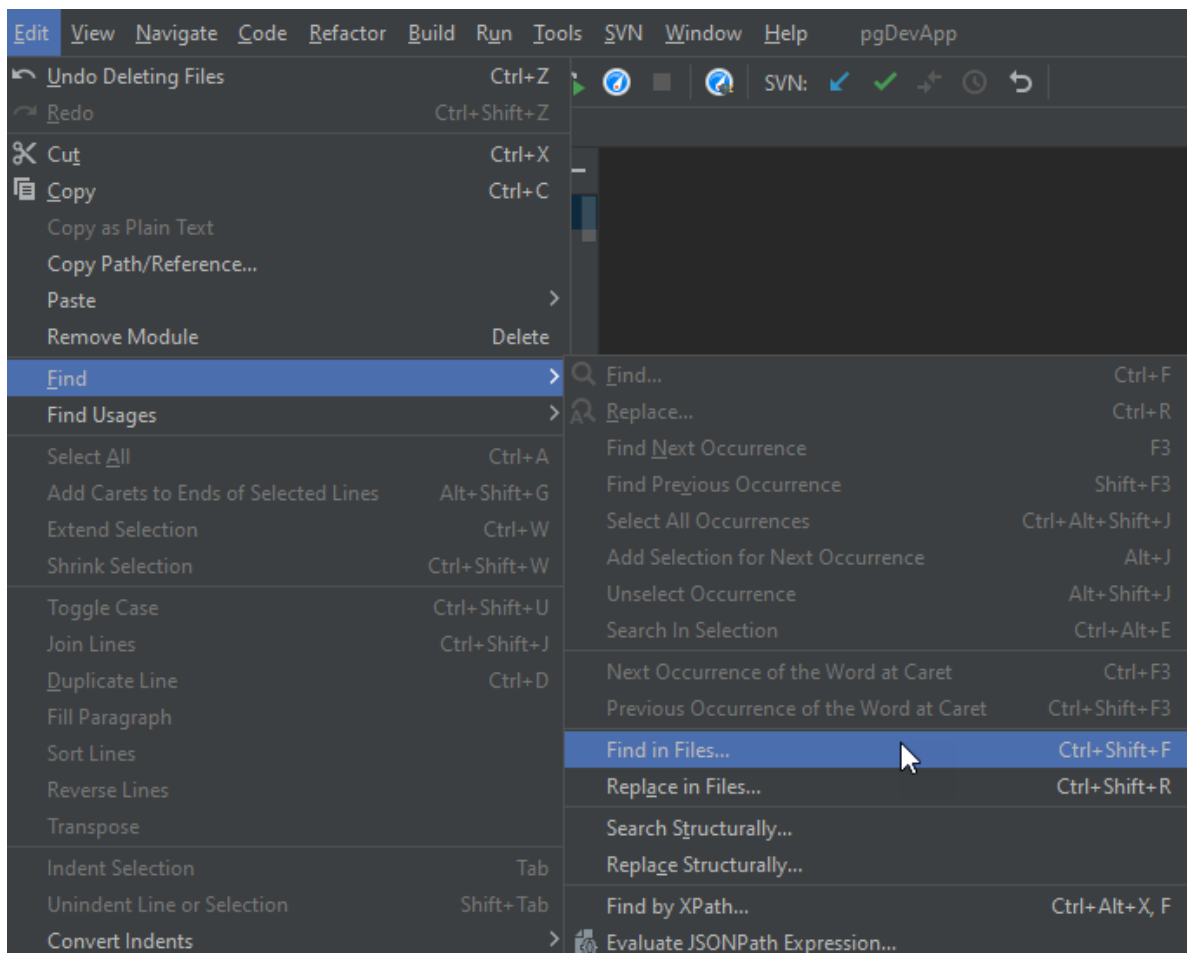
Пример тега collections:

```
<collections>
  <var-collection name="Btk_ObjectGroup" ref.attr="gidSrcObject"/>
</collections>
```

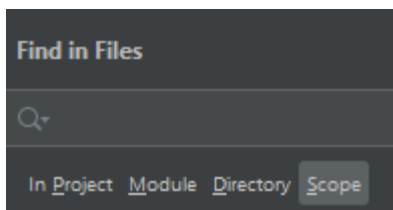
6. Сформируйте исходный код для класса (`generate sources`)
7. Сформируйте таблицы класса (`generate tables`)

Поиск по коду в Idea

1. Для поиска по коду Вызовите пункт меню Edit -> Find -> Find in files (горячая клавиша Ctrl + Shift + F)



2. Для поиска по всем файлам (в т.ч. и исходникам бандлов) выберите категорию Scope



3. Результат поиска можно открыть на отдельной закладке по кнопке `Open in Find Window` в нижнем-правом углу окна поиска

Создание логического атрибута класса

1. Откройте `odm` класса
2. Создайте тег `attributes`, если он не создан
3. Под тегом `attributes` укажите новый тег `attr`
4. Укажите свойства:
 - `name` - Имя атрибута
 - `caption` - Наименование атрибута
 - `attribute-type` = `Number`
 - `editorType` = `check`
5. Под тегом `attr` добавьте новый тег `booleanColumn`

Пример:

```
<attr name="bBool" caption="Логический" attribute-type="Number" editorType="check">
  <booleanColumn/>
</attr>
```

Практики разработки

Полезные практики от опытных разработчиков

Переопределение `checkWorkAbility()`

В `checkWorkAbility` нельзя использовать:

- `thisRop()`, вместо этого лучше использовать `getVar/getSelfVar`
- обходчики, особенно с инвалидацией в БД (запросом на согласование данных в БД) (например, `refreshByParent()`).

Нужно понимать, что `checkWorkAbility` вызывается часто, поэтому его оптимизация сильно сказывается на времени работы бизнес-логики.

session.flush() вместо **session.commit()**

`session.commit()` лучше избегать и использовать `session.flush()`.

`commit()` выгружает данные из кэша в БД.

`flush()` применяет изменения сессии к БД.

В случае использования `flush()` данные ещё не в БД и есть возможность откатить изменения, например, в случае ошибки.

`commit()` вызывает `afterEdit()` с проверкой обязательных полей в отличие от `flush()`.

Избегать использование [Option].get и [Collection].head

Не использовать для:

- `Option .get`, если внутри был `None`, то выпадет исключение.

Использовать `.getOrElse(<значение или исключение, в случае None>)`.

- коллекций `.head` без проверки на пустоту коллекции.

лучше использовать `.headOption`, что вернёт результат в конструкции `Option`, который дальше распаковать через `.getOrElse()`.

Миксин Btk_Object

миксин `Btk_Object` хранит все записи классов супер-типа документ и справочник.

В него нельзя вносить изменения! Это системный миксин.

Практика Avi

Про selectStatement и onRefreshExt

Изначально результат выборки формируется операцией `onRefresh` или `onRefreshItem` одним из двух способов: реляционный или объектный запрос.

Реляционный запрос

В этом случае в теле `onRefresh` вызывается метод `selectStatement` или `prepareSelectStatement` (вызывает тот же `selectStatement`, но с добавлением фильтров, описанных в `avm`). Иначе говоря, результатом операции `onRefresh` является текстовое значение, внутри которого реляционный запрос.

Примечание

`onRefreshExt` в случае формирования выборки реляционным запросом не вызывается!

Так формируется по умолчанию отображение `List` (кроме коллекций). Не учитываются значения хранящиеся в кэше, учитываются значения только из БД.

Пример `selectStatement` для отображения `List`:

```

override protected def selectStatement: String = {
  s"""SELECT
    t.id
    ,t.idClass
    ,t.idState
    ,t1.sHeadLine_dz as idStateHL
    ,t.idStateMC
    ,t.idObjectType
    ,t2.sHeadLine_dz as idObjectTypeHL
    ,t.idDepOwner
    ,t3.sHeadLine_dz as idDepOwnerHL
    ,t.gidSrc
    ,t.idResourceHolder
    ,t4.sHeadLine_dz as idResourceHolderHL
    ,t.sRegNum
    ,t.dReg
    ,t.idPeriod
    ,t5.sHeadLine_dz as idPeriodHL
    ,t.sDescription
    ,t.gid
    ,t.sRegNum_dz
    ,t.sRegNumBMs_dz
    ,t.sRegNumidVer_dz
  FROM Oil_DemandMov t
    LEFT JOIN Btk_ClassState t1 on t.idState = t1.id
    LEFT JOIN Btk_ObjectType t2 on t.idObjectType = t2.id
    LEFT JOIN Bs_DepOwner t3 on t.idDepOwner = t3.id
    LEFT JOIN Bs_Contras t4 on t.idResourceHolder = t4.id
    LEFT JOIN Bs_Period t5 on t.idPeriod = t5.id
  """
}

```

Коммит в БД при выполнении реляционного запроса и @FlushBefore

Важно понимать, что при реляционном запросе сервер приложения делает коммит в БД (особенность работы сервера приложения). Поэтому на выборках, где вводятся значения в поля и поддерживается возможность пользователю отменить изменения, нельзя использовать формирование выборки на реляционном запросе. Иначе данные запишутся в БД без разрешения пользователя (без нажатия пользователем на операцию сохранения (дискетка)).

Однако, иногда есть необходимость в выполнении выборки на реляционном запросе в отображении с редактированием полей. Например, выборка выпадающего списка `Lookup`.

В таком случае перед `onRefresh` необходимо добавить аннотацию `@FlushBefore(mode = FlushBeforeMode.Disabled)`:

```

trait Lookup extends Default with super.Lookup {

  @FlushBefore(mode = FlushBeforeMode.Disabled)
  override protected def onRefresh: Recs = {
    s"""SELECT
      t.id
    """
  }
}

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        ,t.sHeadLine_dz as sHeadLine
        ,t.sMnemoCode_dz as sMnemoCode
        ,coalesce(t.sMnemoCode_dz, '') || ' ' || coalesce(t.sHeadLine_dz, '') as sMnemoCodeHeadLine
    ↪
    from AsfEqp_IntoOperExt t
    order by upper(t.sHeadLine_dz)
    """
}
}
```

Иначе при открытии выпадающего списка данные будут записываться в БД.

Объектный запрос

В этом случае onRefresh формируется из экземпляров объектов SRep (инструменты refreshByParent, load, TxIndex, OQuery) или case class'ов.

В этом случае учитываются значения, хранящиеся в кэше.

Примеры:

По умолчанию отображение List у коллекций формируется с помощью refreshByParent:

```
trait List_idPlanFactoryShip extends Default with super.List_Master {

    override protected def onRefresh: Recs = {
        Oil_PlanFactShipSegrGroupApi().refreshByParent(getIdMaster)
    }
}
```

По умолчанию отображение Card у коллекций формируется с помощью load:

```
trait Card extends Default with super.Card {

    override protected def onRefreshItem: Recs = {
        Oil_PlanFactShipSegrGroupApi().load(getVar(getPKFieldName).asNLong)
    }
}
```

onRefreshExt

В случае объектного запроса сервер сам вызывает метод onRefreshExt, в который можно дописать получение нехранимых полей на синтаксисе SQL, НЕ учитывая значения из кэша:

```
override protected def onRefreshExt: String = {
    s"""with t as ( select
        :id as id
        ,:idPlanFactoryShip as idPlanFactoryShip
        ,:idSegregationGroup as idSegregationGroup
    )
    SELECT
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
t.id
,t1.sHeadLine_dz as idPlanFactoryShipHL
,t2.sHeadLine_dz as idSegregationGroupHL
,t2.sMnemonicCode_dz as idSegregationGroupMC
FROM t
  LEFT JOIN Oil_PlanFactoryShip t1 on t.idPlanFactoryShip = t1.id
  LEFT JOIN Oil_SegregationGroup t2 on t.idSegregationGroup = t2.id
"""
}
```

Примечание

Также существует другой способ реализовать нехранимые поля (через case class AdditionalInfo), который будет описан дальше. Данный способ формирует значения с учётом кэша.

Добавление нехранимых полей

Здесь будет рассмотрено 2 случая, как реализовать нехранимые поля в выборке, реализованные реляционным запросом и объектным.

Для формирования нехранимого поля нужно:

- получить поле в запросе под заданным псевдонимом
- описать поле в разметке `avm` под тем же псевдонимом

Формировать реляционно или объектно

Если нехранимый атрибут вычисляется динамически от изменений на выборке, это значит, что работа идёт с значениями из кэша, т.е. с значениями, которые ещё не закоммичены в БД. Это означает, что нехранимое поле реализовывать нужно на объектном запросе.

Например: в карточке документа нужно посчитать сумму по всем записям в коллекции. Если поле суммы реализовать реляционно, то оно будет формироваться только из согласованных в БД значений. Это значит, для того чтобы учитывались новые записи в коллекции, их нужно закоммитить (сохранить) в БД. Если же поле реализовать объектно, то в нём будут учитываться данные из кэша, т.е. те самые созданные новые записи в коллекции, которые ещё не закоммичены в БД.

Нехранимые поля в реляционном запросе

Примечание

Не учитывает значения из кэша, которые не закоммичены в БД.

Чтобы добавить не хранимый атрибут реляционно, нужно иметь значение атрибута в выборке запроса `selectStatement`.

Примером может служить результат кодогенератора ссылочного поля для отображения `List`, где описывается получение значения хэдайна в `Dvi` и описание атрибута в `dvm`.

Dvi:

```
override protected def selectStatement: String = {
s"""SELECT
  t.id
  ,t.idClass
  ,t.idObjectType
  ,t1.sHeadLine_dz as idObjectTypeHL  --нехранимое поле
  ,t.dReg
  ,t.sRegNum
  ,t.idState
  ,t2.sHeadLine_dz as idStateHL  --нехранимое поле
FROM Rzd_Train t
  LEFT JOIN Btk_ObjectType t1 on t.idObjectType = t1.id
  LEFT JOIN Btk_ClassState t2 on t.idState = t2.id
""""
}
```

dvm:

```
<representation name="Default">
  <attributes>
    <attr name="idObjectType" caption="Тип документа" isVisible="false" editorType=
↪"edit" order="10" isRequired="true"/>
    <attr name="idObjectTypeHL" caption="Тип документа" order="10.2" isRequired="true
↪">
      <editor>
        <buttonsEdit canEditText="true" changeableAttr="idObjectType">
          <buttons lookup="true" openCard="true" reset="true"/>
        </buttonsEdit>
      </editor>
      <ref class="Btk_ObjectType"/>
    </attr>
    <attr name="idState" caption="Состояние" isVisible="false" editorType="edit"
↪order="40" isRequired="true"/>
    <attr name="idStateHL" caption="Состояние" order="40.2" isRequired="true">
      <editor>
        <lookup changeableAttr="idState" isLookupLazyLoad="true" lookupKeyAttr=
↪"id" lookupListAttr="sHeadLine" lookupQuery="gtk-ru.bitec.app.btk.Btk_ClassStateAvi
↪#Lookup_Class"/>
      </editor>
      <ref class="Btk_ClassState"/>
      <grid columnWidth="12"/>
    </attr>
  </attributes>
</representation>
```

Примечание

Напомню, что Dvi и dvm - это результат работы кодогенератора по данным описанным в разметке odm. Разработчик работает в соответствующих файлах Avi и avm, переопределяя содержимое Dvi и dvm.

ВНИМАНИЕ! Изменять Dvi и dvm нельзя.

А точнее бесполезно, потому что изменения будут перетёрты при следующем запуске кодогенератора.

Нехранимые поля в объектном запросе

Примечание

Учитывает значения из кэша, которые не закоммичены в БД.

onRefreshExt

Данный метод вызывается, если результатом `onRefresh` является набор объектов, а не текст, представляющий реляционный запрос.

С помощью `onRefreshExt` в отображении `Card` получены хэдлайны в `Dvi`:

```
override protected def onRefreshExt: String = {
  s"""with t as ( select
    :id as id
    ,:idObjectType as idObjectType
    ,:idState as idState
    ,:gidSrc/*@NString*/ as gidSrc
  )
  SELECT
    t.id
    ,t1.sHeadLine_dz as idObjectTypeHL
    ,t2.sHeadLine_dz as idStateHL
    ,t3.sHeadLine as gidSrcHL
  FROM t
    LEFT JOIN Btk_ObjectType t1 on t.idObjectType = t1.id
    LEFT JOIN Btk_ClassState t2 on t.idState = t2.id
    LEFT JOIN Btk_Object t3 on t.gidSrc = t3.gidRef
  """
}
```

Здесь используется синтаксис postgresql.

Применяется конструкция `with` (ссылка).

Через ":" подставляются текущие значения атрибутов с выборки (в том числе из кэша). Так, например, `:id as id` означает, что с выборки будет получено текущее значение атрибута `id` и подставлено в таблицу `t` под псевдонимом поля `id`. Теперь обращение в основном запросе `t.id` будет возвращать текущее значение `id`.

Примечание

Под фразой «текущее значение» понимается значение на момент обновления выборки и выполнения метода `onRefreshExt`.

Примечание

Описание в `dvm` не меняется в зависимости от метода получения значения нехранимого поля (реляционного или объектного) и останется таким же, как в случае формирования нехранимого поля в `selectStatement`.

case class AdditionalInfo

Нехранимое поле можно сформировать, переопределив `onRefresh` и `onRefreshItem`.

В этом случае объект представляют как кортеж (`SROp`, экземпляр `case class`'а). Тем самым объект имеет поля записи, описанных в БД (хранимые поля), и поля `case class`'а (нехранимые).

В этом случае принято называть `case class AdditionalInfo`, а заполнение описывать в методе `getAdditionalInfo`.

Нехранимое поле в отображении `Card`:

```
case class AdditionalInfo(
    var nQtyLoadRecievActs: NNumber,
    var nQtyLoadTransferCertificate: NNumber,
    var nQtyLoadTransferCertificateAccounted: NNumber,
    var nQtyRemains: NNumber,
    var nQtyReserv: NNumber
)

protected def getAdditionalInfo(rop: Oil_ExternalMovApi#ApiRop): AdditionalInfo = {
    AdditionalInfo(
        nQtyLoadRecievActs = None.nn
        , nQtyLoadTransferCertificate = None.nn
        , nQtyLoadTransferCertificateAccounted = None.nn
        , nQtyRemains = None.nn
        , nQtyReserv = None.nn
    )
}

override protected def onRefresh: Recs = {
    val rop = thisApi().load(getVar(CardRep.IdItemSharp).asNLong)
    (rop, getAdditionalInfo(rop))
}

override protected def onRefreshItem: Recs = {
    val rop = thisApi().load(getVar(getPKFieldName).asNLong)
    (rop, getAdditionalInfo(rop))
}
```

Примечание

Переменные внутри `case class` необходимо указывать `var` для `Avi`

Нехранимое поле в отображении `List_idBrigade` коллекции:

```

trait List_idBrigade extends Default with super.List_idBrigade {

  case class AdditionalInfo(var sEmpId: NString = None.ns,
                             var sPosition: NString = None.ns)

  def getAdditionalInfo(rop: Bs_BrigadeStaffApi#ApiRop): AdditionalInfo = {
    if (rop.get(_.idEmployee).isNotNull) {
      val avEmployee = Bs_EmployeeApi().load(rop.get(_.idEmployee)).copyAro()
      AdditionalInfo(
        sEmpId = avEmployee.sEmpId,
        sPosition = avEmployee.sPosition)
    } else AdditionalInfo()
  }

  override protected def onRefresh: Recs = {
    Bs_BrigadeStaffApi().refreshByParent(getVarWithDep("super$id").asNLong)
      .map(rop => {
        (rop, getAdditionalInfo(rop))
      })
  }
}

```

Такой способ самый универсальный.

Например, с помощью `onRefreshExt` не получится посчитать сумму по всем записям коллекции с учётом кэша (для этого необходимо присоединить таблицу коллекции по условию `idparent = t.id`, что обеспечит учёт только тех записей и их значений, которые закоммичены в БД, т.е. без учёта кэша).

В рассматриваемой реализации, можно получить все записи коллекции с учётом кэша с помощью метода `refreshByParent(rop.get(_.id))` и далее в обходчике сложить необходимые значения полей.

Нехранимые строки

Здесь будет описан пример реализации пустых предзаполненных строк в списке, как частный случай, в коллекции. Такие строки не хранятся в БД, а существуют визуально в интерфейсе (иными словами нехранимые строки). Запись в БД создаётся при вводе пользователем какого-либо поля.

Пример в проекте pgDev: `ru.bitec.app.oil.Oil_RecievTaskDetAvi.List_idRecievTaskByFlyOverWay`.

№	Наименование местоположения	Вагон	Досмотровая ЖДН	ЖДН	Состав	Индекс состава	Выгружен	На перепод	Кол-во выгружено	Кол-во негружено	Кол-во п
1	ПТЗ-2-01				ЖД состав	123			0	0	
2	ПТЗ-2-02								0	0	
3	ПТЗ-2-03								0	0	
4	ПТЗ-2-04								0	0	
5	ПТЗ-2-05								0	0	
6	ПТЗ-2-06				ЖД состав	123			0	0	
7	ПТЗ-2-07								0	0	
8	ПТЗ-2-08								0	0	
9	ПТЗ-2-09								0	0	
10	ПТЗ-2-10								0	0	
11	ПТЗ-2-11								0	0	
12	ПТЗ-2-12								0	0	
13	ПТЗ-2-13								0	0	
14	ПТЗ-2-14								0	0	
15	ПТЗ-2-15								0	0	
16	ПТЗ-2-16								0	0	
17	ПТЗ-2-17								0	0	
18	ПТЗ-2-18								0	0	
19	ПТЗ-2-19								0	0	
20	ПТЗ-2-20								0	0	

case class Row

По сколку поля строки редактируемые, то не нужно использовать реляционный запрос для формирования выборки. Выборка будет формироваться экземплярами case class'a. Таким образом onRefresh вернёт список экземпляров case class'a.

В примере в коллекции «Подача» имеет столько строк, сколько вагонов указано в записи «Путь эстакады», на который ссылается мастер-документ.

Case class должен иметь такие же поля, как поля класса (таблицы), описанные в odm и нехранимые, формируемые не реляционно.

При добавлении нового хранимого поля, необходимо это поле добавить в case class.

```
/**
 * Представление строки, поля соответствуют хранимым полям строки
 *
 * Поле id выступает idFlyOverPos, т.к. для работы некоторых инструментов (например,
 * onRefreshExt) необходимо
 * наличие уникального идентификатора
 */
case class Row(
  var id: NLong //Oil_RecievTaskDet.idFlyOverPos
  , var gid: NGid = None.ng
  , var idRecievTask: NLong
  , var nRow: NNumber
  , var idWagon: NLong = None.n1
  , var idRailwayInvoice: NLong = None.n1
  , var bUnloaded: NNumber = None.nn
  , var bReFeed: NNumber = None.nn
  , var idReFeedType: NLong = None.n1
  , var bWagonRemoved: NNumber = None.nn
  , var bRequiredMeasure: NNumber = None.nn
  , var nQtyMeasure: NNumber = None.nn
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
, var idStorageTank: NLong = None.n1
, var sCertificateList: NString = None.ns
, var idRecievAct: NLong = None.n1
, var bRepeatedFeed: NNumber = None.nn
, var bCommAct: NNumber = None.nn
, var bRecievActCreated: NNumber = None.nn
, var idFlyOverPos: NLong = None.n1
, var idLockDeviceOut: NLong = None.n1
, var idRecievTaskDet: NLong = None.n1 //Oil_RecievTaskDet.id
, var idTrain: NLong = None.n1
, var bRecievByMeasure: NNumber = None.nn
)
```

onRefresh

ru.bitec.app.oil.Oil_RecievTaskDetApi.List_idRecievTaskByFlyOverWay#onRefresh.

Описание алгоритма:

- Получаем все записи данной коллекции по мастеру (т.е. те, что хранятся в БД или в кэше).

```
RecievTaskDetApi().refreshByParent(getIdMaster)
```

- Получаем все вагоны выбранной у мастера «Пути эстакады».

```
val ropParent = Oil_RecievTaskApi().load(selection.master.getSelfVar("id").asNLong)
Oil_FlyOverPosApi().txidFlyOverWay.refreshByKey(ropParent.get(_.idFlyOverWay))
```

Примечание

В примере в коллекции «Подача» имеет столько строк, сколько вагонов указано в записи «Путь эстакады», на который ссылается мастер-документ.

- Реализуем обходчик по каждому вагону и заполняем экземпляры case class'a
 - Если на данный вагон есть запись из таблицы БД, то заполняем данными из БД
 - Иначе устанавливаем предзаполненные значения необходимых полей или None

```
ropaFOP.map(ropFOP => {
    //поиск роны в БД
    val ropOpt = ropa.find(_.get(_.idFlyOverPos) === ropFOP.get(_.id))
    getRowByRop(ropOpt, ropFOP, nvRow)
})

protected def getRowByRop(ropOpt: Option[SRop[_ <: JLong, _ <: Oil_
↪RecievTaskDetAro]]
    , ropFOP: Oil_FlyOverPosApi#ApiRop
    , npRow: NNumber = None.nn
): Row = {
```

(продолжается на следующей странице)

```

Row(
  idPlacement = ropOpt.map(_.get(_.idPlacement)).nl
    .nl(thisApi().getIdPlacementByFlyOverPos(ropFOP.get(_.id)))
)
}

```

При добавлении нового хранимого атрибута, необходимо в этом методе заполнения описать правило заполнения нового поля.

На данном этапе предположим, что `id` заполняется от `ropOpt` (реальной записи), в случае отсутствия `None.n1`. Т.е. пустая строка имеет `id = None.n1`, что дальше будет использоваться, как признак пустой строки.

Примечание

Внимание: такая реализация не совсем корректна. В таком случае не будет работать ряд системных операций, таких, как `onRefreshExt`, которые требуют уникального значения `id`. Но для понимания будет рассмотрена такая реализация, а вопрос уникальности `id` будет рассмотрен далее.

`Row` - `case class`, представление строки, поля соответствуют хранимым полям записи.

- Возвращаем полученный список экземпляров `case class`'а.

insert при вводе значения в нехранимую строку

Необходимо переопределить операцию `beforeEdit()`.

```

override def beforeEdit(): Unit = {
    if (getSelfVar("id").isNull) {
        regRow()
    }
}

```

`beforeEdit()` вызывается при каждой попытке редактировать поле. Здесь необходимо инициализировать, что заполнение введётся в пустой нехранимой строке или в хранимой. Это можно сделать, проверив заполнен ли `id` (на данном этапе предполагается, что у пустой строки `id = None.n1`).

Если заполнение ведётся в пустой строке, то необходимо создать запись с переносом всех предустановленных значений и введённое пользователем значение применить к только что созданной записи.

Переносить предустановленные значения можно двумя способами:

- Прописать, какой сеттер вызывать и какое значение из `case class`'а подставлять на каждое поле в отдельности. Такая реализация усложняет поддержание кода. При добавлении нового хранимого атрибута, в случае если оно имеет предустановленное значение, нужно добавлять код в этот фрагмент.
- Реализовать обходчик по всем имеющимся полям `case class`'а, отсеять поля, которые не имеют сеттеров (`gid`, например) и значение которых `null`, по оставшимся полям вызывать сеттеры и предоставлять соответствующие значения:

```

/**
 * insertByParent(ropParent) + сеттеры заполненных полей case class'a Row
 *
 * @param ropParent
 * @param row
 * @return
 */
def insertByParentAndRow(ropParent: Oil_RecievTaskApi#ApiRop, row: Row): ApiRop = {
  insertByParent(ropParent) :/ { rop =>
    //setter'ы для переноса умолчательных значений из Row
    row.getClass.getDeclaredFields.map(_.getName.ns).zip(row.productIterator.to)
      .filterNot(field => saFieldsNotSetter.contains(field._1) || field._2.
→asInstanceOf[Nullable[_ <: Any, _]].isNull)
      .foreach(field => {
        setattrValue(rop, field._1, field._2)
      })
    rop
  }
}

```

Чтобы введенное пользователем значение относилось к только что созданной записи, необходимо принудительно после регистрации задать id на выборке:

```

val rop = thisApi().registerByRow(thisRow())
setVar("id", rop.get(_.id))

```

afterEdit()

Также необходимо добавить проверку, является ли строка пустой, в afterEdit().

Иначе load() внутри afterEdit() будет вызван по некорректного id пустой строки и будет вызвана ошибка.

Поддержание уникального id в нехранимых полях.

Без уникального id ряд системных операций не будет работать или будет работать некорректно, в том числе для работы onRefreshExt.

Необходимо понять, что на выборке является уникальным, в случае примера это ссылка соответствующая вагону: idFlyOverPos .

Теперь параметр id на выборке будет иметь не значение null в случае пустой строки и значение записи коллекции из БД, а значение idFlyOverPos.

А сам id записи коллекции будет храниться в поле case class'a idRecievTaskDet Это нужно учесть в:

- Структуре case class'a
- В переносе значений из записи из БД в case class для формирования onRefresh
- В условии по признаку пустая ли строка (теперь строка пустая, если idRecievTaskDet == None.
nl):

```

override def beforeEdit(): Unit = {
  if (getSelfVar("idRecievTaskDet").isNull) {
    regRow()
  }
}

```

- При сеттере в поле пустой строки в методе переноса умолчательных значений:

```

/**
 * insertByParent(ropParent) + сеттеры заполненных полей case class'a Row
 *
 * @param ropParent
 * @param row
 * @return
 */
def insertByParentAndRow(ropParent: Oil_RecievTaskApi#ApiRop, row: Row): ApiRop = {
  insertByParent(ropParent) :/ { rop =>
    //setter'ы для переноса умолчательных значений из Row
    row.getClass.getDeclaredFields.map(_.getName.ns).zip(row.productIterator.
    ↳to(scala.collection.immutable.IndexedSeq))
      .filterNot(field => saFieldsNotSetter.contains(field._1) || field._2.
    ↳asInstanceOf[Nullable[_ <: Any, _]].isNull)
      .foreach(field => {
        if (field._1 == sid.ns) {
          setAttrValue(rop, sidFlyOverPos, field._2)
        } else {
          setAttrValue(rop, field._1, field._2)
        }
      })
    })
  rop
}

```

- При сеттере в поле пустой строки после создания записи проставить параметр выборки idRecievTaskDet, вместо id, в id только что созданной записи:

```

val rop = thisApi().registerByRow(thisRow())
setVar("idRecievTaskDet", rop.get(_.id))

```

- При взятии параметра у дочерних выборок по super\$id -> super\$idRecievTaskDet
- В CWA управление свойством isEnabled у операций:

```

selection.opers().setEnabled("Delete",selection.canDelete && selection.getSelfVar(
    ↳"idRecievTaskDet").notNull())

```

- Удаление:

```

thisApi().delete(thisApi().load(getSelfVar("idRecievTaskDet").asNLong))

```

- thisRop()

```

thisApi().load(getSelfVar("idRecievTaskDet").asJLong)

```

- onInvalidateItem():

```

override protected def onInvalidateItem(): Unit = {
  if (!getSelfVar(thisApi().sidRecievTaskDet).isNull && thisRop() != null) {
    session.invalidateObject(thisRop())
  }
}

```

- учесть в onRefreshExt() в тексте запроса

Динамическое присоединение столбцов

Если такое отображение только для чтения и результат не использует значения, которые могут быть в кэше, тогда можно реализовать на реляционном запросе, иначе необходимо использовать инструменты DynMetaBuilder и DynRecBuilder.

Реляционная реализация

В этом случае необходимо собрать текст запроса selectStatement.

Ниже пример формирования одного из столбца:

```

s"""
,(select string_agg(cast(t.id as varchar), ', ')
  from Oil_Task t
 where t.idResource = ${rv.idResource()}
       and (t.dBegin < ${sMainFromTab.dEndTime} and t.dExec > ${sMainFromTab.dBegTime})
       and t.idObjectType = :${Oil_TaskMonitorPkg().sfltIdObjectType}
       and t.idStateMC >= 200
       and t.idDepOwner = :super${idGlobalDepOwner}
 group by t.idResource
 ) as \"${sNameFieldForGST[${rv.id()}]}\"
"""

```

Пример текста после подстановки значения:

```

(select string_agg(cast(t.id as varchar), ', ')
  from Oil_Task t
 where t.idResource = 115
       and (t.dBegin < '2023-08-25 12:00:00' and t.dExec > '2023-08-25 14:00:00')
       and t.idObjectType = 225
       and t.idStateMC >= 200
       and t.idDepOwner = 336
 group by t.idResource
 ) as "idFlyOverWay[13]"

```

Мы получим столбец с названием idFlyOverWay[13], результатом будет значение подзапроса.

Примечание

В примере выше у атрибута указывается его название, по которому в автм для всех атрибутов с таким названием можно будет задать параметры, и индекс в квадратных скобках, указывающий на конкретный атрибут.

Пример задания параметров для всех атрибутов idFlyOverWay

```
<attr name="idFlyOverWay">
  <style attr="sAvailStyle"/>
  <grid columnWidth="6" isColumnWidthFixed="true"/>
</attr>
```

Пример установки параметров атрибута по его индексу

```
for ((svCaption, nvIdx) <- Pro_ResLoadChartPkg().getPeriodCaptions(svExpandTypeMC,
→ dvPeriodBegin, dvPeriodEnd, periods).zipWithIndex) {
  selection.attrs(s""nWreQty[$nvIdx]""").caption = s"Плановая|${svCaption.get}"
  selection.attrs(s""nWreQty[$nvIdx]""").order = 100 + 100 * nvIdx
  selection.attrs(s""nResAvail[$nvIdx]""").isVisible = getVar("super$filter$Flt_
→ bShowResourceAvailability").asNNumber.nvl(0.nn).toBoolean
}
```

Примечание

В примере выше у атрибута задается бенд группа через символ „|“. Сначала идет наименование бенда „Плановая“, после наименование самого атрибута.

DynMetaBuilder и DynRecBuilder

Пример использования DynMetaBuilder и DynRecBuilder является отображение Card_ObjectAttr для объектных характеристик, генерируемый кодогенератором в Dvi.

Структура формирования:

Примечание

Recs(<объекты, представляющие строку (список row или case class)>) .extend(.build()) .foreach((row, builder) => (row, .build))

- Пример использования .extend()

Метод расширяет исходный набор данных, добавляя к ним данные о динамических столбцах

```
val dynMetaBuilder = getObjAttrDynMetaBuilder()
thisApi().refreshByParent(getIdMaster)
  .extend(dynMetaBuilder.build())
  .foreach{(rop, builder) =>
    val dynamicInfo = getDynamicObjAttr(rop, builder)
    (rop, dynamicInfo.build)
  }
```

Стоит понимать, что в примере foreach не привычный обходчик по коллекции, который ничего не возвращает, а отдельный метод для динамического формирования столбцов, который **должен вернуть результат выборки!**

В данном методе столбец получает построчно значения.

- Пример getObjAttrDynMetaBuilder

Здесь формируются метаданные о добавляемых столбцах: название столбца, тип данных, отображаемый заголовок колонки.

```
def getObjAttrDynMetaBuilder(): DynMetaBuilder = {
  val dynMetaBuilder = DynMetaBuilder()
  thisApi().jsonAttrsWithPrefix.foreach { case (rvAttr, sAttrPrefix) =>
    val svAttr = getJsonAttrName(rvAttr.id, sAttrPrefix)
    val attrClass = Btk_AttributeApi().getAttrValueClassById(rvAttr.id)
    // Добавляем метаданные об атрибуте
    dynMetaBuilder.add(svAttr, attrClass, rvAttr.sCaption)
    // для ссылочных атрибутов отображаем HL поля
    if (rvAttr.sType === AttrTypes.RefObject.toString.ns) {
      dynMetaBuilder.add(s"idValueHL[${rvAttr.id}]", classOf[String], rvAttr.
→sCaption)
    }
    // И добавляем скрытый столбец для редактора
    dynMetaBuilder.add(Btk_AttributeLib().getEditorAttrName(rvAttr.id, rvAttr.id.
→isNull), classOf[String], s"Редактор ${rvAttr.sCaption}")
  }
  dynMetaBuilder
}
```

- Пример getDynamicObjAttr

Здесь формируются данные строк для каждого добавленного атрибута

```
def getDynamicObjAttr(rop: Rop[_], builder: DynRecBuilder): DynRecBuilder = {
  val svLetterInRuleCaption = thisApi().getLetterInProcessRule(rop).sCaption
  builder.set(A.sLetterInProcessRule.name + "HL", svLetterInRuleCaption.get)
  builder
}
```

В данном примере для атрибута sLetterInProcessRule будет выведен дополнительный столбец с заголовком

В некоторых случаях требуется использовать методы DynMetaBuilder и DynRecBuilder вместе с классом AdditionalInfo. В таком случае вместо extend используется метод extend2.

- Пример использования .extend2()

Метод расширяет исходный набор данных, добавляя к ним данные о динамических столбцах и данные из класса AdditionalInfo

```
thisApi().refreshByParent(getIdMaster)
  .extend2(classOf[AdditionalInfo], dynMetaBuilder.build())
  .foreach{(rop, builder) =>
    val dynamicInfo = getDynamicObjAttr(rop, builder)
    (rop, getAdditionalInfo(rop.asInstanceOf[SomeApi#ApiRop]), dynamicInfo.
→build)
  }
```

,где SomeApi - какая-либо Api класса.

Динамическое изменение свойств avm

Для этого необходимо использовать следующий метод:

```
ru.bitec.app.gtk.gl.Rep#setMetaProp
```

Описание

Пример использования:

```
setMetaProp(attr.sSystemName.get //для какого поля меняется свойство
//какое свойство
//в данном случае свойство задает системное имя атрибута в выборке,
//которое хранит настройки типа редактора
, s"View.Representation.Attributes.Attribute.Editor.editorTypeAttr"меняется
, thisPkg.getSEditorAttrName(attr) //значения свойства
)
```

OnrefreshExt и значение даты с выборки

В onRefreshExt нет приведения /*NSDate*/, нужно писать через /*NSString*/.

В onRefreshExt в конструкции with объявляются поля, которые нужно получить с выборки (из кэша). Иногда требуется указать тип данных. Тип данных указывается в /*...*/, но парсер не знает NSDate, поэтому необходимо указать, как NSString и далее использовать в запросе cast(... as timestamp) или as date.

```
override protected def onRefreshExt: String = {
  s"""with t as ( select
    :id as id
    ,:idState as idState
    ,:gidSrc/*NSString*/ as gidSrc
    ,:idObjectType as idObjectType
    ,:idDepOwner as idDepOwner
    ,:idDischargePlace as idDischargePlace
    ,:idSegregationGroup as idSegregationGroup
    ,:idService as idService
    ,:idFlyOverWay as idFlyOverWay
    ,:idStateMC as idStateMC
    ,:dPlanEnd /*NSString*/ as dPlanEnd
    ,:dPlanBegin /*NSString*/ as dPlanBegin
  )
  SELECT
    t.id
    ,to_char(cast(t.dPlanEnd as timestamp) - cast(t.dPlanBegin as timestamp), 'dd д.␣
→hh24 ч.') as nPlanBusy
    ,t1.sHeadLine_dz as idStateHL
    ,t2.sHeadLine_dz as idObjectTypeHL
    ,t3.sHeadLine_dz as idDepOwnerHL
    ,t4.sHeadLine_dz as idDischargePlaceHL
    ,t5.sMnemoCode_dz as idSegregationGroupHL
    ,t6.sHeadLine_dz as idServiceHL
    ,t7.sHeadLine as gidSrcHL
    ,t8.sHeadLine_dz as idFlyOverWayHL
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
FROM t
  LEFT JOIN Btk_ClassState t1 on t.idState = t1.id
  LEFT JOIN Btk_ObjectType t2 on t.idObjectType = t2.id
  LEFT JOIN Bs_DepOwner t3 on t.idDepOwner = t3.id
  LEFT JOIN Bs_Placement t4 on t.idDischargePlace = t4.id
  LEFT JOIN Oil_SegregationGroup t5 on t.idSegregationGroup = t5.id
  LEFT JOIN Gds_Service t6 on t.idService = t6.id
  LEFT JOIN Btk_Object t7 on t.gidSrc = t7.gidRef
  LEFT JOIN Oil_FlyOverWay t8 on t.idFlyOverWay = t8.id
"""
}
```

Поиск отображения на выборке

```
selection.form.findSelection(...)
```

Ссылка

Полезный метод, который позволяет найти отображение по системному имени среди видимых в сессии.

Пример использования:

```
val sel = selection.form.findSelection(Bdg_ForecastAvi.card())
```

Пользовательская блокировка

Пользовательская блокировка включается при взаимодействии пользователя с интерфейсом в методе `Dvi#beforeEdit`, который является результатом кодогенератора.

Принудительно можно включать блокировку с помощью вызова метода:

- `Btk_FormSessionApi().lockObject(gid)`
- `Btk_FormSessionApi().lockObjectMulti(gida)`

Больше про пользовательскую блокировку можно узнать [здесь](#).

Объект класса в процессе создания и другие состояния объекта rop

Иногда, есть необходимость в `Avi` знать, что запись находится в процессе создания, т.е. не имеет реализации, как запись в таблице БД.

Это можно узнать по объекту записи `rop`:

- используйте метод `Avi thisRop()` чтобы получить объект `rop`, чья карточка открыта или на котором стоит фокус, если отображение `list`.
- если `rop.ropMode == InsertRopMode`, то объект находится в процессе создания, т.е. есть в кэше приложения, но не имеет реализации в таблице БД.

Где поле объекта `rop.ropMode` хранит в себе информацию состояния объекта. Есть и иные состояния объекта:

- `ReadRopMode`

- UpdateRopMode
- DeleteRopMode
- InsertRopMode

Пример кода из проекта:

```
//Документ должен быть закоммичен, чтобы были пройдены все соответствующие проверки,
// т.к. по закрытию card_ReadFromFile будет flush()
if (rop.ropMode == InsertRopMode) {
    throw ApplicationException("Для вызова операции необходимо, чтобы документ был заполнен и
↪сохранён.")
}
```

Как узнать, что выборка является главным меню или главной выборкой формы

В системе есть 3 типа форм:

- главная
- модальная
- MDI (отображается в качестве закладки на главной форме)

У каждой формы есть главная выборка. Флаг `selection.isMainOnForm` и указывает, что выборка - главная на форме.

Условие `application.mainSelection == selection` определит, является ли выборка главным меню.

Фильтрация в древовидном списке (отображение Tree классов-деревьев)

Как выводить ветку дерева до объекта, который удовлетворяет условию фильтрации

Решение такой задачи в каждом случае может быть индивидуальным.

Примеры решений:

Через денормализованную таблицу

Такое решение будет иметь те же преимущества, что денормализованная таблица, которая обеспечивает более быстрый доступ к данным по иерархичным связям, чем рекурсивные запросы.

Что такое Денормализация классов-деревьев и как её создать можно узнать *здесь*.

Пример условия:

```
t.id in (
    select denorm.idParent
    from Btk_TreeExampleDenormalization denorm
    join Btk_TreeExample ex on ex.id = denorm.idChild
    where (upper(ex.sCode) like upper('%'||:filter$flt_sCode||'%'))
)
```

Постройка запроса recursive снизу вверх

Строить запрос с нижних объектов, которые необходимо отфильтровать, дальше строить ветки вверх.

```
--рекурсия
with recursive RecursiveTreeExample as (
  --нижнии объекты дерева, которые фильтруются
  select id, idParent, sCode
  from Btk_TreeExample
  --условие
  where upper(sCode) like upper('%' || :filter$flt_sCode || '%')
  --union на случай, если в одной ветке несколько объектов подходят под фильтр
  union
  --постройка ветки вверх - присоединение родителей idParent
  select e.id, e.idParent, e.sCode
  from Btk_TreeExample e
  --присоединение родителей idParent
  join RecursiveTreeExample rc on e.id = rc.idParent
)
select id, idParent, sCode
from RecursiveTreeExample;
```

Примечание

PostgreSQL оптимизирует рекурсивные запросы, поэтому они могут быть достаточно эффективными, особенно для относительно небольших деревьев.

Практика SQL

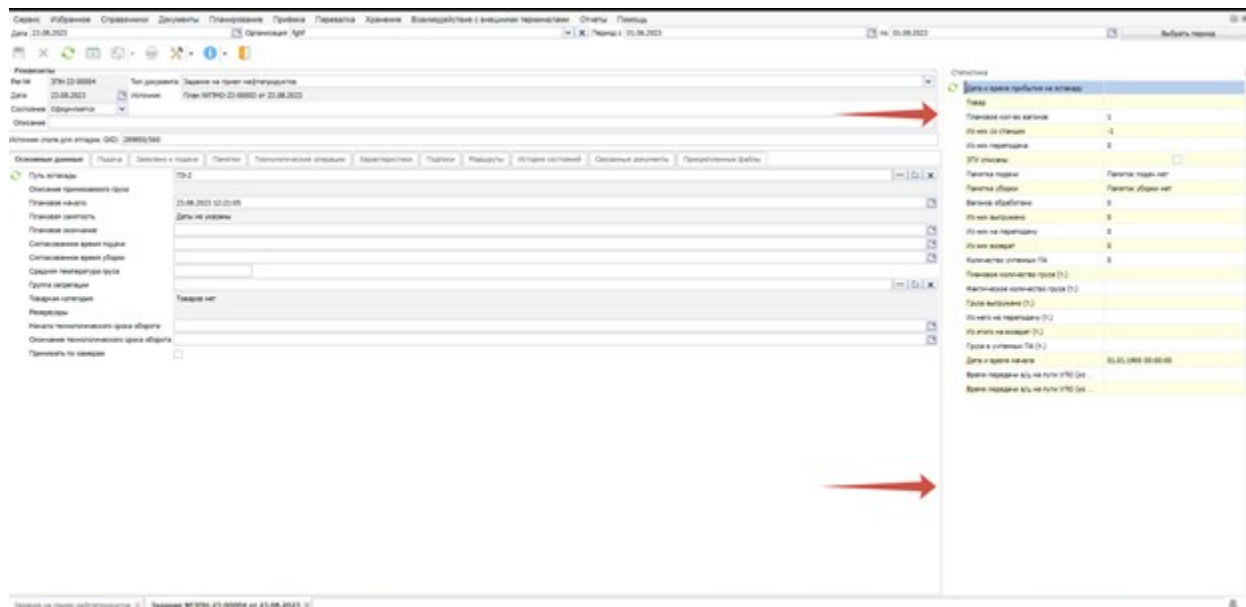
Функции getmnemocode, getheadline и getattribute

Читайте [здесь](#).

Практика avm, примеры интерфейсов

В данном разделе будет расписана некоторая практика, связанная с разметкой в avm. Будут рассмотрены различные реализации интерфейсов.

Присоединение нескольких отображений (одновременно с закладками и без)



Для этого в разметке нужно выбрать `tabDynamicComposer`, который позволяет присоединять, как с помощью `dynamicItems`, так и с помощью `tabItems`.

Примечание

Напомню, что:

- `dynamicItems` - обычное присоединение отображения с возможностью выбора выравнивания;
- `tabItems` - присоединение отображения, как вкладка.

```
<representation name="Card" editMode="edit" stdFilter.isAvailable="false">
  <layout>
    <tabDynamicComposer>
      <frame filter.isVisible="false">
        <card/>
      </frame>
      <dynamicItems>
        <dynamicItem selection="gtk-ru.bitec.app.oilOil_RecievTaskAvi"
        ↪representation="List_Statistics" align="right"/>
      </dynamicItems>
      <tabItems isVisible="true"
        selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"
        representation="List_Tab"
        selection.selectionAttr="SSELECTIONNAME"
        selection.representationAttr="SREPRESENTATIONNAME"
        selection.captionAttr="SCAPTION"
        selection.imageIndexAttr="NIMAGE"
        selection.paramsAttr="JSONPARAMS"/>
      </tabDynamicComposer>
    </layout>
  </representation>
```

(продолжается на следующей странице)

```
</representation>
```

Примечание

В примере разметки в `tabItems` присоединены вкладки, настроенные на типе объекта. Это часто встречающаяся практика, данный пример полезно понимать.

Как убрать кнопку сворачивания отображения в модальное окно

Чтобы убрать кнопку сворачивания отображения, присоединенного `dynamicItems`,  необходимо выключить `header` у `frame`:

```
<representation name="Card_Body" editMode="edit" stdFilter.isAvailable="false">
  <layout>
    <simpleComposer>
      <frame header.isVisible="false">
        <card/>
      </frame>
    </simpleComposer>
  </layout>
</representation>
```

Таким способом будет убран заголовок с кнопкой.

Заголовок можно вернуть с помощью конструкции `vGroup`.

```
<representation name="Card_Body" editMode="edit" stdFilter.isAvailable="false">
  <layout>
    <tabComposer>
      <frame filter.isVisible="false" toolBar.isVisible="false" header.isVisible=
→ "false">
        <card>
          <layout>
            <vBox>
              <vGroup caption="Задание на приём нефтепродуктов">
                <hBox>
                  <vBox>
                    <attr name="sRegNum"/>
                    <attr name="dReg"/>
                    <attr name="idStateHL"/>
                  </vBox>
                  <vBox>
                    <attr name="idObjectTypeHL"/>
                    <attr name="gidSrcHL"/>
                    <attr name="gidSrc"/>
                  </vBox>
                </hBox>
                <attr name="sDescription"/>
              </vGroup>
            </vBox>
          </layout>
        </card>
      </frame>
    </tabComposer>
  </layout>
</representation>
```

```

        </layout>
    </card>
</frame>
<tabItems isVisible="true"
    selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"
    representation="List_Tab"
    selection.selectionAttr="SSELECTIONNAME"
    selection.representationAttr="SREPRESENTATIONNAME"
    selection.captionAttr="SCAPTION"
    selection.imageIndexAttr="NIMAGE"
    selection.paramsAttr="JSONPARAMS"/>
</tabComposer>
</layout>
</representation>

```

Отображение с кнопкой сворачивания:

Задание на прием нефтепродуктов

Рег.№	ЗПН-23-00004	Тип документа	Задание на прием нефтепродуктов
Дата	23.08.2023	Источник	План №ПМО-23-00003 от 23.08.2023
Состояние	Оформляется		
Описание			

Отображение с выключенным header:

Рег.№	ЗПН-23-00004	Тип документа	Задание на прием нефтепродуктов
Дата	23.08.2023	Источник	План №ПМО-23-00003 от 23.08.2023
Состояние	Оформляется		
Описание			

Отображение с vGroup:

Задание на приём нефтепродуктов

Рег.№	ЗПН-23-00002	Тип документа	Задание на прием нефтепродуктов
Дата	07.06.2023	Источник	
Состояние	Оформляется	Источник (поле для отладки, GID)	
Описание			

Динамическая смена отображения

Эта тема отвечает на вопрос, как при изменении значения поля менять присоединяемое отображение.

Примечание

Так работает tabItems на типе объекта: при смене типа объекта набор вкладок в tabItems меняется на те, которые настроены на выбранном типе объекта.

Смена отображений при изменении значения фильтра

Пример реализации:

ru.bitec.app.oil.Oil_TaskMonitor.avm.xml - разметка отображения Card,

ru.bitec.app.oil.Oil_TaskMonitorAvi.Card - результат выборки отображения Card.

avm:

```
<representation name="Card">
  <filter name="Oil_TaskMonitorFilter">
    <macros name="DefFltReferenceMacro">
      <condition logicalOperator="and" id="attrs" isExpression="false">
        <filterAttr name="flt_idObjectType" attribute-type="Long" caption="Тип
↳ задания"
                    isLastInLine="false" order="10" editorType="edit" isVisible=
↳ "false"/>
        <filterAttr name="flt_idObjectTypeHL" attribute-type="Varchar" caption=
↳ "Тип задания"
                    isLastInLine="false" order="10.2" editorType="lookup">
          <editor>
            <lookup lookupQuery="gtk-Btk_ObjectTypeAvi#MainLookup"
↳ lookupKeyAttr="id"
                    lookupListAttr="sHeadLine" changeableAttr="flt_
↳ idObjectType"
                    isLookupLazyLoad="true"/>
          </editor>
        </filterAttr>
        <filterAttr name="flt_dBegin" attribute-type="Date" caption="Дата, время
↳ c" isLastInLine="false"
                    order="20" editorType="dateTimePick">
          <card controlWidth="60" isControlWidthFixed="true"/>
        </filterAttr>
        <filterAttr name="flt_dEnd" attribute-type="Date" caption="по"
↳ isLastInLine="true"
                    order="30" editorType="dateTimePick">
          <card controlWidth="60" isControlWidthFixed="true"/>
        </filterAttr>
      </condition>
    </macros>
  </filter>
  <layout>
    <tabDynDetComposer>
      <frame filter.isVisible="true">
        <card/>
      </frame>
      <tabItems selection="gtk-Oil_TaskMonitorAvi" representation="List_RecievTask
↳ " isVisible="false"/>
      <dynDetail masterAlign="top" detailAlign="client" selectionAttr=
↳ "sSelectionAttr"
                    representationAttr="sRepresentationAttr" paramsAttr="jsonParams"
↳ isVisible="true"/>
    </tabDynDetComposer>
```

(продолжается на следующей странице)


```

</layout>
<attributes>
  <attr name="sSelectionAttr" isVisible="false"/>
  <attr name="sRepresentationAttr" isVisible="false"/>
  <attr name="sCaptionAttr" isVisible="false"/>
  <attr name="nImageIndexAttr" isVisible="false"/>
  <attr name="jsonParams" isVisible="false"/>
</attributes>
</representation>

```

Это отображение Card.

В тэге `filter` определены фильтры. Панель фильтров включена свойством тэга `<frame filter.isVisible="true">`.

Присоединение отображений выполнено инструментом `tabDynDetComposer`, который позволяет присоединять как `tabItems` (закладками), так и `dynDetail` (аналогия `dynamicItem`). Закладка `tabItems` присоединена, как пустышка, чтобы заработал `tabDynDetComposer` (проблема была актуальна на 05.07.2023, обещали исправить).

В закладке `dynDetail` определены свойства, значения которых являются атрибутами отображения (см. перечень атрибутов отображения). Тем самым отображение Card, как результат выборки должно иметь настройку для `dynDetail`.

Для формирования такой выборки необходимо рассмотреть `Avi`:

```

trait List_CommonResource extends Default with List_CoreResource {
  override protected val sNameFieldForGST = "Brigade"
  override protected val sNameTableForGST = "Bs_Brigade"
}

trait Card extends Default with super.Card {
  //Типы объекта класса Oil_Task и соответствующие отображения к ним
  lazy val mapRepByOT: Map[NString, NString] =
    Map("Oil_TaskReceive".ns -> "List_RecievTask".ns
      , "Oil_TaskShip".ns -> "List_ShipTask".ns
      , "Oil_TaskCommon".ns -> "List_CommonTask".ns
    )

  @FlushBefore(mode = FlushBeforeMode.Disabled)
  override protected def onRefresh: Recs = {
    val idvOT = getVar("flt_idObjectType").asNLong
    if (idvOT.isNotNull) {
      val sRep = mapRepByOT.getOrElse(Btk_ObjectTypeApi().getMnemoCode(idvOT), throw_
↳ AppException("Не найдено отображение."))
      s"""
        select
          'gtk-Oil_TaskMonitorAvi' as sSelectionAttr
          , '$sRep' as sRepresentationAttr
          , '${Btk_ObjectTypeApi().getShortCaption(idvOT)}' as sCaptionAttr
          , cast(null as numeric) as nImageIndexAttr
          , '{}' as jsonParams
        """
    } else {

```

```

s"""
    select cast(null as int8) as id
      ,cast(null as numeric) as nOrder
      ,cast(null as varchar) as sCaptionAttr
      ,cast(null as numeric) as nIndexAttr
      ,cast(null as varchar) as sRepresentationAttr
      ,cast(null as varchar) as sSelectionAttr
      ,'{' as jsonParams
    where false
  """
}
}

override protected def selectStatement: JClob = s"""
}

```

У отображения есть фильтр по типу объекта `flt_idObjectType` с редактором `lookup`. При выборе различных типов объекта к `Card` должны присоединяться различные отображения (их соответствие описано в `Map mapRepByOT`).

Алгоритм метода `onRefresh`:

- Проверка, не пустое ли значение фильтра типа объекта

```

val idvOT = getVar("flt_idObjectType").asNLong
if (idvOT.isNull) {}

```

- Получение соответствующего отображения для выбранного типа объекта из `mapRepByOT`:

```

val sRep = mapRepByOT.getOrElse(Btk_ObjectTypeApi().getMnemonicCode(idvOT), throw
↳ AppException("..."))

```

- Формирование выборки:

`sSelectionAttr` - выборка, отображение которого присоединяется (в рассматриваемом случае все отображения принадлежат одной и той же выборке);

`sRepresentationAttr` - соответствующее отображение

`sCaptionAttr` - описание отображения

`nImageIndexAttr` - картинка отображения

`jsonParams` - json параметры

Смена отображений при изменении значения поля отображения

Если нужно менять отображения в зависимости от значения полей отображения `Card`, то нужно сделать промежуточное отображение с механикой, описанной выше (например, `List_Tab`), которое присоединить к `Card`.

Фиксация ширины столбца/поля и задание значения ширины

В карточке:

Необходимо для атрибута задать свойства в тэге `card`:

```
<attr name="sRegNum" caption="Пер.№" editorType="edit" order="10" isReadOnly="true">
  <card controlWidth="30" isControlWidthFixed="true"/>
</attr>
```

В списке:

Необходимо для атрибута задать свойства в тэге `grid`:

```
<attr name="sRegNum" caption="Пер.№" editorType="edit" order="10">
  <grid controlWidth="30" isControlWidthFixed="true"/>
</attr>
```

`controlWidth` - ширина поля

`isControlWidthFixed` - `true` - ширина будет зафиксирована, `false` - доступна для редактирования в интерфейсе.

Убрать описание у поля

Чтобы убрать заголовок у поля, писать `<arrt caption="">` неправильно.

Необходимо задать свойство `labelPosition`:

```
<attr name="idTankerHL" caption="Танкер" order="120.1">
  <editor labelPosition="none"/>
</attr>
```

Помимо значения `"none"` свойство `labelPosition` имеет значения:

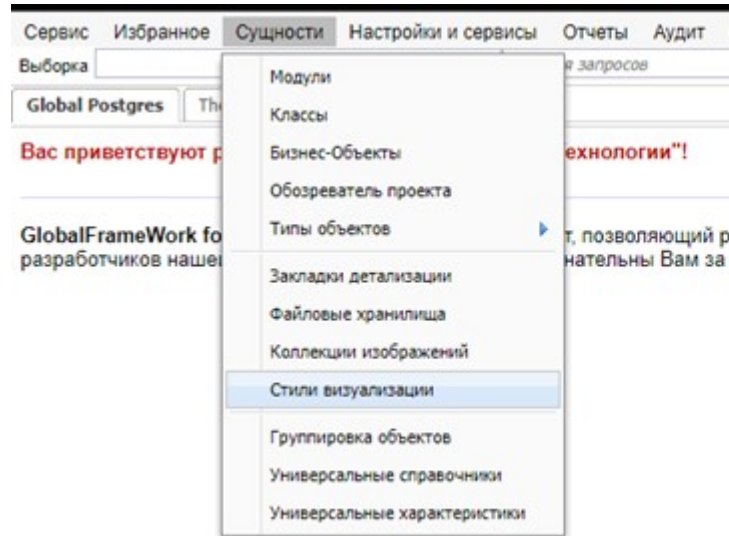
- `"right"` - справа
- `"left"` - слева
- `"above"` - сверху
- `"under"` - снизу

Как в списке покрасить ячейку

Для этого необходимо иметь дополнительный атрибут, в который будет записываться системное имя стиля. Обычно такой атрибут называют `sStyle`.

Далее необходимо указать, что столбец такого-то атрибута имеет стиль, описанный в атрибуте `sStyle`:

```
<attr name="nRating" order="20" caption="Результат">
  <style attr="sStyle"/>
</attr>
```



Стили можно посмотреть и создать в «Настройках систем»:

Практика odm

В данном разделе будет расписана некоторая практика, связанная с разметкой в odm.

Свойство not null для атрибута

В разметке odm нет настройки для того, чтобы указать атрибуту свойство not null.

Примечание

Свойство `isRequired=»true»` влияет на результат работы кодогенератора:

- реализуется настройка в avm для визуального обозначения поля обязательным
- в методе `Dpi.validateRequired()` реализуется проверка на заполненность обязательных полей.

Но генератор таблиц не смотрит на данную настройку и рассматриваемое поле в таблице не будет иметь признак not null.

Для этого необходимо реализовать sql-скрипт для задания данной настройки для поля:

```
<dbSchema>
  <scripts>
    <script name="isNotNull_sKeyResDepOwn0TArchive" version="1">
      <install>
        ALTER TABLE public.oil_task ALTER COLUMN skeyresdepownotarchive SET NOT_
        NULL;
      </install>
    </script>
  </scripts>
</dbSchema>
```

Практика безопасность

Защита от HTML инъекций

При построении HTML строки нужно экранировать все значения которые могли быть введены пользователем

Для экранирования значений можно использовать

1. `ru.bitec.app.btk.utils.HTMLUtils.escape` для вставки строк
2. Использовать `scala.XML` для построения HTML

```
val sName = ropUser.get(_sName) // интересное имя "Ваня</h1><script>alert('XSS')</script><h1>"

/* Произошла HTML инъекция
  <h1>Ваня</h1><script>alert('XSS')</script><h1></h1>
*/
val sHTML = s"""
  <h1>${sName}</h1>
"""

/* Нужно экранировать значения вставляемые в HTML-строку. Выведется как обычный текст
  <h1>Ваня&lt;/h1&gt;&lt;script&gt;alert('XSS')&lt;/script&gt;&lt;h1&gt;</h1>
*/
val sHTML_correct = s"""
  <h1>${HTMLUtils.escape(sName)}</h1>
"""

/* scala XML сама экранирует значения. Выведется как обычный текст
  <h1>Ваня&lt;/h1&gt;&lt;script&gt;alert('XSS')&lt;/script&gt;&lt;h1&gt;</h1>
*/
val HTML = <h1>{sName}</h1>.toString
```

Если нужно использовать пользовательский HTML. То его нужно отчистить от вредоносного кода

1. `ru.bitec.app.btk.utils.HTMLUtils.sanitize` Убирает весь JS, небезопасные элементы, и ссылки не на https ресурсы

```
/* HTML введенный пользователем
  
  
*/
val sHTML = ropDoc.get(_sDescription)

/* Безопасный для использования HTML
  
  
*/
val sHTML_correct = HTMLUtils.sanitize(sHTML)

// Предотвращение на вставке в бд
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
override def setDescription(rop: ApiRop, value: NString): Unit = {  
    super.setDescription(rop, HTMLUtils.sanitize(value))  
}
```

Практика код

Вычисление суммы без использования буфера

Пример, где подсчитывается сумма по записям коллекции Oil_PlanFactoryShipPos:

```
//получаем все записи коллекции по родителю (объект rop)  
Oil_PlanFactoryShipPosApi().byParent(rop)  
//получаем только значение поля nQtyLoad  
//в случае незаполненности поля необходимо иметь значение 0, иначе в дальнейшем сумма с_  
↪null = null  
.map(_.get(_nQtyLoad).nvl(0.nn))  
//сложение, результат которого будет обернут безопасной конструкцией Option  
.reduceOption(_ + _)  
//распаковка Option  
//если внутри был null или в коллекции не было записей, то вернётся указанное значение 0.  
↪nn  
.getOrElse(0.nn)
```

Группировка объектов с использованием null-типов

При группировке объектов коллекций, в которых используется null-типы нужно учитывать, что метод `groupBy` считает хэши группируемых объектов, поэтому перед группировкой, нужно убедиться, что параметр, используемый для группировки, не равен null.

Пример, с NPE:

```
List(  
    asd(123.n1, "asd".ns),  
    asd(None.n1, "asd".ns), // у NLong с underlying = null хэш не посчитается  
)groupBy(_.id).foreach { case (_, name) =>  
    println(name)  
}
```

Перед использованием `groupBy` необходимо отфильтровать значения или использовать метод `nvl`, либо группироваться по `Option[?]` использовав метод `option`.

Для дальнейшего использования Nullable, участвовавших в уникальном ключе есть несколько вариантов.

- Коллекции с сортировкой для простых Nullable

Пример:

```
val avIdEiToRop = ropa.view.map(r => r.get(_idExpenItem) -> r).to(mutable.  
↪SortedMap)  
avIdEiToRop.get(None.n1).foreach { ropInDim => }
```

При создании коллекций в scala с уникальным ключом возможны варианты использования:

- отсортированной коллекции, имеющей обычно префикс Sorted или Tree, например, `collection.SortedMap`, `collection.immutable.TreeMap`
- коллекции, для определения уникальности обращающейся к методу `hashCode` с сравнением результата для определения уникальности. Такие классы используются в часто вызываемых методах: фабричных (`mutable.Map.empty`, `mutable.HashMap`), группировки (`ropa.groupBy(_.get(_.gidObjCalc))`, `immutable.HashMap`), `toMap`, `toSet`.

Взамен применения Hash коллекций, вызывающих `hashCode`, ведущий к NPE, возможно использование Sorted, для чего важно наличие способа упорядочивания для типа (Ordering, что сейчас отсутствует для NGid): `ropa.view.map(r => r.get(_.gidObjCalc)).to(collection.SortedSet)` // compile error

- **Tuple с nvl и nullif**

Пример

```
for (((idvEI, gidvOC), ropa) <- ropa.groupBy(_ => (aro => aro.idExpenItem.nvl(0) -  
-> aro.gidObjCalc.nvl("0".ng)))) {  
  val idvUsedEI = idvEI.nullif(0)  
  val idvUsedOC = gidvOC.nullif("0".ng)  
}
```

В чем есть неудобство при более чем одном обращении к ключу.

- **case class с переопределённым hashCode**

Пример:

```
case class Key(gidObjCalc: NGid, idBisObj: NLong) {  
  override lazy val hashCode: Int = {  
    def hvl(p: Nullable[_], _) = if (p.isNull) -1.nl else p  
  
    util.hashing.MurmurHash3.orderedHash(Seq(hvl(gidObjCalc), hvl(idBisObj)))  
  }  
}  
  
case class Det()  
  
val avKeyToDet = mutable.Map.empty[Key, mutable.Buffer[Det]]  
// Заполнение avKeyToDet  
for ((vKey, avDet) <- avKeyToDet) {  
  processDim(vKey.gidObjCalc, vKey.idBisObj) // прямо без nullif  
}
```

В чем отсутствует недостаток постоянного написания `nullif` и `nvl`. Запись `hashCode` однотипная, шаблонная. В случае хэша `lazy val` всех входящих свойств важно указание как `val` констант.

Сравнение диапазона дат

Допустим документы имеют поля `dBeginDoc` и `dEndDoc`.

Фильтр имеет 2 поля `dBeginFilter` и `dEndFilter`.

Чтобы найти все документы, которые началом или окончанием входят в диапазон, указанный в фильтре, нужно использовать следующее условие:

```
dBeginDoc <= dEndFilter && dEndDoc >= dBeginFilter
```

.distinct или .toSet для scala-коллекций и особенности применения

Если необходимо в scala-коллекции держать только уникальные объекты, можно использовать методы:

- `.distinct`
- `.toSet`

Перед использованием метода `.distinct` scala-коллекцию необходимо подготовить: убрать значения `null`. Иначе в ходе выполнения программы выпадет ошибка `java.lang.NullPointerException`.

Пример использования и демонстрация поведения методов:

```
test("distinctOrToSet") {  
  
    val data: Seq[NString] = Seq("h".ns, "i".ns, "i".ns, None.ns, None.ns)  
  
    println("Результат работы .distinct с null внутри scala-коллекции:")  
    try {  
        println(data.distinct)  
    } catch {  
        case e: Throwable => println("Ошибка:\n" + "java.lang.NullPointerException")  
    } finally {  
  
        println("\nРезультат работы .filter(_ != null).distinct:")  
        println(data.filter(_ != null).distinct)  
  
        try {  
            println("\nРезультат работы .toSet с null внутри scala-коллекции:")  
            println(data.toSet)  
        } catch {  
            case e: Throwable => println("Ошибка:\n" + "java.lang.NullPointerException\n")  
        }  
    }  
}
```

Результат:

Примечание

Результат работы `.distinct` с null внутри scala-коллекции: Ошибка: `java.lang.NullPointerException`

Результат работы `.filter(_ != null).distinct`: `List(h, i)`

Результат работы `.toSet` с null внутри scala-коллекции: `Set(h, i, Null)`

`immutable.Map.builder` **вместо** `mutable.Map`

Если по результату сформированной Map она больше не изменяется, то для формирования лучше использовать конструктор `immutable.Map.builder`, чем `mutable.Map`.

Пример:

```
val map = Map.newBuilder[NString, NString]
map += Map("1" -> "11")
map += "2" -> "22"
map.result() //Map("1" -> "11", "2" -> "22")
```

Признак наличия модуля на проекте

```
val isInstallProModule = session.sbtClassLoader.getModuleMap.containsKey("pro")
```

Вернёт true, если такой модуль есть в проекте, иначе false.

Предупреждение

Поддержка этого метода не гарантируется.

Применение ASQL/ ASelect/ OQuery/ TxIndex/ refreshByParent и byParent

Про данные инструменты можно почитать *здесь*.

ASQL

Выполнение реляционного запроса на чтение.

ASQL

Внимание

Вызывает транзакцию в БД, не учитывая значения в кэше.

Удобен для получения значения по одному столбцу результата запроса.

```
val idaWagonByTrain = ASQL"""
  select string_agg(cast(rwt.idWagon as varchar), ', ')
  from Rzd_TrainWagon rwt
  where rwt.idTrain = $idpTask
  """
//берется 1ый столбец результата запроса, как NString
//если результат запроса вернул несколько строк, то будет взята первая в конструкции
↪ Option,
// но предполагается, что результат вернёт максимум 1 строку
.as(nStr(1).singleOpt)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
//если Option пустой, т.е. в результате запроса не было строк, то вернётся None.ns  
.getOrElse(None.ns)
```

`$idpTask` это подстановка значения из переменной `scala idpTask` в запрос связанной переменной (binding).

Внимание

Нельзя использовать внутри цикла. Это приведёт к многочисленным транзакциям в БД.

Вместо использования **ASQL** внутри цикла с множественными транзакциями в БД нужно перед циклом одной транзакцией сформировать массив данных, который дальше будет использоваться внутри цикла.

ATSQL

Выполнение реляционного запроса с изменением данных или блокировками.

ATSQL

Используется редко, в основном в ядровых процедурах, потому что минует серверную логику, записи в системные миксины, ведение аудитов и другое.

ASelect

Выполнение реляционного запроса на чтение/запись.

ASelect

Внимание

Вызывает транзакцию в БД, не учитывая значения в кэше.

Используется в основном для чтения, потому что при изменении данных минует серверную логику, записи в системные миксины, ведение аудитов и другое.

Удобен для получения значений по нескольким столбцам результата запроса.

```
val mapPerson: Map[NLong, NString] = new ASelect {  
  val sCode = asNString("sCode")  
  val id = asNLong("id")  
  SQL"""  
    select p.id  
          ,p.sCode  
    from Bs_Person p  
    where idObjectType = $idvObjectType  
  """  
}.map { rv => //rv представляет собой одну строку результата запроса  
  rv.id() -> rv.sCode()  
}.toMap
```

`$idvObjectType` это подстановка значения из переменной `scala idvObjectType` в запрос связанной переменной (`binding`).

Внимание

Нельзя использовать внутри цикла. Это приведёт к многочисленным транзакциям в БД.

Вместо использования `ASelect` внутри цикла с множественными транзакциями в БД нужно перед циклом одной транзакцией сформировать массив данных, который дальше будет использоваться внутри цикла.

Когда использовать ASQL, а когда ASelect

ASQL удобен для получения значения по одному столбцу результата запроса.

ASelect удобен для получения значений по нескольким столбцам результата запроса.

Подстановка связанных переменных (`binding`)

Актуально для инструментов ASQL, ATSQL, ASelect.

Примечание

Использование ASQL с подстановкой связанных переменных является полезной практикой, потому что запрос остаётся неизменным, меняется лишь значение параметра. Тем самым запрос не воспринимается системой, как новый, и будет записан в системную таблицу запросов единожды, что не приводит к распуханию БД.

Инструмент `ASQL""<текст запроса>""` подставляет бинды с учётом типа данных переменной.

- Если бы была подстановка `NString`, то ASQL сам обернул бы значение в одинарные кавычки, т.е. нет необходимости их указывать вручную.
- Если `NString` подставляется в текст `s""<текст запроса>""`, то одинарные кавычки необходимо указывать вручную.

Если текст запроса для `ASQL""<текст запроса>""` собирается динамически средствами `scala`, в том числе название таблицы для `select` формируется переменной, то его необходимо подставлять через `#$bind`, чтобы ASQL не обернул подставляемое значение в одинарные кавычки.

```
val sNameTb = {
  if (a = 1) "Bs_Goods".ns
  else "Bs_Person".ns
}

val ida: List[NLong] = ASQL""
  select t.id
  from #sNameTb
  where idObjectType = $idvObjectType
"".as(nLong(1).*)
```

Существует иной формат binding, когда запрос формируется в переменной String/NString, где указываются ключи для подстановки значения, карта подстановки указывается в .on()

```
val svRequest =
  s"""
    select t.id
    from Bs_Goods
    where idObjectType = {idObjectType}
  """
ASQL(svRequest).on("idObjectType" -> idvObjectType)

new ASelect {
  val sCode = asNString("sCode")
  val id = asNLong("id")
  SQL(s"""
    select p.id
           ,p.sCode
    from Bs_Person p
    where idObjectType = {idObjectType}
  """)
  .on("idObjectType" -> idvObjectType)
}.map { rv => //rv представляет собой одну строку результата запроса
  rv.id() -> rv.sCode()
}.toMap
```

Может быть полезно, если запрос собирается в переменной String/NString.

OQuery

- Объектный запрос
- Синтаксис схож с реляционным запросом
- Возвращает результат с учётом кэша
- Результат - список Rop, что удобно для использования методов, которым требуется Rop

OQuery

Пример запроса:

```
new OQuery(Bs_GoodsAta.Type) {
  where(t.sSystemName === spMnemonicCode)
}.toVector
```

Особенности, которые нужно знать про OQuery:

Условие применяется для запроса в БД

Условие применяется для запроса в БД. Поэтому поиск записей идёт в БД без учёта кэша, а результат возвращается с учётом кэша. Пример:

```
val idvGds = 121.nn
val ropGds = Bs_GoodsApi().load(idvGds)
//ТМЦ переводится в состояние Отменено (0)
Bs_GoodsApi().setidState(ropGds, idvStateCancel)
//Поиск неотмененных ТМЦ с условием по категории
new OQuery(Bs_GoodsAta.Type) {
    where(t.idCategory = idvGdsCategory
          and t.idStateMC >> 0.nn)
}.toVector
```

В результате OQuery будет запись Bs_Goods с id = 121, при чём с состоянием «Отменено» , потому что:

- в БД эта запись не Отменена (t.idStateMC > 0) , а изменение по отмене записи находится в кэше и не синхронизировано с БД
- результат OQuery возвращается с учётом кэша , поэтому на запись с id = 121 применён кэш - смена состояния

Решить можно 2 способами:

```
/** Вариант 1 - синхронизация с БД */
//синхронизация с БД
session.flush()
//Поиск неотмененных ТМЦ с условием по категории
new OQuery(Bs_GoodsAta.Type) {
    where(t.idCategory = idvGdsCategory
          and t.idStateMC >> 0.nn)
}.toVector

/** Вариант 2 - вынос условия из OQuery, фильтрация результата OQuery */
//Поиск неотмененных ТМЦ с условием по категории
new OQuery(Bs_GoodsAta.Type) {
    where(t.idCategory = idvGdsCategory)
}
//вынос условия из OQuery
.filter(_.get(_.idStateMC) > 0.nn)
```

Предупреждение

1. Вариант 1 - синхронизация с БД с помощью session.flush() имеет недостаток: у пользователя пропадает возможность отменить изменения на выборке.
2. Вариант 2 - вынос условия из OQuery, фильтрация результата OQuery имеет недостаток: OQuery в память выгружает больше записей из БД, что может требовать приемов по работе с большими данными (см. раздел «Разработка под высокую нагрузку»).

Для предварительной загрузки записей в кэш необходима инициализация OQuery

Про предварительную загрузку записей в кэш можно прочитать *здесь*.

Чтобы данные были загружены в кэш, необходима инициализация OQuery , которая происходит при использовании обходчика `foreach` или преобразовании OQuery к иной коллекции:

```
//Предварительная выгрузка записей в кэш
new OQuery(Bs_GoodsAta.Type) {
    where(t.sSystemName === spMnemonicCode
        and t.idStateMC >> 0.nn)
}
//для инициализации OQuery
.toVector
```

Транзакционный индекс

- Объектный запрос
- Поиск по одному полю по условию равенства
- Нет возможности искать по составному индексу из нескольких полей класса
- Ищет и возвращает результат с учётом кэша
- Результат - список Rop, что удобно для использования методов, которым требуется Rop

Транзакционный индекс.

Пример запроса:

```
//Объявляется обычно в Api класса соответствующего Ata (у примера Bs_GoodsApi)
lazy val idxCategory = TxIndex(Bs_GoodsAta.Type)(_.idCategory)

/** Поиск неотмененных ТМЦ по категории */
def byCategoryNotCanceled(): Iterable[ApiRop] = {
    idxCategory.byKey(idvGdsCategory)
    .filter(_.get(_.idStateMC) > 0.nn)
}
```

Если нужен запрос по нескольким условиям, то используйте индекс по полю, который вернёт меньшее кол-во записей в память, после чего примените остальные условия используя `.filter()`.

Если транзакционный индекс используется в цикле, то необходимо предварительно загрузить результаты в память по множеству ключей. Об этом можно почитать *здесь*.

refreshByParent и byParent

- Объектный запрос
- Имеется только у коллекций
- Поиск по ссылочному полю на мастера
- Возвращает результат с учётом кэша
- Результат - список Rop, что удобно для использования методов, которым требуется Rop

- **refreshByParent** инвалидирует записи из БД и возвращает актуальные данные, гарантированно делает транзакцию в БД.
- **byParent** возвращает данные из памяти, если они там были, иначе первично получает актуальные данные из БД, делая транзакцию.
- Перед использованием **byParent** в цикле для предварительной загрузки в кэш можно использовать **0Query** (*предварительная загрузка*.)

Про разницу между **refreshByParent** и **byParent** можно почитать [здесь](#).

Какой инструмент использовать

Загрузка множества объектов

Если нужны записи с учётом кэша, то используйте:

- объектный запрос
- реляционный запрос с предварительным `session.flush` (не для построения выборок)

Предупреждение

1. Синхронизация с БД с помощью `session.flush()` имеет недостаток: у пользователя пропадает возможность отменить изменения на выборке.
2. Для построения выборок, в которых ведутся изменения объекта, используйте объектный запрос.

Практики при разработке документов

Одм документа

- Для документов создаем отдельную директорию (пакет), чтобы все коллекции были тоже в ней
- У полей с датами в названии системного имени НЕ пишем слово Date
пример:
dReg
dDoc
dExec \
- Для поля с датой регистрации, ставим значение по умолчанию текущую дату
`defaultValue="sysdate"`
- Для полей с номерами, датой рег, состоянием и номером состояния устанавливаем `isCopyInCopyObject="false"`, чтобы значения не копировались при копировании
- Для полей с номером, датой и типом объекта устанавливаем свойство `isHeadLine="true"`, так как они в большинстве случаев участвуют в вычислении заголовка
- Добавляем поле с номером состояния, не видимое

```
<attr name="idStateMC" attribute-type="Number" caption="Номер состояния" order=
↪ "80" type="basic"
    isVisible="false" isCopyInCopyObject="false" isStateMC="true">
```

(продолжается на следующей странице)

```
<mnemoCodeColumn/>
</attr>
```

- Большинство полей можно взять из шаблона `bs\src\main\resources\META-INF\attribute-template.xml`
Доп. поля связанные со складом и тмц `stk\src\main\resources\META-INF\attribute-template.xml`
- Для денежных полей с суммами и ценами устанавливаем сразу денежный редактор `editorType="currency"`
- Для булевых полей ставим значение по умолчанию 0 `defaultValue="0"` и тип редактора галку `editorType="check"`
- Добавляем скрипты регистрации состояний, закладок и типов документа (вместо `Demo_Doc` напишите свой класс)

```
<dbData>
  <script name="regObjectType" version="1">
    <depends>
      <dep on="Demo_Doc.regTab"/>
      <dep on="Demo_Doc.regState"/>
    </depends>
    <install>Demo_DocApi.regObjectType(false);</install>
  </script>
  <script name="regState" version="1">
    <install>Demo_DocApi.regState();</install>
  </script>
  <script name="regTab" version="1">
    <install>Demo_DocApi.regTab();</install>
  </script>
</dbData>
```

- Если нужны прикрепленные файлы устанавливаем свойство в шапку `attachType="simple"` или `attachType="versioned"` для версионного режима
- Если нужна настройка по счетам учета, то добавляем в коллекции настройку `<var-collection name="Bs_AccObjSetting" ref.attr="gidSrc" cascadeOnDelete="true"/>`

Odm коллекции документа

- Наименование коллекций, как правило, содержит имя документа (может быть и сокращенное) и суффикс `Det` или какой то другой, например `Demo_DocDet`
- Порядковый номер для позиций следует именовать как `nRow` - № п/п и делать его числовым, наименование `nOrder` путает с `Bs_Order`. Его не следует делать Autonumbering, так как нумерация срабатывает на сохранение. Нумерацию можно сделать руками в `Api.insertByParent` (ниже будет пример)

Арі документа:

- Вместо `ListBuffer` следует использовать `ArrayBuffer`, так как он меньше занимает места в памяти
- Переопределяем `calcHeadLine`, для вычисления заголовка с учетом типа документа, номера и даты

```
override def calcHeadLine(idp: NLong): NString = {
  var svHeadLine = NString()
  if (idp.isNotNull) {
    load(idp) :> { aro =>
      if (aro != null) {
        svHeadLine = s"${Btk_ObjectTypeApi().getShortCaption(aro.idObjectType).nvl(Btk_
→ClassApi().getHeadLine(aro.idClass))} №" ++ aro.sNumDoc.nvl("") ++ " от " ++ aro.dDoc.
→toString("dd.MM.yyyy").nvl("")
      }
    }
  }
  svHeadLine
}
```

- Переопределяем `insert` для установки типа объекта по умолчанию и других атрибутов (например валюта и тип курса) при создании

```
override def insert(): ApiRop = {
  val rop = super.insert()
  //установка типа объекта по умолчанию
  setIdObjectType(rop, Btk_ObjectTypeApi().getDefaultObjType(idClass))

  //установки даты документа без времени
  //setDoc(rop, NDate.now().truncate())

  //установка нац. валюты Cur_Currency
  //setIdCur(rop, Cur_CurrencySettingsApi().getIdCurrencyMain)
  //установка типа курса Cur_CurrencyRateType
  //setIdCurRateType(rop, Cur_CurrencySettingsApi().getIdCurrencyRateType)

  //установка типа налогообложения Tax_TaxType
  //setIdTaxType(rop, Tax_TaxTypeApi().getDefaults)
  //установка ставки налога по умолчанию Tax_TaxRate
  //setIdVATRate(rop, Tax_TaxRateApi().getDefaults)

  //установка текущего пользователя Btk_User
  //setIdUser(rop, Btk_UserApi().getCurrentUserID)
  //установка текущего физ. лица Bs_Person
  //setIdPerson(rop, Bs_PersonApi().getCurrentPersonID)
  //установка текущего сотрудника Bs_Employee
  //setIdEmployee(rop, Bs_EmployeeApi().getIdCurrentEmployee)

  rop
}
```

- Переопределяем `delete`, чтобы нельзя было удалить документ, если он находится не в состоянии Оформляется

```

override def delete(rop: ApiRop): Unit = {
  if (rop.get(_idStateMC).isDistinct(100.nn))
    throw AppException("Ошибка. Удаление документа возможно только в состоянии \
↪ "Оформляется\".")
  //Удаление из журнала с признаками проведения в учетах
  //Bs_FlagAccKindBookDocApi().delByObj(rop.gid)
  //Удаление из журнала связанных документов
  //Bts_DocLinkApi().delByDoc(rop.gid)
  super.delete(rop)
}

```

- Если есть настройка по счетам учета, то на установку типа объекта и организации прописываем установку счетов учета для документа

```

override def setidDepOwner(rop: ApiRop, value: NLong): Unit = {
  super.setidDepOwner(rop, value)
  Bs_AccObjSettingApi().registerAccObjSetByObjectForDoc(
    gidpObject = rop.gid
  )
  //если есть КПП организации, то устанавливаем его
  //setidSelfKpp(rop, Bs_DepOwnerApi().getIdDefKpp(value))
}

override def setidObjectType(rop: ApiRop, value: NLong): Unit = {
  super.setidObjectType(rop, value)
  Bs_AccObjSettingApi().registerAccObjSetByObjectForDoc(
    gidpObject = rop.gid
  )
}

```

- Для setidState в прикладной логике надо использовать номера состояний, а на их id.
Пример действий на перевод состояний:

```

//проверки для документа при выполнении
protected def validateOnAccept(rop: ApiRop): Unit = {
}

//проверки для документа при откате
protected def validateOnDecline(rop: ApiRop): Unit = {
  //проверка на наличие проведения в бух., нал., упр. учетах
  //Bs_FlagAccKindBookDocApi().validateObj(rop.gid)
}

//действия на выполнен
protected def accept(rop: ApiRop): Unit = {
  //если не заполнена дата исполнения, то заполняем ее текущей датой
  //if (rop.get(_dExec).isNull)
  //  setdExec(rop, NDate.now())
}

//действия на откат из выполнен
protected def decline(rop: ApiRop): Unit = {

```

(продолжается на следующей странице)

```

}

override def setidState(rop: ApiRop, value: NLong): Unit = {
  //номера состояний
  //из какого
  val nvStateFrom = rop.get(_.idStateMC) //если нет поля можно использовать Btk_
  ↪ ClassStateApi().getOrder(rop.get(_.idState))
  //в какое
  val nvStateTo = Btk_ClassStateApi().getOrder(value)
  //Должно выполняться в самом начале. На понижение состояния (с учетом состояния в
  ↪ настройке) - проверка проведения в бух. учете, при синхронном проведении, откат
  ↪ проводок
  Bs_TransPkg().setidStateBefore(rop, rop.get(_.idObjectType), nvStateFrom, nvStateTo)

  //переход в выполнен (вверх по состоянию)
  if (nvStateTo >= 300.nn && nvStateFrom < 300.nn) {
    //проверки для документа при выполнении
    validateOnAccept(rop)
    //действия на выполнен
    accept(rop)
  } else if (nvStateTo < 300.nn && nvStateFrom >= 300.nn) {
    //проверки для документа при откате
    validateOnDecline(rop)
    //действия на откат из выполнен
    decline(rop)
  }

  super.setidState(rop, value)

  //Должно выполняться в самом конце. На повышение состояния (с учетом состояния в
  ↪ настройке): при синхронном проведение - выполнение TXO, при отложенном - добавление в
  ↪ очередь отложенного проведения документов
  Bs_TransPkg().setidStateAfter(rop, rop.get(_.idObjectType), nvStateFrom, nvStateTo)
}

```

- Различные проверки должны выполняться, как правило, в начале перевода состояния, чтобы было как можно меньше логики выполнено, до получения ошибки
- Если у документа есть коллекции, а у коллекций есть еще коллекции, то на перевод состояния сначала можно выполнить загрузку в кэш

```

/**
 * загрузка в кэш самого документа с коллекциями
 *
 * @param idap
 */
protected def loadDocWithCollections(idp:NLong): Unit = {
  new OQuery(entityAta.Type) {
    where(t.id === idp)
    batchIn(Demo_DocDetAta, Demo_DocDetItemAta, Stk_OperationAta)
  }.foreach { rop => }
}

```

- Создаем метод `regState` для регистрация состояний (здесь просто написан пример)

```
def regState(): Unit = {
    session.commit()
    Btk_Pkg().setRWSharedUOWEditType()
    //первоначальное состояние
    Btk_ClassStateApi().register(
        idpMasterClass = idClass,
        spSystemName = "Create",
        spCaption = "Оформляется",
        bpStartState = 1.nn,
        npOrer = 100.nn
    )
    session.flush()
    Btk_ClassStateApi().register(
        idpMasterClass = idvClass,
        spSystemName = "Annulled",
        spCaption = "Аннулирован",
        bpStartState = 0.nn,
        npOrer = 50.nn)
    Btk_ClassStateApi().register(
        idpMasterClass = idClass,
        spSystemName = "Agreed",
        spCaption = "Согласуется",
        bpStartState = 0.nn,
        npOrer = 200.nn)
    Btk_ClassStateApi().register(
        idpMasterClass = idClass,
        spSystemName = "Executed",
        spCaption = "Выполнен",
        bpStartState = 0.nn,
        npOrer = 300.nn
    )
    session.commit()
}
```

- Создаем метод `regTab` для регистрация закладок (здесь просто написаны примеры)

```
//регистрация закладок
def regTab(): Unit = {
    session.commit()
    Btk_Pkg().setRWSharedUOWEditType()

    Btk_TabApi().register(
        spCaption = "Позиции",
        spSel = "gtk-Demo_DocDetAvi",
        spRep = "List_idDoc",
        idprefClass = idClass
    )

    Btk_TabApi().register(
        spCaption = "Прикрепленные файлы"
        , spSel = "gtk-Btk_AttachItemAvi"
        , spRep = "List_SimpleAttach"
```

(продолжается на следующей странице)

```

    , idprefClass = idClass
)

Btk_TabApi().register(
    spCaption = "Объектные характеристики",
    spSel = "gtk-Demo_DocAvi",
    spRep = "Card_ObjectAttr",
    idprefClass = idClass
)

Btk_TabApi().register(
    spCaption = "Связанные документы"
    , spSel = "gtk-Bts_DocLinkAvi"
    , spRep = "Tree"
    , idprefClass = idClass
)

Btk_TabApi().register(
    spCaption = "Проводки",
    spSel = "gtk-Act_TransAvi",
    spRep = "List_gidDoc",
    idprefClass = idClass
)

Btk_TabApi().register(
    spCaption = "Настройка счетов учета",
    spSel = "gtk-Bs_AccObjSettingAvi",
    spRep = "List_gidSrc",
    idprefClass = idClass
)
session.commit()
}

```

- Создаем метод `regObjectType` для регистрация типов объекта (здесь просто написан пример)

```

//регистрация типов
def regObjectType(bpNeedRefresh: Boolean = false): Unit = {
    session.commit()
    Btk_Pkg().setRWSharedUOWEditType()
    //параметр bpNeedRefresh = false нужен, чтобы не затирать данные на проектных базах,
    ↪ если такого типа объекта нет, то он будет создан
    //если хотите все перезаписать, то выполните в Jexl скрипте Demo_DocApi.
    ↪ regObjectType(true);
    if (bpNeedRefresh || Btk_ObjectTypeApi().findByMnemonicCodeAndClass("Demo_DocObjType",
    ↪ idClass).isNull) {
        //регистрируем тип
        val idvOT = Btk_ObjectTypeApi().register(spCode = "Demo_DocObjType"
            , spCaption = "Тип А"
            , spShortCaption = "Документ А"
            , idpRefClass = idClass
            , bpIsDefault = 1.nn)
    }
}

```

```

//ищем закладку и привязываем к типу
var idvTab = Btk_TabApi().findByMnemonicCode("Demo_DocAvi.List_idDoc")
if (idvTab.isNotNull)
  Btk_ObjectTypeTabApi().registerTab(
    idpObjectType = idvObjectType
    , idpTab = idvTab
    , spCaption = "Позиции"
    , npOrder = 10.nn
  )

idvTab = Btk_TabApi().findByMnemonicCode("Demo_DocAvi.Card_ObjectAttr")
if (idvTab.isNotNull)
  Btk_ObjectTypeTabApi().registerTab(
    idpObjectType = idvObjectType
    , idpTab = idvTab
    , spCaption = "Характеристики"
    , npOrder = 20.nn
  )

//регистрируем переходы состояний для типа
Btk_StateChangeApi().registerForObject(idvOT, List(
  "Create" -> "Agreed",
  "Agreed" -> "Executed",
  "Agreed" -> "Create",
  "Executed" -> "Create",
  "Executed" -> "Agreed"
))
}
session.commit()
}

```

Api коллекции

- Переопределяем insertByParent для установки порядкового номера при создании

```

override def insertByParent(ropParent: AnyRop): ApiRop = {
  val nvRow = byParent(ropParent).map(_.get(_).nv1(0.nn)).reduceOption(_ max _).
  ↳getOrElse(0.nn) + 1.nn
  super.insertByParent(ropParent) :/ { rop =>
    setnRow(rop, nvRow)
    rop
  }
}

```

Avi документа:

для отображения List

- Переопределяем `onRefreshItem`, чтобы выполнялся запрос из `selectStatement`

```
override protected def onRefreshItem: Recs = {  
  prepareSelectStatement(s"$t.$getPKFieldName = :$getPKFieldName")  
}
```

- Переопределять `onRefreshExt` в списках не нужно, так как весь запрос должен быть в `selectStatement`
- Переопределяем CWA, чтобы дописать блокировку кнопки удаления, если документ находится не в состоянии оформления

```
override def checkWorkability(): Unit = {  
  super.checkWorkability()  
  val bvR0 = getSelfVar("idStateMC").asNNumber.isDistinct(100.nn)  
  opers().setEnabled("Delete", selection.canDelete && !getVar("ID").isNull && !bvR0)  
}
```

- Если нужно передать параметры в карточку при создании из списка, надо переопределить метод `insert_Params`, и затем обработать данный параметр в отображении для карточки в методе `onInsertItem`.

Пример передачи организации из фильтра списка.

```
override def insert_Params(): Map[String, Any] = {  
  Map("idDepOwner#" -> getVar("flt_idDepOwner").asNLong)  
}
```

- Если нужно сделать фильтр по состоянию или типу, надо добавить на `beforeFirstOpen` переменные со значением `id` нашего класса, чтобы работали стандартные вып. списки для типа и состояния

```
override protected def beforeFirstOpen(): Unit = {  
  super.beforeFirstOpen()  
  addVar("idStateClass#", thisApi().idClass, FieldType.ftInteger) // для фильтра по_↵  
↵ состояние  
  addVar("idObjTypeClass", thisApi().idClass, FieldType.ftInteger) // для фильтра по_↵  
↵ типу  
}
```

- Для установки значений по умолчанию при открытии для полей в фильтре с датами и организацией, значения которых будут браться из глобального фильтра, надо использовать значения по умолчанию через `Avm`, а не через `beforeFirstOpen`

для отображения Card - шапки

- Переопределяем `onInsertItem` для установки организации из глобального фильтра при создании

```
override protected def onInsertItem(): Unit = {
    super.onInsertItem()
    //если был передана организация через параметр
    val idvDepOwner = getSelfVar("idDepOwner#").asNLong.nvl(getVar("super
→$idGlobalDepOwner").asNLong)
    if (idvDepOwner.isNotNull) thisApi().setidDepOwner(thisRop(), idvDepOwner)
}
```

- Если есть закладки или детальные формы, то переопределяем `refresh`, чтобы при нажатии обновить обновлялись детали тоже

```
override def refresh(): Unit = {
    super.refresh()
    selection.refreshDetails()
}
```

- Переопределяем `saveForm`, чтобы прописать обновление карточки, если есть автономерующиеся атрибуты

```
@Oper(refreshAfter = true)
override def saveForm(): Unit = super.saveForm()
//или так
override def saveForm(): Unit = {
    super.saveForm()
    selection.refreshItem()
}
```

- Если нужна операция с молоточком и ключом, то устанавливаем для нее активность

```
@Oper(active = true)
override def extraOperations() = super.extraOperations()
```

- Переопределяем `setidState`, чтобы дописать различные действия

```
override def setidState(event: SetterEvent): Unit = {
    super.setidState(event)

    //завершение транзакции
    session.commit()
    //снятие блокировки
    Btk_FormSessionApi().closeLockUnit()

    //обновление карточки
    selection.refreshItem()
    //вызов checkWorkability в карточке
    selection.checkWorkability()
    //вызов checkWorkability в деталях
    selection.cwaDetails()
    //если нужно обновление деталей вместо cwaDetails используем selection.
}
```

(продолжается на следующей странице)


```
↪ refreshDetails
}
```

- Переопределяем CWA, для блокировки полей и операций в зависимости от состояния

```
//для управлением редактируемостью атрибутов
protected def setAttrReadOnly(): Unit = {
  val avAttrsNotApply = Set("DREG", "SREGNUM", "IDSTATE", "IDSTATEHL")
  val bvStNotForm = thisRop().get(_idStateMC).isDistinct(100.nn)
  attrs().filterNot(f => avAttrsNotApply.contains(f.name.toUpperCase))
    .foreach(_isReadOnly = bvStNotForm)
}

//для управлением видимостью атрибутов
protected def setAttrVisible(): Unit = {
  //(A.idObjectType + "HL").isVisible = false
}

//для управлением активность операций
protected def setEnabledOps(): Unit = {
  val bvStNotForm = thisRop().get(_idStateMC).isDistinct(100.nn)
  val bvStAcc = thisRop().get(_idStateMC) >= 300.nn
  ops("fillDoc").isEnabled = !bvStNotForm
  ops("clonePrint").isEnabled = bvStAcc
}

override def checkWorkability(): Unit = {
  super.checkWorkability()
  //для управлением редактируемостью атрибутов
  setAttrReadOnly
  //для управлением видимостью атрибутов
  setAttrVisible
  //для управлением активность операций
  setEnabledOps
}
```

- реляционные запросы в onRefresh в карточках не следует использовать, так как если будет написан запрос, то перед выполнением произойдет сразу СОХРАНЕНИЕ. Все доп. поля пишем либо через onRefreshExt, additionalInfo или case class.
- если переопределяете LookUp или пишете свои вып. списки, не забывайте дописывать @FlushBefore(mode = FlushBeforeMode.Disabled), так как это тоже приводит к коммиту при вызове запроса

```
import ru.bitec.app.gtk.gl.{FlushBefore, FlushBeforeMode}

@FlushBefore(mode = FlushBeforeMode.Disabled)
override protected def onRefresh: Recs
```

Avi коллекции

- Переопределяем onRefresh, чтобы добавить сортировку по №п/п

```
override def onRefresh: Recs = {  
  thisApi().refreshByParent(getIdMaster).toList.sortBy(_.get(_.nRow))  
}
```

- реляционные запросы в onRefresh в коллекциях не следует использовать, так как если будет написан запрос, то перед выполнением произойдет сразу СОХРАНЕНИЕ. Все доп. поля пишем либо через onRefreshExt, additionalInfo или объектный List[case class].
- Переопределяем CWA, для блокировки полей и операций, в зависимости от состояния

```
//для управлением редактируемостью атрибутов  
protected def setAttrReadOnly(): Unit = {  
  val bvStNotForm = getVar("super$idStateMC").asNNumber.isDistinct(100.nn)  
  attrs().foreach(_.isReadOnly = bvStNotForm)  
}  
  
//для управлением видимостью атрибутов  
protected def setAttrVisible(): Unit = {  
  //(A.idGds + "HL").isVisible = false  
}  
  
//для управлением активность операций  
protected def setEnabledOpsers(): Unit = {  
  val bvStNotForm = getVar("super$idStateMC").asNNumber.isDistinct(100.nn)  
  opsers("insert").isEnabled = getIdMaster.isNotNull && !bvStNotForm  
  opsers("delete").isEnabled = !bvStNotForm && !A.id.isNull  
  opsers("copyObject").isEnabled = !bvStNotForm && !A.id.isNull  
}  
  
override def checkWorkability(): Unit = {  
  super.checkWorkability()  
  //для управлением редактируемостью атрибутов  
  setAttrReadOnly  
  //для управлением видимостью атрибутов  
  setAttrVisible  
  //для управлением активность операций  
  setEnabledOpsers  
}
```

Avm документа

- Для атрибута тип документа в отображении Default устанавливаем тип редактора - выпадающий список, чтобы отображались типы только нашего класса

```
<attr name="idObjectTypeHL" caption="Тип" order="50.2" editorType="lookup" ☐  
↪isLastInLine="false" isVisible="true">  
  <editor>  
    <lookup lookupQuery="gtk-Btk_ObjectTypeAvi#MainLookup" ☐
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
↪isLookupLazyLoad="true"
        changeableAttr="idObjectType" lookupKeyAttr="id"
↪lookupListAttr="sHeadLine"
        isResetButtonVisible="true"/>
    </editor>
</attr>
```

- Для атрибута организация в отображении Default устанавливаем тип редактора - выпадающий список

```
↪">
    <attr name="idDepOwnerHL" caption="Организация" order="70.2" isRequired="true"
        <editor>
            <lookup lookupQuery="gtk-Bs_DepOwnerAvi#Lookup"
                changeableAttr="idDepOwner" isLookupLazyLoad="false"
                lookupKeyAttr="id" lookupListAttr="sHeadLine"
↪isResetButtonVisible="false"/>
            </editor>
        </attr>
```

- Устанавливаем закладки от типа документа, по умолчанию видимые в карточке и не видимые в списке (isVisible="false")

```
<tabItems isVisible="true" selection="gtk-Btk_ObjectTypeTabAvi"
    representation="List_Tab"
    selection.selectionAttr="SSELECTIONNAME"
    selection.representationAttr="SREPRESENTATIONNAME"
    selection.captionAttr="SCAPTION"
    selection.imageIndexAttr="NIMAGE"
    selection.paramsAttr="JSONPARAMS"
/>
```

- Если есть поля в фильтре, значения которых должны браться из глобального фильтра (например по датам и организации), то прописываем значения по умолчанию через defaultValue

```
<!--Период с-->
<condition id="filterFrom" logicalOperator="and" isExpression="true"
    expression=":flt_dFrom &lt;= t.dPeriod">
    <filterAttr attribute-type="Date" name="flt_dFrom" editorType=
↪"datePick" order="10.0"
        caption="Период с" isLastInLine="false" defaultValue=
↪"super$DGLOBALBEGINDATE">
        <card controlWidth="24" isControlWidthFixed="true"/>
    </filterAttr>
</condition>
<!--Период по-->
<condition id="filterTo" logicalOperator="and" isExpression="true"
    expression=":flt_dTo &gt;= t.dPeriod">
    <filterAttr attribute-type="Date" name="flt_dTo" editorType="datePick
↪" order="20.0" caption="по"
        isLastInLine="false" defaultValue="super$DGLOBALENDDATE">
        <card controlWidth="20" isControlWidthFixed="true"/>
    </filterAttr>
```

(продолжается на следующей странице)

```

        </condition>
        <!--План счетов-->
        <condition logicalOperator="and" id="filterIdAdjustMethod" isExpression=
↪ "true"
            expression="t.idAdjustMethod = :flt_idAdjustMethod">
            <filterAttr name="flt_idAdjustMethod" attribute-type="Long"
↪ isVisible="false" order="30"
                defaultValue="super$idGlobalAdjustMethod"/>
            <filterAttr name="flt_idAdjustMethodHL" attribute-type="Varchar"
↪ caption="План счетов" order="30.1"
                editorType="lookup" isLastInLine="false">
                <editor>
                    <lookup lookupQuery="gtk-Bs_AdjustMethodAvi#Lookup"
                        lookupKeyAttr="id" lookupListAttr="sHeadLine"
↪ changeableAttr="flt_idAdjustMethod"
                        isLookupLazyLoad="false" isResetButtonVisible="true"/
↪ >
                </editor>
            </filterAttr>
        </condition>
        <!--Организация-->
        <condition logicalOperator="and" id="filterIdDepOwner" isExpression="true
↪ "
            expression="t.idDepOwner = :flt_idDepOwner">
            <filterAttr name="flt_idDepOwner" attribute-type="Long" caption=
↪ "Организация" order="50"
                defaultValue="super$idGlobalDepOwner" isVisible="false"/>
            <filterAttr name="flt_idDepOwnerHL" attribute-type="Varchar" caption=
↪ "Организация" order="50.2"
                editorType="lookup" isLastInLine="false">
                <editor>
                    <lookup lookupQuery="gtk-Bs_DepOwnerAvi#Lookup"
                        changeableAttr="flt_idDepOwner" isLookupLazyLoad=
↪ "false"
                        lookupKeyAttr="id" lookupListAttr="sHeadLine"
↪ isResetButtonVisible="true"/>
                </editor>
            </filterAttr>
        </condition>

```

Разработка под высокую нагрузку

Работа с кэшем

Применяйте правильно byParent и byKey, вместо refreshByParent и refreshByKey

Методы refreshByParent и refreshByKey всегда перезапрашивают данные из БД, плюс берут данные которые находятся в кэше, чтобы все данные были актуальными. Методы byParent и byKey берут данные из кэша, но если идет первое обращение по ключу, то тоже сначала обращаются к БД. Если было обращение byParent и byKey по ключу и затем уже произошла вставка новых записей по этим же ключам, то последующие обращения могут вернуть не полный перечень данных. Поэтому при написании

логики важно понимать какой метод использовать, чтобы не было постоянных запросов к БД и в то же время не было не корректных данных. Например, при переводе состояния документа, так как коллекции уже не будут добавляться, то для них следует использовать byParent, а не refreshByParent. А например, при удалении документа, следует использовать refreshByParent, чтобы не получить ошибки о зависимых объектах.

Применяйте правильно Session.commit, Session.commitWork, flush и flush(true), чтобы не достичь лимита загружаемых ячеек

session.flush - применяет изменения сессии к базе данных без завершения транзакции. Есть параметр cleanTransactCache, если true очищает транзакционный кэш (загруженные объекты) после операции, обнуляет счетчик загруженных ячеек. Если false (по умолчанию), то происходит только синхронизация данных в БД, загруженные объекты остаются в кеше. session.commit - применяет изменения сессии к базе данных с успешным завершением транзакции, очищает транзакционный кэш после операции, обнуляет счетчик загруженных ячеек. session.commitWork - применяет изменения сессии к базе данных без завершения транзакции, очищает транзакционный кэш после операции, обнуляет счетчик загруженных ячеек.

Чтобы не достичь лимита загружаемых ячеек, надо сбрасывать пачками данные в БД с очисткой транзакционного кэша. Пример:

```
var counter = 0
for (rv <- new ASelect {
  val id: NLongColumn = asNLong("id")
  val idBalAcc: NLongColumn = asNLong("idBalAcc")
  val nSum: NNumberColumn = asNNumber("nSum")
  SQL(sQuery)
  on("dExecDate" -> dExecDate, "idaDep" -> LongPgArray(idaDep), "idaEmployee" ->
↳LongPgArray(idaEmployee), "idaAcc"-> LongPgArray(idaAcc.toList), "idBisObj" ->
↳idBisObj, "idDepOwner" -> idDepOwner)
}) {
  counter += 1
  insertByParent(Asf_InventoryApi().load(idpInventory)) :/ { rop =>
    setnOrder(rop, counter.nn)
    setidInvNumb(rop, rv.id())
    setidAcc(rop, rv.idBalAcc())
    setnSum(rop, rv.nSum())
    setnQtyRemain(rop, 1.nn)
    rop
  }
  if (counter % 1000 === 0)
    session.flush(true)
}

session.flush(true)
```

Загрузка множества объектов

Если требуется загрузить множество объектов, с последующим использованием гор, например для изменения, то можно использовать OQuery или tx индекс или byParent (для коллекций), в других случаях можно обойтись запросами через ATSQL или ASelect, чтобы не загружать оперативную память сервера лишними данными.

1. Используйте OQuery когда нужны гор, нет соединений с другими таблицами, условия фильтрации идут по одному или нескольким полям и все объекты уже обязательно находятся в БД.
2. Используйте tx индекс или byParent (для коллекций) когда нужны гор, нет соединений с другими таблицами, основное условие фильтрации идет по одному полю (доп. условия можно накладывать через filter) и объекты могут не находится в БД.
3. Используйте ATSQL или ASelect когда гор не нужны, но нужны данные по одному или нескольким столбцам, могут быть соединения с другими таблицами, могут быть сложные условия фильтрации и все объекты уже обязательно находятся в БД.
4. Используйте Btk_BulkProcessPkg().chunkedQuery когда нужны гор, могут быть соединения с другими таблицами, могут быть сложные условия фильтрации и все объекты уже обязательно находятся в БД. Так же в этом случае можно применить п.3 для нахождения списка идентификаторов и затем применить п.1

Используйте предварительную загрузку данных в кэш через batchIn и queryKeys

Предварительная загрузка данных в кэш нужна, когда требуется загрузить гор множества объектов или данные коллекций различной вложенности и чтобы запрос к БД по этим объектам происходил один раз, а не множество.

Пример загрузки для tx индекса:

```
//прогружаем данные из БД по всем ключам в кэш через queryKeys
idxGidSrcObject.queryKeys(gidapSrc)
//при использовании byKey по этим ключам, данные будут браться уже из кэша
val result = gidapSrc.flatMap(idxGidSrcObject.byKey)
```

Пример загрузки с помощью OQuery:

```
//прогружаем данные из БД с коллекциями через batchIn
new OQuery(entityAta.Type) {
    where(t.id === idp)
    batchIn(Stk_WarrantInDetAta, Stk_WarrantInDetItemAta, Stk_ObjNeedDetAta, Stk_
↳DMDKByDocAta, Stk_OperationAta)
}.foreach { rop => }
//при использовании load или byParent, данные будут браться уже из кэша, если они_
↳были в запросе в OQuery
val rop = Stk_WarrantInApi().load(idp)
val ropDet = Stk_WarrantInDetApi().byParent(rop)
```

При массовой обработке объектов используйте `Btk_BulkProcessPkg().chunkedQuery`

С помощью реляционного запроса, получает значения первичных ключей для массовой загрузки объектов в память. Количество одновременно загружаемых объектов указывает в параметре `chunkSize`. Пример:

```
Btk_BulkProcessPkg().chunkedQuery(  
  Asf_InvNumbSumRegApi(),  
  5000,  
  s""  
    select t.id  
    from Asf_InvNumbSumReg t  
    where t.gidSrc in (select unnest({gidap}))  
    "",  
  Seq("gidap" -> GidPgArray(idap.map(id => getGid(id))))  
) { ropa =>  
  ropa.foreach { rop =>  
    if (rop.exists())  
      Asf_InvNumbSumRegApi().delete(rop)  
  }  
  session.flush(true)  
}
```

При массовой регистрации объектов используйте `Btk_BulkProcessPkg().findObjects`.

По переданному массиву case-классов ищет записи в классе. Массово загружает найденные записи, и вызывает обработчик для всех переданных объектов case-класса. Пример:

```
case class objStatusPlanReg(idGdsPlanReg: NLong  
  , idEndowStatus: NLong)  
val avObjStatusPlanReg = ArrayBuffer[objStatusPlanReg]()  
...  
Btk_BulkProcessPkg().findObjects(  
  filters = avObjStatusPlanReg.toVector,  
  api = this,  
  sqlConditions =  
    s""t.idGdsPlanReg = f.idGdsPlanReg""  
) { res =>  
  //если запись найдена она будет прогружена в кэш, если не нашли запись то создаем  
  ↪ ee  
  val rop = res.getRopOrElseUpdate {  
    insert() :/ { rop =>  
      setidGdsPlanReg(rop, res.filter.idGdsPlanReg)  
      rop  
    }  
  }  
  if (res.filter.idEndowStatus.isDistinct(rop.get(_idEndowStatus))) {  
    setidEndowStatus(rop, res.filter.idEndowStatus)  
  }  
}
```

Scala код

Используйте View или withFilter, чтоб избежать создания временных коллекций

1. View - это особый вид коллекции в Scala, который берет базовую коллекцию и лениво выполняет методы преобразования в этой коллекции. Мы можем преобразовать каждую коллекцию Scala в отложенное представление и обратно с помощью view метода. Вместо создания новой коллекции после каждой операции, они применяют сразу всю цепочку преобразований к каждому элементу оригинальной коллекции, благодаря чему цепочка преобразований выполняется за один проход. Преобразование виртуальной коллекции в реальную выполняется либо одним из преобразующих методов из серии to{Название коллекции}, либо методом force.

Пример:

```
// До
seq.map(f).flatMap(g).filter(p)

// После
seq.view.map(f).flatMap(g).filter(p).force
```

2. withFilter работает так же, как и view — создает временный объект, который ограничивает область последующих преобразований коллекции (так, что он реорганизует возможные побочные эффекты). Однако, нет нужды явно преобразовывать коллекцию к (или наоборот) от временного представления (вызвав view и force). Если filter используется перед map, flatMap или foreach, то для лучшей производительности вместо него должен использоваться withFilter.

Пример:

```
// До
Stm_OrderInDetApi()
  .byParent(rop)
  .filter(f => f.get(_.idStock).isNotNull)
  .foreach { ropDet =>
    if (Stk_StockApi().load(ropDet.get(_.idStock)).get(_.idBisObj).isDistinct(value))
      Stm_OrderInDetApi().setidStock(ropDet, None.nl)
  }

// После
Stm_OrderInDetApi()
  .byParent(rop)
  .withFilter(f => f.get(_.idStock).isNotNull)
  .foreach { ropDet =>
    if (Stk_StockApi().load(ropDet.get(_.idStock)).get(_.idBisObj).isDistinct(value))
      Stm_OrderInDetApi().setidStock(ropDet, None.nl)
  }
```


Избегайте использования filter несколько раз подряд

Пример:

```
// До
seq.filter(p1).filter(p2)

// После
seq.filter(x => p1(x) && p2(x))
```

Используйте lazy val

Компилятор не сразу вычисляет связанное выражение отложенного значения val. Он вычисляет переменную только при первом обращении к ней.

Пример:

```
// До
val idgForming = Btk_ClassStateApi().findByNameAndIdClass("Forming", idClass)

// После
lazy val idgForming = Btk_ClassStateApi().findByNameAndIdClass("Forming", idClass)
```

Используйте коллекции, имеющие оптимальную сложность поиска, взамен обладающих линейной при частом обращении за находением

Пример:

```
//до:
def registerByParent(ropDoc: RopDoc, idap: CSeq[NLong]): Unit = {
    val idavAlreadyRegistered = byParent(ropDoc).map(_.get(_.idGdsType)).toList

    idap.foreach { idv =>
        if (idv.isNotNull) {
            if (!idavAlreadyRegistered.contains(idv)) {
                val rop = // ...
            }
        }
    }
}

//после:
def registerByParent(ropDoc: RopDoc, idap: CSeq[NLong]): Unit = {
    val idavAlreadyRegistered = byParent(ropDoc).map(_.get(_.idGdsType)).toSet

    idap.foreach { idv =>
        if (idv.isNotNull) {
            if (!idavAlreadyRegistered(idv)) {
                val rop = // ...
            }
        }
    }
}
```

(продолжается на следующей странице)

```
}
}
```

На while и рекурсии добавляйте защиту по кол-ву итераций

Пример:

```
// До
while (dotsNeedUpdate.nonEmpty) {
  val newDotsNeedUpdate = scala.collection.mutable.HashSet[NNumber]()
  ???
  dotsNeedUpdate = newDotsNeedUpdate.toSet
}
// После
var count = 0
while (dotsNeedUpdate.nonEmpty && count < 10000) {
  val newDotsNeedUpdate = scala.collection.mutable.HashSet[NNumber]()
  ???
  dotsNeedUpdate = newDotsNeedUpdate.toSet
  count += 1
}
```

Не эмулируйте существующие методы работы с коллекциями

Пример:

```
// До
seq.map(f).flatten
seq.toSet.toSeq
seq.reverse.iterator
seq.reverse.map(f)
seq.collect{case P => ???}.headOption

// После
seq.flatMap(f)
seq.distinct
seq.reverse.iterator
seq.reverseMap(f)
seq.collectFirst{case P => ???}
```

Используйте ArrayBuffer вместо ListBuffer

ArrayBuffer выигрывает в производительности ListBuffer при операциях добавления и итерации, а это 90% всего использования. Также ArrayBuffer занимает меньший объем в оперативной памяти, потому что ListBuffer основан на связанном списке.

- [Scala Collections Performance](#)

При проверке на существование - не прибегайте к фильтрации

Пример:

```
// До
seq.filter(p).nonEmpty
seq.filter(p).isEmpty

// После
seq.exists(p)
!seq.exists(p)
```

При проверке на пустоту - не вычисляйте длину

Пример:

```
// До
seq.length > 0
seq.length != 0
seq.length == 0

// После
seq.nonEmpty
seq.nonEmpty
seq.isEmpty
```

В checkWorkability (CWA) используйте getVar вместо обращений к БД (ASelect, ATSQL, OQuery, Load, byParent, byKey и т.д.)

Так как CWA вызывается на большое количество событий, выполняемые в нем запросы к БД могут существенно замедлить работу интерфейса пользователя. Вычисления дополнительных полей для CWA можно осуществлять в onRefreshExt, selectStatement или на события при открытии карточки (onLoadMeta, beforeOpen, afterOpen и т.д.). Пример:

```
//До
override def checkWorkability(): Unit = {
  super.checkWorkability()
  if (thisApi.load(getVar("id").asNLong).get(_idStateMC) >= 300.nn)
   opers("delete").isEnabled = false
}

//После
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
override def checkWorkability(): Unit = {
  super.checkWorkability()
  if (getSelfVar("idStateMC").asNNumber >= 300.nn)
    opers("delete").isEnabled = false
}
```

Избегайте использовать copyAro

При использовании copyAro все данные из гор копируются, что увеличивает объем потребляемой оперативной памяти на сервере и может привести к ошибке потребляемой памяти в сессии. Пример:

```
// До
val rvWorkDoc = Mct_DocumentApi().loadByGid(rop.get(_.gidMctDocument)).copyAro()
val idvPrj = rvWorkDoc.idPrj
val idvPrjVer = rvWorkDoc.idPrjVer
// После
val ropWorkDoc = Mct_DocumentApi().loadByGid(rop.get(_.gidMctDocument))
val idvPrj = ropWorkDoc.get(_.idPrj)
val idvPrjVer = ropWorkDoc.get(_.idPrjVer)
```

В Api пишите только бизнес логику для взаимодействия с пользователем, все остальное выносите в методы Api

Пример:

```
// До
override protected def onInsertItem(): Unit = {
  super.onInsertItem()
  if (getSelfVar("idCandidat#").asNLong.isNotNull) thisApi().setidCandidate(thisRop(),
  ↳ getSelfVar("idCandidat#").asNLong)
  if (getSelfVar("idVacancy#").asNLong.isNotNull) thisApi().setidVacancy(thisRop(),
  ↳ getSelfVar("idVacancy#").asNLong)
  if (getSelfVar("nSalaryProbAcc#").asNNumber.isNotNull) thisApi().
  ↳ setnSalaryProb(thisRop(), getSelfVar("nSalaryProbAcc#").asNNumber)
  if (getSelfVar("nProbationAcc#").asNNumber.isNotNull) thisApi().
  ↳ setnProbation(thisRop(), getSelfVar("nProbationAcc#").asNNumber)
  if (getSelfVar("nSalaryAcc#").asNNumber.isNotNull) thisApi().setnSalary(thisRop(),
  ↳ getSelfVar("nSalaryAcc#").asNNumber)
  if (getSelfVar("idCurAcc#").asNLong.isNotNull) thisApi().setidCur(thisRop(),
  ↳ getSelfVar("idCurAcc#").asNLong)
  if (getSelfVar("idEmployment#").asNLong.isNotNull) thisApi().setidEmployment(thisRop(),
  ↳ getSelfVar("idEmployment#").asNLong)
  if (getSelfVar("idWorkGraph#").asNLong.isNotNull) thisApi().setidWorkGraph(thisRop(),
  ↳ getSelfVar("idWorkGraph#").asNLong)
  if (getSelfVar("idWorkMode#").asNLong.isNotNull) thisApi().setidWorkMode(thisRop(),
  ↳ getSelfVar("idWorkMode#").asNLong)
  if (getSelfVar("idMentor#").asNLong.isNotNull) thisApi().setidCurator(thisRop(),
  ↳ getSelfVar("idMentor#").asNLong)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
if (getSelfVar("idHREmpl#").asNLong.isNotNull) thisApi().setidHREmpl(thisRop(),  
↪getSelfVar("idHREmpl#").asNLong)  
if (getSelfVar("dPlan#").asNDate.isNotNull) thisApi().setdPlan(thisRop(), getSelfVar(  
↪"dPlan#").asNDate)  
}  
// После  
override protected def onInsertItem(): Unit = {  
  super.onInsertItem()  
  if (getSelfVar("idOffer#").notNull())  
    thisApi().fillByOffer(thisRop(), getSelfVar("idOffer#").asNLong)  
}
```

Не пишите длинные выражения и запросы в одну строчку

Длинные выражения в одну строку затрудняют чтение кода и отнимает много времени на понимание
Пример:

```
// До  
Stk_InternalWarrantDetApi().byParent(rop).withFilter(f => (gidapDet.contains(f.gid) ||  
↪gidapDet.isEmpty)).foreach { ropDet =>  
  ???  
}  
  
// После  
Stk_InternalWarrantDetApi().byParent(rop)  
  .withFilter(f => (gidapDet.contains(f.gid) || gidapDet.isEmpty))  
  .foreach { ropDet =>  
    ???  
  }
```

Пример:

```
-- До  
select gidObj, nPercent, nQty from Bs_CostDistr d where d.gidSrc = :gid  
-- После  
select d.gidObj  
      ,d.nPercent  
      ,d.nQty  
from   Bs_CostDistr d  
where  d.gidSrc = :gid
```

Создавайте общие trait без реализации для разных отображений с одинаковой функциональностью

Чтобы избежать дублирование кода и упростить дальнейшую поддержку Пример:

```
trait List_gidDocGeneral extends Default with super.List {  
  ???  
}  
  
trait List_gidDocTax extends List_gidDocGeneral {  
  override def idAccKind: NLong = Bs_AccKindApi().idTax  
}  
  
trait List_gidDocMng extends List_gidDocGeneral {  
  override def idAccKind: NLong = Bs_AccKindApi().idMng  
}  
  
trait List_gidDocAit extends List_gidDocGeneral {  
  override def idAccKind: NLong = Bs_AccKindApi().idAit  
}
```

Создавайте библиотеки для использования общих методов в различных выборках

Чтобы избежать дублирование кода и упростить дальнейшую поддержку Пример:

```
Pm_Lib  
Btk_FileLib
```

Не используйте поля Tuple, а по возможности пользуйтесь экстрактором

Пример:

```
val t = ("Vlad", 23, "M")  
//До  
setName(t._1)  
setAge(t._2)  
setSex(t._3)  
  
//После  
val (name, age, sex) = t  
setName(name)  
setAge(age)  
setSex(sex)  
  
Seq(t)  
  .foreach { case (name, age, sex) =>  
    setName(name)  
    setAge(age)  
    setSex(sex)  
  }
```

Используйте «`.lastOption`, `.headOption`, `dropRight(1)`, `.drop(1)`, `.reduceOption`» вместо «`.last`, `.head`, `.init`, `.tail`, `.reduce`»

Использование «`.last`, `.head`, `.init`, `.tail`, `.reduce`» может приводит к исключению в случае если коллекция пустая Пример:

```
val unknownSizeList: List[_] = ???
//До
val sum = unknownSizeList.reduce(_ + _) // есть шанс ошибки
//После
val sum = unknownSizeList.reduceOption(_ + _).nn // мы получим NNumber(null) в случае
→ если список пустой
```

Не складывайте строки через `+`, используйте строковую интерполяцию

Пример:

```
val hello = "hello"
val world = "world"
//До
val str = hello + " " + world

//После
val str = s"$hello $world"
```

При сборе сложной строки не создавайте много промежуточных результатов

Пример:

```
val ropa: List[???] = ???
//До
var str = "Names: "

val result =
  for (rop <- ropa) yield {
    str += rop.sName + ";" // каждый раз создаётся новая строка
    load(rop.idSmth)
  }

str += " end"
//После
val stringBuilder =
  new StringBuilder()
  .append("Names: ")
val result =
  for (rop <- ropa) yield {
    stringBuilder
      .append(rop.sName)
      .append(";")
    load(rop.idSmth)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
}

val names = stringBuilder.append(" end").toString()
// или если есть возможность сформируйте список
val (result, names) = ropa.map { rop =>
    (load(rop.idSmth), rop.sName)
}.unzip

val str = names.mkString("Names: ", ";", " end")
```

Используйте Option при работе с пустыми (null) ссылками

Пример:

```
val rop = ???
val oldJavaLib = ???
//До
val ropSmth = load(rop.idSmth) // забыли обработать null

var result = _
val thing = oldJavaLib.getThing
if(thing != null) {
    val smth = thing.getSmth
    if(smth != null) {
        try {
            result = smth.doWork()
        } catch {
            case _: Exception => {}
        }
    }
}
if(result == null) throw AppException("Не получилось")
//После
val ropSmth = get(rop.idSmth)
    .getOrElse(throw AppException("Чего то нету"))

val result = (for {
    thing <- Option(oldJavaLib.getThing)
    smth <- Option(thing.getSmth)
    result <- Try(smth.doWork()).toOption
} yield result).getOrElse(throw AppException("Не получилось"))
```


Избегайте выполнения запросов внутри вложенных циклов

Пример:

```
case class TaskDoc(/* атрибуты */, aDocDet: CSeq[TaskDocDet])

case class TaskDocDet(/* атрибуты */, aFile: CSeq[TaskDocDetFile])

case class TaskDocDetFile(/* атрибуты */)

//до:
def getTasksData(dpFrom: NDate, dpTo: NDate): CSeq[Task] = {
  val avResult = mutable.ArrayBuffer.empty[TaskDoc]

  for (rvx <- new ASelect {
    // документы Task
  }) {
    // ...

    for (rvxDet <- new ASelect {
      // позиции Task
    }) {
      // ...

      for (rvxDetFile <- new ASelect {
        // файлы позиций Task
      }) {
        // ...
      }
    }
  }

  avResult
}

//после:
def getTasksData(dpFrom: NDate, dpTo: NDate): CSeq[Task] = {
  val avResult = mutable.ArrayBuffer.empty[TaskDoc]

  val avIdToTask = new ASelect {
    val id = asNLong("id")
    // остальные атрибуты
  }.map(rvx => rvx.id() -> TaskDoc(/* ... */)).toMap

  val avIdDocAndTaskDets = new ASelect {
    SQL"""
      select t.idDoc
             ,t.id
             ,t.nRow
      -- ..
      from Md_TaskDet t
      where t.idDoc in (select unnest(${LongPgArray(avIdToTask.keys)}) )
    """
  }
}
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
}.map { rvx =>
  (rvx.idDoc(), rvx.id(), TaskDocDet(/* ... */))
}.toSeq

val gidavTaskDet = avIdDocAndTaskDets.map(vEntry => Md_TaskDetApi().getGid(vEntry._
←2))
val avIdTaskDetAndFiles = new ASelect {
  SQL"""
  --
  where t.gidSrc in (select unnest(${GidPgArray(gidavTaskDet)}) )
  """
}.map { rvx =>
  (rvx.id(), TaskDocDetFile(/* ... */))
}

// агрегация с циклом по Task, поиском TaskDet, подбором File для Det

avResult
}
```

Пример с поисковым ключом:

```
case class objRem(
  idGds: NLong
  , idStock: NLong
  , gidSlitAccount: NGid
  , gidSTKAcc: NGid
  , gidMaster: NGid
  , gidSuite: NGid
  , idCons: NLong
  , var nQtyBase: NNumber
)

val avRem = collection.mutable.Map[NString, ArrayBuffer[objRem]]()
for (rv <- new ASelect {
  val idGds = asNLong("idGds")
  val idCons = asNLong("idCons")
  val idStock = asNLong("idStock")
  val gidMaster = asNGid("gidMaster")
  val gidSTKAcc = asNGid("gidSTKAcc")
  val gidSuite = asNGid("gidSuite")
  val gidSlitAccount = asNGid("gidSlitAccount")
  val nQtyBaseMsr = asNNumber("nQtyBaseMsr")
  SQL"""
SELECT sum(-r.nQtyBaseMsr) nQtyBaseMsr
  ,r.idGds
  ,r.idStock
  ,r.gidMaster
  ,r.gidSTKAcc
  ,r.gidSuite
  ,r.gidSlitAccount
  ,r.idCons
from (select dr.idgds
```

(продолжается на следующей странице)

```

        ,dr.gidMaster
        ,dr.gidObjNeed
        ,sum(dr.nQtyBase) nQtyBase
    from (
        select *
        from unnest(${idavGds}
            ,${gidavMaster}
            ,${gidaObjNeed}
            ,${naQtyBase}
        ) as x(idGds, gidMaster, gidObjNeed, nQtyBase)
        ) dr
    group by dr.idgds
            ,dr.gidmaster
            ,dr.gidObjNeed
    ) d
    join Stk_RegistryGdsMov r on r.idGds = d.idGds and r.gidSlitAccount = d.gidObjNeed
    and r.gidMaster = d.gidMaster
    where r.idDepOwner = $idpDepOwner
    and r.sType = 'res'
    and r.idStock in (select st.idchild
                      from stk_stocktree st
                      where st.idparent = $idpStock)

    group by r.idGds
            ,r.idStock
            ,r.gidMaster
            ,r.gidSTKAcc
            ,r.gidSuite
            ,r.gidSlitAccount
            ,r.idCons
    having sum(-r.nQtyBaseMsr) > 0""
    }) {
        val sKey = rv.idGds() + "_" + rv.gidMaster().nvl("0".ng) + "_" + rv.
    gidSlitAccount().nvl("0".ng)
        avRem.getOrCreateUpdate(sKey, ArrayBuffer[objRem]()) += objRem(
            idGds = rv.idGds()
            , idStock = rv.idStock()
            , gidSlitAccount = rv.gidSlitAccount()
            , gidSTKAcc = rv.gidSTKAcc()
            , gidMaster = rv.gidMaster()
            , gidSuite = rv.gidSuite()
            , nQtyBase = rv.nQtyBaseMsr()
            , idCons = rv.idCons()
        )
    }

    apObjNeed.foreach { vObjNeed =>
        val sKey = vObjNeed.idGds + "_" + vObjNeed.gidMaster.nvl("0".ng) + "_" +
    vObjNeed.gidObjNeed.nvl("0".ng)
        if (avRem.contains(sKey)) {
            avRem(sKey).filter(f => f.nQtyBase > 0.nn).foreach { rem =>
                lazy val ropGds = Bs_GoodsApi().load(vObjNeed.idGds)
            }
        }
    }

```

```

val nvQtyBase = vObjNeed.nQtyBase.min(rem.nQtyBase)
avOperationData += OperationData(
    gidParent = vObjNeed.gidDet
    , gidDet = vObjNeed.gidDet
    , gidSrcObject = vObjNeed.gidSrcObject
    , idGds = vObjNeed.idGds
    , idCons = rem.idCons
    , gidSuite = rem.gidSuite
    , idStock = rem.idStock
    , gidMaster = rem.gidMaster
    , gidSTKAcc = rem.gidSTKAcc
    , gidSlitAccountRes = rem.gidSlitAccount
    , nQtyBase = nvQtyBase
    , nQtyBase2 = {
        if (vObjNeed.nConvertBaseToBase2 > 0.nn) {
            (nvQtyBase / vObjNeed.nConvertBaseToBase2).round(ropGds.get(_
↪nRoundPlaces2))
        } else None.nn
    }
    , idReplacedGds = vObjNeed.idReplacedGds
    , nRateReplacedGds = vObjNeed.nConvertReplacedGds
    , nPrimeCostSum = (nvQtyBase * vObjNeed.nPrimeCostPrc).round(2.nn)
    , nPrimeCostSumTax = (nvQtyBase * vObjNeed.nPrimeCostPrcTax).round(2.nn)
    , nPrimeCostSumMng = (nvQtyBase * vObjNeed.nPrimeCostPrcMng).round(2.nn)
    , dDateTime = dpExec
    , idAddCondGenOper = Stk_AddCondGenOperApi().idvStornResByDetNeed
)
rem.nQtyBase -= nvQtyBase
vObjNeed.nQtyBase -= nvQtyBase
}
}

if (vObjNeed.nQtyBase > 0.nn && params.bValidateRemStorno) {
    throw AppException(s"""По ТМЦ ${Bs_GoodsApi().getMnemonic(vObjNeed.idGds)} $
↪{Bs_GoodsApi().getHeadLine(vObjNeed.idGds)} нет достаточного количества для
↪сторнирования резерва!""")
}
}
}

```

Используйте параллельные вычисления, если требуется множественная обработка независимых объектов

Руководство разработчика: Параллельные вычисления

Sql запросы

Просматривайте план выполнения запроса, чтобы узнать проблемные места

План запроса – последовательность шагов в виде дерева для получения результата SQL-запроса. Каждый шаг – операция: извлечение или обработка данных, сканирование индекса, пересечение или объединение множеств, построение битовых карт или другая. Чтобы построить план, оптимизатор использует правила преобразования, эвристику и алгоритмы оптимизации на базе стоимости. Правило преобразует план в другой план с лучшей стоимостью. Например, операции фильтрации и проекции уменьшают размер набора данных и, следовательно, должны выполняться как можно раньше, правило может переупорядочить операции так, чтобы операции фильтрации и проекции выполнялись раньше. Алгоритм оптимизации выбирает план с наименьшей оценочной стоимостью. Число возможных планов (называемых пространством планов) для запроса, содержащим несколько операций, является огромным - слишком большим, чтобы алгоритм рассмотрел каждый возможный план, поэтому используются эвристики для сокращения числа планов, оцениваемых оптимизатором. Получить план запроса можно в dBeaver с помощью операции на панели слева **Получить план выполнения**, а так же с помощью команды `explain` перед запросом. Пример:

```
explain analyze
SELECT w.*
FROM stk_internalwarrant w
WHERE w.idstateinmc = 300
AND EXISTS (SELECT 1 FROM stk_internalwarrantdet d WHERE d.idinternalwarrant = w.id
↳AND d.idcons IS null)

--Выход
Hash Semi Join (cost=13.79..52.69 rows=14 width=3485)
  Hash Cond: (w.id = d.idinternalwarrant)
  -> Seq Scan on stk_internalwarrant w (cost=0.00..38.44 rows=32 width=3485)
        Filter: (idstateinmc = '300'::numeric)
  -> Hash (cost=12.25..12.25 rows=123 width=8)
        -> Index Scan using idx_stk_internalwarrantdet_idcons on stk_internalwarrantdet
↳d (cost=0.15..12.25 rows=123 width=8)
        Index Cond: (idcons IS NULL)
```

- Использование EXPLAIN
- Курс «PostgreSQL для начинающих»: #4 — Анализ запросов

При написании сложных запросов дробите его на части и используйте подзапросы

При построении сложных запросов, дробите его на части - основной запрос с условиями и с минимальным кол-вом соединений можно вынести в with или в нижеуровневый запрос. Если часть данных надо брать из доп. таблиц и при этом понадобится группировка, то лучше сделать это подзапросом, либо с помощью `left join lateral`. Тщательно анализируйте условия фильтрации, большинство условий старайтесь выносить на нижний уровень, чтобы как можно меньше было данных для последующих соединений. Если условия фильтрации идут по нескольким полям, но индексы есть только по отдельным полям, то создавайте составной индекс. Минимизируйте кол-во соединяемых таблиц, можно использовать денормализованные таблицы или столбцы. Пример:

```
--использование with
WITH t as (
```

(продолжается на следующей странице)

```

SELECT t.*
from   Prc_PriceListRegFull t
where  t.idPriceList = 1
and    t.sFindKey in (select unnest(array['a','b','c']))
)
SELECT t.*
from   t
where not exists(select 1
                  from t t1
                  where t1.idPriceList = t.idPriceList
                  and   t1.sFindKey = t.sFindKey
                  and   t1.dDoc > t.dDoc
                  and   t1.dFrom <= t.dFrom
                  and   t1.dTo >= t.dTo
                  )

--подзапрос с использованием jsonb_build_object, чтобы использовать сразу несколько
--столбцов
SELECT c.jStkRem->>'sMasterCode' as sMasterRem
      ,c.idOrder
      ,cast(c.jStkRem->>'nQtyBase' as numeric) as nQtyBase
      ,c.nQtyBase      as nQtyStockCard
      ,c.sNumSrcDoc
      ,c.dSrcDoc
      ,c.sHLSrcDoc
from (select c.*
      ,(select jsonb_build_object(
          'nQtyBase', sum(rc.nQtyBaseMsr)
          , 'sStockCode' ,string_agg(distinct ss.sCode, ', ')
          , 'sStockCaption', string_agg(distinct ss.sCaption, ', ')
          , 'sMasterCode', string_agg(distinct mm.sMnemonicCode_dz, ', ')
        )
      from (select sum(rc.nQtyBaseMsr) nQtyBaseMsr
            ,rc.idStock
            ,rc.gidMaster
            from   Stk_RegConsMat rc
            where  rc.idGds = c.idGds
            and    rc.gidsuite = c.gid
            and    rc.idCons = getgidid(c.gidCons)
            and    rc.stype = 'mov'
            group by rc.idStock
            ,rc.gidMaster
            having sum(rc.nQtyBaseMsr) > 0
            ) rc
      left join Stk_Stock ss on ss.id = rc.idStock
      left join Stk_Master mm on mm.gidref = rc.gidMaster
      ) jStkRem
from   Gds_StockCard c
) as c

--использование join lateral
select t.id

```

```

        ,t.idNeedObj
        ,t8.nQtySumBySpec
        ,t8.nSumNoTaxBySpec
from Prs_PurchTrans t
left join lateral(select sum(t7.nQty) as nQtySumBySpec
                  ,sum(t7.nSumNoTax) as nSumNoTaxBySpec
                  from Prs_PurchTrans t7 where t7.gidSrcObj = t.gidSrcObj
                  ) as t8 on true

```

- Табличные выражения
- Запросы WITH

Используйте индексы, чтобы ускорить получение запрашиваемых данных

Индекс в таблице базы данных — это дополнительная структура данных, которая создаётся наряду с данными в таблице. Индексы используются для ускорения процесса извлечения данных и повышения производительности запросов. Они обеспечивают эффективный способ обращения к упорядоченным записям. Когда индекс создан, никакие дополнительные действия не требуются: система сама будет обновлять его при изменении данных в таблице и сама будет использовать его в запросах, где, по её мнению, это будет эффективнее, чем сканирование всей таблицы. Вам, возможно, придётся только периодически запускать команду ANALYZE для обновления статистических данных, на основе которых планировщик запросов принимает решения. Создание индекса для большой таблицы может занимать много времени. По умолчанию PostgreSQL позволяет параллельно с созданием индекса выполнять чтение (операторы SELECT) таблицы, но операции записи (INSERT, UPDATE и DELETE) блокируются до окончания построения индекса. Составной индекс - это индексы создаваемый по нескольким столбцам таблицы. Частичный индекс — это индекс, который строится по подмножеству строк таблицы, определяемому условным выражением (оно называется предикатом частичного индекса). Такой индекс содержит записи только для строк, удовлетворяющих предикату. Частичные индексы довольно специфичны, но в ряде ситуаций они могут быть очень полезны.

Примеры:

```

--частичный индекс
CREATE INDEX IF NOT EXISTS idx_stk_cons_bneedcheckin_IsNullIdRes ON stk_cons_
↳(bneedcheckin) WHERE idcontrolresult IS NULL;
--составной индекс
CREATE INDEX IF NOT EXISTS idx_stk_regmovmat_gdsmastertype ON stk_regmovmat (idGds,
↳gidMaster, stype)
--GIN индекс для jsonb поля
create index if not exists idx_Cnt_ContractSumDistr_jBbbObjAttrs on Cnt_ContractSumDistr_
↳using gin (jBbbObjAttrs)

```

- Индексы

Используйте параметры при передаче значений в запрос

При написании запросов в ASelect или ATSQL для передачи параметров не используйте прямую «врезку» значений в тело запроса, нужно передавать через аргументы. При написании запроса в onRefresh или selectStatement для Avi, к параметрам выборки или фильтра можно обращаться через двоеточие. Пример:

```
//До
for (rv <- new ASelect{
  val id: NLongColumn = asNLong("id")
  SQL(s""select o2.id
    from mes_WorkOrderOperation o1
    join Mct_WorkStructureOperation o2 on o1.idWorkStructureOperation = o2.id
    where o1.idWorkOrder in (${idaWork.mkString(",")})
    and coalesce(o1.bAccept,0) = 0
    and o1.dFactEndDate is null
    and not exists (select 1 from mct_WorkStructureOperation c where c.idParent =
↪o2.id)""")
}) {
  val rop = Mct_WorkStructureOperationApi().load(rv.id())
  Mct_WorkStructureOperationApi().setidResource(rop, idResource)
}

//После
for (rv <- new ASelect{
  val id: NLongColumn = asNLong("id")
  SQL(s""select o2.id
    from mes_WorkOrderOperation o1
    join Mct_WorkStructureOperation o2 on o1.idWorkStructureOperation = o2.id
    where o1.idWorkOrder in (select unnest({idaWork}))
    and coalesce(o1.bAccept,0) = 0
    and o1.dFactEndDate is null
    and not exists (select 1 from mct_WorkStructureOperation c where c.idParent =
↪o2.id)""")
  on(Symbol("idaWork") -> LongPgArray(idaWork))
}) {
  val rop = Mct_WorkStructureOperationApi().load(rv.id())
  Mct_WorkStructureOperationApi().setidResource(rop, idResource)
}

//или так
for (rv <- new ASelect{
  val id: NLongColumn = asNLong("id")
  SQL(s""select o2.id
    from mes_WorkOrderOperation o1
    join Mct_WorkStructureOperation o2 on o1.idWorkStructureOperation = o2.id
    where o1.idWorkOrder in (select unnest(${LongPgArray(idaWork)}))
    and coalesce(o1.bAccept,0) = 0
    and o1.dFactEndDate is null
    and not exists (select 1 from mct_WorkStructureOperation c where c.idParent =
↪o2.id)""")
}) {
  val rop = Mct_WorkStructureOperationApi().load(rv.id())
  Mct_WorkStructureOperationApi().setidResource(rop, idResource)
}
```


При группировке минимизируйте кол-во группируемых полей

Чем больше группируемые поля, тем дольше будет выполняться запрос. Пример:

```
// До
select t.idgds
      ,g.smnemocode_dz as idGdsMC
      ,g.sheadline_dz as idGdsHL
      ,greatest(sum(case when ft.scode in ('20', '25') then t.nQtyRelease end), sum(case
↪when ft.scode in ('30') then t.nQtyRelease end)) as nQtyRelease
from   Mes_WorkOrderOperation o
join   Mes_WorkOrderOperationGds t on t.idworkorderoperation = o.id
join   Mct_MaterialFulfillmentType ft on ft.id = t.idmaterialfulfillmenttype
join   bs_goods g on g.id = t.idgds
where  --@resultCondition
and    ft.scode in ('20', '25', '30')
group by t.idgds
      ,g.smnemocode_dz
      ,g.sheadline_dz

// После
select t.idgds
      ,max(g.smnemocode_dz) as idGdsMC
      ,max(g.sheadline_dz) as idGdsHL
      ,greatest(sum(case when ft.scode in ('20', '25') then t.nQtyRelease end), sum(case
↪when ft.scode in ('30') then t.nQtyRelease end)) as nQtyRelease
from   Mes_WorkOrderOperation o
join   Mes_WorkOrderOperationGds t on t.idworkorderoperation = o.id
join   Mct_MaterialFulfillmentType ft on ft.id = t.idmaterialfulfillmenttype
join   bs_goods g on g.id = t.idgds
where  --@resultCondition
and    ft.scode in ('20', '25', '30')
group by t.idgds
```

Фильтруйтесь по индексируемым полям, вместо служебных полей

Например, при фильтрации по мнемокоду надо использовать поле, которое является мнемокодом, а не служебное smnemocode_dz. Потому что индекс создается на поле, которое является мнемокодом. Пример:

```
// До
select crt.id from oil_contrasroletype crt where crt.smnemocode_dz = 'Surveyor'

// После
select crt.id from oil_contrasroletype crt where crt.scode = 'Surveyor'
```

При соединении таблиц по идентификаторам, избегайте различных преобразований

При добавлении в запросе различных преобразований на индексируемые поля не будет использоваться индекс и может существенно замедлиться запрос. Пример:

```
// До
select t.id
from bs_goods t
join msr_measureitem mm on coalesce(t.idmeasureitem,0) = coalesce(mm.id,0)
where t.id = 1
// После
select t.id
from bs_goods t
left join msr_measureitem mm on t.idmeasureitem = mm.id
where t.id = 1
```

Используйте правильное приведение типов, в соответствие, с типом данных поля

При добавлении в запросе условий с не верными типами данных на индексируемые поля не будет использоваться индекс и может существенно замедлиться запрос. Пример:

```
// До
select c.id, c.sName
from oco_query t
join btk_class c on c.id = cast(substring(t.gidobject,'^[^/]*')as numeric)
// После
select c.id, c.sName
from oco_query t
join btk_class c on c.id = cast(substring(t.gidobject,'^[^/]*')as bigint)
```

Пример:

```
// До
select *
from stm_actin t
where to_char(t.dDocIn,'yyyy') >= '2024'
// После
select *
from stm_actin t
where t.dDocIn >= to_date('01.01.2024','dd.mm.yyyy')
```

Добавляйте защиту от закливания в рекурсивных запросах

Пример:

```
// До
with recursive dw_graph as
    (select t.id
     ,t.idClass
     ,t.gidSrc
     ,t.gidRec
```

(продолжается на следующей странице)

```

        ,t.gidSrc as gid
        ,t.gidRec as gidDoc
    from Bts_DocLink t
    where t.gidSrc = :super$$gid
    union all
    select t.id
        ,t.idClass
        ,t.gidSrc
        ,t.gidRec
        ,t.gidSrc as gid
        ,t.gidRec as gidDoc
    from Bts_DocLink t
    ,dw_graph sg
    where t.gidSrc = sg.gidRec
    )
SELECT * FROM dw_graph
// После
with recursive dw_graph as
    (select t.id
        ,t.idClass
        ,t.gidSrc
        ,t.gidRec
        ,t.gidSrc as gid
        ,array[''||t.gidRec] as path
        ,t.gidRec as gidDoc
    from Bts_DocLink t
    where t.gidSrc = :super$$gid
    union all
    select t.id
        ,t.idClass
        ,t.gidSrc
        ,t.gidRec
        ,t.gidSrc as gid
        ,path || cast(t.gidRec as varchar)
        ,t.gidRec as gidDoc
    from Bts_DocLink t
    ,dw_graph sg
    where t.gidSrc = sg.gidRec
    and t.gidRec != any(path)
    )
SELECT * FROM dw_graph

```

Используйте not exists вместо not in

Фильтрация будет срабатывать без использования индекса и то что находится внутри not in, в большинстве случаев тоже Пример:

```
// До
select emp.gid as gidEmployee
      , per.idStockOut as idStock
      , row_number() over (order by emp.gid) nCountCreated
      , count(emp.gid) over () as nTotalCount
from Bs_Employee emp
join Bs_Person per on per.id = emp.idPerson
where per.idStockOut is not null
and emp.id not in (select r.idEmployee from Ppm_PpmNeedReg r where r.idEmployee is not null)
and per.idStockOut is not null
and emp.gid in (select det.gidEmployee
                from Ppm_GoodsNeedDet det
                join Bs_Goods gds on gds.id = det.idGds
                where det.idDoc = $idvGoodsNeed
                and gds.idObjectTypeSuite in (select unnest($LongPgArray(Ppm_
ObjectTypeIdGoodsNeedOverrideApi().getidaSuiteType(ropGoodsNeed.get(_idObjectType))))))
order by emp.gid
// После
select emp.gid as gidEmployee
      , per.idStockOut as idStock
      , row_number() over (order by emp.gid) nCountCreated
      , count(emp.gid) over () as nTotalCount
from Bs_Employee emp
join Bs_Person per on per.id = emp.idPerson
where per.idStockOut is not null
and not exists (select 1 from Ppm_PpmNeedReg r where r.idEmployee = emp.id) -- так
--смотрим новых сотрудников
and per.idStockOut is not null
and emp.gid in (select det.gidEmployee
                from Ppm_GoodsNeedDet det
                join Bs_Goods gds on gds.id = det.idGds
                where det.idDoc = $idvGoodsNeed
                and gds.idObjectTypeSuite in (select unnest($LongPgArray(Ppm_
ObjectTypeIdGoodsNeedOverrideApi().getidaSuiteType(ropGoodsNeed.get(_idObjectType))))))
order by emp.gid
```

14 Приложение 2. Руководство администратора

14.1 Предисловие

Добро пожаловать в Руководство администратора GlobalFramework. Цель руководства предоставить информацию о принципах и способах администрирования решений на основе GlobalFramework.

На кого ориентировано руководство

Руководство ориентировано на:

- Разработчиков системы
- Аналитиков, настраивающих доступ к системе на проектах

14.2 Введение

Действие пользователя в системе требует наличия у него тех или иных привилегий. Администрирование доступа производится путем выдачи пользователям прав на объектные привилегии или элементарные привилегии.

Пользователь

Для работы в системе необходимо зайти под пользователем системы. С точки зрения системы пользователем является учетная запись, на которую выдается перечень прав, дающий возможность юридическому или физическому лицу выполнять действия в системе.

Список пользователей можно открыть из пункта меню:

- Администратор \ Доступ > Пользователи

Ограничение имени пользователя

Имя пользователя имеет вид <локальная_часть>@<домен>, домен не является обязательной частью имени пользователя.

Имя пользователя уникально в пределах системы.

Общая длина имени пользователя не должна превышать 1024 байт (октет), включая символ @

Ограничения локальной части (До последнего символа @):

- Прописные и строчные буквы латинского алфавита: (A-Z), (a-z)
- Цифры: (0-9)
- Специальные символы: !#\$%&*+,-/=^_`{|}~
- Символ (.), если он не является первым или последним, а так же если они не идут друг за другом.
- Следующие специальные символы разрешены, только если локальная часть выделена в кавычки: пробел и «(),,;<>[]

Ограничения домена (После последнего символа @):

- Прописные и строчные буквы латинского алфавита: (A-Z), (a-z)

- Цифры: (0-9)
- Специальный символ: -, если он не является первым или последним

Имя пользователя должно составляться по стандарту [RFC 822](#)

Длина имени используется из стандарта [Active Directory](#)

Отключение администрирования

В случае, если у пользователя стоит признак **Супер-пользователь**, настройки администрирования не применяются, под таким пользователем можно совершать в системе любые действия. Данный признак удобно использовать для разработки и тестирования, так как это позволяет отделить администрирование от разработки.

Администрируемый объект

Администрируемый объект - это логически связанный для массовой раздачи прав набор привилегий. Административный объект генерируется по объектам системы и позволяет единым образом выдавать права на них.

Администрируемый объект генерируется в момент установки или обновления системы по таким объектам как:

- Произвольная выборка
- Класс
 - Справочник
 - Настройка
 - Журнал
 - Документы
- Пакеты
- Приложения

Список административных объектов можно открыть из пункта меню:

- Администратор \ Настройки > Администрируемые объекты

Отключение администрирования

Возможно отключить администрирование в разрезе административного объекта, данным механизмом удобно пользоваться в случае поэтапной настройки прав, когда администрирование включается только для тех объектов, на которых оно уже настроено. Для отключения администрирования на административном объекте существуют следующие поля:

- **Не распространяются настройки администрирования**
Если флаг установлен, на объекты представляющие состав административных объектов не распространяются настройки администрирования
- **Не требуется настройка прав доступа на состояния**
Если флаг установлен, на документе данного административного объекта выключается контроль доступных переходов между состояниями.

Узел администрирования

Группирует элементы администрируемого объекта для массовой раздачи прав, по умолчанию создается одна основная группа на каждый администрируемый объект.

Узел администрирования зарезервирован на случай, если в системе потребуется настраивать права в зависимости от **типа объекта** или **состояния**

Элементы администрируемого объекта

Элемент административного объекта создается на объекты которые требуют раздачи прав на элементарные привилегии.

Выборка

По умолчанию выборка со всеми отображениями регистрируется в один **элемент администрируемого объекта** с именем {SelName}#Default. При этом на каждую операцию и атрибут в разрезе системного имени создается **элементарная привилегия**.

Для переноса отображения в отдельный **элемент администрируемого объекта** существует аннотация:

```
@AcItemRep(name = "{Name}")
```

В таком случае элемент администрируемого объекта получит имя {SelName}#{Name} Так же элемент администрируемого объекта можно задать в avm.xml через атрибут отображения acItemRep

Примечание

Изменения вступят в силу после обновления административного объекта (см. главу Групповая настройка привилегий ролей)

Для создания элементов администрируемого объекта по выборкам, не имеющим класса, в avm файле обязательно должен быть тег <acObject/>. Для выборок без класса формируется администрируемый объект и элемент администрируемого объекта с суффиксом Avi в имени.

Дискретные привилегии

Дискретные привилегии используют правила дискретного доступа при проверке прав, что позволяет раздавать доступ по объектно (в зависимости от значений атрибутов строки). В интерфейсе настройки ролей дискретные привилегии можно выдать с ограничением по дискретном правилу.

Элементарные привилегии

Элементарные привилегии используются для разграничения действий пользователя и создаются следующим образом:

- по выборке на:
 - Операции
Создаются дискретные привилегии
 - Атрибуты
 - Сеттеры атрибута
Создаются дискретные привилегии

Типы элементарных привилегий

- Чтение
Содержит настройки видимости атрибутов.
- Редактирование
Содержит сеттеры атрибутов.
- Добавление
Содержит операции вставки и копирования.
- Удаление
Содержит операция удаления.
- Интерактивные права
Содержит остальные операции.

Для изменения типа привилегии, которой соответствует операция выборки, существует аннотация `@AcPrivilegeType`. Пример:

```
@AcPrivilegeType(value = PrivilegeTypes.Read)
def myOperation(): Unit = {}
```

Примечание

Изменения вступят в силу после обновления административного объекта (см. главу Групповая настройка привилегий ролей)

Объектные привилегии

Объектные привилегии позволяют раздавать права на бизнес логику.

Авто генерируемые объектные привилегии

- edit#
Создаются дискретные привилегии
- view#
Создаются дискретные привилегии
- viewReport#
Создаются дискретные привилегии

Привилегии на переход состояний

Привилегии на переход состояний ограничивают доступные переходы состояния. Ограничения на переходы реализовано в стандартном выпадающем списке для состояний `ru.bitec.app.btk.Btk_ClassStateAvi.Lookup_Class`

Имя привилегии формируется по правилу:

```
s"StateChange_OT:${idObjectType}_SS:${idStartState}_FS:${idFinishState}"
```

Функция для определения возможности перехода: `ru.bitec.app.btk.Btk_AdminPkg#hasStateChangePriv.`

Ручное создание объектных привилегий

Для ручного создания объектных привилегий на классе(`odm.xml`) или пакете(`pkg.xml`) используются xml разметка, пример:

```
<admin>
  <privileges>
    <privilege name="SbtManage" caption="Управление решением"/>
  </privileges>
</admin>
```

Примечание

Изменения вступят в силу после обновления административного объекта (см. главу Групповая настройка привилегий ролей)

Приложение

Для администрирования приложений создается адм. объект `Приложения(Btk_ApplicationsListPkg)`. Объектные привилегии импортируются в него из xml файлов `META-INF/applications.xml` всех подключенных модулей.

Выданные права на такую привилегию обеспечивают отображение приложения в списке в меню выбора приложений.

Дискретные ограничения

Дискретные ограничения позволяют задавать выражения для создания правил выдачи дискретных привилегий на конкретной роли. Подробности смотрите в главе **Дискретный доступ** Принцип настройки дискретных ограничений:

1. Администратор определяет дискретные ограничения на административном объекте
2. Администратор определяет дискретные правила в роли
Каждое правило содержит дискретное ограничение и набор параметров
3. Администратор выдает грант на дискретную привилегию роли
При этом при синхронизации пользователей проходит расчет дискретных привилегий данных пользователю по алгоритму объединения грантов

При проверке привилегии в бизнес логике, в функцию проверки передается перечень строк, для которых выполняется проверка. При ограничении запроса по дискретной привилегии, каждое ограничение добавляется в where условие.

Алгоритм объединение грантов:

1. Пусть $g1$ и $g2$ гранты пользователю на объектную привилегию p
2. Если $g1$ или $g2$ с типом полный доступ, вернуть грант на p с полным доступом
3. Иначе вернуть грант на p объединив ограничения

Алгоритм объединения ограничений:

1. Пусть A массив ограничений для грантов $g1$ и $g2$
2. Пусть S пустое множество ограничений(параметры ограничения не входят в ключ множества)
3. Для каждой g из A
 1. Если g существует в S , добавить новые параметры к ограничению в S
 2. Иначе добавить g к S
4. Вернуть S

Скрипт фильтрации

Используется в универсальной фильтрации для ограничения видимости по объектам.

Скрипт проверки объектного кэша

Используется в автономной бизнес логики для проверки возможности действия со строкой.

Роль

Роль - административная единица для выдачи прав. Для раздачи дискретных привилегий на роли могут быть настроены дискретные правила, которые определяют дискретные ограничения и передаваемые в них на гранте параметры.

В роле выдаются гранты на привилегии.

Список ролей можно открыть из пункта меню:

- Администратор \ Доступ > Роли

Типы ролей

Система поддерживает следующие типы ролей:

- Обычная роль - стандартная роль с набором привилегий
- Мастер-роль - специальная роль, позволяющая определять параметры дискретного доступа для подчиненных профилей

Профиль пользователя

Профили пользователя используется для группировки ролей для более удобной раздачи их пользователю.

Список профилей можно открыть из пункта меню:

- Администратор \ Доступ > Профили

Типы профилей

Система поддерживает следующие типы профилей:

- обычный профиль - стандартный профиль с набором ролей
- Мастер-профиль - профиль, содержащий мастер-роли, от которых наследуются права
- Подчиненный профиль - профиль, который должен быть связан с мастер-профилем

Особенности подчиненных профилей

- Обязательно указание мастер-профиля
- Не могут содержать собственные роли (права транслируются от мастер-профиля)
- Требуют указания значений параметров с типом «Определяется профилем» из мастер-ролей
- Если параметры разных бизнес-объектов имеют одинаковые системные имена, значение указывается один раз

{note} Пользователям можно назначать только **обычные** и **подчиненные** профили. Деривация мастер-профилей Система предоставляет удобный интерфейс для быстрого создания подчиненных профилей на основе мастер-профиля с указанием необходимых параметров.

Замещение прав

Позволяет указать какой пользователь будет обладать всеми правами другого и на какое время. Настройки замещения прав учитываются при выполнении индексации прав пользователя. Для отслеживания окончания периода действия замещения каждую ночь выполняется задание, которое выполняет индексацию пользователей.

Список замещений можно открыть из пункта меню:

- Администратор \ Доступ > Замещение прав

Индексация привилегий пользователей

Для быстрой проверки привилегий пользователя используются **индексы привилегий**. Индексы привилегий необходимо пересчитывать после завершения редактирования ролей и их выдаче пользователю.

Предустановленные профили

Профиль «Отладчик»

Профиль "Отладчик" предназначен для пользователей, для которых нужно предоставить доступ к инструментам отладки. Таким как:

- доступ к операциям управления решением (управление SBT)
- возможность открыть дебаггер
- выполнение jexl скриптов в дебаггере
- доступ к инструментам меню Сервиса
- доступ к инструментам отладки клиента

Профиль «Разработчик»

Профиль "Разработчик" предназначен для пользователей, которые являются разработчиками. Включает в себя все права профиля «Отладчик». А также имеет доступ к следующим инструментам:

- доступ к редактированию справки

14.3 Приложение администратор

Приложение Администратор используется для администрирования пользователей.

Список пользователей

Список пользователей можно открыть из пункта меню:

- Администратор \ Доступ > Пользователи

Операции списка пользователей:

- **Пересчитать индексацию привилегий для текущего пользователя**
Операция предназначена для обновления индексов системных привилегий для текущего пользователя. В результате операции обновятся записи в индексе(`Btk_AcUserItemPrivFlat`) по всем элементам администрируемых объектов на привилегии, которых у пользователя есть права, а также записи в индексе(`Btk_AcUserObjPrivFlat`) по всем администрируемым объектам, на объектные права которых у пользователя есть права.
- **Пересчитать индексацию привилегий для всех пользователей**
Операция предназначена для обновления индексов системных привилегий для всех пользователей.

Карточка пользователя

Операции карточки пользователя:

- **Изменить пароль**
Операция предназначена для установки пароля для входа в систему для пользователя.
- **Пересчитать индексацию привилегий для текущего пользователя**
Операция предназначена для обновления индексов привилегий для текущего пользователя.

Детализация пользователя

В карточке и списке доступна детализация, содержащая закладки:

- **Профили доступа**
Список записей в классе-развязке(`Btk_AcUserGrant`) со ссылкой на профиль. Для закладки **Профили доступа** доступна детализация с закладкой **Роли доступа профиля**, отображающая список записей в классе-развязке(`Btk_AcProfileGrant`) со ссылкой на роль
- **Роли доступа**
Список ролей, связанных с пользователем через профиль
- **Индексация элементарных привилегий**
Список записей в `Btk_AcUserItemPrivFlat` для текущего пользователя.
- **Индексация объектных привилегий**
Список записей в `Btk_AcUserObjPrivFlat` для текущего пользователя.

Для закладок **Индексация элементарных привилегий** и **Индексация объектных привилегий** доступна операция

- **Посмотреть значение индекса**
Операция открывает в модальном окне содержимое поля(`cPriv`) с перечнем доступных привилегий для элемента или объекта.

Синхронизация с Active Directory

Для синхронизации:

1. Выполните операцию **Синхронизация с Active Directory**
Операция доступна в списке пользователей. При этом откроется интерфейс синхронизации отображающий список логов.
2. При необходимости укажите шаблон в шапке интерфейса
Вновь созданные пользователи будут созданы по этому шаблону
3. Нажмите кнопку **Синхронизировать**
При этом будут просканированы домены на наличие групп и пользователей. Произойдет синхронизация пользователей

Синхронизируемые поля пользователей:

- **Системное имя**
Атрибут `sAMAccountName`, если он пустой, то берётся из `uid`
- **Пароль**
Равен системному имени при создании пользователя
- **Группа**
Определяется через атрибут `memberOf`
- **Подразделение Active Directory** Атрибут `department`
- **Наименование Active Directory** Атрибут `title`
- **Описание Active Directory** Атрибут `description`
- Для физ. лица:
 - **Фамилия**
Первое слово из атрибута `displayName`, если в нём три слова. В ином случае атрибут `sn`.
 - **Имя**
Второе слово из атрибута `displayName`, если в нём три слова. Первое слово из атрибута `givenName`, если в нём два слова. В ином случае атрибут `givenName` целиком.
 - **отчество**
Третье слово из атрибута `displayName`, если в нём три слова. Второе слово из атрибута `givenName`, если в нём два слова.
 - **email**
Атрибут `mail`

Синхронизируемые поля групп:

- **Наименование**
Атрибут `name`, если он не указан, то полное имя. Например: `CN=Domain users,CN=Users`
- **Системное имя**
Полное имя группы. Например: `CN=Domain users,CN=Users`

Пользователи создаются только в случае, если они не заблокированы в домене. Если пользователь заблокирован, и уже зарегистрирован в системе Global, то ему выставляется признак «Заблокирован».

Помимо непосредственной синхронизации пользователей, происходит поиск/создание физ. лиц для пользователей. Для тех пользователей, у которых нашлось/создалось физическое лицо, создаётся ещё и сотрудник в `Bs_Employee`. В случае если у группы пользователя настроена ещё и связь с ОФС, то для сотрудника создаётся соответствующая запись в `Bs_OFSEmployee`. Если у связанной ОФС настроены профили доступа, то эти пользователи применяются и на пользователя.

Настройка синхронизации

Для настройки подключения используется универсальный класс настроек (`Btk_Setting`). Открыть настройки можно из пункта меню **Настройка системы \ Настройки и сервисы > Настройки модулей системы > Общие настройки системы**. Далее следует открыть настройки модуля `btk` и выбрать настройку `ldapSync`. Доступные параметры:

- **Создавать физ. лицо**

Если параметр выключен - для пользователей, которые появились в результате синхронизации, будут искаться соответствующие физ. лица (`Bs_Person`). Поиск осуществляется по совпадению фамилии, имени и отчества. В физ. лице будет установлена ссылка на пользователя и почта (при наличии в AD).

Если параметр включен - для пользователей, у которых не удалось найти существующее физ. лицо, будет создано новое физ. лицо с ФИО и почтой из AD

- **Тип синхронизации**

Все пользователи - Настройка фильтра по группе отключается и синхронизация возьмёт всех пользователей из выбранных доменов вместе с их группами. Группа переносится из AD только в случае, если она находится в одном из выбранных доменов. Если группа находится вне списка выбранных доменов, то она не переносится.

Пользователи группы - Настройка фильтра по группе включается и в выбранных доменов идёт поиск всех подгрупп группы указанной в фильтре. Далее ищутся пользователи, которые входят в найденные подгруппы.

- **Фильтр по группе**

Механика работы описана выше. Задаётся в виде `CN=...,OU=...,DC=...`

- **Домены**

В этом параметре можно задать список доменов, в которых будет осуществляться поиск, у каждого домена есть свои параметры:

- **Добавить домен к пользователю**

Если параметр активен, то к имени пользователя будет добавляться имя домена, в котором он был найден. В противном случае - имя пользователя будет браться из AD без изменений

- **Домен**

Пример: `part.gs.local`

- **Логин**

- **Пароль**

- **URL**

Пример: `ldap://192.168.2.11:389`

Так же для применения профилей доступа на пользователей, можно задать связь между пользовательскими группами и ОФС в классе «Соответствие групп пользователей и офс» (`Bs_OFStructureUserGroupLink`). Тогда ко всем пользователям группы в будущем при синхронизации будет применяться профиль доступа настроенный на ОФС.

Список ролей

Список ролей можно открыть из пункта меню:

- Администратор \ Доступ > Роли

Карточка роли

Используется для выдачи привилегий для роли от дерева административных объектов.

Права роли

В данном интерфейсе отображается дерево административных объектов с уровнями:

- Группа администрируемых объектов(только чтение)
 - Объект администрирования
 - * Узел администрирования
 - Элемент администрируемого объекта

Совет

По умолчанию отображаются только административные объекты использующиеся в роли. Для отображения всех административных объектов выключите флаг **Только объекты, на которые у роли имеются права** в панели фильтрации.

В данной интерфейсе каждый тип элементарной привилегии отображен отдельным столбцом, что позволяет выдать права массово по типу для всех элементарных привилегий этого типа для текущего уровня и уровней ниже.

Расшифровка прав

В случае если выбран объект администрирования отображает:

- перечень типов элементарных привилегий
- перечень объектных привилегий

В случае если выбран элемент администрируемого объекта отображает:

- перечень элементарных привилегий для данной строки

Перевод состояний

Отображается в случае если выбран объект администрирования. Позволяет настроить возможные переходы между состояниями.

Функции отображения

Для работы системы администрирования в отображении объявлены следующие функции:

- `onLoadAdminMeta` - Применяет настройки администрирования.
- `acObject` - Возвращает имя администрируемого объекта.
- `acObjectBundle` - Имя узла администрирования.
- `acObjectItem` - Имя элемента администрирования.
- `isAdministraded` - Признак, что выборка администрируется.

Функции проверки привилегий и ролей

- `ru.bitec.app.btk.Btk_AdminPkg#hasObjPriv` - Проверяет наличие объектной привилегии у пользователя.
- `ru.bitec.app.btk.Btk_AdminPkg#hasStateChangePriv` - Проверяет наличие привилегии на перевод состояния.
- `ru.bitec.app.btk.Btk_AdminPkg#hasRole` - Проверяет наличие у пользователя роли.

Выдача прав на объектные привилегии

1. Выберите требуемый адм. объект.
При этом в расшифровке прав отобразятся объектные привилегии
2. Раздайте права на объектные привилегии роли \
3. Обновите индексы системных привилегий

Выдача прав на тип элементарных привилегий

В случае если выдано право на тип элементарной привилегии, доступ будет дан всем элементарным привилегиям данного типа во всех элементах администрируемого объекта.

1. Выберите требуемый адм. объект
При этом в расшифровке прав отобразятся типы элементарных привилегий.
2. Раздайте права на типы элементарных привилегий
Права на типы можно раздавать либо в расшифровке пара либо в дереве административных объектов
3. Обновите индексы системных привилегий

Выдача прав на элементарную привилегию

1. Выберите требуемый элемент администрируемого объекта
При этом в расшифровке прав отобразятся типы элементарных привилегий.
2. Раздайте права на элементарные привилегий
Для этого в расшифровке прав выберите нужный элемент и предоставьте к нему доступ.
3. Обновите индексы системных привилегий

Примечание

Если установить признак «Запрещено», то пользователи, обладающие этой ролью, не будут иметь прав на запрещенную привилегию, даже если другие роли дают на нее доступ.

Администрирование универсальных характеристик

Для разграничения доступа пользователей к универсальным характеристикам в динамически расширенных (с помощью свойства атрибута «Видимость в списке») списках и карточках объектных характеристик предусмотрены стандартные привилегии на чтение и изменение. Они названы в соответствии с системными именами УХ и имеют вид `UC_<системное имя УХ>`

Примечание

Если внутри элемента администрируемого объекта отсутствуют права на чтение и изменение уже наложенных универсальных характеристик, необходимо обновить администрируемый объект!

Кроме того, определены привилегии, предоставляющие доступ ко всем универсальным характеристикам сразу. При выдаче для роли такой «общей» привилегии открывается доступ ко всем универсальным характеристикам **класса**, **шаблона** и **типа объекта**. Для ссылочных атрибутов в имени привилегии используется постфикс HL, для значимых - без него.

Список «общих» привилегий:

- Для расширенных списков
 - Чтение
 - * objAttrValue
 - * objAttrValueHL
 - Редактирование
 - * setobjAttrValue
 - * setobjAttrValueHL
- Для карточек объектных характеристик
 - Чтение
 - * sValue
 - * sValueHL
 - Редактирование
 - * setsValue

* setValueHL

Явные настройки на уровне конкретной характеристики имеют приоритет над общими:

Сценарий	Глобальная привилегия	Настройка для конкретной УХ	Результат
Глобально редактирование разрешено, одна УХ запрещена	setobjAttrVa	setUC_nNum	Все УХ доступны для редактирования, кроме nNum
Глобально чтение запрещено, одна УХ разрешена	sValueHL	UC_idDepOwner	Чтение всех УХ запрещено, но idDepOwner доступна

Выдача прав на использование Rest-пакетов

1. В карточке Роли на закладке Права на Rest-пакеты отображен список пакетов. Чекбокс Rest-пакет указывает, является ли пакет Rest-пакетом.
2. Для выдачи\отзыва доступа используйте чекбокс Имеет доступ.
3. Обновите индексы системных привилегий по роли.

Примечание

Обновление списка Rest-пакетов происходит при обновлении адм.объектов по модулю. С пакетов, которые перестали быть Rest-пакетами, чекбокс Rest-пакет автоматически снимается.

Групповая настройка привилегий ролей

Выборка Групповая настройка привилегий ролей предназначена для массовой раздачи привилегий по ролям от административного объекта.

Выборку можно открыть из пункта меню:

- Администратор \ Доступ > Групповая настройка привилегий ролей

Общий принцип работы с выборкой:

1. Выберите требуемый административный объект
2. Выберите требуемые роли
Для этого в панели фильтрации привилегий откройте список подбора для поля Перечень ролей
3. Проставьте гранты на требуемых привилегиях
4. Пересчитайте индексы привилегий
Для этого выполните операцию Пересчитать индексацию привилегий в интерфейсе Перечень ролей

Дерева администрируемых объектов

Дерево администрируемых объектов состоит из следующих уровней:

- Группа администрируемых объектов (Опционально)
 - Объект администрирования
 - * Узел администрирования
 - Элемент администрируемого объекта

Объекты администрирования могут не входить в группу, тогда они будут считаться корневыми записями.

Операции дерева администрируемых объектов

- **Обновить выбранный объект**
Операция предназначена для обновления выделенного объекта или элемента адм. объекта. При обновлении административного объекта происходит сканирование исходного кода по которому добавляются недостающие привилегии и элементы.
- **Обновить адм. объект по имени**
При этом произойдет запрос имени и произойдет обновление административного объекта по выбранному имени
- **Обновить адм. объекты модуля**
Операция предназначена для обновления адм. объектов по всему модулю

Примечание

Классы-коллекции не имеют собственных объектов администрирования и входят в объект мастера. Поэтому для обновления настроек для коллекций необходимо обновлять адм. объект по имени класса мастера.

Настройка прав

Закладка отображает настройку прав для ролей в зависимости от выделенной записи в дереве администрируемых объектов:

1. Для административного объекта:
 - объектные привилегии
2. Для элемента администрирования:
 - типы элементарных привилегий
 - элементарные привилегии

Колонки ролей в настройке прав формируются динамически по ролям из фильтра.

Операции настройки прав:

- **Подобрать роли**

Операция предназначена для подбора в фильтр ролей, для которых есть какие-либо права для текущего объекта или элемента.

Пользователи

Отображаются пользователи для роли в активной ячейки.

Профили

Отображаются профили для роли в активной ячейки.

Индексация привилегий для пользователей

Индексация пользователя

Для пересчета индекса по 1-му пользователю существует операция **Пересчитать индексацию привилегий для текущего пользователя**. Операция доступна:

- в списке и карточке **Пользователей**
- на закладке **Пользователи** роли в списке и карточке **Ролей**.

Индексация всех пользователей

Для индексации привилегий всех пользователей предназначена операция **Пересчитать индексацию привилегий для всех пользователей**. Операция доступна:

- в списке **Пользователей**.

Аудит прав доступа

Историю изменений прав доступа можно посмотреть в **Аудит прав доступа**, находящемся в пункте меню **Аудит**.

Примечание

Для включения аудита изменений прав доступа необходимо выполнить операцию **Включить аудит для структур прав доступа** в этом меню.

В самом типе необходимо выбрать **Вид объекта** (**Пользователь**, **Профиль** или **Роль**), **Объект** и, если необходимо, период времени.

Пользователи

Для пользователей в аудите отображаются **Профили**, которые были выданы пользователю или отобраны у него, дата изменения и изменивший пользователь.

Профили

Для профилей в аудите отображаются **Роли**, которые включали в профиль или убрали из него, и **Пользователи**, которым выдавался профиль или его отбирали, а также дата изменения и изменивший пользователь.

Примечание

В списке изменений для **Пользователей** и **Профилей** есть операция **Изменения по объекту**, которая позволяет перейти к аудиту другого объекта, который является выделенным.

Роли

Для ролей список изменений представляет собой две выборки:

- **Дерево администрируемых объектов**
Отображаются администрируемые объекты или элементы администрирования, по которым для данной роли были совершены изменения
- **Список привилегий конкретного администрируемого объекта**
Список привилегий конкретного адм. объекта или элемента администрирования, которые изменялись для данной роли

Список привилегий представляет из себя:

1. Имя адм. объекта или элемента администрирования
2. Тип привилегии:
 - Изменение атрибута (например, при простановке **Не распространяются настройки администрирования** для адм. объекта),
 - Объектная привилегия,
 - Элементарная привилегия,
 - Группа элементарных привилегий (когда изменяются доступ не на конкретные элементарные привилегии, а на всю группу, например, Редактирование)
 - Перевод состояния
3. Системное имя привилегии
4. Привилегия - наименование привилегии
5. Изменивший пользователь
6. Дата изменения
7. Системное имя атрибута
8. Наименование атрибута
9. Старое значение - старое значение атрибута

10. Новое значение - новое значение атрибута

Примечание

Для списка привилегий есть фильтр **С входящими**, который в случае, когда в дереве адм. объектов выбран адм. объект, а не элемент администрирования, отображает изменения не только администрируемого объекта, но и элементов администрирования, входящих в этот объект.

Трассировка прав доступа

Запуск трассировки прав пользователя

Для запуска трассировки прав пользователя, выполните следующие шаги:

- Перейдите в раздел «Сервис» в меню системы.
- В меню «Инструменты» найдите опцию «Начать трассировку прав» и выберите её.

После начала трассировки, система будет отслеживать все ваши действия. Выполняйте необходимые действия с данными в системе, так как вам требуется.

Чтобы завершить трассировку прав пользователя, выполните следующие шаги:

- Перейдите снова в раздел «Сервис» в меню системы.
- В меню «Инструменты» выберите опцию «Закончить трассировку прав».

Отчет сессии трассировки прав

Отчет «сессии трассировки прав пользователей» выведен в приложении «Администратор», в меню «Отчеты». Отображает список сессий трассировки. При открытии карточки сессии в детализации будет отображена более подробная информация по проверкам прав, которые вызывались в ходе сессии трассировки.

Очистка записей

Сессии трассировки сохраняются в таблицу `Btk_AcTraceSession`, а детализация сессии в `Btk_AcTraceJournal`. Для очистки этих таблиц предусмотрен по умолчанию регистрируется задание очистки данных по классу «`Btk_AcTraceSession`», удаляющий записи, хранящиеся больше 30 дней.

Реализация трассировки

При запуске трассировки

- в параметрах рабочей сессии сохраняется параметр активности трассировки;
- создается новая запись в `Btk_AcTraceSession` в логирующей транзакции;
- в параметрах рабочей сессии сохраняется параметр с идентификатором запущенной сессии трассировки;
- для уже открытых выборок подключается подписка на события, так же она подключается на `onLoadAdminMeta` при открытии новой выборки; При действиях пользователя во время трассировки

- на методах проверки прав доступа, вызов операции/сеттера на администрируемых выборках, создается запись в логирующей транзакции в Btk_AcTraceJournal; При завершении трассировки
- вызывается коммит логирующих транзакций со всех запущенных сессий выборок;
- в сессию трассировки Btk_AcTraceSession сохраняется дата завершения;
- отключается подписка на события, подключенная для логирования;
- из параметров рабочей сессии удаляются параметр активности трассировки и идентификатор сессии трассировки.

14.4 Дискретный доступ

Дискретный доступ позволяет выдавать привилегии в рамках объектов классов или записей в администрируемой выборке.

Настройка привилегии

Административный объект

1. Открыть карточку Адм. объекта
2. Установить признак Дискретный доступ
3. Перейти на закладку Дискретные ограничения доступа
4. Создать Дискретное ограничение.

Дискретное ограничение доступа

1. Создать новую запись или встать на запись для редактирования
2. Указать уникальное в рамках адм. объекта имя и наименование
3. Настроить тип правила
4. Настроить параметры правила.
5. На закладке Скрипт для фильтрации объектных привилегий написать sql-запрос, который будет фильтровать списки объектов
6. На закладке Скрипт проверки строк по объектному кешу написать jexl-скрипт, который будет по гор-у объекта или строке выборки возвращать признак, что условие выполнилось для этого объекта.

Тип правила

Доступны 3 варианта типа правила:

- Примитивное
- Составное
- Без параметров

Примитивное правило

Состоит из одного параметра.

При анализе прав пользователя со всех его ролей будут собраны значения параметров и собраны в один массив примитивных значений. Например для числового правила будет собран массив вида: [10, 20, 30]

Составное правило

Состоит из нескольких параметров.

При анализе прав пользователя используется следующий алгоритм:

1. Для каждого набора значений, указанных на роли, формируется json-объект вида:

```
{
  <ИД параметра 1>: [<Массив указанных значений>],
  <ИД параметра 2>: [<Массив указанных значений>],
  ...
  <ИД параметра n>: [<Массив указанных значений>]
}
```

2. Каждая роль может обладать несколькими наборами значений на одно правило. По этому результатом агрегации настроенных значений на роли будет массив, содержащий объекты, указанные в п.1:

```
[
  {json-объект для набора 1},
  {json-объект для набора 2},
  ...
  {json-объект для набора n},
]
```

3. При агрегации прав пользователя:

- со всех ролей анализируются массивы, описанные в п.2
- в json-е заменяются ИД параметров на их имена
- каждый элемент массива приводится к строке
- собирается результирующий массив, содержащий уникальные строки.

Таким образом исключаются дублирующие настройки на ролях.

Пример агрегации прав пользователя:

```
[
  {
    "paramName1": ["a", "b", "c"],
    "paramName2": [1, 2, 3]
  },
  {
    "paramName1": ["c", "d", "e"],
    "paramName2": [3, 4, 5]
  }
]
```

(продолжается на следующей странице)

```
}
]
```

Правило без параметров

Не содержит параметров.

При анализе прав пользователя, если хотя на одной роли будет правило без параметров, оно будет проверяться.

Параметры правила

Позволяют настроить тип данных и ссылочность для значений, которые будут указаны в зависимости от выбранного типа определения:

- „Обычный“: значения параметров определяются на ролях.
- „Динамический“: значения параметров определяются динамически по скрипту динамического определения под параметром в настройке дискретного доступа на администрируемом объекте. на ролях.
- „Определяется профилем“: значения параметров в этом случае определяются от параметров с подчиненных профилей пользователя.

{important} Если для роли назначен параметр с типом «**Определяется профилем**», сохранить роль можно только если она имеет тип «**Мастер-роль**».

Если правило примитивное, то у него доступен для настройки только один параметр с системным именем «SimpleAcObjectRuleParam». Для составного правила доступны несколько параметров.

Динамическое определение значения параметров

Для некоторых случаев бывает удобнее не задавать на каждую роль набор параметров, а определять их динамически, например от текущего пользователя или даты. Для этого на роли в настройке правила дискретного доступа нужно:

- В карточке администрируемого объекта, на закладке с параметрами под динамическими параметрами написать jexl-скрипт, который возвращает массив строк Пример:

```
var idvUser = Btk_UserApi.getCurrentUserID(); //получаем id текущего пользователя
var res = [...]; //объявляем массив для сбора значений параметра
if (idvUser != null){
    var rvUser = Btk_UserApi.load(idvUser);
    var idvObjAttr = Btk_UserApi.getAttrValue(rvUser, "idObjAttr"); // получаем значение_
    атрибута с именем "idObjAttr", хранящийся для текущего пользователя
    if (!empty idvBisObj){
        res.add(idvBisObj.toString()); // добавляем полученное значение в массив
    }
}
res; // возвращаем массив со значениями
```

- На роли, на закладке с настройками дискретного доступа для нашего правила у параметра выбираем тип «Динамический».

Скрипт для фильтрации объектных привилегий

Используется для фильтрации списков объектов, через наложение макроса универсального фильтра `&DefUniFltMacros`.

Доступные макросы внутри текста скрипта:

- `(<Имя атрибута>)`
Будет заменено на указанное имя атрибута. Например, `t.id`
Для фильтрации доступны все атрибуты дата-сета выборки шапки БО.
- `(&params)`
Массив строк, в котором будут параметры, собранные со всех ролей пользователя.
- `(&CurrentUserID)`
Будет заменено на ИД текущего пользователя.
- `(&CurrentUserName)`
Будет заменено на имя текущего пользователя.

Запрос чаще всего представляет из себя `exist`, в котором джойнится таблица класса, и через массив значений накладываются условия фильтрации.

Скрипт проверки строк по объектному кешу

Используется для проверки ограничения по объектам, используя объектный кеш.

Представляет из себя `jexl`-скрипт, который возвращает `true`, если объект проходит условие, и `false` - если не проходит.

Доступные переменные:

- `row \`
 - Если адм. объект создан по классу, то в этой переменной будет `rop` проверяемого объекта. Это не исходный `rop`, а его обертка `JexlRop`, которая позволяет обращаться к полям объекта.

Совет

Для получения исходного `rop` воспользуйтесь следующим примером:

```
//в переменной jexlRop находится объект класса JexlRop
var rop = jexlRop.data()
```

- Если адм. объект создан не по классу, то здесь будет json-представление строки дата-сета выборки.
- `params`
массив значений параметров, собранных со всех ролей пользователя.
 - если правило примитивное - то будет содержать примитивы (строка или число)
 - если правило составное - то будет содержать строки, каждая из которых - json

Роль

1. Открыть карточку роли или выбрать ее в списке;
2. перейти на закладку **Дискретный доступ**;
3. создать новую запись, выбрав адм. объект;
4. указать значение параметров для ограничений;
5. сопоставить объектные и элементарные привилегии с нужным значением параметров ограничений, заполнив поле **Ограничение дискретного доступа**

Ограничение доступа к объектам класса.

Выдается через объектные привилегии с системными именами **edit#** и **view#**

Совет

Если таких привилегий нет у адм. объекта, то выполните его синхронизацию.

Дискретные ограничения применяются только, если:

- это главная выборка адм. объекта (шапка БО);
- это не супер-пользователь;
- к адм. объекту применяются настройки администрирования;
- адм. объект имеет признак **Дискретный доступ**.

Определение объектов для проверки дискретных прав

При проверке дискретных привилегий строки, в контексте которых требуется проверить наличие прав, определяются методом **acRows**

Алгоритм работы метода:

1. Если выборка принадлежит классу (**thisApi() != null**), то для всех выделенных записей через поле первичного ключа загружаются **провайдеры строк**, из них через поле **gidRoot_dz** определяются шапки БО
2. Если выборка не принадлежит классу (**thisApi() == null**), то по всем выделенным строкам собирается json-массив, содержащий на каждую строку json-объект, в котором ключ - имя атрибута, значение - значение атрибута.

Ограничение списков

Используя значения параметров всех ролей и настроенные «Скрипты для фильтрации объектных привилегий» формируется условие ограничения и накладывается фильтрация через макрос `&DefUniFltMacros`.

Ограничение открытия карточек

На открытие карточки используя значения параметров всех ролей выполняется `jexl`-скрипт, настроенный в Скрипт проверки строк по объектному кешу. Если у пользователя нет прав на объект, то будет выдана ошибка.

Ограничение на редактирование объектов

На `beforeEdit` или `delete` проверяется наличие привилегии `edit#` (выполняется `jexl`-скрипт, настроенный в Скрипт проверки строк по объектному кешу). Если у пользователя нет прав на объект, то будет выдана ошибка.

Ограничение на объектные привилегии

Если для объектной привилегии указано **Ограничение дискретного доступа**, то такая привилегия может быть проверена только в контексте какого-либо объекта. Для проверки объектных привилегий используется метод `Btk_AdminPkg().hasObjPriv`.

Пример:

```
if (Btk_AdminPkg().hasObjPriv("SomeObjectName", "SomePrivName", Seq(rop))) {  
  //код, выполняемый при наличии привилегии  
}
```

Если для привилегии, имеющей ограничение, выполнить проверку без передачи объектов, то будет выдана ошибка.

Ограничение на элементарные привилегии

Для элементарной привилегии может быть указано ограничение дискретного доступа. Если ограничение указано, то при выполнении операции будут проверены права в контексте записей в выборке.

На загрузку выборки или перезагрузку `ClassLoader`-а происходит подписка на событие выполнения операции.

При выполнении операции проверяются дискретные права на эту элементарную привилегию:

1. Получение `acRows`
2. Для полученных строк выполняется `jexl`-скрипт, настроенный в Скрипт проверки строк по объектному кешу
3. Если прав нет, то выдается ошибка.

Принудительное включение и отключение проверки дискретного доступа

Принудительное включение или отключение проверки дискретного доступа позволяет игнорировать настройки дискретного доступа на административном объекте. Признак устанавливается через параметр `UseDiscreteAccess` при открытии lookUp-формы через метод-обработчик.

Таким образом, установка параметра `UseDiscreteAccess` в значение равное 1 позволит принудительно применить настройки дискретного доступа, независимо от настройки на административном объекте. Пример принудительного включения проверки дискретного доступа:

```
Bs_BisObjAvi.processIdMCEvent(  
    event,  
    { id => setClientidBisObjFrom(editRef(), id) },  
    params = Map(RefEventParam.UseDiscreteAccess -> 1)  
)
```

При установке значения параметра `UseDiscreteAccess` в значение равное 0 настройки дискретного доступа будут игнорироваться, вне зависимости от настройки дискретного доступа на административном объекте. Пример принудительного выключения проверки дискретного доступа:

```
Bs_BisObjAvi.processIdMCEvent(  
    event,  
    { id => setClientidBisObjFrom(editRef(), id) },  
    params = Map(RefEventParam.UseDiscreteAccess -> 0)  
)
```

Если параметр не был передан при вызове метода-обработчика то проверка дискретного доступа осуществляется по настройке на административном объекте.

Параметр учитывается на всех методах-обработчиках (`processRefEvent`, `processHLEvent`, `processIdMCEvent` и т.д.).

Проверка прав доступа на объект после его редактирования

1. Открыть карточку Адм. объекта
2. Установить признак Дискретный доступ
3. Установить признак Проверять изменение прав объектов на сохранении

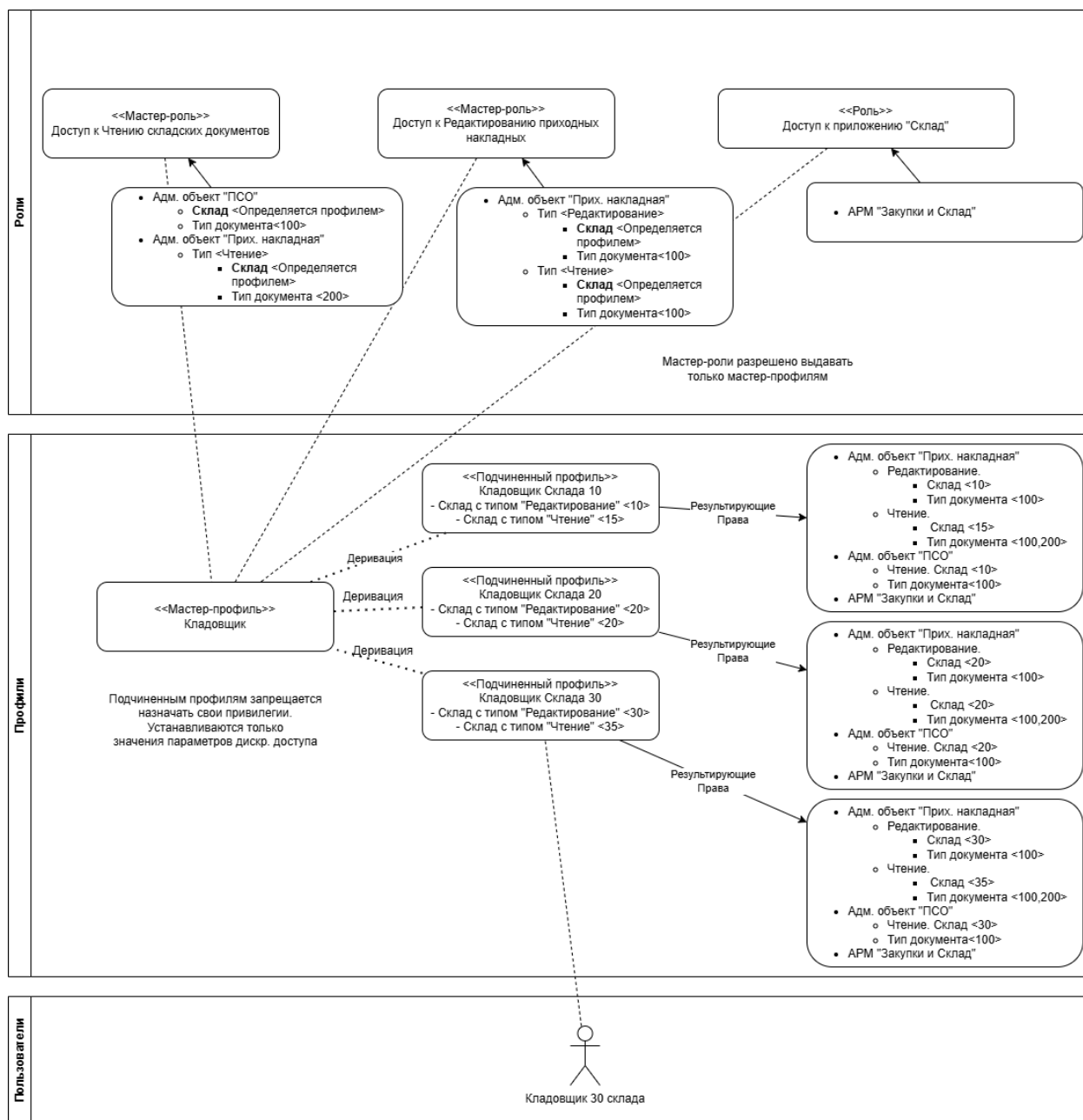
При включенном дискретном доступе и проверке изменения прав на сохранении, если при редактировании пользователем объекта, объект становится недоступным даже для чтения, система выдаст ошибку с откатом изменений.

Деривация профилей

Для дифференцирования доступа в разрезе требуемых организационных уровней (складов, бизнес-единиц, организаций и пр.), реализован следующий подход:

1. Роли разделяются на обычные и мастер-роли. Для мастер-ролей при назначении полномочий для бизнес-объектов появляется возможность указать, что значение параметров дискретного доступа определяется профилем доступа.
2. Профили доступа разделяются на мастер-профили, подчиненные и обычные. Мастер-роли можно включать только в мастер-профили доступа.

3. Для подчиненного профиля указываются конкретные значения параметров дискретного доступа.
4. При проверке прав пользователя система определяет параметры для дискретного доступа от параметров, указанных в подчиненном профиле.



Подправила

Особенности подправил

Для каждого правила можно создать подправило. Оно будет использовать те же параметры, что и основное правило, но при этом автоматически нигде не проверяется. Программист должен самостоятельно внедрить соответствующую проверку в коде. После этого администратор, зная имя подправила, сможет задать для него нужные ограничения и проверки.

Для администрируемого объекта можно создать подправило, выполнив следующие шаги:

1. Открыть карточку администрируемого объекта.
2. Перейти в закладку Дискретные ограничения доступа.
3. Выбрать существующие правило или создать его.
4. Под выбранным или созданным правилом добавить подправило.
5. Назвать подправило именем, которое используется в коде для проверки доступа.
6. Задать необходимые скрипты для подправила.

После выполнения этих шагов подправило будет создано и применено в соответствии с заданными параметрами.

Пример проверки подправила в выборке:

```
override protected def onRefresh: Recs = {
  lazy val sSubRule = "SomeSubRuleName"

  val ropav = SomeApi().refreshByParent(getIdMaster)

  if (needApplyDiscreteAccess) {
    lazy val bHasGroup = Btk_AcLib().hasBtkGroup(acObject, acObject)
    ropav.filter { rv =>
      val rows = AcRows(Seq(rv))

      val viewPriv = if (bHasGroup) Btk_AdminPkg.viewGroupPriv.get else Btk_AdminPkg.
↪viewObjPriv.get
      val editPriv = if (bHasGroup) Btk_AdminPkg.editGroupPriv.get else Btk_AdminPkg.
↪editObjPriv.get

      Btk_AdminPkg().hasObjPrivByRows(acObject.get, viewPriv, rows, spSubRule =
↪sSubRule) ||
      Btk_AdminPkg().hasObjPrivByRows(acObject.get, editPriv, rows, spSubRule
↪= sSubRule)
    }.map { rop =>
      Row(rop.idJ, SomeApi().getHeadLine(rop.idJ))
    }
  } else {
    ropav.map { rop =>
      Row(rop.idJ, SomeApi().getHeadLine(rop.idJ))
    }
  }
}
```


Проверка полномочий в печатных формах

Для реализации проверки доступа к объектам, которые получаются внутри запроса при построении печатных форм, необходимо использовать макросы в запросах.

При построении печатной формы эти макросы будут заменены на скрипты для фильтрации объектных привилегий. Фильтрация выполняется на основе:

- Настроек административного объекта
- Параметров, настроенных в ролях пользователя

В результате будут использованы только те объекты, на которые у пользователя есть права.

Важно: Если в печатной форме используется файл с запросом, где не прописаны необходимые макросы, но при этом:

1. На закладке настроен доступ к администрируемым объектам
2. В запросе есть обращение к соответствующей таблице

Тогда система не позволит пользователю без прав распечатать документ и выведет сообщение об ошибке с просьбой исправить запрос, добавив корректные макросы.

Настройка дискретного доступа в отчетах

Для реализации проверки дискретного доступа в запросах отчетов необходимо выполнить следующие настройки:

1. В отчете **Rpt_Report** на закладке «Настройка контроля доступа к данным печатной формы» выбрать администрируемый объект для ограничения доступа
2. Указать объектную привилегию (если не выбрана, будет проверяться привилегия «Доступ на чтение из печатных форм» - **viewReport#**)
3. В поле «Список алиасов» указать алиасы таблиц через «;» (если поле пустое, по умолчанию используется «t»)

Для удобства система автоматически сгенерирует макросы для вставки в запрос печатной формы.

Использование макросов дискретного доступа в Jasper-отчетах

В Jasper-отчетах макрос передается как параметр. Имя параметра формируется по шаблону: **DM_(<системное_имя_административного_объекта>)_(<алиас_таблицы>)**

Чтобы использовать параметр как макрос для подстановки скрипта дискретного доступа, в запросе его нужно указать со знаком «!» (восклицательный знак).

Пример объявления параметра и использования в запросе для ограничения доступа к таблице **Btk_User**:

```
<parameter name="DM_(BTK_USER)_(T)" class="java.lang.String">
  <defaultValueExpression><![CDATA["1=2"]]></defaultValueExpression>
</parameter>
<queryString>
  <![CDATA[select sHeadLine_dz
              from btk_user t
              where $P!{DM_(BTK_USER)_(T)}]]>
</queryString>
```

Рекомендация: Для тестирования отчета можно указать значение по умолчанию как «1=1», но перед загрузкой формы в приложение его следует убрать, чтобы избежать случайного предоставления полного доступа при ошибках в настройках.

Использование макросов дискретного доступа в отчетах типа «Текстовый», «Excel», «Word»

В запросе необходимо указать макрос, который будет заменен скриптом дискретного доступа. Макрос формируется по шаблону:

`&DM_(<системное_имя_административного_объекта>)_(<алиас_таблицы>)`

Пример запроса для печатной формы типа «Word» с ограничением доступа к таблице Btk_User:

```
[#&Order= select sHeadLine_dz, sMnemonic_dz from BTK_USER t
where &DM_(Btk_User) #]
```

Примеры скриптов

Примитивное правило по атрибуту класса

Правило значимое, тип данных строка. Фильтрация по like атрибута sCode

Скрипт для фильтрации объектных привилегий:

```
select 1
  from RplTst_ClassTree tt
  join (
    select cast(json_array_elements_text(cast((&params) as json)) as varchar) as sCode
    ) as codes
    on tt.sCode like codes.sCode
 where tt.id = (&id)
```

Скрипт проверки строк по объектному кешу:

```
for (p: params) {
  if (row.sCode != null && p != null) {
    if (like(row.sCode, p)) {
      return true;
    }
  }
}
```

Примитивное правило по атрибуту коллекции

Правило ссылочное на Btk_Group. Фильтрация по прямому вхождению объекта в группу (фильтрация коллекции Btk_ObjectGroup).

Скрипт для фильтрации объектных привилегий:

```
select 1
  from RplTst_TestGroup tt
  join Btk_ObjectGroup og
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
on tt.gid = og.gidSrcObject
join (
  select cast(json_array_elements_text(cast((&params) as json)) as int8) as id
) as params
on og.idGroup = params.id
where tt.id = (&id)
```

Скрипт проверки строк по объектному кешу:

```
for (p: params) {
  var rops = toJRops(Btk_ObjectGroupApi.byParent(row.data())).asList();
  if (p != null) {
    for (r: rops) {
      if (r.idGroup == p) {
        return true;
      }
    }
  }
}
```

Примитивное правило по адм. объекту, созданному не по классу (произвольная выборка)

Правило значимое, тип данных строка. Фильтрация регистронезависимая по like поля sClass

Скрипт для фильтрации объектных привилегий:

```
select 1
from json_array_elements_text(cast((&params) as json)) as p
where upper((&sClass)) like upper(p)
```

Скрипт проверки строк по объектному кешу:

```
for (p: params) {
  if (row.sClass != null && p != null) {
    if (like(row.sClass.toUpperCase(), p.toUpperCase())) {
      return true;
    }
  }
}
```

Составное правило по 3 атрибутам класса

Параметры правила:

1. dDate - Значимый, тип данных дата.
2. nNumber - Значимый, тип данных число
3. sCaption - Значимый, тип данных строка

Фильтрация отбирает записи у которых дата меньше указанной, число равно указанному, и Наименование по like регистронезависимо совпадает с указанной

Скрипт для фильтрации объектных привилегий:

```
select 1
  from RplTst_AllDbTypes tt
 where tt.id = (&id)
    and exists (
      select 1
        from (
          select v -> 'dDate' as dDate
                ,v -> 'nNumber' as nNumber
                ,v -> 'sCaption' as sCaption
          from json_array_elements(cast((&params) as json)) as v
        ) p
      where exists (
        select 1
          from json_array_elements_text(p.dDate) pp
          where tt.dDate <= to_timestamp(pp, 'DD.MM.YYYY HH24:MI:SS')
      )
    and exists (
      select 1
        from json_array_elements_text(p.nNumber) pp
        where tt.nNumber = cast(pp as numeric)
    )
    and exists (
      select 1
        from json_array_elements_text(p.sCaption) pp
        where upper(tt.sCaption) like upper(pp)
    )
  )
```

Скрипт проверки строк по объектному кешу:

```
for (p: params) {
  var jsonObj = toJsonObject(p);
  var res = true

  //проверка даты
  if (res) {
    res = false
    var aValues = jsonObj.childJSONArray("dDate");
    var i = 0
    while(i < aValues.size() && !res) {
      var value = aValues.getDate(i);
      if (row.dDate <= value) {
        res = true;
      }
      i = i + 1;
    }
  }

  //проверка числа
  if (res) {
    res = false
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
var aValues = jsonObj.childJSONArray("nNumber");
var i = 0
while(i < aValues.size() && !res) {
    var value = aValues.getNumber(i);
    if (row.nNumber == value) {
        res = true;
    }
    i = i + 1;
}

//проверка наименования
if (res) {
    res = false
    var aValues = jsonObj.childJSONArray("sCaption");
    var i = 0
    while(i < aValues.size() && !res) {
        var value = aValues.getString(i);
        if (row.sCaption != null && value != null) {
            if (like(row.sCaption.toUpperCase(), value.toUpperCase())) {
                res = true;
            }
        }
        i = i + 1;
    }
}

if (res) {
    return true;
}
}
```

Составное правило по атрибуту класса и коллекции

Параметры правила:

1. sCaption - Значимый, тип данных строка
2. idGroup - ссылочный на Btk_Group

Фильтрация отбирает записи, у которых Наименование по like регистронезависимо совпадает с указанным, и есть прямое вхождение в группу (фильтрация коллекции Btk_ObjectGroup).

Скрипт для фильтрации объектных привилегий:

```
select 1
  from RplTst_TestGroup tt
 where tt.id = (&id)
    and exists (
      select 1
      from (
        select v -> 'sCaption' as sCaption
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        ,v -> 'idGroup' as idGroup
        from json_array_elements(cast((&params) as json)) as v
    ) p
    where exists (
        select 1
        from json_array_elements_text(p.sCaption) pp
        where upper(tt.sCaption) like upper(pp)
    )
    and exists (
        select 1
        from Btk_ObjectGroup og
        join json_array_elements_text(p.idGroup) pp
        on og.idGroup = cast(pp as int8)
        where og.gidSrcObject = tt.gid
    )
)
```

Скрипт проверки строк по объектному кешу:

```
for (p: params) {
    var jObj = toJObject(p);
    var res = true

    //проверка наименования
    if (res) {
        res = false
        var aValues = jObj.childJArray("sCaption");
        var i = 0
        while(i < aValues.size() && !res) {
            var value = aValues.getString(i);
            if (row.sCaption != null && value != null) {
                if (like(row.sCaption.toUpperCase(), value.toUpperCase())) {
                    res = true;
                }
            }
            i = i + 1;
        }
    }

    //проверка группы
    if (res) {
        res = false
        var rops = toJRops(Btk_ObjectGroupApi.byParent(row.data())).asList();
        var aValues = jObj.childJArray("idGroup");
        var i = 0
        while(i < aValues.size() && !res) {
            var value = aValues.getLong(i);
            for (r: rops) {
                if (r.idGroup == value) {
                    res = true;
                }
            }
        }
    }
}
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        i = i + 1;
    }
}

if (res) {
    return true;
}
}
```

Составное правило по 2 атрибутам адм. объекта, созданного не по классу (произвольная выборка)

Параметры правила:

1. sClass - Значимый, тип данных строка
2. idRefClass - ссылочный на Btk_Class

Фильтрация отбирает записи, у которых поле sClass по like регистронезависимо совпадает с указанным, и поле idRefClass совпадает с указанным

Скрипт для фильтрации объектных привилегий:

```
select 1
  from (
    select v -> 'sClass' as sClass
           ,v -> 'idRefClass' as idRefClass
    from json_array_elements(cast((&params) as json)) as v
  ) p
 where exists (
    select 1
    from json_array_elements_text(p.sClass) pp
    where upper((&sClass)) like upper(pp)
  )
 and exists (
    select 1
    from json_array_elements_text(p.idRefClass) pp
    where (&idRefClass) = cast(pp as int8)
  )
)
```

Скрипт проверки строк по объектному кешу:

```
for (p: params) {
    var jsonObj = toJsonObject(p);
    var res = true

    //проверка sClass
    if (res) {
        res = false
        var aValues = jsonObj.childJSONArray("sClass");
        var i = 0
        while(i < aValues.size() && !res) {
            var value = aValues.getString(i);
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
        if (row.sClass != null && value != null) {
            if (like(row.sClass.toUpperCase(), value.toUpperCase())) {
                res = true;
            }
        }
        i = i + 1;
    }
}

//проверка idRefClass
if (res) {
    res = false
    var aValues = jsonObj.childJSONArray("idRefClass");
    var i = 0
    while(i < aValues.size() && !res) {
        var value = aValues.getLong(i);
        if (row.idRefClass == value) {
            res = true;
        }
        i = i + 1;
    }
}

if (res) {
    return true;
}
}
```

14.5 Связь выборок с администрируемыми объектами

основные понятия

Элемент администрирования

Элемент администрирования (класс `Btk_AcItem`) - логический объект, который соответствует одной выборке.

На каждую выборку, обладающую своей `avm.xml`, создается отдельный элемент администрирования, обладает перечнем элементарных привилегий

Элементарная привилегия

Элементарная привилегия (класс `Btk_AcItemPrivilege`, коллекция к `Btk_AcItem`) - логический объект, который соответствует одному из элементов выборки: атрибуту или операции.

Элементарные привилегии обладают **типом привилегии**. Например, атрибуты относятся к типу **Чтение**.

Тип элементарной привилегии

Определяет группу **Элементарной привилегии**. Позволяет управлять доступом к группам привилегий, а не по отдельности к каждой элементарной привилегии

Объект администрирования

Объект администрирования (класс `Btk_AcObject`) - логический объект, который соответствует **Бизнес объекту**, и обладает перечнем **элементов администрирования**, входящих в этот объект.

Бизнес объект

Бизнес объект (класс `Btk_BoEntity`) - логическое объединение нескольких **классов** в единую сущность.

Бизнес объект создается на **корневые классы**, и включает все его коллекции.

Базовая выборка

Выборка, которая соответствует классу. Например, выборка `Btk_ClassAvi` является базовой выборкой класса `Btk_Class`

Произвольная выборка

Выборка, которая не принадлежит какому-либо классу. В том числе и выборки-наследники от базовой выборки

Корневой класс

Класс, на который формируется отдельный **бизнес объект**. Является логическим отдельным объектом, который обладает своими **коллекциями**.

Это классы с типом **Справочник**, **Документ**, **Настройка**, **Журнал**.

Принцип построения адм. объектов

Формирование на основе структуры odm.xml

1. На каждый корневой класс создается свой администрируемый объект, имя которого равно имени класса.
2. В администрируемый объект включаются элементы администрирования, которые соответствуют выборкам всех классов входящий в бизнес объект (сам класс и все его коллекции).

Пример структуры бизнес объекта:

```
Документ Some_Document
+ Коллекция Some_DocumentPosition
+ Коллекция Some_DocumentPositionDet
+ V-коллекция Коллекция Some_DocumentLink
```

Пример сформированного администрируемого объекта:

```
Адм. объект Some_Document
+ Элемент Some_DocumentAvi
+ Элемент Some_Document\Some_DocumentPositionAvi
+ Элемент Some_Document\Some_DocumentPosition\Some_DocumentPositionDetAvi
+ Элемент Some_Document\Some_DocumentLinkAvi
```

Элементы администрирования в составе объекта администрирования имеют имя, которое соответствует их пути внутри бизнес объекта. Зачем используются такие пути будет описано в главе **Определение связи выборки и адм. объекта**

Подключение произвольной выборки в структуру адм. объекта

По мимо элементов администрирования, соответствующих базовым выборкам классов бизнес объектов, есть возможность добавить произвольную выборку в структуру администрируемого объекта. Для этого существует 2 способа:

1. Подключение в исходном коде
В avm.xml базовой выборки корневого класса добавить тег `acObject`, в котором указать дополнительные элементы администрирования.

Пример разметки:

```
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.global-system.ru/xsd/global3-view-1.0"
      name="Some_Document">

  <acObject>
    <acItems>
      <acItem name="Some_Document\Some_CustomAvi" caption="Произвольная выборка"/
    </acItems>
  </acObject>
</view>
```

В этом примере объявлен один дополнительный элемент администрирования в составе объекта «Some_Document», с именем «Some_Document\Some_CustomAvi» и наименованием «Произвольная выборка». Наименование используется для визуального отображения этого элемента в приложении администратор.

2. Подключение через приложение администратор

- Открыть карточку нужного объекта администрирования
- добавить новый элемент на закладке Элементы администрируемого объекта

Определение связи выборки и адм. объекта

Каждая выборка на открытии в событии `onLoadAdminMeta` через методы выборки `acObject` и `acObjectItem` определяет свою принадлежность к элементу администрирования и объекту администрирования. Все настройки прав доступа к выборке будут искаться по паре значений, которые вернули эти два метода.

Значение, которые вернули эти два метода для выборки, можно увидеть через окно отладки.

Т.е. чтобы к выборке корректно применялись настройки доступа, в приложении администратор должен быть объект администрирования, имя которого равно результату метода `acObject`, и в составе этого объекта должен быть элемент администрирования, имя которого равно результату метода `acObjectItem`.

Метод `acObject`

Возвращает системное имя объекта администрирования к которому относится выборка. Например, «Some_Document»

Основной принцип:

1. Если выборка принадлежит классу, и класс **корневой**, то имя этого класса = имя адм. объекта
2. Иначе вызывается метод `acObject` мастер-выборки.

Метод `acObjectItem`

Возвращает системное имя элемента администрирования в составе объекта. например, «Some_Document\Some_DocumentPositionAvi»

Основной принцип:

1. Вызывается метод `acObjectItem` мастер-выборки.
2. К полученному значению добавляется текущее имя выборки.

Базовые выборки классов

Для базовых выборок классов в большинстве случаев корректно отрабатывает стандартная логика определения элемента администрирования и объекта администрирования.

Исключением может служить базовая выборка *v-коллекций*, если она открывается вне формы, на которой расположена базовая выборка мастер-объекта. Например, если открыть список объектов *v-коллекции*, как главную выборку формы.

Произвольные выборки

Для произвольных выборок определить принадлежность к объекту администрирования автоматически не так просто, как для базовых выборок, которые соответствуют структуре бизнес объектов.

В большинстве случаев разработчику требуется самостоятельно переопределить работу методов `acObject` и `acObjectItem`.

Выборка как самостоятельный адм. объект

Если произвольную выборку предполагается использовать как самостоятельный объект администрирования, то требуется:

1. в `avm.xml` объявить тег `acObject`
2. в `Avi` переопределить методы `acObject` и `acObjectItem` следующим образом:

```
override def acObject: NString = {  
    baseAvi.simpleName  
}  
  
override def acObjectItem: NString = {  
    baseAvi.simpleName  
}
```

Выборка как часть другого адм. объекта

Если произвольную выборку предполагается использовать как часть существующего объекта администрирования, то требуется:

1. В `avm.xml` базовой выборки корневого класса этого бизнес объекта объявить тег `acObject` и внутри него добавить новый элемент администрирования

```
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://www.global-system.ru/xsd/global3-view-1.0"  
    name="Some_Document">  
  
    <acObject>  
        <acItems>  
            <acItem name="Some_Document\Some_CustomAvi" caption="Произвольная выборка"/  
↪>  
        </acItems>
```

(продолжается на следующей странице)

```
</acObject>
</view>
```

2. в `Avi` произвольной выборки переопределить методы `acObject` и `acObjectItem` следующим образом:

```
override def acObject: NString = {
  //имя объекта
  "Some_Document"
}

override def acObjectItem: NString = {
  //путь, который мы указали в тегах acItem
  "Some_Document\\Some_CustomAvi"
}
```

3. в `Api` произвольной выборки зарегистрировать с помощью метода `Btk_AcObjectItemApi().registerByAcObjectItemName`

```
def regAcObjectItem(): Unit = {
  val bvDefinedAcObjectItem = TxIndex(Btk_AcObjectItemAta.Type)(_sCode).byKey("Some_
  ↳Document\\Some_Document\\Some_CustomAvi#Default").isEmpty
  if (bvDefinedAcObjectItem) Btk_AcObjectItemApi().registerByAcObjectItemName("Some_
  ↳Document\\Some_Document\\Some_CustomAvi#Default")
}
```

Часто задаваемые вопросы

Проблема: при заходе не под суперпользователем выборка не отображается/не применяются права

Возможная причина: некорректное определение `acObjectItem` или `acObject`

1. Войдите под суперпользователем или пользователем с правами:
 - Доступ к отображению
 - Права на открытие окна отладки
2. Откройте окно отладки (`Ctrl+Alt+Shift+W`) и проверьте:
 - `acObjectItem` должен быть цепочкой «мастер-деталь»
 - Начало `acObjectItem` должно совпадать с `acObject`

Примеры корректного определения:

Dct_GdsForDocAvi		List	ТМЦ
Btk_GroupAvi		FilterTreeNotCur	Группы ТМЦ
Системное имя	ru.bitec.app.dct.Dct_GdsForDocAvi.List		
Ави-класс	ru.bitec.app.dct.Dct_GdsForDocAvi		
Класс фрейма			
Проектное переопределение	☐		
AcObject	Dct_GdsForDocAvi		
AcObjectItem	Dct_GdsForDocAvi		

Для выборки как администрируемого объекта

Btk_SelListSetSourceAvi		List_idSelListSetting	Источники настройки атрибутов ...
Системное имя	ru.bitec.app.btk.sel.service.list.setting.Btk_SelListSetSourceAvi.List_idSelListSetting		
Ави-класс	ru.bitec.app.btk.sel.service.list.setting.Btk_SelListSetSourceAvi		
Класс фрейма			
Проектное переопределение	☐		
AcObject	Btk_SelListSetting		
AcObjectItem	Btk_SelListSetting\Btk_SelListSetSourceAvi		

Для детализации

Prs_PurchReq		ListAppr	Согласование заявок на закупку	☒	☐	☐
Системное имя	gtk-ru.bitec.app.prs.purchreq.Prs_PurchReq#ListAppr					
Рер-класс	ru.bitec.app.zsngprs.purchreq.Prs_PurchReqOverrideAvi.ListAppr					
Класс фрейма						
Проектное переопределение	☒					
AcObject	Prs_PurchReq					
AcObjectItem	Prs_PurchReqOverrideAvi					

Для переопределенной выборки

Prs_PurchReqDet		List_idPurchReqVe	Позиции заявки на закупку	☒	☐	☐
Системное имя	gtk-ru.bitec.app.prs.purchreq.Prs_PurchReqDet#List_idPurchReqVer					
Рер-класс	ru.bitec.app.zsngprs.purchreq.Prs_PurchReqDetOverrideAvi.List_idPurchReqVer					
Класс фрейма						
Проектное переопределение	☒					
AcObject	Prs_PurchReq					
AcObjectItem	Prs_PurchReq\Prs_PurchReqVer\Prs_PurchReqDetOverrideAvi					

Для переопределенной детализации

Примеры некорректных определений:

1. Переопределение thisApi в выборке без класса.

	Tree	Дерево проектов и работ
Pro_ProjectActionTreeApi		
Pm_TaxOperDocApi	List	Документы учета налогов

Системное имя	ru.bitec.app.pro.Pro_ProjectActionTreeApi.Tree
Api-класс	ru.bitec.app.pro.Pro_ProjectActionTreeApi
Класс фрейма	
Проектное переопределение	☐
AcObject	Pro_ProjectActionTreeApi
AcObjectItem	Pro_Project\Pro_ProjectActionTreeApi

Решение:

- Убрать ненужное переопределение
- Или переопределить acObject соответствующим образом

2. Коллекция переменной ссылочности без мастера

Системное имя	ru.bitec.app.srm.Srm_AttachItemApi.List
Api-класс	ru.bitec.app.btk.Btk_AttachItemApi
Класс фрейма	ru.bitec.engine.model.ui.frame.EGridFrameImpl
Проектное переопределение	☒
AcObject	Srm_AttachItemOverrideApi
AcObjectItem	null\Srm_AttachItemOverrideApi

Решение:

- Не открывать такие списки как самостоятельные отображения
- Или переопределить acObject как один из возможных БО

3. Ошибка в переопределении acObject

Системное имя	ru.bitec.app.pro.Pro_ProjectActionTreeApi.Tree
Api-класс	ru.bitec.app.pro.Pro_ProjectActionTreeApi
Класс фрейма	
Проектное переопределение	☐
AcObject	123Pro_ProjectActionTreeApi
AcObjectItem	Pro_ProjectActionTreeApi

Решение:

- Исправить или удалить некорректное переопределение

Если объект определяется правильно, но проблема осталась:

1. Откройте приложение «Администратор»
 2. Обновите администрируемый объект
-

Проблема: новый атрибут/колонка не отображается у обычного пользователя

Проверка:

1. В приложении «Администратор» найдите нужный элемент администрирования
2. Убедитесь в наличии нужной привилегии

Если привилегии нет:

- Обновите администрируемый объект
 - Проверьте avm:
 - атрибут определен в dvm, но отсутствует в avm: добавьте ссылку на шаблон dvm.
 - Для динамических строителей: добавьте атрибут в avm
-

Проблема: SQL-ошибка из-за дискретного доступа

Скрипт фильтрации может вызывать ошибки, если:

- Обращается к несуществующим в отображении колонкам
- Используется параметр (например &id), отсутствующий в запросе

Рекомендации:

1. Используйте только гарантированно существующие колонки (лучше id)
2. Для специальных отображений:
 - Если администрирование не нужно: переопределите `isAdministered: Boolean = false`
 - Если нужно разграничение прав: вынесите в отдельную выборку

14.6 Телеметрия

Основные понятия

Отправка телеметрической информации во внешние аналитические системы по стандарту [OpenTelemetry](#).

Телеметрия - удобный процесс сбора информации о состоянии приложения.

Смотрите также:

- Документация [OpenTelemetry](#)
- Сигналы

Сигналы

Трассировка

Детальный отчёт о действиях происходящих на сервере при обработке входящих запросов.

Метрики

Периодически измеряемый количественный показатель.

Пример:

- количество http запросов к серверу по нарастанию
- процент использование процессора в минуту

Счетчик

Позволяет получать информацию о количестве случившихся событий.

Спидометр

Позволяет получать информацию о текущей загрузке ресурса.

Административный объект

Сигналы собирающиеся в разрезе административного объекта могут быть включены или выключены для конкретного административного объекта

Объектная метрика

Используется в случаи когда необходимо измерить специфичный показатель через сигнал метрики. В таком случаи прикладной программист регистрирует объектную метрику в метаданных модуля.

Объектная трассировка

Используется в случаи когда необходимо измерить специфичный показатель через сигнал трассировки. В таком случаи прикладной программист регистрирует объектную трассировку в метаданных модуля.

Серверная телеметрия

Отслеживает общее время взаимодействия между тонким клиентом и сервером приложения.

Настройка телеметрии

Для настройки телеметрии откройте форму Администратор: Настройки->Телеметрия

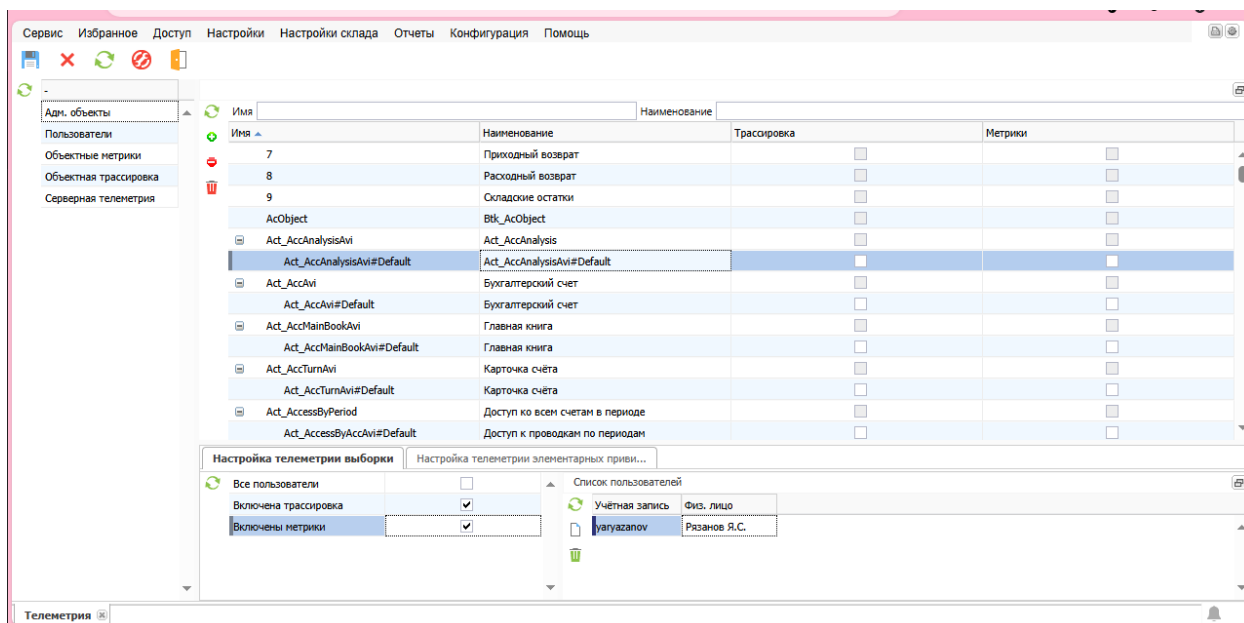
Разделы

Адм. объекты

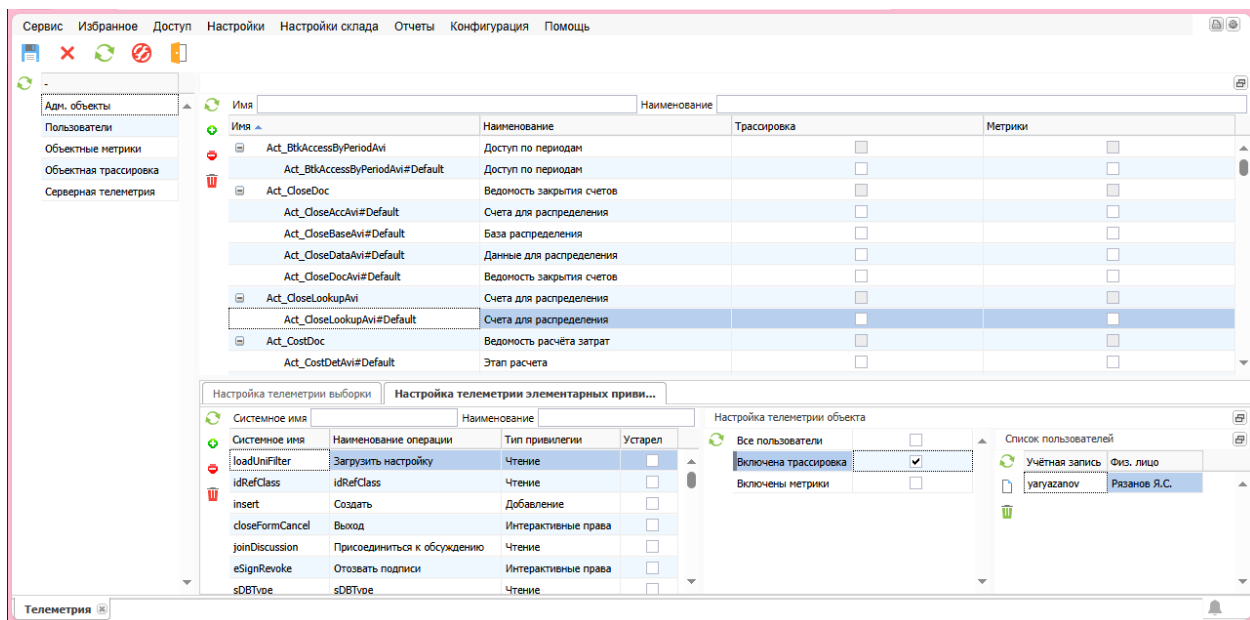
Позволяет включать сбор *сигналов* по административным объектам в разрезе всех или конкретных пользователей.

Примеры

Включение метрик и трассировки на административном объекте `Act_AccAnalysisAvi` для пользователя `yaryazanov`.



Включение трассировки элементарной привилегии `loadUniFilter` на административном объекте `Act_CloseLookupAvi` для пользователя `yaryazanov`.

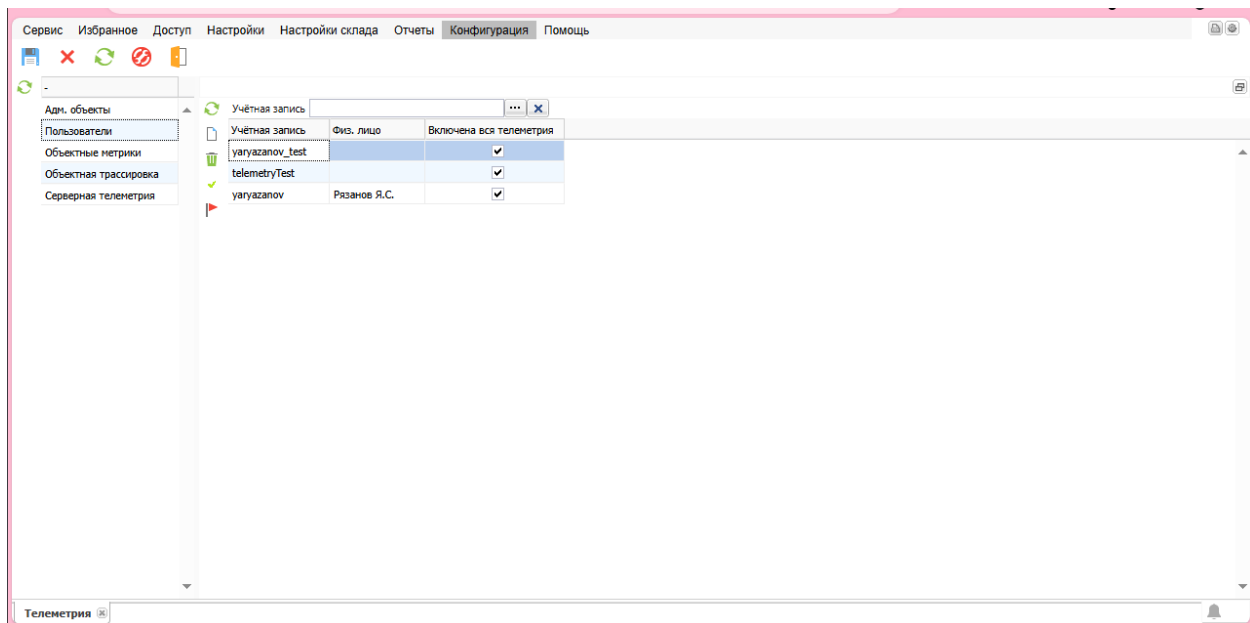


Пользователи

Позволяет включать сбор *сигналов* по всем адм. объектам на конкретном пользователе

Примеры

Включение сбора *сигналов* по всем адм. объектам для пользователей *yaryazanov*, *telemetryTest* и *yaryazanov_test*.



Объектные метрики

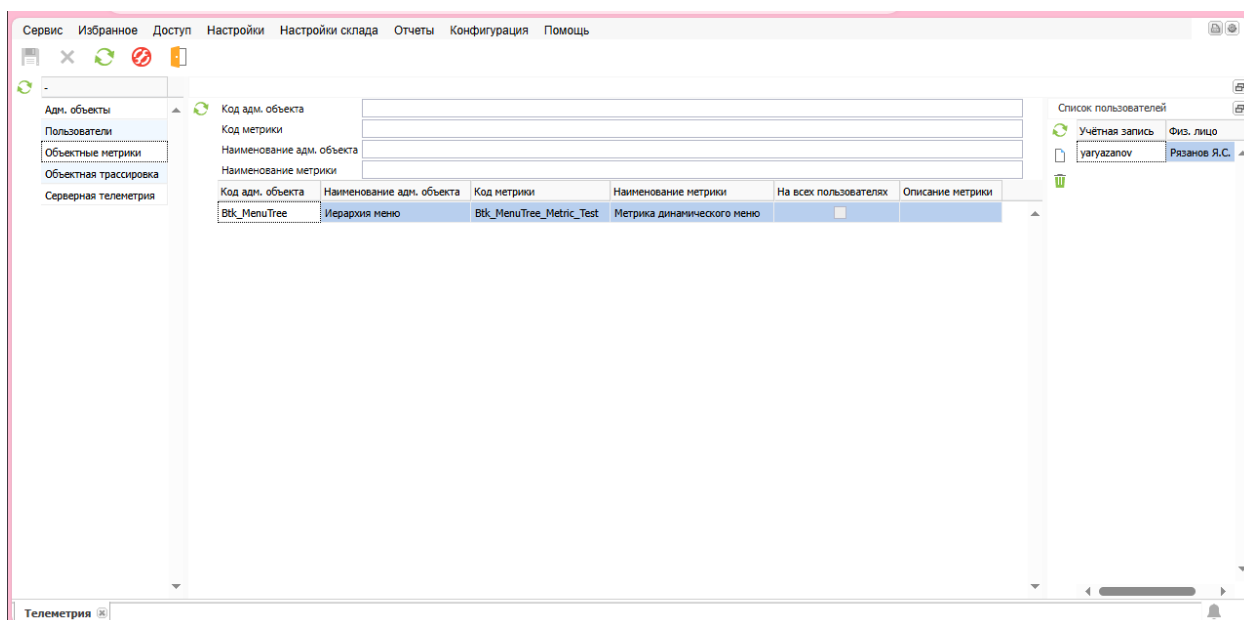
Данный раздел позволяет увидеть общий список объектных метрик, а так же настроить их сбор для всех или конкретных пользователей.

Внимание

Регистацией объектных метрик занимается разработчик в прикладном коде.

Примеры

Включение сбора объектной метрики `Btk_MenuTree` для пользователя `yaryazanov`.



Объектная трассировка

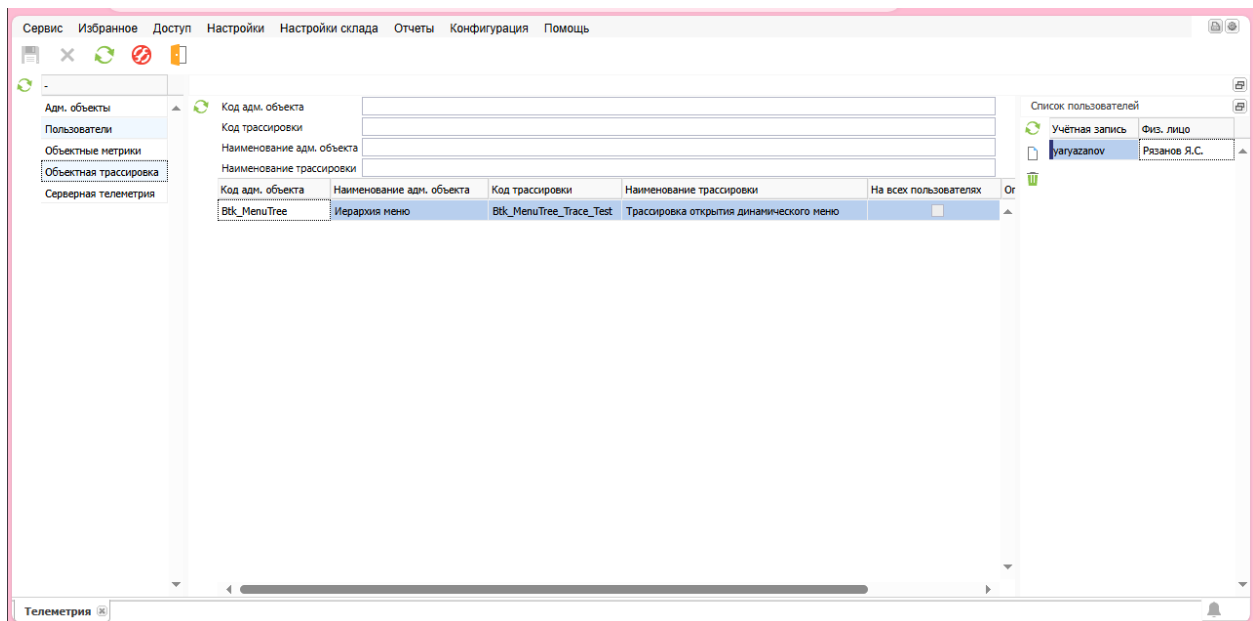
Данный раздел позволяет увидеть общий список объектных трассировок, а так же настроить их сбор для всех или конкретных пользователей.

Внимание

Регистацией объектных трассировок занимается разработчик в прикладном коде.

Примеры

Включение сбора объектной трассировки `Btk_MenuTree` для пользователя `yaryazanov`.



Серверная телеметрия

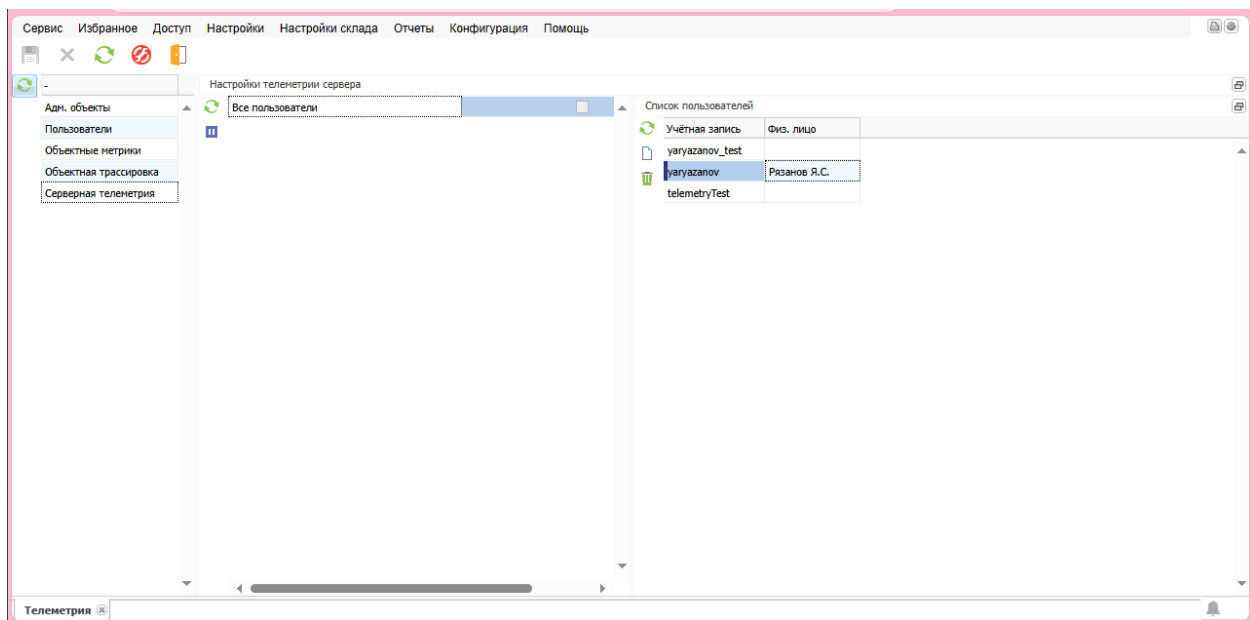
Данный раздел позволяет настроить серверную телеметрию всех или конкретных пользователей.

Внимание

В данном разделе можно отключить всю телеметрию полностью.

Примеры

Включение сбора *сигналов* серверной телеметрии для пользователей `yaryazanov`, `telemetryTest` и `yaryazanov_test`.



Объектные метрики и трассировка

У программистов есть возможность создавать свои метрики и трассировку по администрируемым объектам.

Что бы создать метрику, необходимо в классе «Метрики объектов» зарегистрировать запись:

```
val ropAcMetric = Btk_AcObjectMetricApi().register(
    idParent = Btk_AcObjectApi().findByMnemonicCode("Btk_MenuTree"),
    sCode = "Btk_MenuTree_Metric_Test",
    sCaption = "Метрика динамического меню"
)
```

Внимание

Код метрического прибора или трассировки должен быть не более 255 символов, должен начинаться с буквы и может содержать только следующие специальные символы: „_“, „“, „“, „“.

После регистрации администратор в настройках телеметрии во вкладке «Объектные метрики» может включить её для всех пользователей или определённого набора.

Для использования метрики в прикладном коде, необходимо получить метрику и увеличить счётчик.

```
val longCounterOpt = Btk_TelemetryPkg().getAcObjectLongCounter(
    idUser = Btk_UserApi().getCurrentUserID(),
    idAcObjectMetric = ropAcMetric.idj
)

longCounterOpt.foreach(_.add(1))
```

Внимание

Методы получения метрических приборов или трассировки возвращают монаду Option. В контейнере значение присутствует, только если на текущем пользователе включены настройки.

Для использования трассировки аналогичные методы, только регистрация происходит в классе Btk_AcObjectTrace.

Справка

Метрика

Имена

- «btk_item_telemetry_user_operation_exec_time» - Гистограмма времени выполнения пользовательской операции.
- «btk_item_telemetry_user_operation_calls_successful_total» - Счётчик количества успешных вызовов пользовательской операции.
- «btk_item_telemetry_user_operation_calls_failed» - Счётчик количества вызовов пользовательской операции, которые завершились ошибкой.
- «btk_item_telemetry_user_operation_calls_total» - Счётчик количества вызовов пользовательской операции.

Трассировка

Имена

- «btk_item_telemetry_user_operation_trace» - Трассировка пользовательских операций.

Атрибуты

Каждая метрика содержит в себе перечень атрибутов по которым можно фильтровать значения в мониторе.

- «solution» - Алиас БД.
- «solution.image» - Алиас SBT.
- «userName» - Логин пользователя.
- «session.secret» - «Секретный» ключ рабочего сеанса.
- «session.sid» - Идентификатор сессии.
- «work.session.sid» - Идентификатор рабочей сессии.
- «cluster.node» - Наименование кластерной ноды.
- «selection.name» - Системное наименование выборки.
- «selection.caption» - Наименование выборки.
- «oper.name» - Системное наименование операции.

- «oper.caption» - Наименование операции.
- «representation.name» - Системное наименование отображения.
- «form.name» - Системное наименование формы.
- «form.caption» - Наименование формы.
- «form.representation.name» - Системное наименование отображения формы.

Серверные метрики

Имена

- «rpc_client_duration» - Длительность выполнения RPC с точки зрения клиента.
- «rpc_server_duration» - Длительность обработки RPC сервером.
- «rpc_transport_duration» - Длительность передачи по сети (Разница между «rpc_duration_client_ms» и «rpc_duration_server_ms»).

Объектные метрики

Имена

- «file_storage_availability» - Наличие доступа к файловому хранилищу (1 - есть доступ, 0 - нет доступа).
- «file_storage_write_speed» - Скорость записи файлового хранилища.
- «config_manager_export_count» - Количество экспортов конфигураций.
- «config_manager_import_count» - Количество импортов конфигураций.
- «config_manager_errors_count» - Количество ошибок при импорте конфигурации.
- «config_manager_fatal_error» - Наличие фатальной ошибки при импорте конфигурации (1 - есть ошибка, 0 - нет ошибки).
- «license_file_expiry_days» - Количество дней до истечения лицензии файла.
- «license_file_expired» - Истёк ли срок лицензии файла (1 - срок истёк, 0 - срок не истёк).

14.7 Политика паролей

Реализация в системе

За включение парольной политики в системе отвечает флаг «Настройка политики паролей на типе объекта» (`bPasswordSettingsInOT`) в группе «Безопасность и аутентификация» в классе `Btk_Setting` (по умолчанию — `false`).

- Если флаг установлен (`true`), то парольная политика настраивается на типе объекта пользователя (`Btk_UserPasswordSetting`)
- Если флаг не установлен или равен `false`, используются стандартные параметры паролей из класса `Btk_User`. Они настраиваются отдельно на каждом пользователе.

Параметры из класса `Btk_User`

Атрибуты, определяющие требования к паролю:

- `bPassSpecSymb` — необходимость наличия хотя бы одного специального символа. По умолчанию — 0 (не требуется);
допустимые символы: `!@#$$%^&*()-+\\?/.,#;:`
- `bPassLetterDigit` — обязательное наличие как букв, так и цифр. По умолчанию — 0.
- `bPassUpperLowerLetter` — необходимость использования как строчных, так и прописных букв. По умолчанию — 0.
- `nPassMinSymbCnt` — минимально допустимое количество символов в пароле. Значение по умолчанию отсутствует (ограничение не применяется).

Настройки политики на типе объекта (`Btk_UserPasswordSetting`)

Атрибуты, определяющие политику управления паролем:

- Все атрибуты из класса `Btk_User` (`bPassSpecSymb`, `bPassLetterDigit`, `bPassUpperLowerLetter`, `nPassMinSymbCnt`) — действуют аналогично вышеописанным.
- `nNonRepeatPassAmount` — количество последних паролей, которые нельзя повторно использовать. По умолчанию отсутствует (повтор разрешён).
- `nDayInactivityCntForUserBlocking` — число дней без активности, после которого пользователь будет заблокирован. Значение по умолчанию — 180.
- `nMinLifetime` — минимальное количество дней между сменами пароля. Запрещает слишком частую смену. По умолчанию отсутствует.
- `jTempBlockingForUnsuccessfulLogin` — параметры временной блокировки при неудачных попытках входа. JSON-объект с полями:
 - `nNumberOfAttempts` — допустимое число ошибочных попыток входа;
 - `nBlockingTimeMin` — время блокировки (в минутах) при превышении лимита.
- `nValidityPeriod` — срок действия постоянного пароля (в днях).
- `nValidityPeriodTemp` — срок действия временного пароля (в днях).
- `sDaysForNotifyChangePass` — количество дней до окончания срока действия пароля, за которое будет отправлено уведомление о необходимости его смены.

Настройка политики через типы объектов

При включении флага `bPasswordSettingsInOT` в классе `Btk_Setting` (установке галочки «Настройка политики паролей на типе объекта») политика применяется ко всей системе.

- После включения управление политикой осуществляется через типы объектов класса `Btk_User`.
- Каждому пользователю назначается определённый тип объекта, который определяет применяемые к нему правила парольной политики.
- По умолчанию доступны предопределённые типы объектов, однако допускается создание новых.

Настройка параметров на типе объекта

Параметры парольной политики настраиваются непосредственно на типе объекта:

1. Перейдите: Настройки системы → Сущности → Типы объектов.
2. В фильтре укажите класс `Btk_User`.
3. Выберите нужный тип объекта из списка.
4. В левой панели перейдите в раздел «Настройки пользователей».
5. Установите требуемые значения параметров парольной политики.

Каждый пользователь, имеющий данный тип объекта, будет подчиняться соответствующей политике при создании или изменении пароля.

Валидация политики

Проверка требований к сложности пароля

Происходит при смене пароля в методе `Btk_ChangePasswordAvi.Card.beforeCloseForm`. Используются следующие методы:

- `Btk_UserApi.validateUserPassword` — проверяет соблюдение минимального срока жизни пароля (`nMinLifetime`).
- `Btk_UserApi.validateTempPassword` — проверяет, не истёк ли срок действия текущего пароля (`nValidityPeriod` или `nValidityPeriodTemp`).
- `Btk_UserApi.checkPass` — производит детальную проверку структуры нового пароля по заданным параметрам (`bPassSpecSymb`, `bPassLetterDigit`, `bPassUpperLowerLetter`, `nPassMinSymbCnt`).

Проверка уникальности пароля

Метод `Btk_UserApi.setsPass` вызывает `Btk_UserPassHistoryApi.register`, который сравнивает новый пароль с предыдущими.

- Если `nNonRepeatPassAmount > 0`, установка пароля, совпадающего с одним из последних, запрещается.
- Например, при значении 2 система не позволит использовать пароль, совпадающий с любым из двух предыдущих.

Обработка устаревших паролей и уведомлений

Методы:

- `processUpdatingOutdatedPass` — используется в составе job-процесса. При истечении срока действия пароля:
 - устанавливает `Btk_User.bChangePasswd = true`;
 - автоматически генерирует новый временный пароль.
- `emailNotifyUserForNeedChangePassword` — отправляет уведомление пользователю, если до окончания срока действия пароля осталось количество дней, заданное в `sDaysForNotifyChangePass`.

Временная блокировка при неудачных попытках входа

Метод `validateBlockingUserResult` контролирует количество подряд неудачных попыток входа:

- При превышении лимита `jTempBlockingForUnsuccessfulLogin.nNumberOfAttempts` пользователь временно блокируется.
- Длительность блокировки задается параметром `jTempBlockingForUnsuccessfulLogin.nBlockingTimeMin`.

Рекомендации по парольной политике

Требования к паролю

- **Минимальная длина:**
 - 8 символов — для обычных пользователей;
 - 12 символов — для администраторов и привилегированных учётных записей.
- **Сложность:** пароль должен содержать символы из трёх и более категорий:
 - прописные латинские буквы (A–Z);
 - строчные латинские буквы (a–z);
 - цифры (0–9);
 - специальные символы (!, @, #, \$, %, ^, &, * и др.).
- Допускается использование кириллицы.
- Максимальная длина — до 64 символов.
- **Запрещено использовать:**
 - простые или предсказуемые пароли (например, `Password123`, `Qwerty!`);
 - персональные данные (имена, даты рождения, номера телефонов);
 - клавиатурные последовательности (например, `qwerty`, `1qaz2wsx`);
 - повторяющиеся символы (например, `aaaaaa`);
 - пароли из списков скомпрометированных.

Жизненный цикл пароля

- **Срок действия:**
 - не более 180 дней — для обычных пользователей;
 - не более 90 дней — для привилегированных учётных записей.
- Смена обязательна при:
 - подозрении на утечку;
 - передаче пароля третьим лицам;
 - компрометации АИБ.
- При первом входе — временный пароль должен быть изменён.

- **История паролей:**

- новый пароль не должен совпадать с двумя последними;
- должен отличаться от старого минимум на 4 символа;
- не должен содержать фрагменты старого пароля длиной 2 символа на тех же позициях;
- рекомендуется хранить не менее 24 последних паролей.

- Блокировка после 10 неудачных попыток на минимум 3 минуты.

Учётные записи

- При увольнении, переводе без доступа или прекращении отношений — учётная запись немедленно блокируется в течение следующего рабочего дня.
- Запрещено передавать пароль уволенного или перемещённого сотрудника.
- После блокировки — отключается во всех системах.
- Отключённая запись сохраняется в каталоге (например, Active Directory) для аудита:
 - срок хранения — от 6 до 36 месяцев;
 - по истечении срока — удаляется из всех систем.

Использование и хранение паролей

- Пароль должен быть известен только владельцу.
- Запрещено:
 - передавать пароль (включая руководителей);
 - работать под чужим паролем;
 - регистрировать других пользователей под своим аккаунтом.
- Запрещено хранить пароли:
 - на бумаге (кроме сейфа);
 - в записках, файлах, на стикерах, под клавиатурой.
- Допускается:
 - хранение на бумажном носителе в запирающемся шкафу или сейфе;
 - хранение в зашифрованном виде (bcrypt, scrypt, PBKDF2).
- Рекомендуется использование корпоративного менеджера паролей (Bitwarden, 1Password, Keeper).
- При вводе:
 - обеспечивать конфиденциальность (защита от «глазков»);
 - не произносить пароль вслух;
 - учитывать регистр.

Особые положения

- MFA обязательно для:
 - администраторов;
 - пользователей с расширенными правами;
 - доступа к системам с персональными данными.
 - Предпочтительно: мобильные аутентификаторы, аппаратные ключи.
 - SMS-коды как единственный фактор — запрещены.
- Групповые учётные записи — строго запрещены.
- Восстановление пароля:
 - только через генерацию нового;
 - после личной верификации (визит или видеозвонок с документом);
 - запрещено отправлять временные пароли по email без подтверждения.
- Аудит аутентификации:
 - все попытки входа (успешные и нет) должны логироваться;
 - логи хранятся не менее 180 дней.

14.8 Администрирование Jexl

Безопасный диалект Jexl обеспечивает контроль доступа при выполнении Jexl-скриптов. Проверка прав применяется только к скриптам, выполняемым напрямую пользователем через:

- **Специальные формы выполнения Jexl**
- **Дебаггер**
- **Rest-пакеты**
- **Консоль WebSocket**
- **Soap-сервисы**
- **Библиотеки скриптов**

Для включения безопасного диалекта, необходимо зайти в приложение **Настройки системы**, зайти в пункт меню **Настройки и сервисы** -> **Настройки модулей системы** -> **Общие настройки модулей**. В настройках модуля **btm** включите пункт **Выполнение jexl-скриптов через безопасный диалект**.

Так же в настройках модуля **btm** можно включить пункт **Заполнять журнал доступа к объектам системы через jexl**, что позволит отслеживать в аудите к каким объектам обращался Jexl-скрипт.

Все настройки производятся в карточке роли. Что бы в неё попасть, необходимо зайти в приложение **Администратор**, в пункт меню **Доступ** -> **Роли** и открыть карточку.

Доступ к Api / Pkg

Для ограничения доступа к Api, необходимо в карточке роли зайти в закладку Серверные полномочия -> Доступ к Api.

В отображении необходимо найти нужные вам Api и на детали Права на вызов методов из Jexl для нужных методов проставить галочки Имеется право.

Если нужен полный доступ к Api, то необходимо поставить галочку Доступны все методы Api через Jexl.

Доступ к Api

Настройка доступа к Pkg осуществляется аналогично, но на закладке Доступ к пакетам

Разрешения Jexl

Для настройки доступа к классам, не являющимися часть бизнес-объекта системы GlobalERP, используется закладка Разрешения Jexl.

- **Выдача прав на пакет:** Для выдачи прав на пакет, необходимо выбрать тип привилегии `package` и имя привилегии соответствующее пакету. Таким образом из Jexl-скрипта будут доступны все классы данного пакета.
- **Выдача прав на класс:** Для выдачи прав на класс, необходимо выбрать тип привилегии `class` и имя привилегии соответствующее текущему классу. Что бы выдать права на конкретные методы класса, необходимо в поле Методы в виде Json-массива указать их наименования. При необходимости выдать права на все методы, кроме указанных, следует установить галочку Указанные методы исключяющие.

Разрешения Jexl

Выдача прав на таблицы в SQL запросах

Выдать права на доступ можно в карточке Роли.

1. Перейдите на закладку Доступ к таблицам в SQL запросах.
2. Найдите необходимые вам таблицы и выдайте/отзовите права с помощью соответствующего чек-бокса (слева на право - права на `Select`, `Update`, `Delete`, `Insert`)

Примечание

Права на `Insert` можно выдать только на таблицы «не наших» (`public`, `aud`, `global_dev`, `global_system`) схем.

Внимание

Права на `Select` так же могут быть получены при наличии любой не запрещающей привилегии на администрируемом объекте, соответствующем данной таблице.

14.9 Журнал доступа к объектам системы

Флаг в настройках модуля `btk` **Заполнять журнал доступа к объектам системы через Jexl**, позволяет отслеживать в аудите скриптов Jexl к каким именно объектам обращался Jexl-скрипт.

Для просмотра аудита, необходимо зайти в приложение **Настройки системы** и в пункт меню **Аудит** -> **Аудит Jexl**.

Аудит Jexl

14.10 Администрирование Rest-сервисов

Класс `Btk_AcPackage` содержит информацию об администрируемых пакетах:

- Имя пакета
- id модуля, которому он принадлежит
- является ли он rest-пакетом (`bIsRest`)
- администрируются ли серверные полномочия (`bControlServerPriv`)

Пакет считается Rest-пакетом, если он наследуется от трейта `RestPkg`, в том числе от его потомков. Rest-пакеты регистрируются при обновлении администрируемых объектов по модулю. Если при обновлении окажется, что пакет больше не наследуется от `RestPkg`, то признак `bIsRest` в таблице будет снят.

Внимание

После изменения статуса администрируемости пакета, необходимо сбросить shared-кэш по классу `Btk_AcPackage`.

Выдача прав на вызов Rest-пакетов

В карточке роли на закладке **«Права на Rest-пакеты»** отображен список пакетов из класса `Btk_AcPackage` и имеет ли роль доступ к ним. Информация по доступу для роли хранится в таблице `Btk_AcRolePackagePriv`. Для выбора\снятия доступа роли к пакету используйте чекбокс **Имеет доступ**. Во время индексации прав пользователей по роли для всех Rest-пакетов, у которых стоит галка **«Контролировать серверные полномочия»**, будут выданы объектные привилегии пользователям, которые имеют данный профиль. Привилегии регистрируются на адм. объект `Btk_AcPackage`.

Примечание

Супер-пользователь имеет полный доступ

14.11 Администрирование SOAP-вызовов

Администрирование SOAP-вызовов включается в Администратор \ Настройки > Настройки администрирования соответствующим чекбоксом. Для выдачи права роли необходимо дать доступ к объектной привилегии UseSoap адм. объекта Btk_ManagementPkg.

Примечание

Супер-пользователь имеет полный доступ

15 Приложение 3. Ответы на задания

Ответы временно не доступны.

16 Приложение 4. GlobalFrameworkStarterKit

16.1 Описание

GlobalFramework Starter Kit - это универсальный набор инструментов и настроек, созданный для ускорения старта прикладной разработки.

Состав

Шаблоны для IntelliJ IDEA (GigaIDE)

Готовые шаблоны для быстрого создания объектов системы.

Вспомогательные утилиты разработчика для IntelliJ IDEA (GigaIDE)

Набор утилит для обслуживания проекта.

Плагин разработчика Global для IntelliJ IDEA (GigaIDE)

Плагин предоставляет дополнительные возможности при работе с проектом.

Документация по GlobalERP ToolKit

16.2 Установка

Шаблоны и вспомогательные утилиты разработчика для IntelliJ IDEA (GigaIDE)

Выгрузите шаблоны и утилиты из репозитория по [ссылке](#).

Скопируйте содержимое каталога IntelliJIdea.files в каталог с настройками IntelliJ IDEA или GigaIDE.

Примеры каталогов для IntelliJ IDEA:

- Windows: C:\Users\<username>\AppData\Roaming\JetBrains\<IDEA_version>
- Alt Linux: /home/<username>/.config/JetBrains/<IDEA_version>

Примеры каталогов для GigaIDE:

- Windows: C:\Users\<username>\AppData\Roaming\GIGAIDE\<GigaIDE_version>
- Alt Linux: /home/<username>/.config/GIGAIDE/<GigaIDE_version>

16.3 Шаблоны IntelliJ IDEA (GigaIDE)

Готовые шаблоны для быстрого создания объектов системы.

Включают в себя:

- Файловые шаблоны для быстрого создания файлов модуля (Odm, Pkg, Lib и т.д.).
- Шаблоны кода.

Структура каталогов

- IntelliJIdea.files
 - fileTemplates
 - * AllTypes.odm.xml
 - * Application.avm.xml
 - * Global Avi Override.scala
 - * Global Ept.scala
 - * Global Ext.scala
 - * Global Lib.scala
 - * Global Pkg.scala
 - * Scala Test Api.scala
 - * odm collectionext.odm.xml
 - * odm mixinext.odm.xml
 - * odm simpleext.odm.xml
 - * odm v-collectionext.odm.xml
 - templates
 - * avm.xml
 - * odm.xml

* scala.xml

Состав файловых шаблонов

- AllTypes.odm.xml - Odm файл класса
- odm collectionext.odm.xml - Odm файл класса (коллекция)
- odm v-collectionext.odm.xml - Odm файл класса (v-коллекция)
- odm mixinext.odm.xml - Odm файл класса (миксин)
- odm simpleext.odm.xml - Odm файл класса (справочник/разделяемый справочник/документ)
- Application.avm.xml - Создание разметки выборки основного приложения
- Scala Test Api.scala - Шаблон scala теста для отладки автономной бизнес-логики
- Global Avi Override.scala - Создание Avi-файла для переопределения существующего класса
- Global Ept.scala - Создание точки расширения
- Global Ext.scala - Создание модуля расширения
- Global Pkg.scala - Создание файла пакета
- Global Lib.scala - Создание библиотеки

Состав шаблонов кода

Шаблоны для Odm

- refAttr - Создание ссылочного атрибута

Шаблоны для Avm

- tabitem - Создание простой закладки
- fltCaption - Условие фильтрации по наименованию
- fltmain - Создание фильтра
- fltscode - Условие фильтрации по коду
- printclone - Добавление клонируемой операции
- collTabSimple - Добавление композитора с простыми закладками из Btk_ObjectType
- collTabDyn - Добавление композитора с динамическими закладками из Btk_ObjectType
- lookup State - Редактор выпадающий список для состояния
- lookup Main - Редактор выпадающий список для любой выборки

Шаблоны Scala

- `onrefresh` - Переопределение операции `onRefresh`
- `addOper` - Добавление операции для выборки
- `trait list_master` - Трейт для коллекций по мастеру
- `def list_master` - Объявление `list_master`
- `forQ` - Цикл `for` по объектному запросу
- `ins` - Вставка записи с обработкой
- `selidOQ` - Получение `id` объекта с пессимистической блокировкой
- `forS` - `Aselect` с получением `id`
- `msel` - Получение значений полей по мультиселекту в `ListBuffer` по имени атрибута
- `setter` - Шаблон сеттера
- `selidASQL` - Получение `id` через `ASQL`
- `selidaList` - Получение списка `id` через `ASQL`

16.4 External Tools

Набор утилит для обслуживания проекта

Состав

- `Generate Sources` - Кодогенерация по каталогу/модулю/проекту
- `Generate Sources (only this item)` - Кодогенерация по выбранному классу
- `Generate tables` - Синхронизация схемы БД с метаданными
- `Generate tables force` - Принудительная синхронизация схемы БД с метаданными
- `Run Migrations` - Запустить скрипты миграции бизнеслогики
- `Run Schema Upgrade` - Запустить скрипты миграции БД
- `Generate build.sbt` - Пересоздать `build.sbt`
- `Export data to XML` - Экспортировать файл в виде XML
- `Import data from XML` - Импортировать данные из XML файла
- `Open in VSCode` - Открыть файл в редакторе кода VSCode