Бозененков Ю.М.

Основы практического программирования на платформе IsFusion

2024

Оглавление

Предисловие

Около пяти лет назад белорусская компания НТ ООО «ЛюксСофт» публично представила открытую и бесплатную платформу для разработки бизнес-приложений. С момента официального выпуска платформы вокруг нее успело вырасти комьюнити разработчиков и пользователей бизнес-решений на ее базе, появились новые прикладные решения, написанные как самой компанией «ЛюксСофт» и ее партнерами, так и сторонними разработчиками.

Платформа lsFusion с момента выпуска сопровождается полной и актуальной бесплатной документацией, опубликованной на официальном сайте, однако официальная документация к платформе – это не учебник, а, скорее, справочное пособие. Рассчитана она, в первую очередь, на тех, кто уже освоил разработку на базе lsFusion. Даже беглое знакомство с чатами комьюнити показывает, что существует запрос на простое и понятное руководство по платформе, позволяющее начинающему разработчику, следуя простыми и понятными шагами, постепенно от простого к сложному начать изучать программирование на языке lsFusion. Представленная книга, как мне кажется, призвана, в какой-то мере, решить эту задачу. В книге автор, через призму своего предыдущего опыта, попытался провести читателя по пути создания небольшого бизнес-приложения. Начиная от формализации и создания простых прикладных объектов бизнес-логики, таких как справочники и документы, заканчивая простыми рецептами по построению отчетности и интеграции с внешними системами.

Представленную работу не стоит рассматривать как учебник по программированию – автор в книге не всегда использует подходы, которые принято называть "best practice", описывая, иногда, не самые очевидные пути решения задач. В частности, автор, на мой взгляд, слишком увлекся метапрограммированием. Само по себе метапрограммирование – это мощный инструмент платформы, однако злоупотребление им обычно негативно сказывается на читаемости кода и, как следствие, усложняет дальнейшую поддержку бизнес-решений. Большинство задач переиспользования кода, которые автор решает метапрограммированием, можно решить и другими инструментами, которыми располагает язык платформы. Несмотря на это, в книге приводится множество готовых решений, которые можно использовать в качестве образца, при работе над типовыми практическими задачами.

В качестве небольшого резюме можно сказать, что автором проделана большая работа, значение которой сложно переоценить. И, даже несмотря на то, что работа не раскрывает в полной мере большинство возможностей платформы, она дает, как минимум, общее представление о процессе разработки прикладных решений, учит эффективно читать код, написанный на языке платформы, дает некоторые распространенные паттерны решения типовых задач, встречающихся в реальной разработке.

Алексей Муренков
Разработчик НТ ООО "ЛюксСофт"

От автора

О себе. Программированием занимаюсь очень-очень давно.

При этом работал и на фирмах и самостоятельно, как ИП, решая разные экономические и технические задачи. Были проекты, которые разрабатывались с нуля, когда ничего нет от слова «совсем» – ни платформы, ни библиотек, ни наработок, есть только голый лист бумаги, пустой экран компьютера и расплывчатая постановка задачи от заказчика. Для решения задач довелось использовать различные программные средства, затрачивая при этом много времени и усилий, связанных с разработкой приложений.

Находкой стало использование платформы IsFusion. Кроссплатформенное бесплатное решение, сокращающее разработку в разы. Первый проект был сложным, не в плане реализации идей, а в плане изучения платформы по ее документации. Главное было понять и принять новые подходы к процессу разработки. Собственно, тогда и родилось решение написать небольшую книгу, рассказывающую о платформе, как об эффективном средстве разработки, воплотить наработанные знания в небольшом решении, где последовательно изложить накопленный опыт.

Хочется надеяться, что изложенный материал, в свою очередь, поможет независимым разработчикам легче освоить работу с платформой для разработки собственных бизнес-приложений.

Буду благодарен за Ваши отклики и комментарии по этому материалу.

Как правильно пользоваться

Как правильно пользоваться представленным материалом?

Приводимый материал не является справочным пособием.

Цель материала – показать на связанных примерах основы программирования на платформе lsFusion.
В основу излагаемого материала положено рассмотрение простого приложение типа складского учета. Изложение строится от простого к сложному, охватывая разные темы изучения.
Структурно материал разбит на темы, которые подразделяются на вопросы. В начале каждой темы приводится кликабельный список рассматриваемых вопросов.

В конце рассматриваемого вопроса приводится пункт "См. также", включающий:

- ссылки на официальную документацию или статьи
- пример программного кода, соответствующий рассматриваемому вопросу
- архив SQL сервера, соответствующий рассматриваемому вопросу
- другие материалы

Например:

См. также
Документация, Классы

Пример кода:
examples\t208_статический_объект\src

Архив SQL:
examples\t208_статический_объект\sql

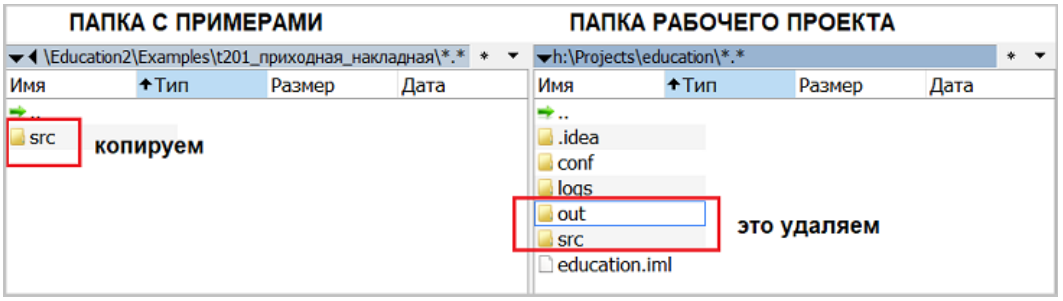
Предлагаемая методика изучения заключается:

- в последовательном рассмотрении вопросов
- в самостоятельном наборе программного кода в рамках изучаемого вопроса, с выходом на конечный результат, для этого:
    - излагаемый материал содержит фрагменты программного кода с описанием
    - полный пример программного кода src
    - архив SQL
- в личных экспериментах внутри созданного (восстановленного) программного кода с анализом результатов.

При изучении рассматриваемых вопросов **важно** не просмотреть их, а именно работать с программным кодом, нарабатывая практический опыт. Каждый рассматриваемый вопрос – это отдельный элемент изучения, но пособие построено таким образом, что все изучаемые элементы между собой связаны. Не надо опасаться сломать что-либо или испортить данные – в любой момент, если "что то пошло не так", есть копия программного кода и архив базы данных.

Для лучшего понимания рассматриваемых вопросов присутствует легенда о необходимости разработки приложения для учета товаров на оптовой базе, при этом группы вопросов или тем содержат такие параграфы как "Постановка задачи" и "Что надо сделать".

Примечание:
Для восстановления архива SQL сервера, смотри: "Восстановление из архива".
Для восстановления программного код в IDEA необходимо код примера скопировать в рабочую папку проекта с именем src. Для этого в папке проекта удаляют каталоги src и out, и далее копируют каталог src из папки выбранного примера в каталог проекта.



Примеры программного кода и SQL находятся в архиве Examples.zip.
Скачать архив.

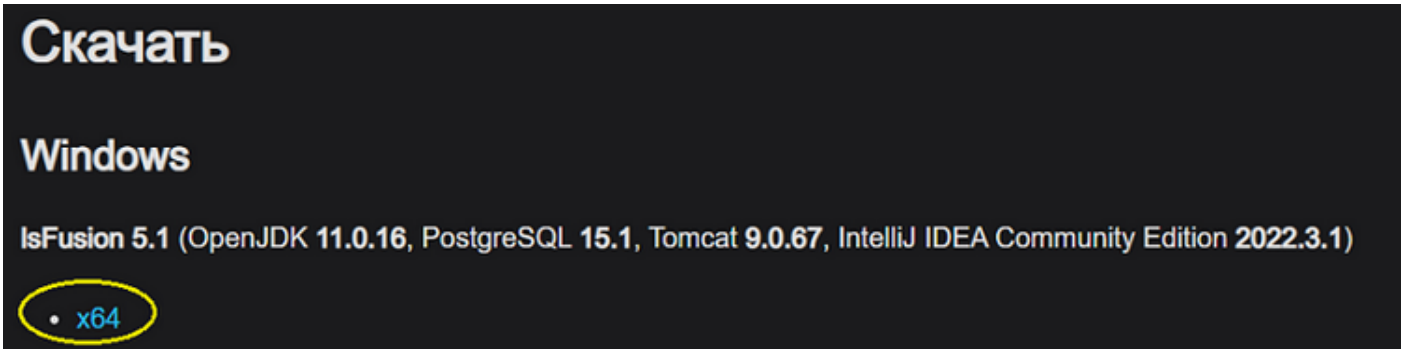
Тема 1. Установка платформы

Рассматриваемые вопросы:

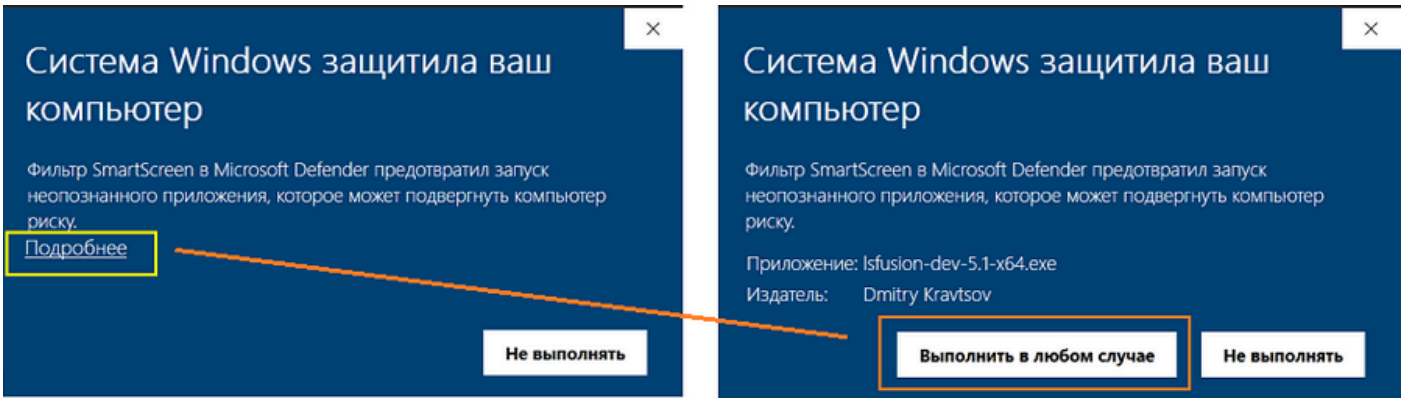
1. Установка платформы
2. Создание и запуск пустого приложения
3. Настройки приложения
4. Полезные настройки IDEA
5. Клавиши управления IDEA

Установка платформы

Начинать изучать lsFusion надо с установки платформы, если платформа не была установлена ранее.
Для установки платформы необходимо обратиться к официальной документации из раздела "Установить - Автоматическая установка - Для разработки" и кликнуть на пункт "x64".



В процессе установки может возникнуть ситуация, что ваша антивирусная система "гневно" отреагирует на установку, например Nod32, и не даст скачать дистрибутив. В этом случае необходимо приостановить защиту на момент скачивания или добавить исключение в свою антивирусную программу. Информация, предназначенная для скачивания, не содержит вирусов, проблема в сертификатах. Также при установке уже скачанного дистрибутива (например, lsfusion-dev-5.1-x64.exe) ОС Windows 10 может отобразить такое сообщение:



В этом случае нажимаем на кнопку "Подробнее" (левая картинка), окно немного изменит содержание: появится кнопка "Выполнить в любом случае". Нажимаем на кнопку "Выполнить в любом случае", и при этом начнется процесс инсталляции платформы. В процессе установки будут установлены пакеты, необходимые для работы платформы.

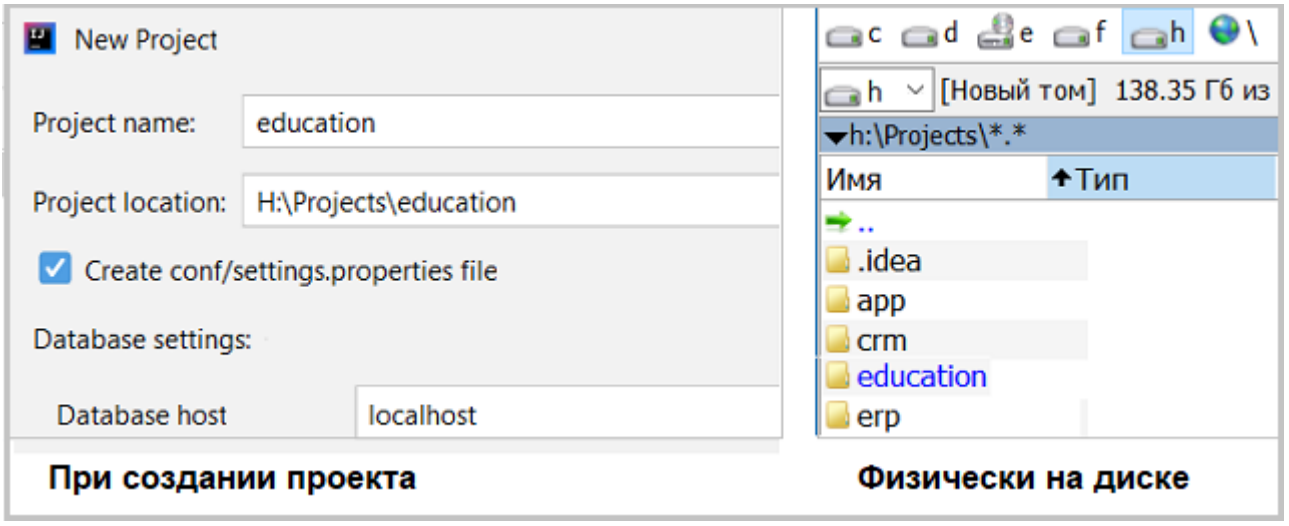
Основные рекомендации при установке:

- отвечать "да"
- не игнорировать пункты установки
- запоминать придуманные пароли

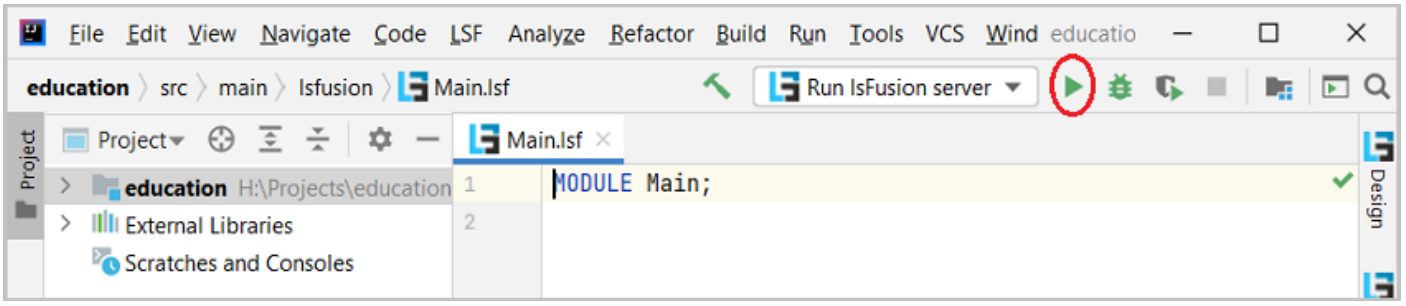
Создание и запуск пустого приложения

Процесс создания приложения хорошо описан в официальной документации в разделе "Создание нового lsFusion проекта".

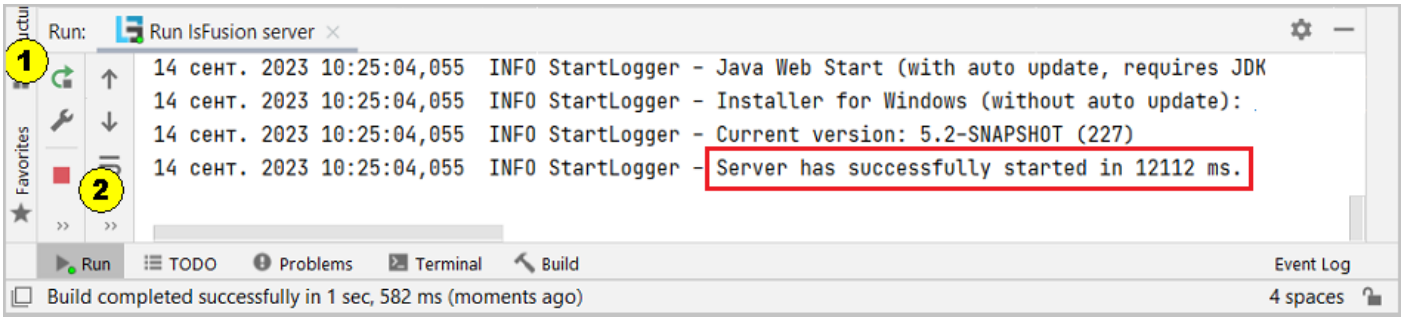
Можно рекомендовать при создании нового проекта определить на доступном диске в корне папку с проектами lsFusion, а не тот подкаталог, который предлагается по умолчанию - будет проще навигация и замена программного кода из каталога примеров (папка src). При этом, при создании следующего проекта, редактор IDEA будет помнить существующий каталог, например:



Чтобы выполнить созданное приложение, в редакторе IDEA надо нажать "волшебную кнопочку", как показано на рисунке:



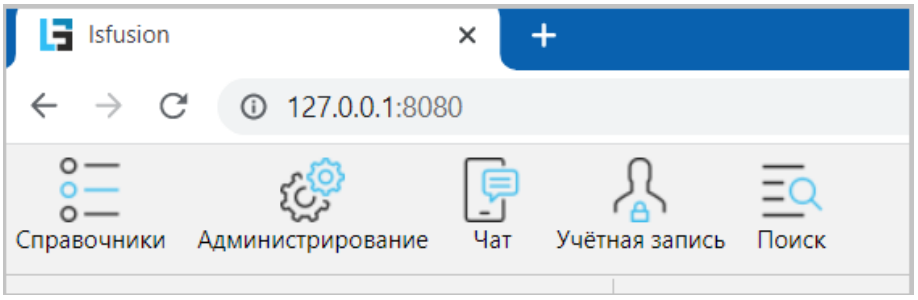
Если все было правильно установлено, то при запуске из IDEA сервера приложений, в окне сообщений появиться фраза "Server has succesfully started in *nnn* ms.", которая укажет, что все ранее выполненные действия были сделаны правильно:



На изображении отмечены кнопочки:

1. Быстрый рестарт сервера
2. Остановка сервера

После этого можно запустить WEB или Desktop клиента:



Примечание:

- в основном все приводимые в изложении копии экранов клиентской части приложения получены из Web версии
- при изучении учебников по языкам программирования, приводится обычно пример вывода сообщения "Hello world". Запуск пустого приложения можно также считать своеобразным "Hello world", говорящим о том, что все предварительные действия были сделаны правильно.

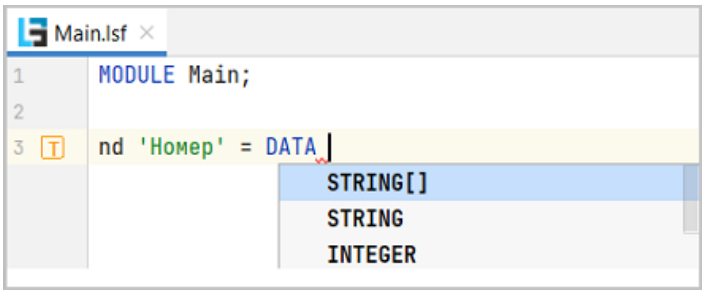
Хотя модуль условно пустой, тем не менее можно составить первое впечатление о синтаксисе языка lsFusion.

Синтаксическая конструкция состоит из одной инструкции:

- заголовок MODULE - имя программируемого модуля
- имя модуля без указания расширения
- обязательная точка с запятой, завершающая синтаксическую конструкцию

Все модули на платформе lsFusion всегда начинаются с инструкции MODULE.

При разработке на платформе lsFusion можно и нужно пользоваться контекстной подсказкой по коду (ключевым словам, именам классов, именам параметров и т. д.), которая облегчит написание выражений.
Например:



Подсказка вызывается по комбинации клавиш CTRL+Пробел.

Подсказка является контекстно-зависимой и отображает информацию, основываясь на том, что вы написали ранее. Подробнее читайте в разделе Справка по ключевым словам.

См. также

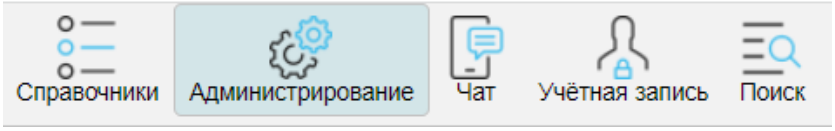
Документация, Заголовок модуля
Документация, Язык

Настройки приложения

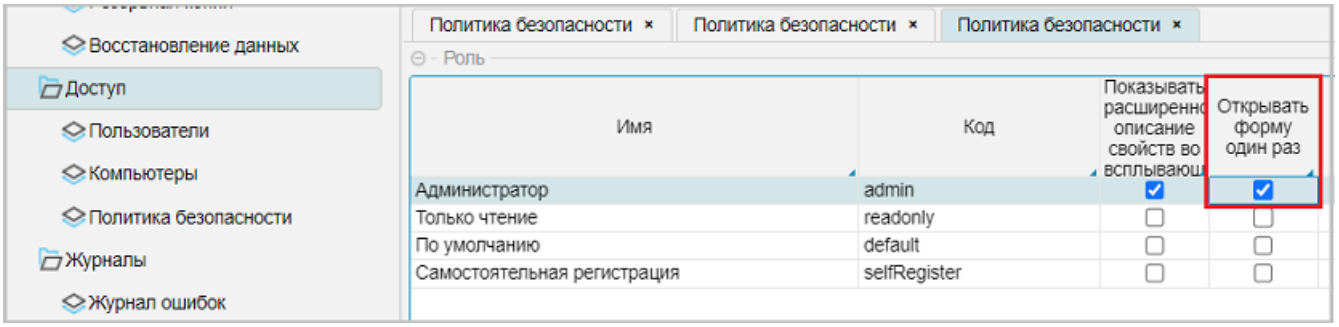
Для комфортной работы в приложении:

- определить настройки открытия форм
- завести кроме пользователя по умолчанию 2-х других

Эти действия выполняются через элемент навигатора "Администрирование", который выглядит как пункт горизонтального меню в верхней части окна приложения. В дальнейшем для простоты изложения элементы навигатора будем называть пунктами и подпунктами меню.



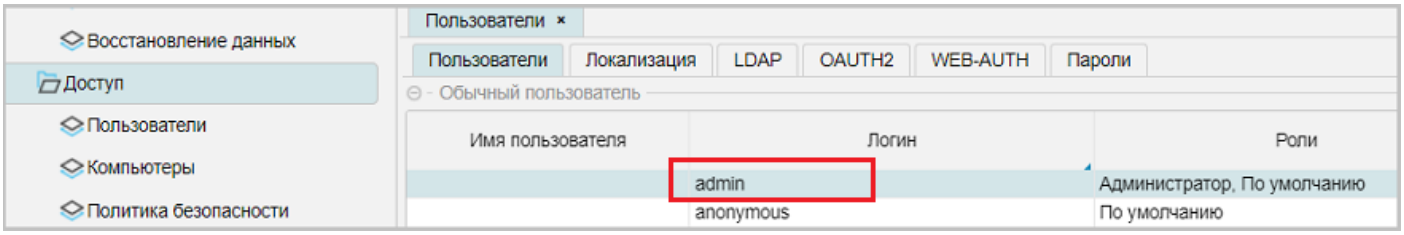
При появлении подменю слева надо найти "Система - Доступ" и несколько раз нажать на пункт "Политика безопасности". Вверху откроется несколько одинаковых вкладок. Это может быть неудобно, и для текущей роли можно назначить, что форму можно открывать только один раз, если она не была открыта ранее.



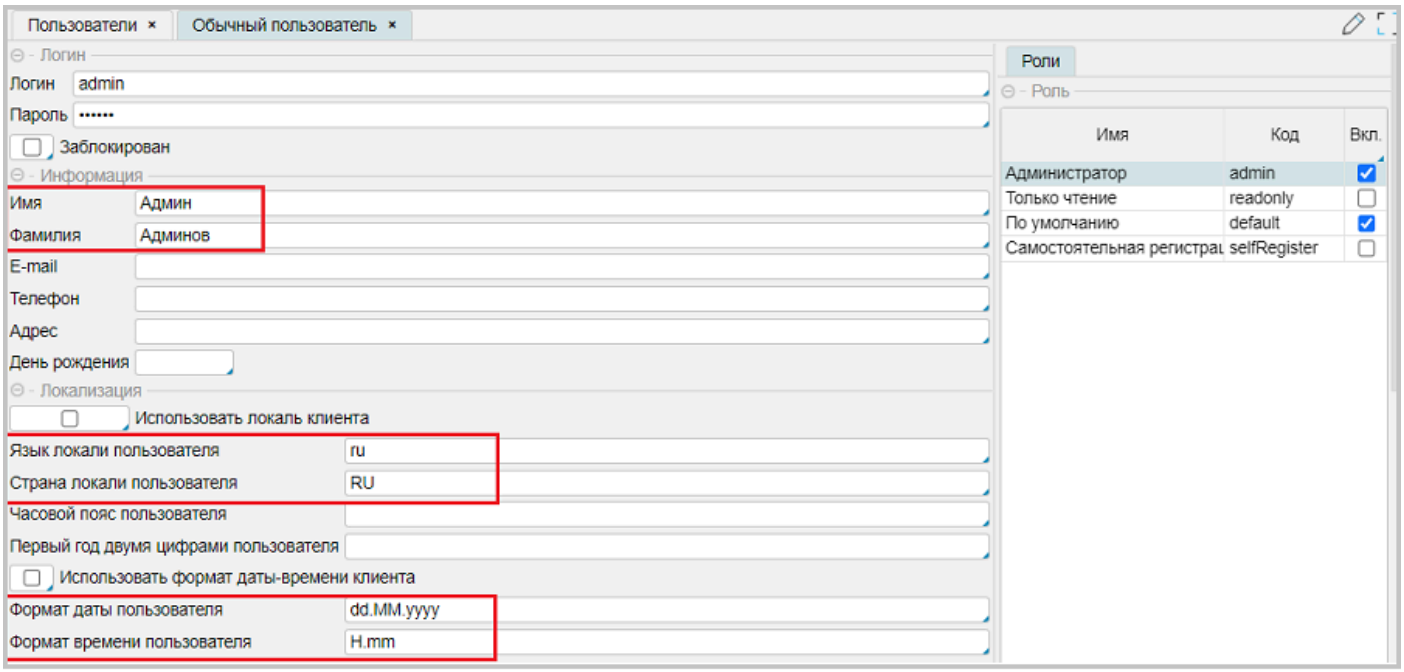
Для текущей роли "Администратор" установим галочку "Открывать форму один раз" и нажмем на кнопку "Сохранить" в самом низу экрана. Опция начнет действовать при перезапуске клиента (завершить и открыть заново). Если все сделали правильно, то форма в пункте "Политика безопасности" будет открываться только один раз.

Важной также является соседняя галочка "Показывать расширенное описание свойств…". Для программиста, который правит код – это важно, так как при наведении курсора мыши на элемент интерфейса работающего приложения будет всплывать подсказка, где этот элемент используется (об этом речь пойдет ниже). Это сильная сторона IsFusion - фишка, которая значительно сокращает время, связанное с доработками программ. Для обычных пользователей (непрограммистов) галочка должна быть снята (см. здесь Интерфейс подсветка свойств).

Заводим новых пользователей.
Выбираем пункт меню "Администрирование - Система - Доступ - Пользователи", при этом на экране откроется новая форма отображения со списком текущих пользователей системы:



Устанавливаем курсор на пользователя с логином admin и нажимаем на кнопку "Редактировать" под табличной частью, при этом откроется форма редактирования, в которой заполняем следующие реквизиты:



- Имя, например Админ (или другое)
- Фамилия, например Администов (или другое)
- Язык локали пользователя: ru
- Страна локали пользователя: RU
- Формат даты пользователя: dd.MM.yyyy
- Формат времени пользователя: H.mm

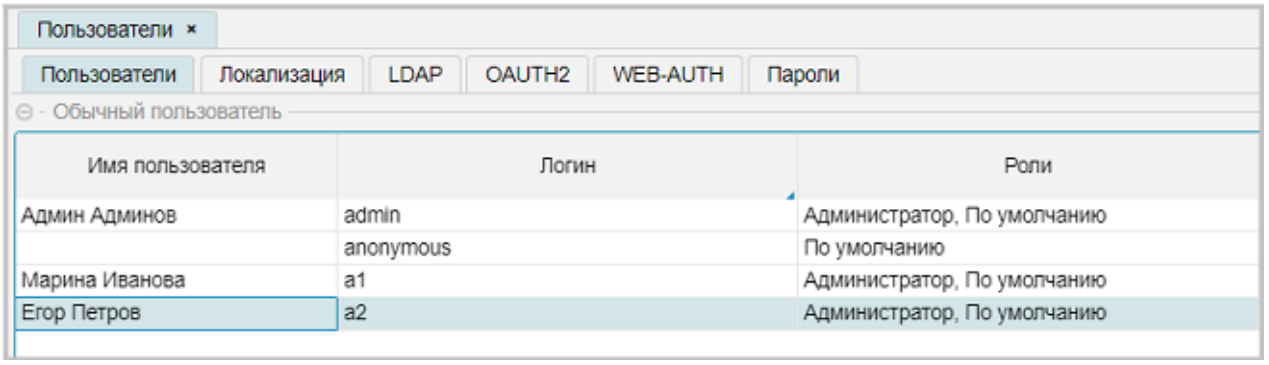
Значение	Описание
yyyy	кол-во цифр года
MM	месяц
dd	день
HH	часы (24-часовой формат)
mm	минуты
ss	секунды
SSS	миллисекунды

После завершения ввода нажимаем на кнопку "Ок" в правом нижнем углу формы.

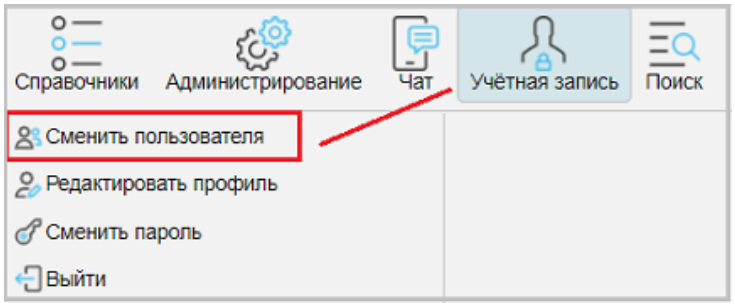
Затем нажимаем кнопку "Добавить" в нижнем левом углу под табличной частью и заводим 2-х новых пользователей, например:

Поле ввода	Значение
Логин	a1
Пароль	123
Имя	Марина
Фамилия	Иванова
Логин	a2
Пароль	123
Имя	Егор
Фамилия	Петров

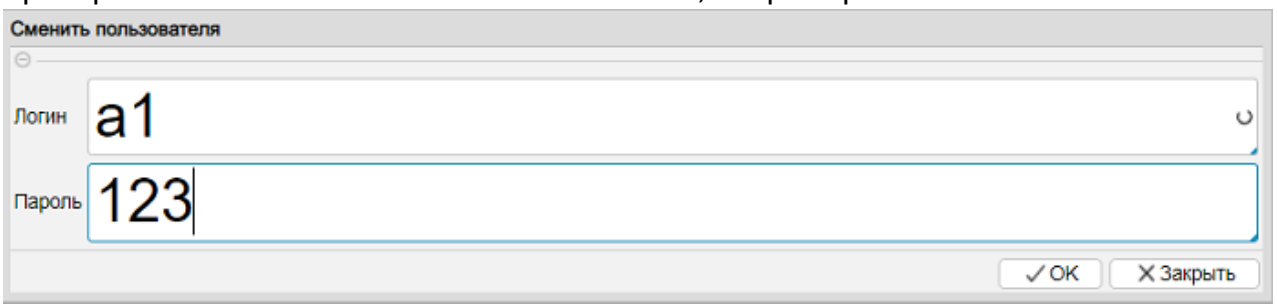
Локали, форматы и роли такие же, как у администратора *Админа Администова*.
Не забываем по окончании ввода нажимать на кнопку "Ок" в правом нижнем углу формы редактирования. По окончании ввода табличная часть будет иметь вид:



После внесения исправлений проверяем сделанные изменения, для этого выбираем пункт горизонтального меню "Учетная запись - Сменить пользователя":



Проверяем возможность смены пользователей, например:



Примечание:

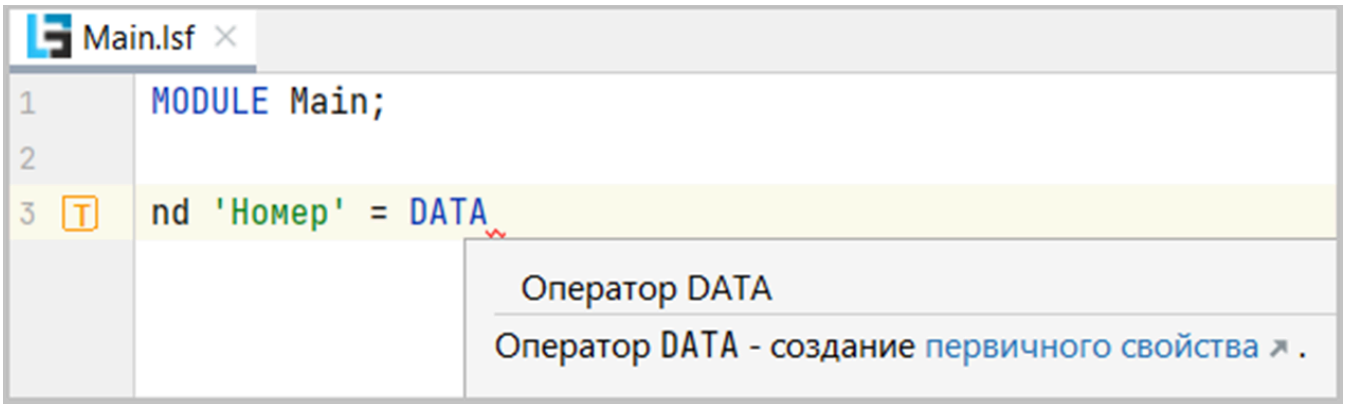
- если кнопка "Ок" не видна, то попробуйте растянуть форму вниз до появления кнопок в самом низу формы. В последующем форма запомнит свое состояние

- для Админа Админова вносим только логин admin и нажимаем "Ок"

См. также
Архив SQL:
examples\t101_настройки\sql

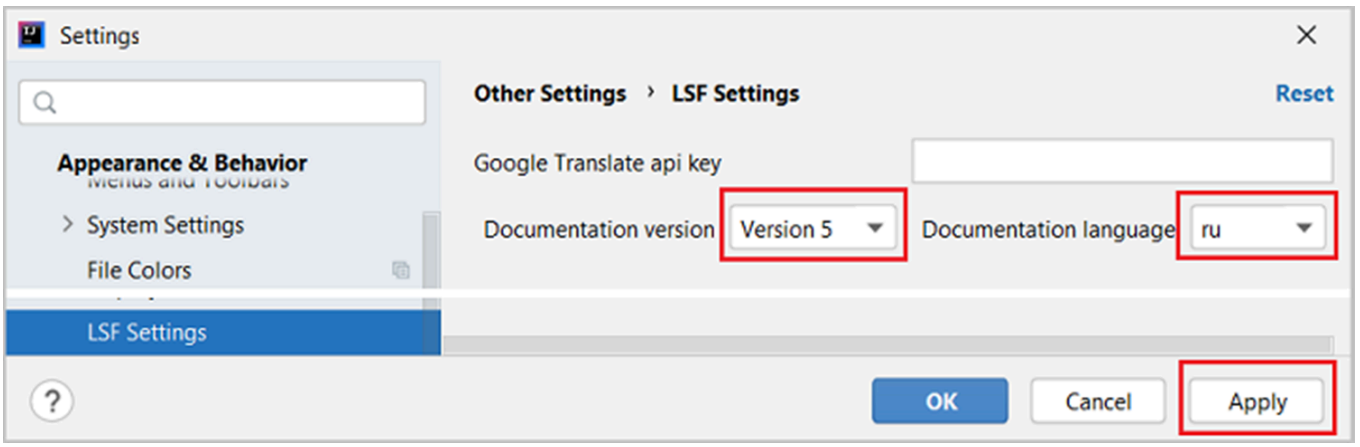
Полезные настройки

При установке плагина lsFusion в редакторе IDEA станет доступна автоматическая справка при наведении указателя мыши на ключевое слово, например:



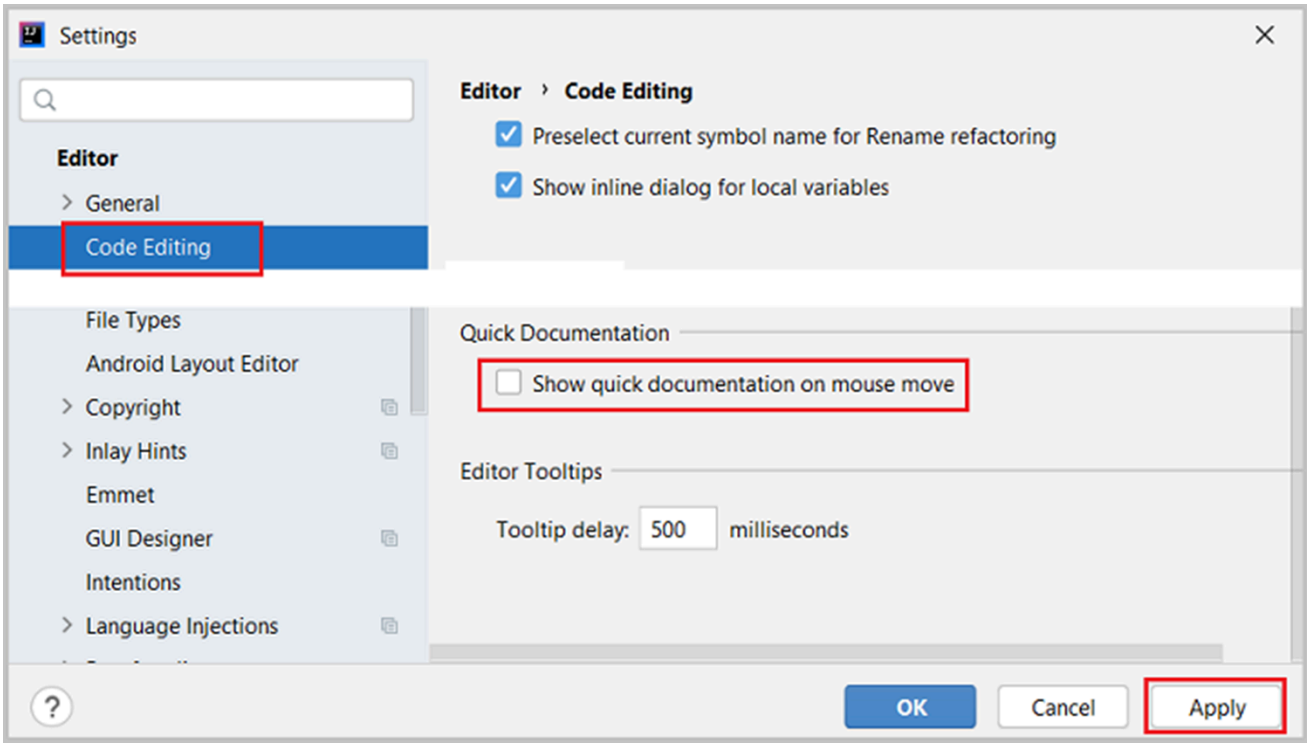
При этом можно настроить отображение, версию и язык справки.

Для изменения языка справки в IDEA выбираем в меню "File - Settings". В открывшемся окне "Settings" выбираем "Other Settings - LSF settings" (возможен вариант LSF Settings *без Other Settings*)



- "Ducumentation version - Version 5" - версия документации, соответствующая текущей установленной версии платформы
- "Documentation language" - язык документации, выбираем "ru" (русский) или "en" (английский)

Для изменения режима автоматического всплытия справки при наведении курсора на ключевое слово в IDEA выбираем в меню "File - Settings". Меню формы "Editor - Code Editing", флажок "Show quick ..."



- если галочка установлена, то подсказка автоматически всплывает при наведении мыши на ключевое слово
- если галочка снята, то вызов справки по ключевому слову может быть вызван наведением курсора мыши на ключевое слово и нажатием комбинации клавиш CTRL + Q.

Полезные сочетания клавиш

IDEA предлагает обширный набор сочетаний клавиш, который значительно облегчает и упрощает работу и навигацию по проекту. Установка плагина IsFusion расширяет эти возможности, адаптируя существующие сочетания клавиш для работы с кодом на языке IsFusion и добавляя уникальные сочетания, специфичные для этого языка. Приведем небольшой список некоторых часто употребляемых сочетаний клавиш.

Клавиши	Описание	Примечание
CTRL+F	текстовый поиск в текущем модуле	Поиск может быть регистрозависимым или нет Поиск может содержать несколько аргументов для поиска, например: слово 1 .*слово 2 .*слово 3 и т.д.
CTRL+SHIFT+F	текстовый поиск по проекту	
двойной SHIFT	глобальный поиск файлов, классов, свойств и других элементов	
CTRL+ALT+F7	поиск всех использований элемента (свойства, класса и т.д.)	
CTRL+B	переход к объявлению элемента	
CTRL+ALT+левая стрелка	возврат к предыдущему месту в коде, например, после CTRL+B или CTRL+ALT+F7	
CTRL+U	переход к метакоду, где был объявлен элемент	
CTRL+Q	вызов справки	

Полный список стандартных клавиш IDEA

Тема 2. Простое приложение

Рассматриваемые вопросы:

1. Постановка задачи
2. Приходная накладная
3. Интерфейс. Табличный процессор
4. Новые свойства. Расширение форм
5. Контроль ввода
6. Анализ приложения
7. Модульность
8. Наследование
   - Родительский класс Document
   - Дочерний класс Reception
   - Дочерний класс Expenses
9. Метапрограммирование
10. Краткие итоги

Постановка задачи

Надо создать программу для учета товародвижения на оптовой базе.

Принцип работы базы с товарами заключается в приеме товаров от поставщиков и продаже товаров розничным торговым объектам.

Структурно программа будет представлена плоскими документами (без явного выделения шапки и строк документа). Реестр приходных документов, реестр расходных документов, остатки. При этом приходная накладная выступает в качестве документа для списания остатков.

При разработке приложения отталкиваемся от партионного учета, то есть позиции для расхода списываются с позиций прихода (пришедшей партии товара). Для расходной накладной свойства название товара и цена берутся из строки приходной накладной, с которой товар списывается. При списании товара действует оптовая надбавка в 10%. Товары учитываются в ценах поставщика.

В программе должны быть предусмотрены механизмы контроля:

- вводимых данных
- остатков для списания
- информации, кто и когда вносил данные

Для документов прихода и расхода приняты следующие реквизиты:

- номер документа - number
- дата - date
- организация (поставщик/получатель) - organization
- название товара - name
- количество - quantity
- цена - price

Основная цель:

- минимально научится писать программный код на языке lsFusion
- понять содержание основных этапов, связанных с разработкой приложений

Приходная накладная

Если бы мы создавали свою первую программу на каком-либо общем языке программирования, то скорее всего, наши действия были бы следующими:

1. Создали БД командой CREATE DATABASE ...
2. Создали таблицу командой CREATE TABLE ...
3. Придумали какой-то класс формы, который бы позволял вносить и отображать данные, а также их сохранять в БД SQL сервера
4. Разработали меню для вызова формы

Алгоритм имеет упрощенный вид и над его полнотой можно поспорить в сторону усложнения. Но в данном случае это неважно. Если кто-либо сталкивался с подобного рода задачами, наверняка знает, что эта работа занимает определенное время, связанное с созданием и отладкой программного кода.

А теперь представим, что, используя платформу lsFusion, эта задача решается элементарно быстро.

1. Создание БД. После создания проекта и первого запуска сервера приложений база данных будет создана автоматически.
2. Создание таблицы. Платформа lsFusion автоматически создает необходимые таблицы для работы, от разработчика требуется определить свойства, которые будут хранить информацию. При этом никакого администрирования базы данных, связанного с этими действиями, не требуется. После каждого нового старта сервера приложений все необходимые изменения, если таковые были, выполняются автоматически.
3. Процесс создания формы, который бы позволял вносить и отображать данные, прост и состоит из простых правил: определение имени формы, заголовка, объектов формы, перечисления созданных свойств, а также операторов для работы с объектами формы (создание, удаление).
4. Разработать меню для вызова формы. В терминах платформы необходимо будет создать элемент навигатора, который визуально будет выглядеть как пункт меню.

Вот, собственно, и все. Описание того, что надо сделать, по времени занимает больше, чем сам процесс программирования.

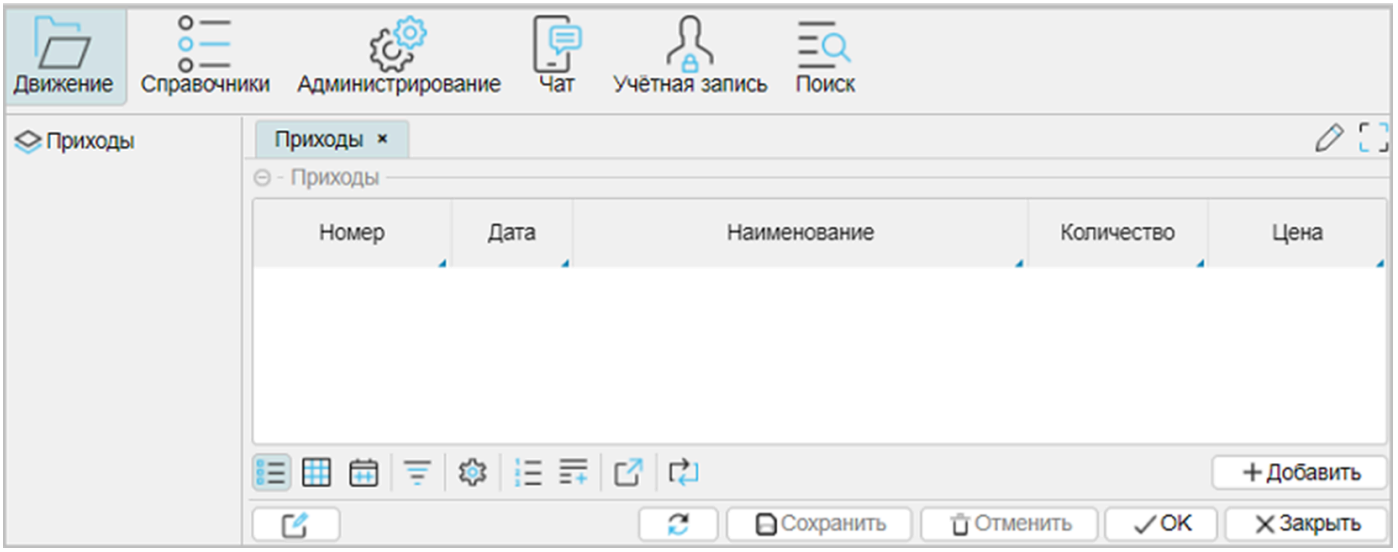
Так как сервер приложений запускался ранее для выполнения настроек приложения, проверим некоторые утверждения пункта 2. Для этого воспользуемся программой pgAdmin 4 (программный продукт для администрирования и разработки баз данных *PostgreSQL*), найдем и откроем базу данных с именем education и обратим внимание на текущее количество таблиц (см. education - Schemas (1) - Public - Tables (119)). 119 таблиц. Запомним это число.

Наполним содержанием модуль Main, в соответствии с ранее описанным планом: создание свойств, определение формы и элементов навигатора:

```
MODULE Main;

// 1. БД была создана ранее при создании проекта

// 2. Создаем новый класс: Приходная накладная
CLASS Reception 'Приходы';

// 2. Описываем свойства класса Приходная накладная
number 'Номер' = DATA STRING[10] (Reception);
```

```
date 'Дата' = DATA DATE (Reception);
name 'Наименование' = DATA STRING[50] (Reception);
quantity 'Количество' = DATA NUMERIC[10,3] (Reception);
price 'Цена' = DATA NUMERIC[10,2] (Reception);

// 3. Назначаем форму для отображения и редактирования свойств класса
FORM viewReception 'Приходы' // название формы и заголовок
    OBJECTS r = Reception    // объект формы
    PROPERTIES (r) number, date, name, quantity, price // список свойств
    PROPERTIES (r) NEW, DELETE // операторы работы с объектами
;

// 4. создаем пункт меню
NAVIGATOR {
    NEW FOLDER move 'Движение' WINDOW toolbar FIRST { // Новый пункт горизонтального меню
        NEW FORM viewReception; // В качестве подпункта меню Заголовок формы
    }
}
```

После старта сервера приложений и запуска клиента, должна получится такая картинка:



Для проверки утверждения пункта 2 нужно повторно воспользоваться программой pgAdmin 4 и обратить внимание на количество таблиц в базе данных - их стало на одну больше (120). Получается, что программный код, который был создан, повлиял на структуру SQL БД данных.

Это важно!

- Программный код управляет содержанием и изменением структуры базы данных SQL. Это очень удобно, когда не надо "админить" физическую структуру базы данных: мы описываем свои действия, которые хотим воплотить в жизнь, а платформа сама решает, как и что добавить или изменить.
- Можно найти и имя созданной таблицы, которая появилась в результате наших действий: _auto_main_reception. Разбирая название таблицы и просматривая ее структуру, можно увидеть использование имен в некоторой комбинации, которые были определены в процессе написания кода приложения. Отсюда следует вывод, что платформа создала для себя физическое представление структуры, с которой будет работать. Попытка вмешаться в физическую сущность структуры базы данных приведет к нарушению работы платформы. Поэтому непосредственно с физическими

структурами базы данных программист на платформе lsFusion не работает, все изменения делаются в программном коде или через меню "Администрирование".

Примечание:

- При создании класса, его можно было бы связать с физической таблицей, используя инструкцию TABLE. Но, это не обязательно и этим можно пренебречь, так как использование TABLE оправдано в случаях оптимизации хранимых данных, когда используется "ручное" управление местами хранения свойств.
- Для ускорения выборки данных можно было проиндексировать данные, используя инструкцию INDEX. Учитывая незначительный объем данных в примерах, работу с локальной базой данных, инструкцией INDEX можно пренебречь. Ее наличие или отсутствие не повлияет на скорость выполнения при отражении или редактировании данных, то есть мы не ощутим преимуществ ее использования.

При этом надо понимать:

- индексирование ускоряет чтение данных из базы данных SQL
- индексирование замедляет запись данных в SQL при большом количестве индексов, так как серверу базы данных необходимо время, чтобы обновить индексные структуры при очередной порции записи данных

См. также

Статья, Не очередной язык программирования. Часть 1: Логика предметной области
Статья, Не очередной язык программирования. Часть 2: Логика представлений
Статья, Не очередной язык программирования. Часть 3: Физика
Документация, Именования
Документация, Политика оформления кода
Документация, Заголовок модуля
Документация, Инструкция CLASS
Документация, Таблицы
Документация, Инструкция TABLE
Документация, Инструкция INDEX
Документация, оператор DATA
Документация, Инструкция FORM
Документация. Интерактивное представление. Операторы работы с объектами
Документация, Блоки Объектов
Документация, Инструкция NAVIGATOR

Пример кода:
examples\t201_приходная_накладная\src

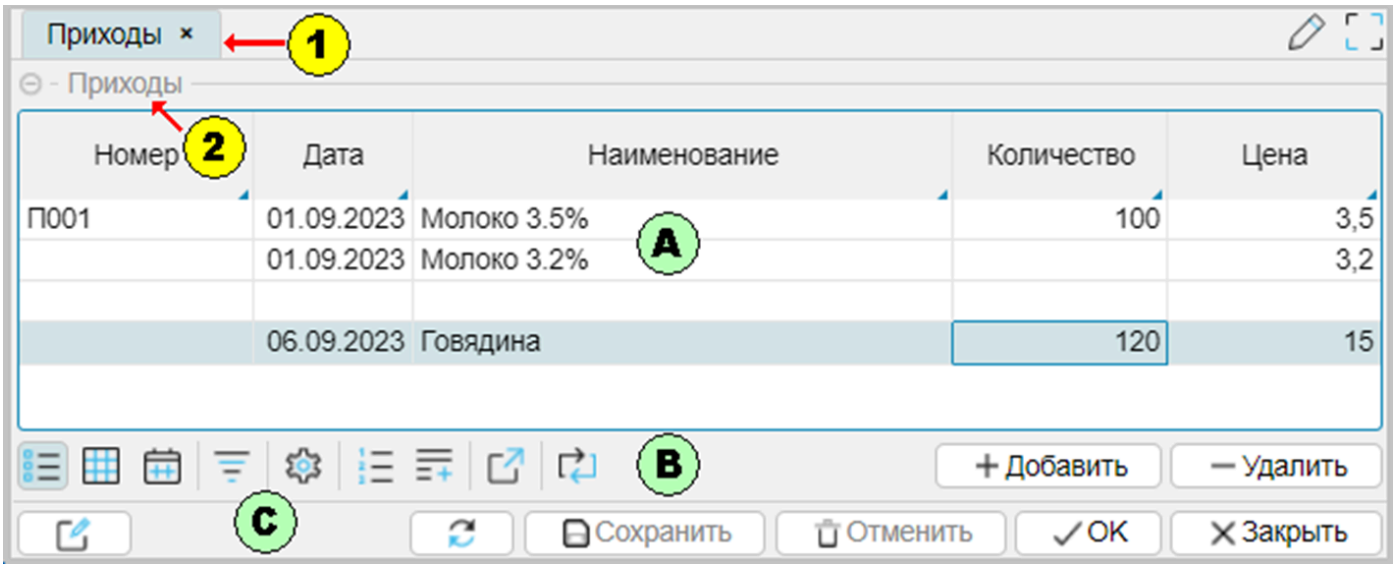
Интерфейс. Табличный процессор

Наполним данными созданное приложение.
Обратим внимание, что, когда табличная часть пуста, видна только кнопка "Добавить", кнопки "Удалить" нет, она появится с первой записью.
Это особенность платформы lsFusion: отображать только то, что существует.

Например, можно используя инструкцию NAVIGATOR, создавать кучу пунктов меню, так сказать на перспективу, что иногда встречается в некоторых программах, но не наполнять их конкретным содержанием. Такие пункты меню отображаться не будут.

Вносим данные, как угодно, особо не заботясь о правильности и полноте наполнения - все равно контроля ввода данных пока нет.
При появлении первой записи активными также станут кнопки "Сохранить" и "Отменить".

В результате действий по вводу содержание формы примет такой вид:



Разберем форму:

- <1> - заголовок формы
- <2> - заголовок табличной части
- <A> - табличная часть формы с заголовками колонок и данными
- <B> - управление табличной частью, состоящее:
  - слева - стандартный набор действий, связанный с табличной частью

	управление отображением: таблица/сводная таблица
	фильтрация данных (пользовательские фильтры) • может быть назначено несколько фильтров
	настройка отображения, в т.ч. выводимые колонки
	количество записей • на результат влияет значение фильтра
	суммовой результат по числовой колонке • на результат влияет значение фильтра

	экспорт содержимого табличной части в Excel • имя созданного отчета lsfReport.xlsx • на результат влияет значение фильтра
	принудительное обновление табличной части • для текущего пользователя изменения, выполненные кем-либо, отобразятся через некоторое время. Можно ускорить обновление данных. Актуально в многопользовательском режиме

- справа - кнопки действий, определенные при проектировании формы (NEW, DELETE)

```
// 3. Назначаем форму для отображения и редактирования свойств класса
FORM viewReception 'Приходы' // название формы и заголовок
    OBJECTS r = Reception    // объекты, с которыми работает форма
    PROPERTIES (r) nd, dd, entity, name, quantity, price // список свойств
    PROPERTIES (r) NEW, DELETE // опции редактирования
;
```

- <C> - кнопки управления формой, где:
  - <Сохранить> - сохраняет все сделанные на форме изменения. Если просто выйти из формы, то сделанные изменения потеряются
  - <Отменить> - отменить сделанные изменения
  - <Ок> - в данном контексте сохранит изменения и закроет форму
  - <Закрыть> - завершит работу с формой
  - Кнопки с пиктограммами

	если версия клиента Desktop и с формой связана печатная форма (jrxml), то печатная форма будет открыта в JasperSoft Studio - редактор отчетов
	принудительное обновление всех табличных частей и свойств на форме

По окончании внесения данных не забываем нажать на кнопку "Сохранить" данные.

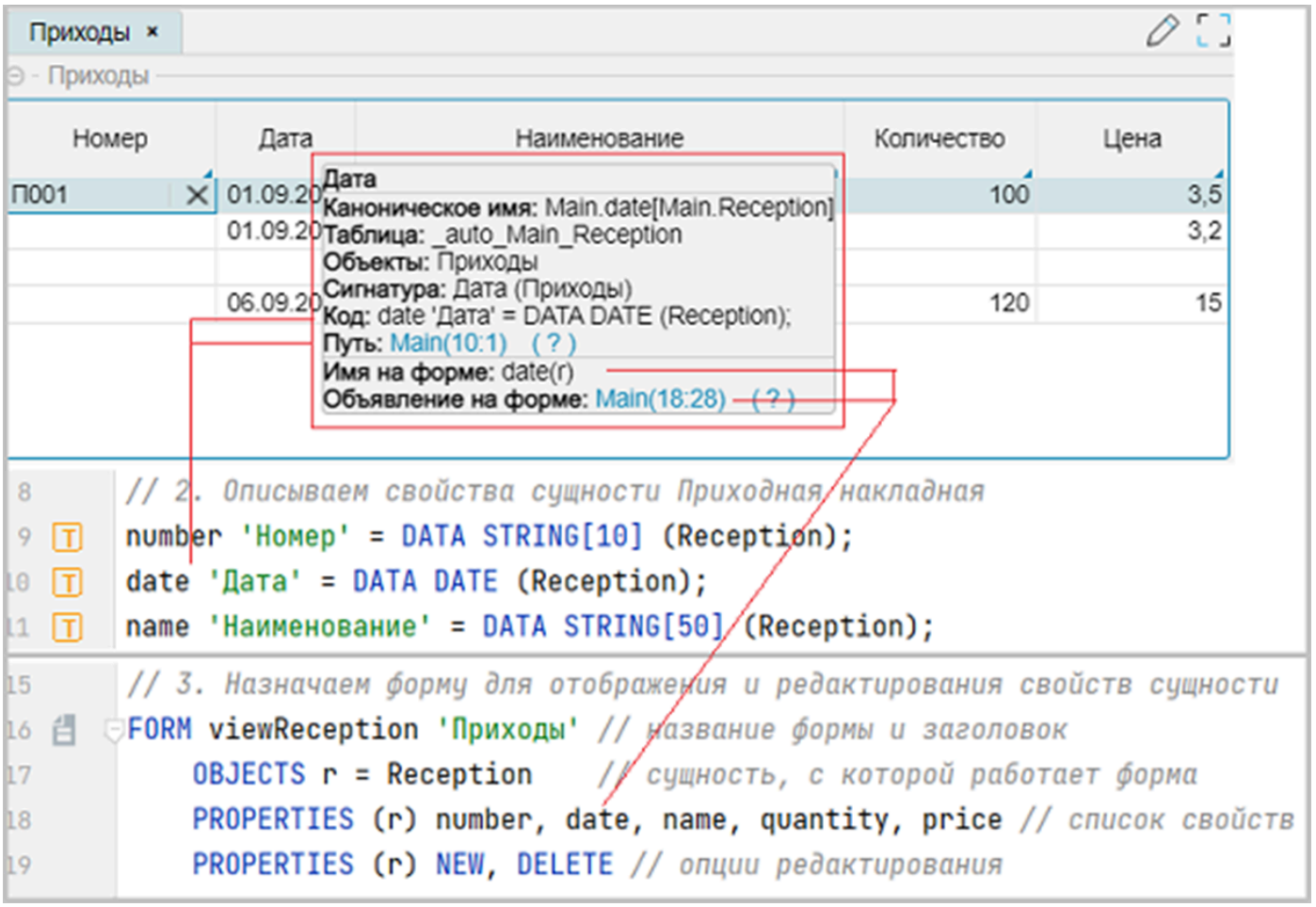
См. также
Документация. Интерактивное представление.

Пример кода:
examples\t202_интерефейс\src

Архив SQL:
examples\t202_интерефейс\sql

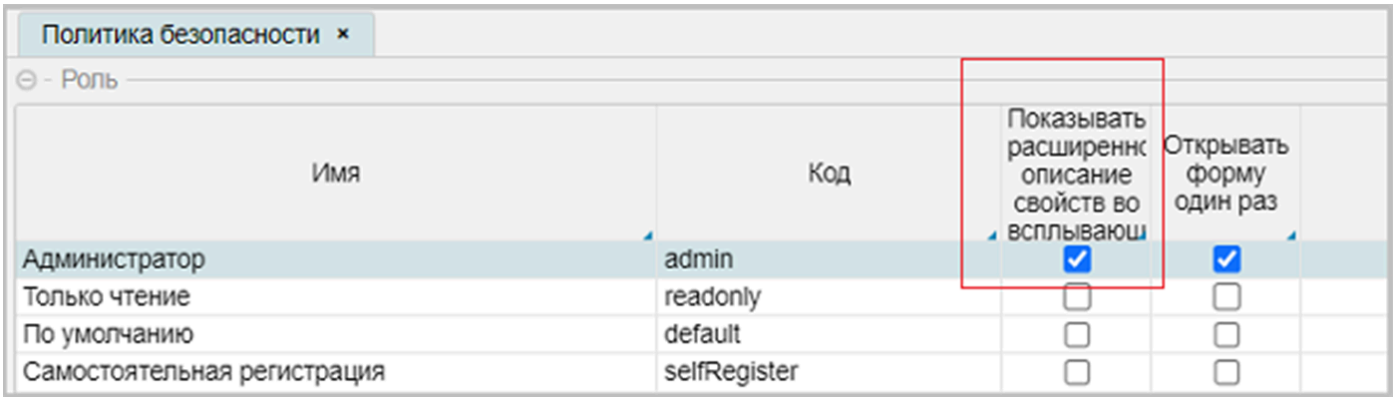
Интерфейс. Подсветка свойств

При наведении курсора мыши на объекты интерфейса - подписи колонок, кнопки, элементы форм - всплывают подсказки, связанные с информацией об элементе.

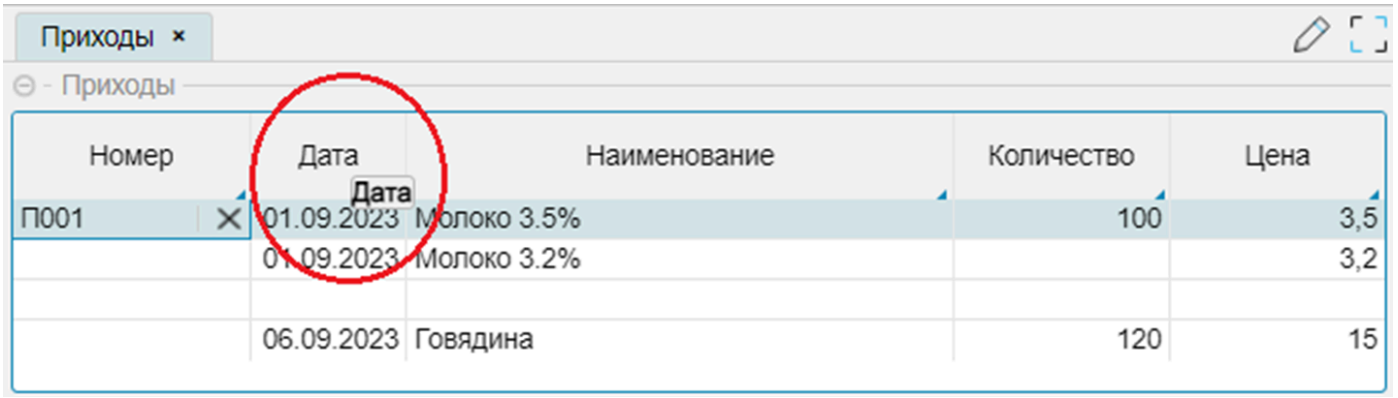


Можно узнать: модуль, который связан с элементом, тип свойства, место определения свойства и размещение на форме, а также другую менее востребованную информацию. Это очень удобно! Пока программный код приложения сосредоточен в одном модуле, это не особо актуально. Но когда приходится иметь дело с большим проектом, тысячами модулей, входящих в проект, – это актуально.

Такое поведение по умолчанию закреплено за ролью "Администратор".
Это удобно при разработке приложения, но не совсем удобно при его эксплуатации.
Поведение регулируется пунктом меню "Администрирование - Система - Доступ - Политика - Безопасности" и значением галки "Показывать расширенное описание свойств во всплывающих подсказках" для разных ролей пользователей.



Если галочка отключена, то вместо расширенного описания элемента будет подсвечиваться только его заголовок.



Примечание:

- изменения не требуют рестарта сервера приложений, а только перезапуска клиентского приложения

Новые свойства. Расширение форм

Необходимо выполнить доработку класса Reception:

- добавить новые свойства:
  - Пользователь - имя пользователя, под которым вносились данные, userName
  - Дата ввода - дата ввода данных пользователем, userDate
- и изменить форму для отображения новых свойств.

Все изменения выполним в конце модуля.

Добавляем новый свойства:

```
// Новые свойства
userName 'Пользователь' = DATA STRING[50] (Reception);
userDate 'Дата ввода' = DATA DATE (Reception);
```

Дорабатываем форму, используя механику "Расширения форм":

```
// Расширяем форму
EXTEND FORM viewReception
    PROPERTIES (r) READONLY userName, userDate
;
```

Примечание:

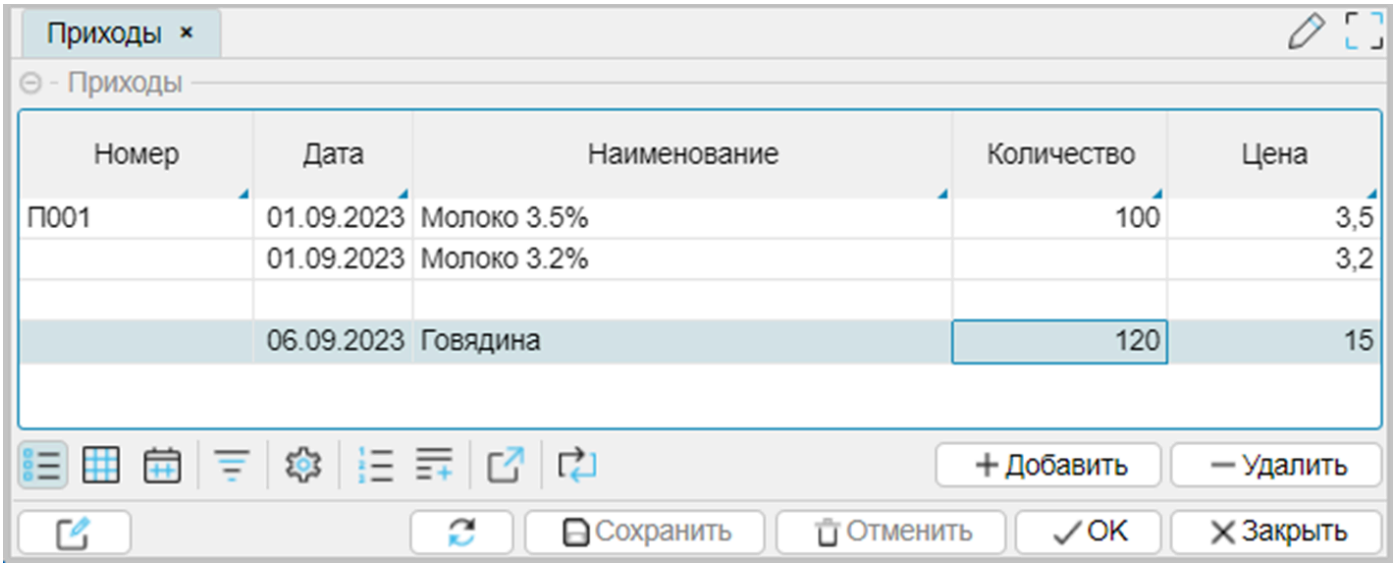
- новые свойства можно было внести там, где были описаны все свойства, а изменения для формы viewReception, можно было выполнить там, где эта форма была описана первый раз. Но такой подход бывает не всегда удобен или возможен в случае больших проектов. Создавать новые свойства или изменять формы можно в любом месте текущего модуля или других модулей. Логика такого поведения проста: *в какой-то момент возникла необходимость каких-то действий, и эти действия были описаны там, где они возникли*.
- ключевое слово READONLY запрещает редактирование свойств на форме. Предполагается, что в ходе последующей разработки приложения эти свойства будут заполнятся автоматически.

См. также
Документация, Расширение форм

Пример кода:
examples\t203_расширения_форм\src

Контроль ввода

Ввод данных осуществлялся намеренно некачественно, с пропусками записей или свойств.



Полагаться на оператора, и не принимать при этом никаких контролирующих действий - плохо. Результат всегда будет некачественный. Отсюда следует вывод, что необходим контроль вводимых данных.

Существует несколько способов решения задач контроля:

1. Запрет пустых значений
2. Проверка на уникальность вносимых значений
3. Создание ограничений
4. Использование простых событий
5. Использование событий форм

1. Запрет пустых значений

Это самое простое, что может быть.

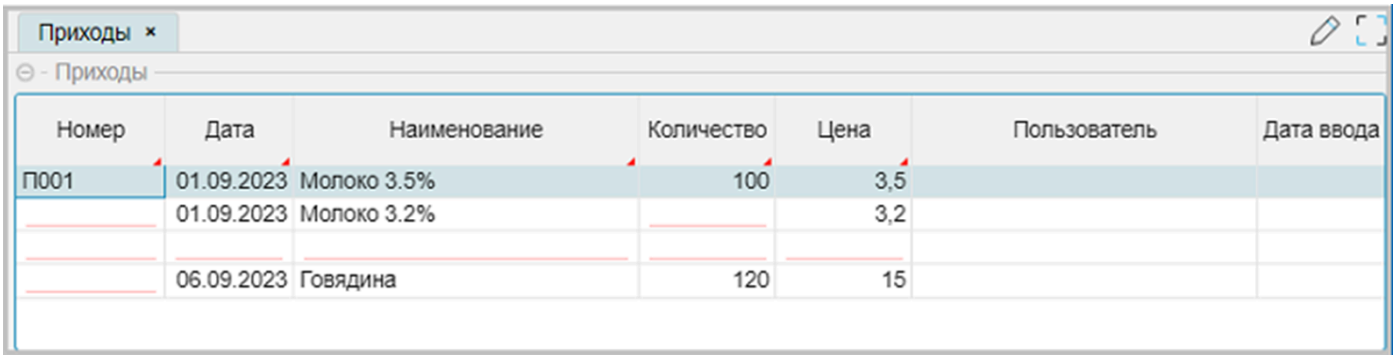
Доработаем все свойства, кроме userName, userDate, используя ключевое слово NONULL.

```
// 1. Запрещаем пустые значения
number 'Номер' = DATA STRING[10] (Reception) NONULL;
date 'Дата' = DATA DATE (Reception) NONULL;
organization 'Организация' = DATA STRING[50] (Reception) NONULL;
name 'Наименование' = DATA STRING[50] (Reception) NONULL;
quantity 'Количество' = DATA NUMERIC[10,3] (Reception) NONULL;
price 'Цена' = DATA NUMERIC[10,2] (Reception) NONULL;
```

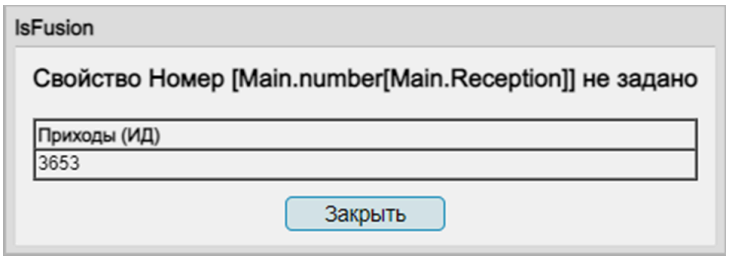
Примечание:

- планируется, что свойства userName, userDate будут заполнятся автоматически

Теперь после перезапуска сервера приложений форма приходной накладной будет выглядеть так:



Все свойства, которые обязательно должны быть заполнены, но они пусты - имеют красное подчеркивание. При попытке записать данные, для которых есть проверка на NULL и которые пусты, платформа не даст сохранить данные.



Примечание:

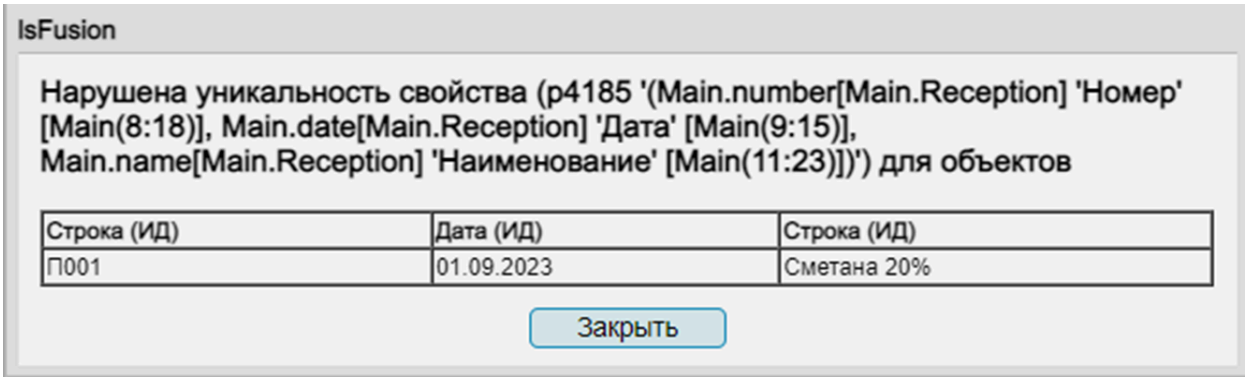
- При этом старые, ранее введенные и сохраненные значения, в процессе контроля не участвуют.

2. Проверка на уникальность вносимых значений

В контексте приходной накладной можно предположить, что уникальными являются совокупность свойств: Номер, Дата, Наименование. То есть рассуждаем, что для одной накладной (пара номер и дата) не может быть повторяющихся наименований товара:

```
// 2. Проверка на уникальность
unique = GROUP AGGR Reception o BY number(o), date(o), name(o);
```

Если уникальность при сохранении данных будет нарушена, то платформа не даст сохранить данные.



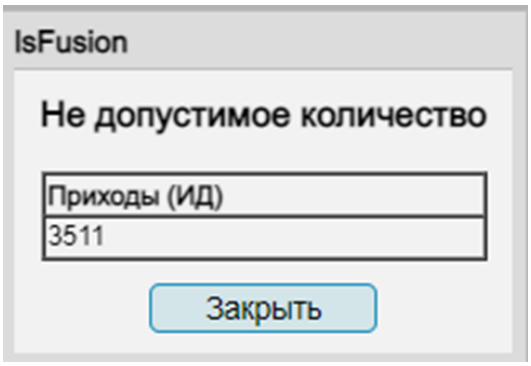
3. Ограничения

Ограничения задаются инструкцией CONSTRAINT.

С помощью ограничений мы можем проконтролировать правильность вносимых данных на соответствие их каким-либо критериям. Например, определим, что вносимое количество должно быть больше 0 и не больше 1000.

```
// 3. Ограничение
CONSTRAINT quantity(Reception o) > 1000 OR quantity(o) < = 0 MESSAGE 'Недопустимое количество';
```

Если соответствие будет нарушено, то платформа не даст сохранить данные.



4. Использование простых событий

Например, будем автоматически заполнять дату ввода накладной, то есть реализуем механизм автозаполнения:

```
// 4. Простые события
WHEN SET(Reception o IS Reception) AND NOT userDate(o) DO userDate(o) ← currentDate();
```

5. Использование событий форм

Например, будем заполнять свойство userName - пользователь, который внес данные:

```
// 5. Использование событий форм
EXTEND FORM viewReception
    EVENTS ON APPLY {
        IF NOT userName(r) THEN userName(r) ← STRING[50](currentUserName());
    }
;
```

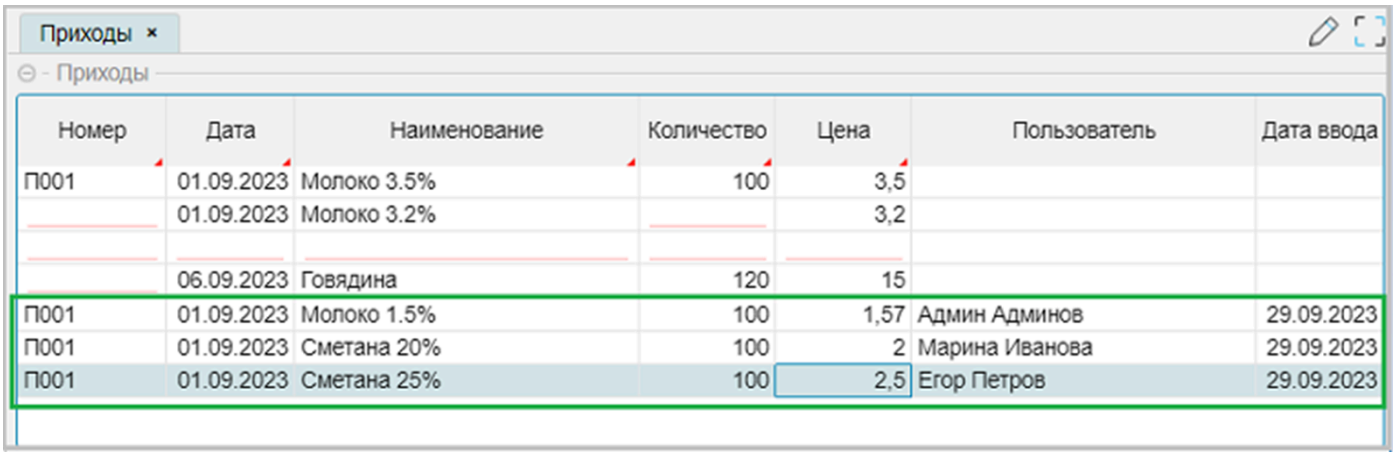
Событие сработает в самом начале транзакции после сохранения сессии в базу данных (APPLY).

Примечание (для п. 4, 5):

- в состав платформы включен ряд модулей, которые можно рассматривать как библиотечные, включающие в свой состав набор действий и свойств. Модули для работы со временем (Time) и пользователями (Authentication) необходимы для получения текущей даты (currentDate) и пользователя (currentUserName), работающего с данными. Для подключения других модулей в заголовок модуля нужно добавить инструкцию REQUIRE.

```
REQUIRE Time, Authentication;
```

При использовании контроля ввода данных вид вносимых данных будет принципиально другой - аккуратный и полный:



Номер	Дата	Наименование	Количество	Цена	Пользователь	Дата ввода
П001	01.09.2023	Молоко 3.5%	100	3,5		
	01.09.2023	Молоко 3.2%		3,2		
	06.09.2023	Говядина	120	15		
П001	01.09.2023	Молоко 1.5%	100	1,57	Админ Админов	29.09.2023
П001	01.09.2023	Сметана 20%	100	2	Марина Иванова	29.09.2023
П001	01.09.2023	Сметана 25%	100	2,5	Егор Петров	29.09.2023

Примечание:

- три последних записи намеренно введены под разными пользователями, чтобы продемонстрировать автоматическое заполнение свойств

См. также

Документация, Заголовки модуля
Документация, Оператор GROUP
Документация, Агрегации
Документация, оператор AGGR
Документация, Инструкция WHEN
Документация, Операторы изменений
Документация, Блок событий
Документация, Инструкция CONSTRAINT
Модуль Time

Пример кода:
examples\t204_контроль_ввода\src

Архив SQL:
examples\t204_контроль_ввода\sql

Форма редактирования

Данные можно вносить и в табличной части формы. Но это бывает не всегда удобно. Для этой цели больше подходит форма редактирования. Форма редактирования вызывается поверх формы отображения как модальная форма. При этом, как правило, свойства формы отображения выводятся с признаком READONLY, так как иначе теряется смысл в наличии еще одной формы.

Что надо сделать:

- Описать форму редактирования примерно так, как это было сделано для формы отображения.
- В описании формы редактирования использовать блок формы EDIT. **Важно!** Относительно блока формы EDIT платформа понимает, какую форму надо использовать для редактирования данных.
- Указать для формы редактирования сколько данных мы редактируем одновременно.
- По умолчанию считается, что любая форма отображает все данные в табличном виде, режим GRID. Если необходимо редактировать одну запись, то необходимо использовать опцию PANEL.
- Исправить форму отображения, добавив признак READONLY.
- При описании операторов редактирования формы для создания, редактирования и удаления объекта использовать операторы работы с объектами NEW, EDIT, DELETE с модификатором NEWSESSION, который означает, что действия будут производится в новой сессии данных. По завершению операции нажимать на кнопку "Сохранить" внизу формы отображения уже не надо. Соответственно, если не использовать NEWSESSION, то будет требоваться отдельное сохранение данных, как это было ранее.

Перекомпануем и исправим содержание модуля, то есть выполним небольшой рефакторинг.

```
MODULE Main;

REQUIRE Time, Authentication;

CLASS Reception 'Приходы';

number 'Номер' = DATA STRING[10] (Reception) NONULL;
date 'Дата' = DATA DATE (Reception) NONULL;
name 'Наименование' = DATA STRING[50] (Reception) NONULL;
quantity 'Количество' = DATA NUMERIC[10,3] (Reception) NONULL;
price 'Цена' = DATA NUMERIC[10,2] (Reception) NONULL;
userDate 'Дата ввода' = DATA DATE (Reception);
userName 'Пользователь' = DATA STRING[50] (Reception);

// контроль данных
unique = GROUP AGGR Reception o BY number(o), date(o), name(o);
CONSTRAINT quantity(Reception o) > 1000 OR quantity(o) < = 0 MESSAGE 'Недопустимое количество';
WHEN SET(Reception o IS Reception) AND NOT userDate(o) DO userDate(o) ← currentDate();
WHEN SET(Reception o IS Reception) AND NOT userName(o) DO userName(o) ← STRING[50](currentUserName());



FORM editReception 'Приход'
    OBJECTS r = Reception PANEL // опция panel - только одна запись
    PROPERTIES (r) number, date, name, quantity, price
    EDIT Reception OBJECT r
;
```

```
FORM viewReception 'Приходы'
    OBJECTS r = Reception // опция по умолчанию GRID
    PROPERTIES (r) READONLY number, date, name, quantity, price, userName, userDate
    PROPERTIES (r) NEWSESSION NEW, EDIT, DELETE
;


NAVIGATOR {
    NEW FOLDER move 'Движение' WINDOW toolbar FIRST { // Новый пункт горизонтального меню
        NEW FORM viewReception; // В качестве подпункта меню Заголовок формы
    }
}
```

Примечание:

- После перехода на форму редактирования изменился набор управляющих кнопок внизу формы. Вместо кнопок "Добавить" и "Удалить" появился другой набор: "Добавить", "Редактировать", "Удалить".

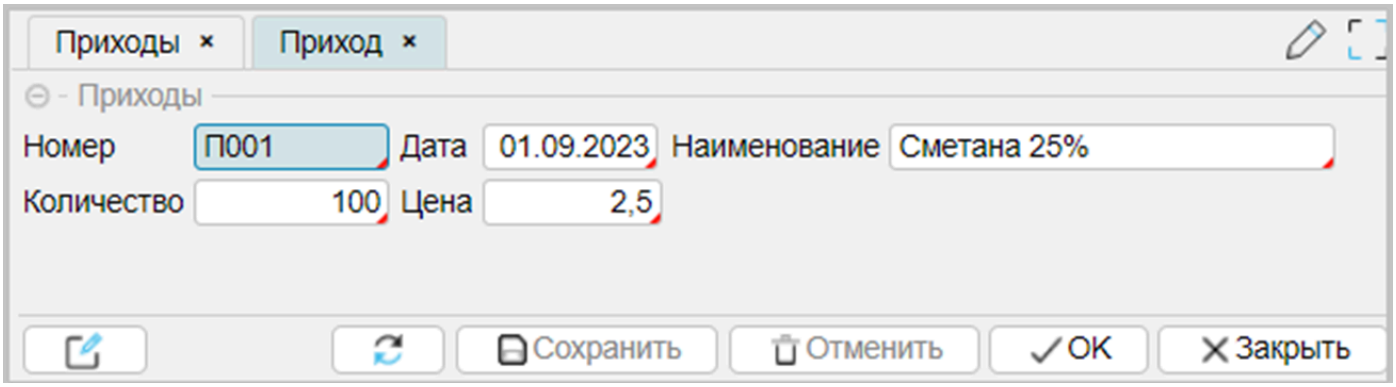
См. также

Документация, Опции свойства или действия
Документация. Инструкция FORM. Блоки форм, EDIT
Документация, Блоки объектов, Опции группы объектов

Пример кода:
examples\t205_форма_редактирования\src

Дизайн формы

После создания формы редактирования и ее отображения, форма имеет такой вид:



Это вид по умолчанию, который устанавливает автоматика формы: слева направо, сверху вниз на ширину формы в порядке определения свойств.

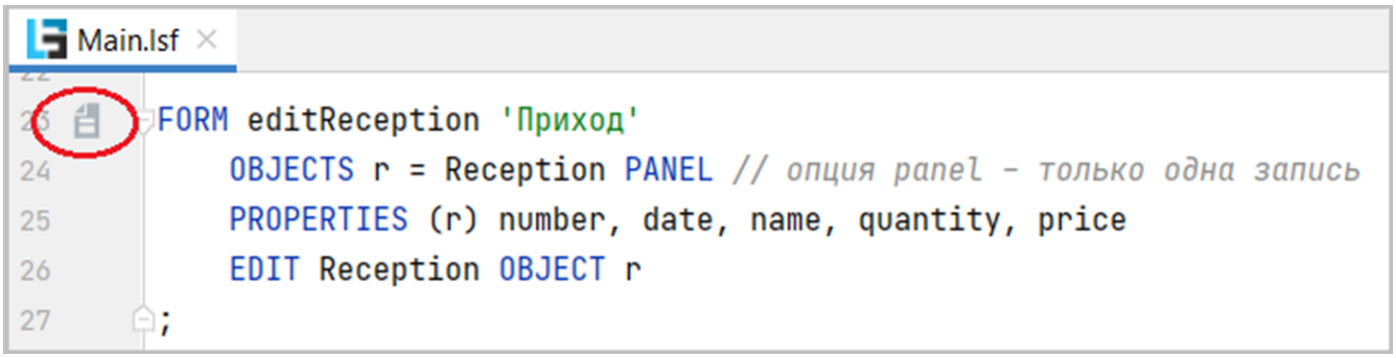
Но, предположим, это не устраивает заказчика, который хочет видеть вносимые данные более информативно, в виде отдельных блоков:

- Номер и Дата накладной, заголовок блока "Накладная"
- Наименование, количество, цена одним блоком с заголовком "Строки накладной"

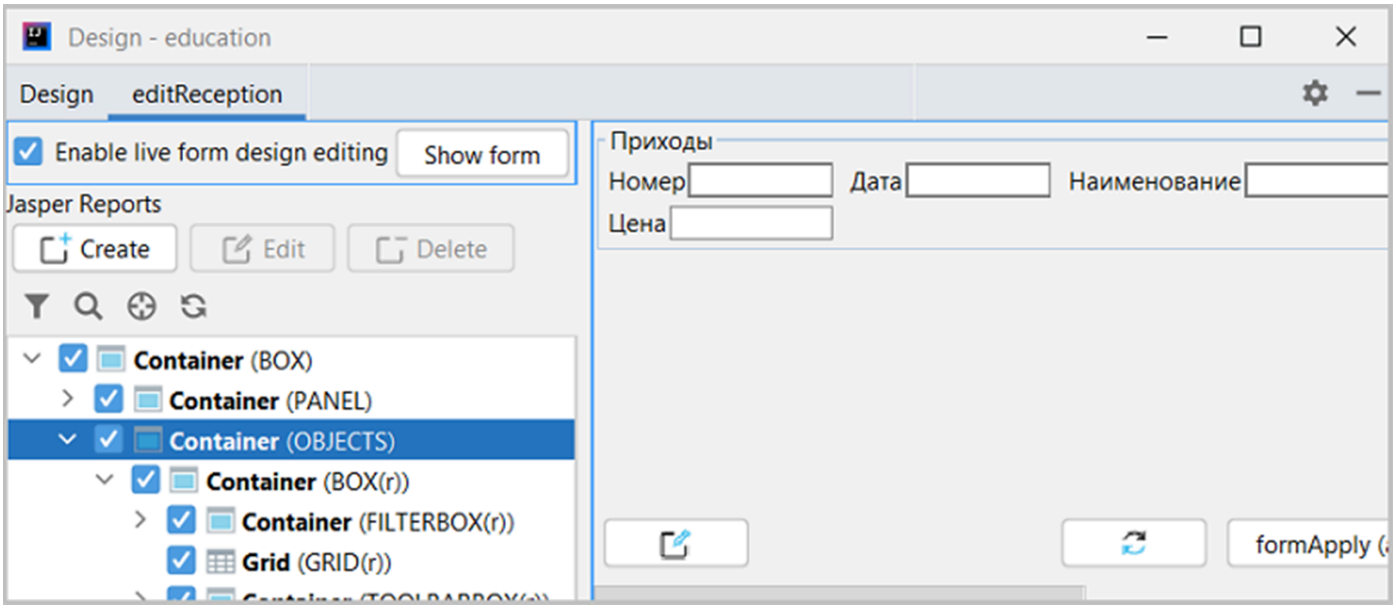
Для управления дизайном формы используется инструкция DESIGN, которая позволяет:

- добавлять контейнеры и размещать в них элементы
- перемещать и удалять элементы
- менять цвета и шрифты элементов

В редакторе на служебном поле, где отображается нумерация строк, высвечивается значок формы в виде серенького листика:



При нажатии на этот значок вызывается дизайнер формы.
Если посмотреть на содержание дизайнера формы, то следует обратить внимание на то, что форма состоит из набора функциональных контейнеров, которые расположены в определенном порядке.



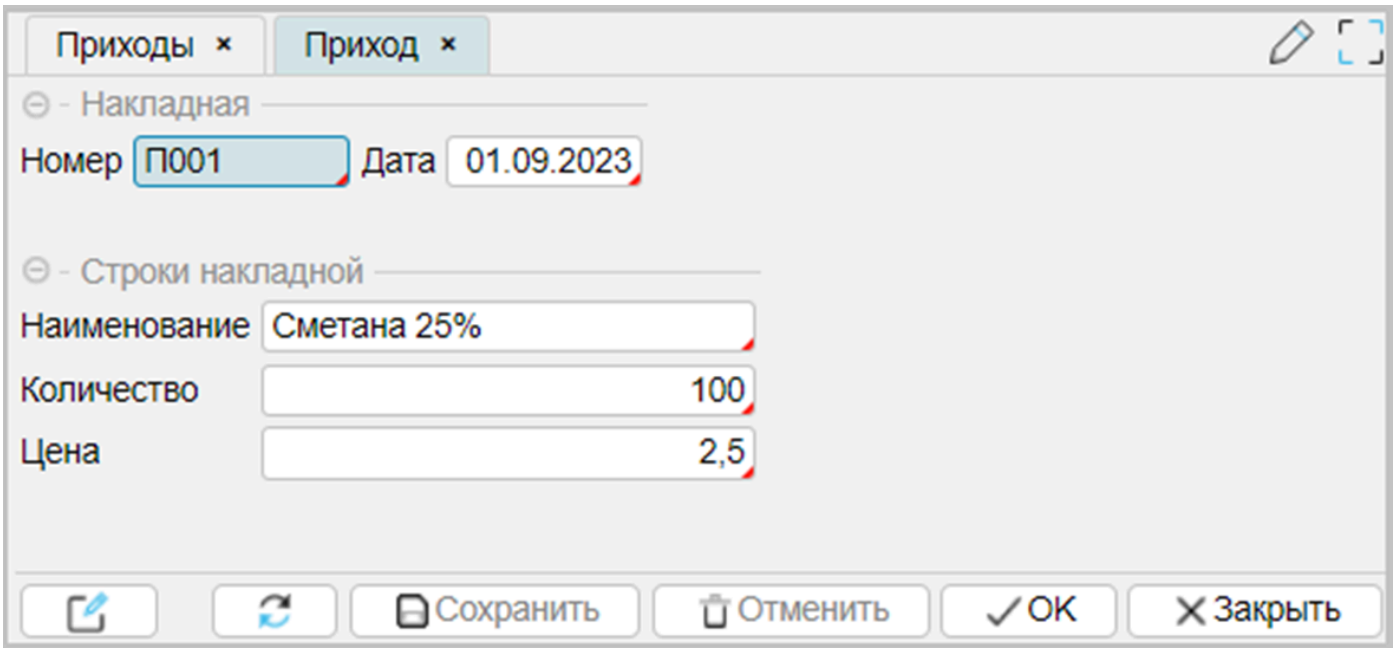
Используя определенные инструкции в DESIGN, можно:

- создавать новые контейнеры и перемещать туда элементы формы
- перемещать контейнеры
- управлять шрифтами и цветами элементов
- управлять отступами

Создадим новый дизайн формы внутри контейнера OBJECTS и опишем новый дизайн:

```
DESIGN editReception {
   OBJECTS {
       NEW cntDoc {
           horizontal = TRUE; // распределяем элементы слева направо и сверху вниз
           caption = 'Накладная';
           height = 60;
           MOVE PROPERTY (number(r));
           MOVE PROPERTY (date(r));
       }
       NEW cntRow {
           horizontal = FALSE ;
           caption = 'Строки накладной';
           MOVE PROPERTY (name(r));
           MOVE PROPERTY (quantity(r));
           MOVE PROPERTY (price(r));
       }
   }
}
```

После изменения дизайна и перезапуска сервера приложений, вид формы редактирования примет следующий вид, как показано на рисунке:



Описание программного кода:
Внутри контейнера OBJECTS создали 2 отдельных контейнера (*для выполнения пожелания заказчика*), каждый из которых будет отражать свою информацию. Дополнительно для cntDoc установили вертикальный размер height = 60px, это визуально отделило контейнеры на форме друг от друга.

См. также
Документация, Инструкция DESIGN

Пример кода:
examples\t206_дизайн_формы\src

Справочник организаций

Работа любого приложения не может обойтись без справочников.
Например, в рамках текущего приложения было бы неплохо связать накладные с поставщиком товара.

Что надо сделать:

- Создать класс Organization, Организации.
- Для класса Organization определить свойство name, Название.
- Для класса Organization создать форму отображения, форму редактирования, форму выбора и пункт меню.
- Для класса Reception задать свойство organization, класс свойства Organization.
- В накладной создать композицию с именем organizationName, для выражения названия организации.
- Для накладной доработать форму отображения и редактирования.

Примечание:

- все доработки будут выполнены в конце модуля, используя возможности платформы lsFusion вносить изменения в любом месте
- для доработки форм Reception активно используется расширение форм - EXTEND FORM

Доработки, связанные с Organization

```
// Новый класс Organization и его формы
//=============================================================

CLASS Organization 'Организации';
name 'Организация' = DATA STRING[50] (Organization) NONULL ;

FORM viewOrgaization 'Организации'
    OBJECTS o = Organization
    PROPERTIES (o) READONLY name
    PROPERTIES (o) NEWSESSION NEW, EDIT, DELETE
;

FORM editOrgaization 'Организация'
    OBJECTS o = Organization PANEL
    PROPERTIES (o) name
    EDIT Organization OBJECT o
;

// Эта форма будет выводится, как справочник
FORM listOrgaization 'Организации'
    OBJECTS o = Organization
    PROPERTIES (o) READONLY name
    LIST Organization OBJECT o
;

NAVIGATOR {
    move {
        NEW FOLDER directory 'Справочники' {
            NEW FORM viewOrgaization;
        }
    }
}
```

Особенности:

- Для формы listOrganization использование блока формы LIST превращает форму в форму выбора значения - типичный случай вызова формы справочника. Как и обычная форма, форма выбора значения может иметь любые блоки, в том числе, можно указать опции редактирования. В этом случае вызовется форма, для которой определен блок EDIT (*для данного примера – это форма editOrganization*).

Примечание:

- Справочник организаций, для дальнейшего изложения материала, должен быть заполнен так, как это указано в приводимой ниже таблице (как минимум, эти записи должны присутствовать):

Организация
ООО Молочные продукты
УП Комбинат хлебопродуктов
УП Мясокомбинат
Магазин 1
Магазин 2
Магазин 3

Доработки, связанные с Reception

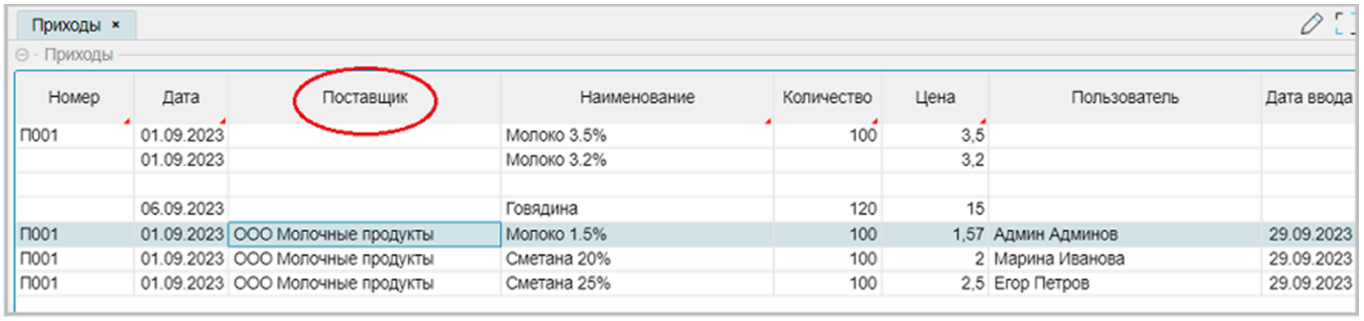
```
// доработки Reception
//=============================================================

// новое свойство - ссылка на организацию
organization = DATA Organization (Reception);

// композиция
organizationName 'Поставщик' (Reception o) = name(organization(o));

EXTEND FORM viewReception
   PROPERTIES (r) READONLY organizationName AFTER date(r)
;

EXTEND FORM editReception
   PROPERTIES (r) organizationName
;

DESIGN editReception {
   OBJECTS {
       NEW cntOrg AFTER cntDoc {
           horizontal = TRUE;
           caption = 'Поставщик';
           height = 60;
           MOVE PROPERTY (organizationName(r)) { caption = 'Фирма'; }
       }
   }
}
```

Особенности:

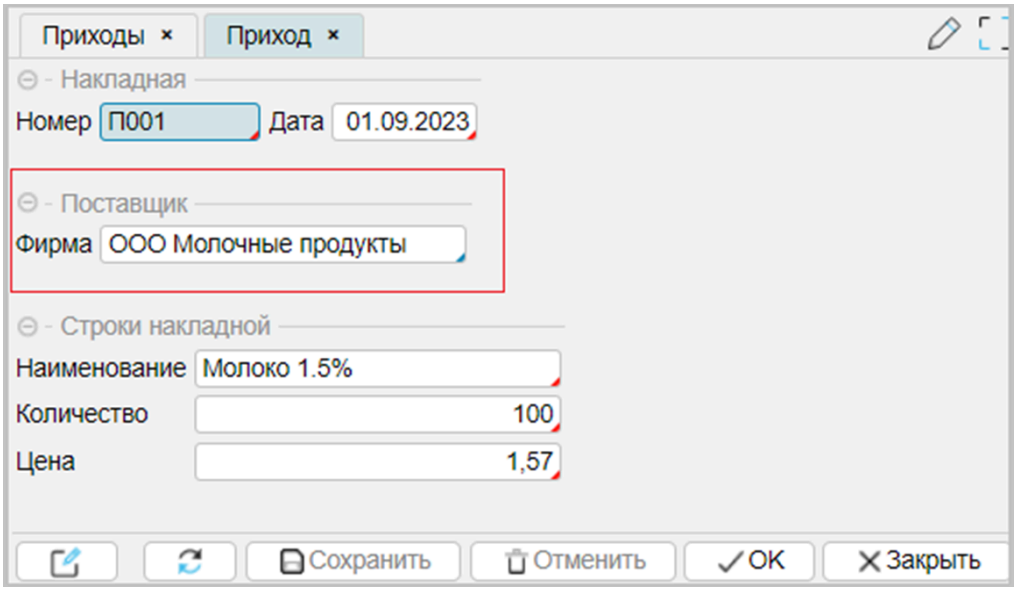
- Новое свойство organization для класса Reception, возвращающее объект класса Organization. Если идет обращение к свойству напрямую или через композицию, будет вызываться форма выбора, связанная с классом возвращаемого значения данного

свойства. При этом свойство получит ссылку на выбранное значение. Для текущего примера: форма выбора listOrgaization, определенная для выбора позиций среди объектов класса Organization.

- Использование композиции organizationName. Композиция – это выражение одних свойств через другие свойства.
- Использование EXTEND FORM с опцией AFTER для формы viewReception, с помощью которой новое свойство organizationName вставили в нужную позицию отображения между датой накладной и наименованием товара:



- Для формы редактирования editReception, чтобы добавить новое свойство organizationName, использовалась инструкция EXTEND FORM. Но для отображения нового свойства не использовалась опция AFTER, так как особого смысла в этом нет. Перемещение нового свойства organizationName выполнено через дизайн формы editReception, добавлением нового контейнера с опцией AFTER, которая вставила контейнер с заголовком "Поставщик" после ранее определенного контейнера cntDoc (Накладная). Кроме того, заголовок свойства organizationName "Поставщик" в дизайнере формы изменен на слово "Фирма".



См. также

Документация. Инструкция FORM. Блоки форм, LIST
Документация. Композиция

Пример кода:
examples\t207_справочник_организаций\src

Архив SQL:
examples\t207_справочник_организаций\sql

Статические объекты

После рассмотрения вопроса "Справочник организаций" появился вполне законный вопрос. В соответствии с постановкой задачи, можно выделить два вида организаций, с которыми работает оптовая база: поставщики (поставляют товар на базу) и покупатели (покупают товар у базы). Можно ли разделить для выбора поставщиков и покупателей так, чтобы при вызове формы выбора организаций при приходе высвечивалась одна группа организаций, а при расходе другая?

Так можно сделать, например, если определить для класса Organization свойство, которое бы отвечало за тип организации: Поставщик или Покупатель. А на форме выбора организаций установить фильтр, в зависимости от того, какая операция используется.

Для примера, текущий справочник организаций содержит 6 записей, где первые три – это поставщики товары, а последние три - покупатели:

Организация
ООО Молочные продукты
УП Комбинат хлебопродуктов
УП Мясокомбинат
Магазин 1
Магазин 2
Магазин 3

В этом случае можно создать класс OrganizationType, отвечающий за тип организации, с двумя статическими объектами, отражающими соответствующий тип: supplier (поставщик) и customer (покупатель).

Добавим доработки в конец модуля:

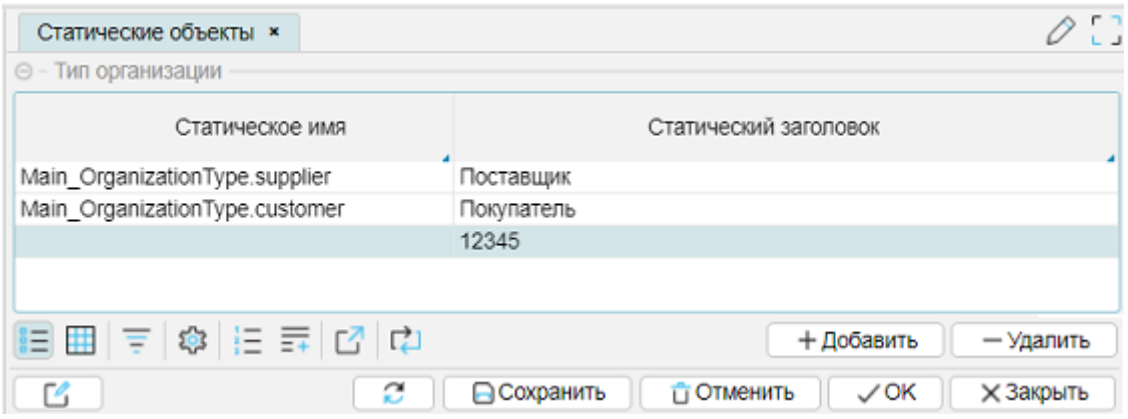
```
// доработки для использования статических объектов
CLASS OrganizationType 'Тип организации' {
   supplier 'Поставщик', customer 'Покупатель'
}

organizationType 'ID' = DATA OrganizationType (Organization) NONULL;
organizationTypeName 'Тип организации' (Organization o) = staticCaption(organizationType(o));

EXTEND FORM viewOrgaization
   PROPERTIES (o) READONLY organizationTypeName
;

EXTEND FORM editOrgaization
   PROPERTIES (o) organizationTypeName
;

CONSTRAINT SETCHANGED(organization(Reception o)) AND
               NOT organizationType(organization(o)) = OrganizationType.supplier
               CHECKED BY organization[Reception]
               MESSAGE 'Для приходной накладной организация должна быть поставщиком';
```

Примечание:

- Класс OrganizationType включает в себя два статических объекта: supplier и customer. Статические объекты характеризуются тем, что после старта сервера приложений они будут автоматически записаны в базу данных. Причем каждая запись (в данном случае их две, по количеству объектов) будет характеризоваться двумя системными свойствами staticName и staticCaption. Если на вход системного свойства подать соответствующий объект, то можно получить полное наименование объекта (например: *Main_OrganizationType.supplier*) или его заголовок (например: *Поставщик*). Можно исследовать содержание класса OrganizationType, содержащего статические объекты supplier и customer, добавив небольшой программный код в конец модуля для примера (*в дальнейшем использоваться не будет*). Код создаст пункт в навигаторе и отобразит форму со статическими объектами:

```
FORM static 'Статические объекты'
   OBJECTS o = OrganizationType
   PROPERTIES (o) staticName, staticCaption
   PROPERTIES (o) NEW , DELETE
;

NAVIGATOR {
   directory {
       NEW FORM static;
   }
}
```

После старта сервера приложений получим форму:

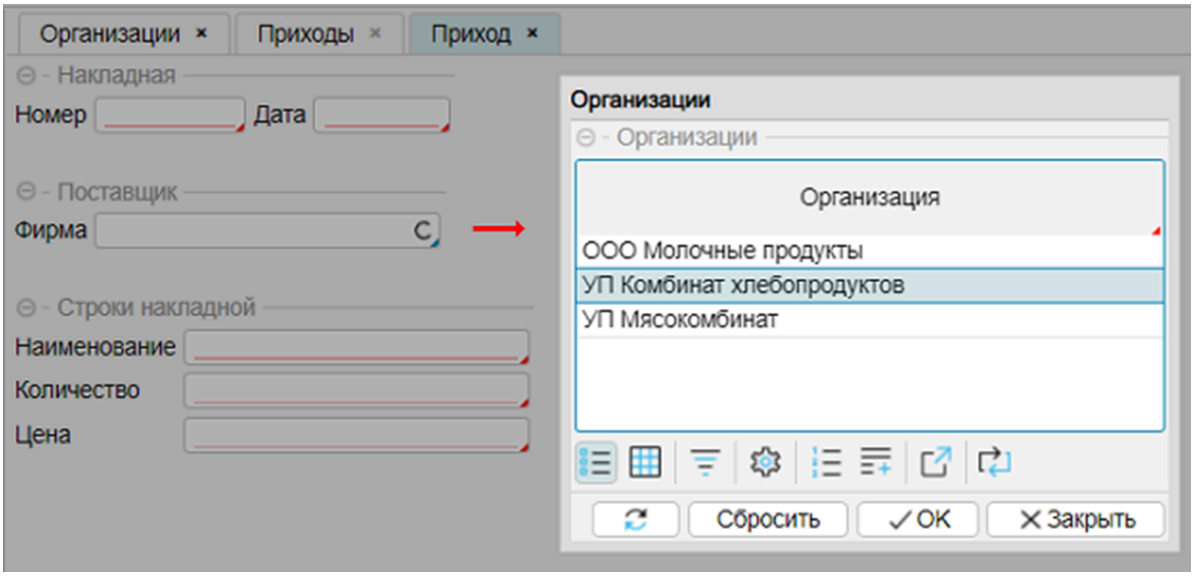


Можно добавлять новые объекты (типы организаций), редактировать или удалять их, *но изменить значения системных свойств* статических объектов, определенных в коде lsFusion, нельзя. Само по себе использование статических объектов удобно для создания справочников с предопределенными значениями, например: дни недели, название месяцев, пол человека, ну и так далее. При этом специально вносить информацию для статических объектов не нужно - они всегда существуют на момент старта сервера приложений. Так как статические объекты автоматически создаются в базе данных, то соответственно, отпадает необходимость в создании дополнительных интерфейсов по их отображению и редактированию, если в этом отсутствует прямая необходимость.

- Работа с классом OrganizationType, использующим статические объекты, строится как с обычным справочником: создаем свойство organizationType и композицию

organizationTypeName, которая выразит выбранное значение. Используя EXTEND FORM, дорабатываем формы отображения и редактирования справочника Организации.

- Наиболее интересным в описании представляется использование ограничения CONSTRAINT, которое работает как входной фильтр для формы выбора listOrganization благодаря использованию опции CHECKED. Если не использовать CHECKED, то CONSTRAINT сработает стандартно - при попытке записать организацию, не являющуюся поставщиком товара, выскочит сообщение MESSAGE, указывающие на нарушение условий ограничения. Если будет использована опция CHECKED, то форма выбора будет отображать только организации поставщиков (фильтр), поэтому физически будет невозможно выбрать что-то другое.



Аналогично, при редактировании расходной накладной, *которая будет создана позже*, должны будут высвечиваться только организации с признаком 'Покупатель', для чего по аналогии с приходами, будет создано ограничение, связанное с покупателями.

См. также

Документация, Классы
Документация, Статические объекты
Документация, Локальные первичные свойства
Документация. Инструкция FORM. Блоки формы
Документация. Блоки фильтров и сортировок. Блок фиксированных фильтров

Пример кода:
examples\t208_статический_объект\src

Архив SQL:
examples\t208_статический_объект\sql

Анализ приложения

Приходная накладная создана.
Следующая задача: создать расходную накладную.
По аналогии с приходной накладной действия представляются вполне очевидными:

- создать новый класс с набором свойств, связанный с расходами
- создать новые формы отображения и редактирования
- создать новый элемент навигатора, для вызова формы отображения расходов

Такой план может иметь место, но это не конструктивное решение:

- если все создаваемые элементы программного кода "складировать" в один модуль, это ухудшит навигацию по проекту и читаемость программного кода. Правильно будет, если программный код разбить на функциональный модули, каждый из которых отвечает за свою часть. Это упрощает навигацию и доработки проекта в будущем
- второй недостаток: упускается важное свойство платформы lsFusion - наследование классов, которое значительно может упростить программный код

Поэтому, перед созданием расходной накладной, необходимо рассмотреть два вопроса:

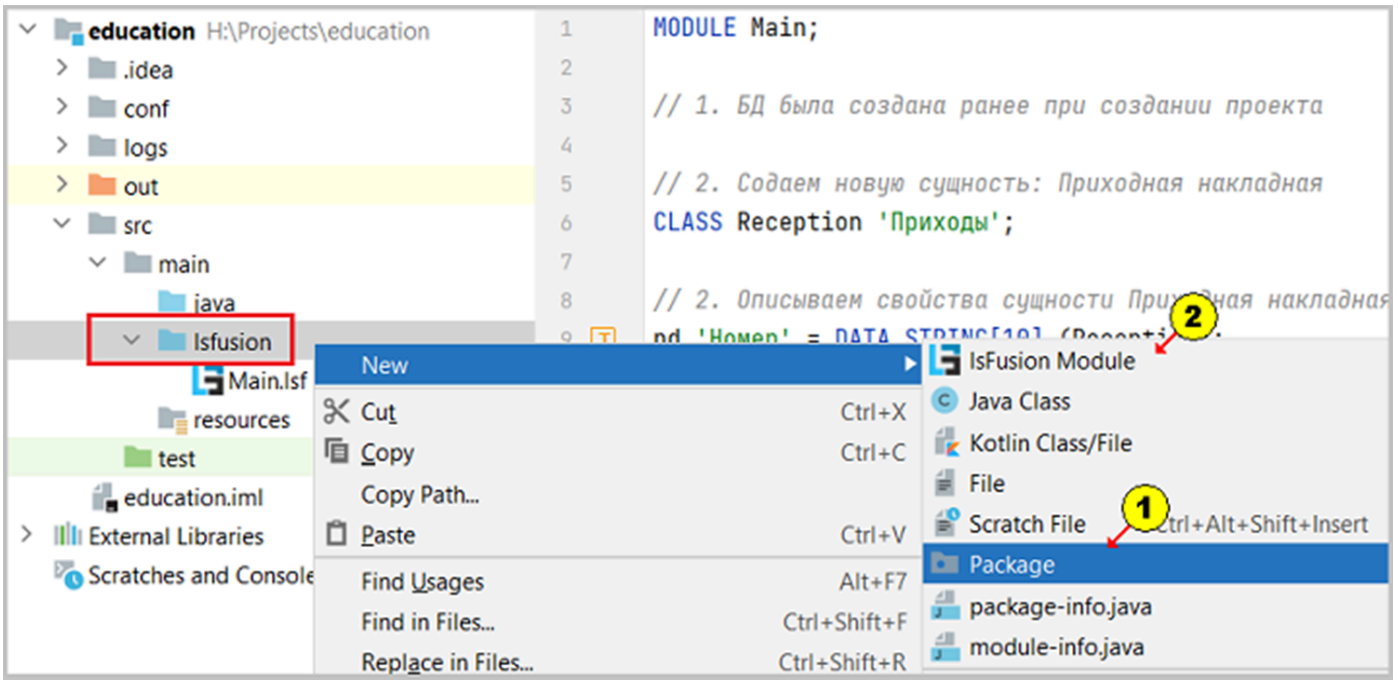
1. Модульность
2. Наследование

Модульность

Анализ предыдущих действий подсказывает, что присутствуют технологическая проблема размещения программного кода. Модуль Main содержит очень много функциональных частей. Если продолжать "запихивать" все подряд в модуль по мере изучения платформы, это сделает содержание модуля Main мало информативным, плохо воспринимаемым и плохо масштабируемым. Для решения этой задачи необходимо разбить существующий проект на отдельные папки (Package в терминах IDEA) и функциональные модули. Должна получиться следующая древовидная структура:

- Package lsfusion - корень проекта
  - модуль AppMenu - описание разделов меню
- Package documents - папка для модулей приходных и расходных документов
  - модуль Reception
- Package references - папка справочников
  - модуль Organization

Для создания нового Package или модуля надо стать на соответствующий пункт в дереве проекта и нажать на правую кнопку мыши, выбрать верхний пункт "New" и в следующем подменю выбрать один из 2- пунктов: lsFusion Module (2) или Package (1), как показано на рисунке:



На экране появится окно для внесения имени модуля или папки, в которое впишем название папки или модуля.

Содержимое для наполнения модулей возьмем из модуля Main. Все используемые ранее отдельные EXTEND FORM, DESIGN объедим на уровне форм или общего участка дизайна. По окончании рефакторинга модуль Main очистим от содержимого.

Первоначально все участки программного кода "видели" друг друга. Но после разбиения приложения на функциональные модули, свойства, пункты меню, выражения потерялись друг для друга. Для того, чтобы модули могли видеть друг друга, используется инструкция REQUIRE в заголовке модуля (см. Reception, Organization).

Содержание модуля AppMenu:

```
MODULE AppMenu;

NAVIGATOR {
   NEW FOLDER move 'Движение' WINDOW toolbar FIRST {
       NEW FOLDER documents 'Документы движения';
       NEW FOLDER references 'Справочники';
   }
}
```

Содержание модуля Reception:

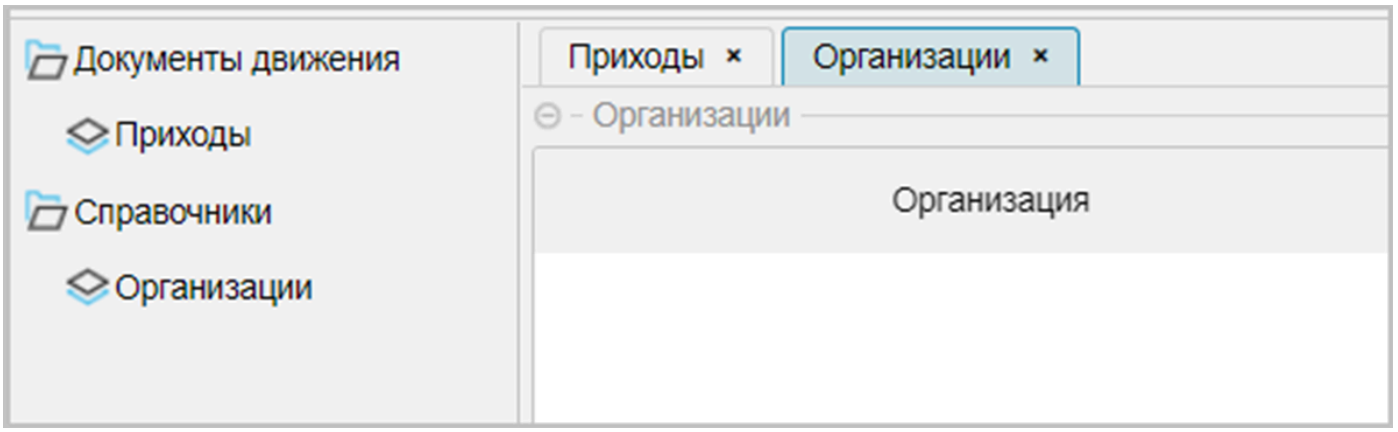
```
MODULE Reception;

REQUIRE Time, Authentication, AppMenu, Organization;

CLASS Reception 'Приходы';

number 'Номер' = DATA STRING[10] (Reception) NONULL;
date 'Дата' = DATA DATE (Reception) NONULL;
name 'Наименование' = DATA STRING[50] (Reception) NONULL;
quantity 'Количество' = DATA NUMERIC[10,3] (Reception) NONULL;
price 'Цена' = DATA NUMERIC[10,2] (Reception) NONULL;
userDate 'Дата ввода' = DATA DATE (Reception);
userName 'Пользователь' = DATA STRING[50] (Reception);

// контроль данных
unique = GROUP AGGR Reception o BY number(o), date(o), name(o);
CONSTRAINT quantity(Reception o) > 1000 OR quantity(o) ⩽ 0 MESSAGE 'Недопустимое количество';

WHEN SET(Reception o IS Reception) AND NOT userDate(o) DO userDate(o) ← currentDate();
WHEN SET(Reception o IS Reception) AND NOT userName(o) DO userName(o) ←
STRING[50](name(currentUser()));

organization = DATA Organization (Reception);
organizationName 'Поставщик' (Reception o) = name(organization(o));

CONSTRAINT SETCHANGED(organization(Reception o)) AND
    NOT organizationType(organization(o)) = OrganizationType.supplier
    CHECKED BY organization[Reception]
    MESSAGE 'Для приходной накладной организация должна быть поставщиком';

FORM editReception 'Приход'
   OBJECTS r = Reception PANEL // опция panel - только одна запись
   PROPERTIES (r) number, date, organizationName, name, quantity, price
   EDIT Reception OBJECT r
;

DESIGN editReception {
   OBJECTS {
       NEW cntDoc {
           horizontal = TRUE; // распределяем элементы слева направо и сверху вниз
           caption = 'Накладная';
           height = 60;
           MOVE PROPERTY (number(r));
           MOVE PROPERTY (date(r));
       }
       NEW cntRow {
           horizontal = FALSE ;
           caption = 'Строки накладной';
```

```
            MOVE PROPERTY (name(r));
            MOVE PROPERTY (quantity(r));
            MOVE PROPERTY (price(r));
        }
    }
}

FORM viewReception 'Приходы'
    OBJECTS r = Reception // опция по умолчанию GRID
    PROPERTIES (r) READONLY number, date, organizationName, name,
                            quantity, price, userName, userDate
    PROPERTIES (r) NEWSESSION NEW, EDIT, DELETE
;


NAVIGATOR {
    documents {
        NEW FORM viewReception;
    }
}
```

Содержание модуля Organization:

```
MODULE Organization;

REQUIRE AppMenu;

CLASS OrganizationType 'Тип организации' {
    supplier 'Поставщик', customer 'Покупатель'
}

CLASS Organization 'Организации';
name 'Организация' = DATA STRING[50] (Organization) NONULL ;

organizationType 'ID' = DATA OrganizationType (Organization) NONULL;
organizationTypeName 'Тип организации' (Organization o) = staticCaption(organizationType(o));

FORM viewOrgaization 'Организации'
    OBJECTS o = Organization
    PROPERTIES (o) READONLY name, organizationTypeName
    PROPERTIES (o) NEWSESSION NEW, EDIT, DELETE
;

FORM editOrgaization 'Организация'
    OBJECTS o = Organization PANEL
    PROPERTIES (o) name, organizationTypeName
    EDIT Organization OBJECT o
;

FORM listOrgaization 'Организации'
    OBJECTS o = Organization
    PROPERTIES (o) READONLY name
    LIST Organization OBJECT o
;

NAVIGATOR {
    move {
        NEW FOLDER directory 'Справочники' {
            NEW FORM viewOrgaization;
        }
    }
}
```

Визуально содержание модулей должно стать проще, вместо "каши", которая была до этого. Выполним запуск сервера приложений и посмотрим, что получилось.



Приложение интерфейсно выглядит, как и планировали.
Но!!! пропали все данные, как в приходах, так и из справочника организаций.

Это важно!
Почему пропали данные? Напомним еще раз, что программный код влияет на структуру базы данных: он может создавать новые классы и свойства или удалять неиспользуемые. Это сильная сторона платформы IsFusion, так как исключает дополнительное администрирование сервера баз данных, но требует внимательности.

Пространство имен для приходов и справочник организаций ранее было Main, а в связи с рефакторингом приложения, пространство имен для приходов стало Reception, а для справочника организаций Organization. Соответственно, изменилась физическая структура базы данных SQL сервера. Но как быть, если данные должны быть сохранены и при этом необходимо выполнить рефакторинг программного кода?

Есть несколько способов и рекомендаций:

- использовать инструкцию NAMESPACE при создании новых модулей, описывающих классы, или рефакторинге
- изучить миграцию данных на платформе
- регулярно создавать архивы базы данных
- изначально продумывать архитектуру приложения
- экспортировать и затем импортировать данные в новую структуру

Когда формируются имена таблиц и полей на физическом уровне, если не указана инструкция NAMESPACE, то за основу берется имя модуля. Поэтому для инструкции NAMESPACE нам надо указать имя модуля, где были первоначально описаны классы – это модуль Main и восстановить данные.

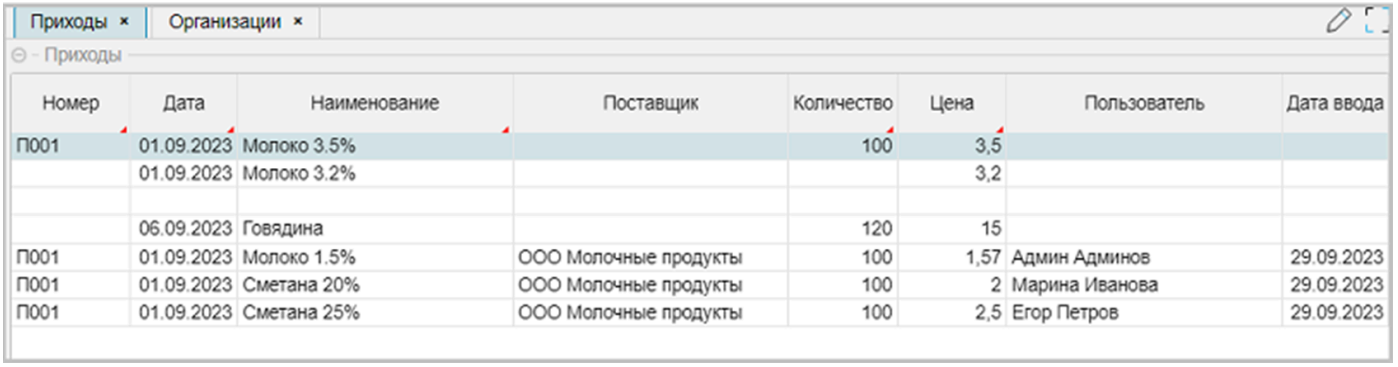
Заголовки модуля Reception:

```
MODULE Reception;
REQUIRE Time, Authentication, AppMenu, Organization;

NAMESPACE Main;
.........................................
```

Заголовки модуля Organization:

```
MODULE Organization;
REQUIRE AppMenu;

NAMESPACE Main;
.........................................
```

Остановим сервер приложений и восстановим базу данных из архива SQL (см. Examples\t208_статический_объект\sql). Выполним старт сервера приложений, и если посмотреть на приходы или организации, то данные будут восстановлены в первоначальном виде:


Приходы × Организации ×

Приходы

Номер	Дата	Наименование	Поставщик	Количество	Цена	Пользователь	Дата ввода
П001	01.09.2023	Молоко 3.5%		100	3,5		
	01.09.2023	Молоко 3.2%			3,2		
	06.09.2023	Говядина		120	15		
П001	01.09.2023	Молоко 1.5%	ООО Молочные продукты	100	1,57	Админ Админов	29.09.2023
П001	01.09.2023	Сметана 20%	ООО Молочные продукты	100	2	Марина Иванова	29.09.2023
П001	01.09.2023	Сметана 25%	ООО Молочные продукты	100	2,5	Егор Петров	29.09.2023

Примечание:

- В дальнейшем не будем "тянуть" пространство имен Main ради нескольких, непонятно как внесенных данных, поэтому исходный код модулей возвращаем без использования инструкции NAMESPACE.
- Отдельно заполним справочник организаций (внесем 6 записей), так как эти данные впоследствии будут нужны.
- При переносе дизайна формы редактирования исключили контейнер cntOrg, поэтому выбор организации будет на форме редактирования первым, далее пойдут контейнер номера и даты документа и контейнер наименования товара, количество, цена.

См. также

Документация, Заголовок модуля
Документация, Модульность
Документация, Именования
Документация, Миграция

Пример кода:
examples\t209_модульность\src

Архив SQL:
examples\t209_модульность\sql

Наследование

Сильная сторона платформы IsFusion – это возможность использования наследования.

Для рассматриваемой задачи, в которой присутствуют приходные и расходные документы, нужно создать родительский класс для всех создаваемых позже документов. На уровне родительского класса нужно создать такой набор свойств, ограничений и действий, который будет справедлив на уровне всех последующих создаваемых документов.

Стратегия построения:
Из ранее созданного модуля Reception выделяем общую часть, которая может быть использована для приходного и расходного документа со своими контролями, ограничениями, автозаполнением реквизитов. Классы приходного и расходного документа будут наследоваться от базового класса, соответственно, они получат все характеристики родительского класса. Дополнительно должен быть разработан механизм контроля списания, в соответствии с постановкой задачи: нельзя списать больше, чем поступило по приходу. Из всего пройденного ранее материала, нет ничего нового, кроме механизма наследования классов: создание классов и свойств, композиции, формы отображения и редактирования, дизайн, формы выбора, ограничения и события.

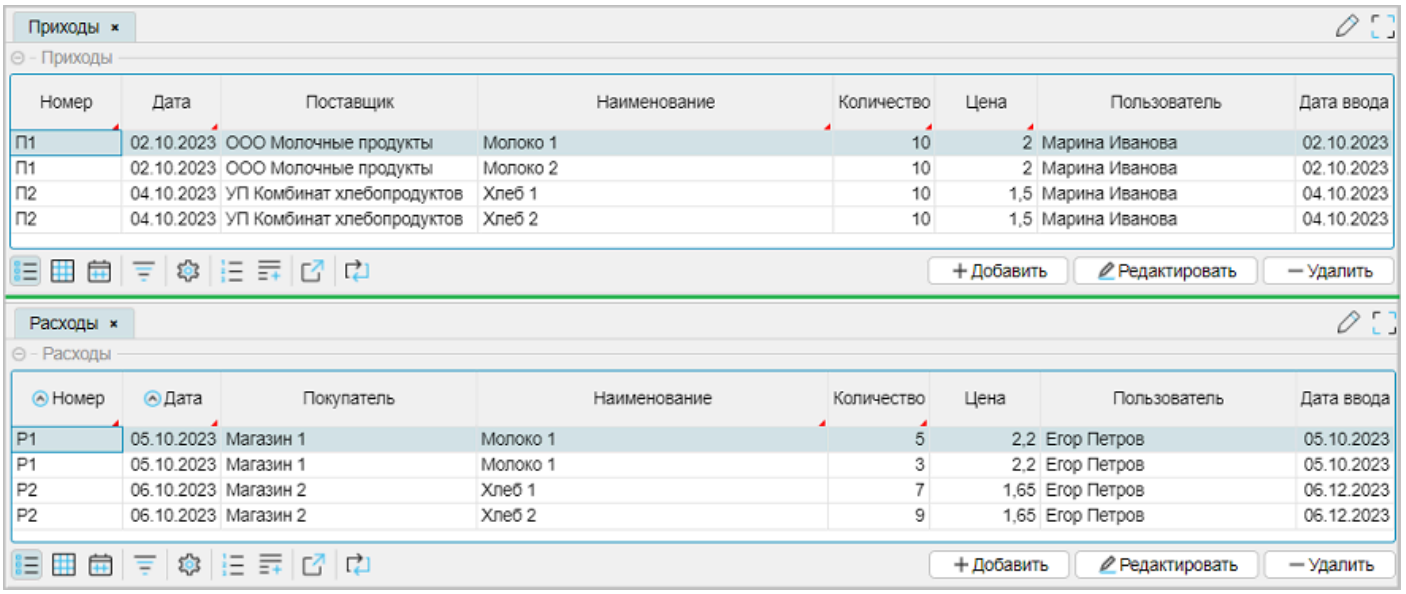
Что необходимо сделать:

1. Создать родительский класс, Document
2. Создать дочерний класс приходов, Reception
3. Создать дочерний класс расходов, Expenses

Примечание:

- Нумерация в комментариях, создаваемых блоков кода в модулях Reception и Expenses, и в описании выбрана намеренно одинаковая, чтобы подчеркнуть схожесть программного кода, унаследованного от общего класса.

После создания родительского класса и классов потомков выполним старт сервера приложений. Внесем данные по приходам и расходам, проконтролируем работу приложения:



Общий итог:

Можно считать, что постановка задачи выполнена полностью:

- присутствует интерфейс создания приходных документов
- присутствует интерфейс создания расходных документов
- создаваемые документы связаны с организациями
- присутствует фильтр для организаций в приходах и расходах
- присутствует контроль уникальных документов в приходах
- присутствует контроль на недопустимое количество в приходах
- присутствует контроль списания в расходах

См. также

Документация, Классы
Документация, Пользовательские классы

Пример кода:
examples\t210_наследование\src

Архив SQL:
examples\t210_наследование\sql

Родительский класс Document

Создадим в папке "Package documents" новый модуль с именем "Documents", в котором создадим *общий* класс с именем Document, который будет родительским для всех последующих документов: приходных и расходных. Создаваемый класс будет характеризоваться набором следующих свойств, часть которых была определена в постановке задачи:

- номер документа - number
- дата - date
- количество - quantity
- пользователь - userName
- дата ввода - userDate

Примечание:

- Общий класс будет являться абстрактным классом (CLASS ABSTRACT). Абстрактный класс предоставляет возможность наследовать от него другие классы, при этом часть общей логики работы для всех классов можно описать на уровне абстрактного класса. Потомки абстрактного класса наследуют логику и свойства, которые для каждого потомка отдельно описывать не надо. При этом каждый потомок может обладать дополнительными свойствами, которые отсутствуют у родительского класса. Основное отличие абстрактного класса заключается в том, что создание объектов (экземпляров) этого абстрактного класса невозможно.
- Из базового класса исключены композиция organizationName (название организации) и CONSTRAINT, отвечающий за фильтрацию при выборе поставщиков в приходах и покупателей в расходах, так как это свойства разной природы.
- Из базового класса исключены свойства name (наименование товара) и price (цена товара), так как для приходного документа эти реквизиты вносятся вручную, а для расходного документа являются отражением строки прихода, с которой списывается нужное количество. То есть по своему характеру свойства name и price для приходного и расходного документа – это свойства разной природы и для разных типов документов будут определяются по-разному.
- Для свойства number, date, quantity определим, что они должны редактироваться. При этом должно действовать ограничение, что редактируемые свойства не могут быть пустыми (ограничение на NULL).
- Для внесения количества будет действует ограничение не больше 1000 и не меньше нуля.
- Для свойств userName, userDate будем исходить из условия, что их наполнение будет формироваться автоматически.

```
MODULE Documents;

REQUIRE Time, Authentication, Organization;

CLASS ABSTRACT Document 'Документ';

number 'Номер' = DATA STRING[10] (Document) NONULL;
date 'Дата' = DATA DATE (Document) NONULL;
quantity 'Количество' = DATA NUMERIC[10,3] (Document) NONULL;
userDate 'Дата ввода' = DATA DATE (Document);
userName 'Пользователь' = DATA STRING[50] (Document);

// контроль данных
CONSTRAINT quantity(Document o) > 1000 OR quantity(o) ⩽ 0 MESSAGE 'Недопустимое количество';
WHEN SET(Document o IS Document) AND NOT userDate(o) DO userDate(o) ← currentDate();
WHEN SET(Document o IS Document) AND NOT userName(o) DO userName(o) ← STRING[50](name(currentUser()));
```

Класс-потомок Reception

Модуль Reception уже существует, поэтому необходимо в его содержание внести отдельные правки,
с учетом предстоящего наследования от базового класса Document:

1. Подключить необходимые для работы модули и исключить ненужные - Time, Authentication.
2. Определить общее пространство имен, как Documents. *Возможно, это необязательное действие, но, если возникнет необходимость рефакторинга, будет проще оперировать данными. Напомним, что если пространство имен (NAMESPACE) не задано, то модуль создает собственное пространство имен с именем, равным имени модуля.*
3. Создать класс Reception, унаследованный от класса Document.

Пункты 4, 5, 6 и 7 уже существуют в модуле Reception, поэтому перечислим:

4. Удалить свойства, которые присутствуют в Document, оставить свойства name и price, так как эти свойства отсутствуют в Document.
5. Организовать проверку на уникальность вносимых данных: (номер + дата) накладной + товар.
6. Создать формы отображения, редактирования и дизайн формы редактирования.
7. Определить пункт меню для отображения формы.

```
MODULE Reception;

// 1. Необходимые для работы модули
REQUIRE Documents, AppMenu, Organization;

// 2. Задаем общее пространство имен для всех документов
NAMESPACE Documents;

// 3. Наследуем класс приходы от класса Document
CLASS Reception 'Приходы':Document;

// 4. Добавляем свойства, которых нет в базовом классе
name 'Наименование' = DATA STRING[100] (Reception) NONULL;
price 'Цена' = DATA NUMERIC[10,2] (Reception) NONULL;

// 4. Композиция
supplier = DATA Organization (Reception);
organizationName 'Поставщик' (Reception o) = name(supplier(o));

// 5. Контроль данных
unique = GROUP AGGR Reception o BY number(o), date(o), name(o);
CONSTRAINT SETCHANGED(supplier(Reception o)) AND
     NOT organizationType(supplier(o)) = OrganizationType.supplier
     CHECKED BY supplier[Reception]
     MESSAGE 'Для приходной накладной организация должна быть поставщиком';

// 6. Формы
FORM editReception 'Приход'
   OBJECTS r = Reception PANEL // опция panel - только одна запись
   PROPERTIES (r) number, date, organizationName, name, quantity, price
   EDIT Reception OBJECT r
;

DESIGN editReception {
```

```
    OBJECTS {
        NEW cntDoc {
            horizontal = TRUE; // распределяем элементы слева направо и сверху вниз
            caption = 'Накладная';
            height = 60;
            MOVE PROPERTY (number(r));
            MOVE PROPERTY (date(r));
        }
        NEW cntOrg {
            horizontal = TRUE;
            caption = 'Поставщик';
            height = 60;
            MOVE PROPERTY (organizationName(r)) { caption = 'Фирма'; }
        }
        NEW cntRow {
            horizontal = FALSE ;
            caption = 'Строки накладной';
            MOVE PROPERTY (name(r));
            MOVE PROPERTY (quantity(r));
            MOVE PROPERTY (price(r));
        }
    }
}

FORM viewReception 'Приходы'
    OBJECTS r = Reception // опция по умолчанию GRID
    PROPERTIES (r) READONLY number, date, organizationName, name, quantity, price, userName,
userDate
    PROPERTIES (r) NEWSESSION NEW, EDIT, DELETE
;


// 7. Навигатор
NAVIGATOR {
    documents {
        NEW FORM viewReception;
    }
}
```

Класс-потомок Expenses

Создадим в папке "Package documents" новый модуль с именем "Expenses", в котором выполним следующие действия, очень похожие на те, которые выполнялись для модуля "Reception". *Можно, как вариант, скопировать весь код из Reception в созданный модуль и внести правки.*

1. Подключить необходимые для работы модули.
2. Определить общее пространство имен, как Documents.
3. Создать класс Expenses, унаследованный от класса Document.
4. Добавить новые свойства name (товар) и price (цена) в созданный класс Expenses, так как эти свойства отсутствуют в Document. При этом добавляемые свойства должны ссылаться на строку прихода, с которой идет списание (см. постановку задачи). Поэтому необходимо организовать композицию для выражения name и price.
5. Создать свойство текущего остатка и организовать контроль списываемого товара.
6. Создать форму отображения, редактирования и дизайн формы редактирования.
7. Определить пункт меню для отображения формы.

```
MODULE Expenses;

// 1. Необходимые для работы модули
REQUIRE Documents, AppMenu, Organization, Reception, Utils;

// 2. Задаем общее пространство имен для всех документов
NAMESPACE Documents;

// 3. Наследуем класс приходы от класса Document
CLASS Expenses 'Расходы':Document;

// 4. Добавляем свойства, которых нет в базовом классе
// так как товар и цена берутся из прихода, то нужна композиция
reception 'ID' = DATA Reception (Expenses) NONULL ;
name 'Наименование' (Expenses o) = name(reception(o));
price 'Цена' (Expenses o) = round(1.1 * price(reception(o)),2); // 1.1 - оптовая надбавка

// 4. Композиция
customer = DATA Organization (Expenses);
organizationName 'Покупатель' (Expenses o) = name(customer(o));

// 5. Контроль данных
allExpenses  (Reception r) = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
balance 'Остаток' (Reception r) = quantity(r) (-) allExpenses(r);
CONSTRAINT balance(reception(Expenses o)) < 0 MESSAGE 'Расход превышает остаток';
CONSTRAINT SETCHANGED(customer(Expenses o)) AND NOT organizationType(customer(o)) =
OrganizationType.customer
   CHECKED BY customer[Expenses] MESSAGE 'Для расходной накладной организация должна быть
покупателем';

// 6. Формы
FORM editExpenses 'Расход'
   OBJECTS e = Expenses PANEL
   PROPERTIES (e) number, date, organizationName, name, quantity, price
   EDIT Expenses OBJECT e
;

FORM viewExpenses 'Расходы'
   OBJECTS e = Expenses
   PROPERTIES (e) READONLY number, date, organizationName, name, quantity, price, userName,
userDate
   PROPERTIES (e) NEWSESSION NEW, EDIT, DELETE
   ORDERS date(e), number(e)
```

```
;

DESIGN editExpenses {
    OBJECTS {
        NEW cntDoc {
            horizontal = TRUE; // распределяем элементы слева направо и сверху вниз
            caption = 'Накладная';
            height = 60;
            MOVE PROPERTY (number(e));
            MOVE PROPERTY (date(e));
        }
        NEW cntOrg {
            horizontal = TRUE;
            caption = 'Покупатель';
            height = 60;
            MOVE PROPERTY (organizationName(e)) { caption = 'Магазин'; }
        }
        NEW cntRow {
            horizontal = FALSE ;
            caption = 'Строки накладной';
            MOVE PROPERTY (name(e));
            MOVE PROPERTY (quantity(e));
            MOVE PROPERTY (price(e));
        }
    }
}

FORM listReception 'Остатки' // форма выбора остатков для списания
    OBJECTS r = Reception
    PROPERTIES (r) READONLY name, price, quantity, balance
    LIST Reception OBJECT r
    FILTERS balance(r)
;

// 7. Задаем пункт меню
NAVIGATOR {
    documents {
        NEW FORM viewExpenses;
    }
}
```

Некоторые пояснения:

- Пункт 4. Новые свойства name и price выражаются через композицию. При этом для цены списания действует оптовая надбавка в 10% - множитель 1.1, и используется внутренняя функция платформы round из модуля Utils для выполнения арифметического округления до 2-х знаков.
- Пункт 5. Вычисление текущего остатка. Общая идея заключается в том, что надо суммировать все расходы, связанные с текущей позицией товара, свойство allExpenses. От количества по приходу отнять allExpenses, которое и будет текущим остатком, свойство balance. *При этом вместо арифметического знака минус используется оператор (-), который трактует NULL параметры как 0, при этом нулевой результат этого вычитания будет трактоваться как NULL*. Таким образом возможны 3 варианта значений для свойства balance:
  - не равен NULL, если не было расхода, balance = quantity;
  - не равен NULL, если итоговый расход меньше прихода;
  - равен NULL, если товар списан полностью.

При расчете свойства allExpenses задействовано ключевое слово MATERIALIZED, которое зафиксирует расчет в отдельной колонке в базе данных. Что это дает? При обращении к свойству (выражения, формы) значение повторно не рассчитывается и выдается готовый результат, что значительно экономит время при обращении к свойству, а также при отображении экранных форм, в котором это свойство участвует. Расчет выполняется только тогда, когда изменилось количество. *Для не больших объемов данных использование MATERIALIZED несущественная опция, а для больших объемов информации может быть источником оптимизации выполнения.*

См. также

Документация. Классы.
Документация. Пользовательские классы. Абстрактные классы.
Документация. Материализации
Документация. Вычисления. GROUP SUM

Метапрограммирование

Анализ программного кода модулей Reception и Expenses выявил, что в разных модулях присутствуют практически одинаковые участки. С точки зрения развития и сопровождения программного кода, это не очень хорошо, так как практически одинаковые функциональные изменения при необходимости придется вносить в разных модулях. В текущем приложении таких модулей два, а в реальном приложении таких модулей может быть много. Поэтому легко может возникнуть ситуация, в которой при внесении необходимых изменений не все программные модули получат нужные изменения, а при выполнении приложения у клиента могут возникнуть нежелательные нюансы.

Выходом из сложившейся ситуации может быть использование Метапрограммирования, которое задает шаблон формирования программного кода, который автоматически формируется в процессе работы приложения. Метакод пишется внутри блока META ... END. Для управления блоком метакода *должны* использоваться входные параметры и специальные операции объединения лексем.

Примечание:

- Возможна такая ситуация, что параметры фактически не нужны. Но так как функция метакода требует входной параметр, то такой параметр нужно определить, но не использовать его в описании метакода.

Использование метапрограммирования позволяет:

- улучшить читаемость программного кода,
- улучшить сопровождение программного кода.

Оформим, используя метакод:

- Создание формы отображения.
- Создание формы редактирования.
- Дизайн формы редактирования.
- Определение пункта меню.

Что нужно сделать:

- Создать в модуле Documents метакод.
- Использовать метакод в модулях Reception и Expenses, удалив описание форм редактирования, отображения, дизайна и создания пункта меню.

Метакод в модуле Documents:

```
META forms(object, title1, title2, title3, title4)
   // форма редактирования
   FORM edit##object title1
       OBJECTS o = object PANEL
       PROPERTIES (o) number, date, organizationName, name, quantity, price
       EDIT object OBJECT o
   ;
   // форма отображения
   FORM view##object title2
       OBJECTS o = object
       PROPERTIES (o) number, date, organizationName, name, quantity, price, userName, userDate
       PROPERTIES (o) NEWSESSION NEW, EDIT, DELETE
       ORDERS date(o), number(o)
   ;
   // дизайн формы редактирования
   DESIGN edit##object {
       OBJECTS {
           NEW cntDoc {
               horizontal = TRUE; // распределяем элементы слева-направо-сверху-вниз
               caption = 'Накладная';
               height = 60;
               MOVE PROPERTY (number(o));
               MOVE PROPERTY (date(o));
           }
           NEW cntOrg {
               horizontal = TRUE;
               caption = title3;
               height = 60;
               MOVE PROPERTY (organizationName(o)) { caption = title4; }
           }
           NEW cntRow {
               horizontal = FALSE ;
               caption = 'Строки накладной';
               MOVE PROPERTY (name(o));
               MOVE PROPERTY (quantity(o));
               MOVE PROPERTY (price(o));
           }
       }
   }
   NAVIGATOR {
       documents {
           NEW FORM view##object;
       }
   }
END
```

Примечание:

- Блок метакода в редакторе IDEA выделяется по умолчанию желтым цветом фона, поэтому для всех приводимых участков кода, имеющих отношения к метакоду, будет использоваться такое оформление.
- Так как изначально формы отображения и редактирования имели имена view*Имя_класса* и *edit*Имя_класса, то их оказалось легко заменить с использованием передаваемого в качестве параметра имени класса. При этом использовалось объединение лексем, как view##*Имя_класса* и *edit*##*Имя_класса*, что предопределило уникальность имен форм.

Код в модуле Reception с использованием метакода записывается так:

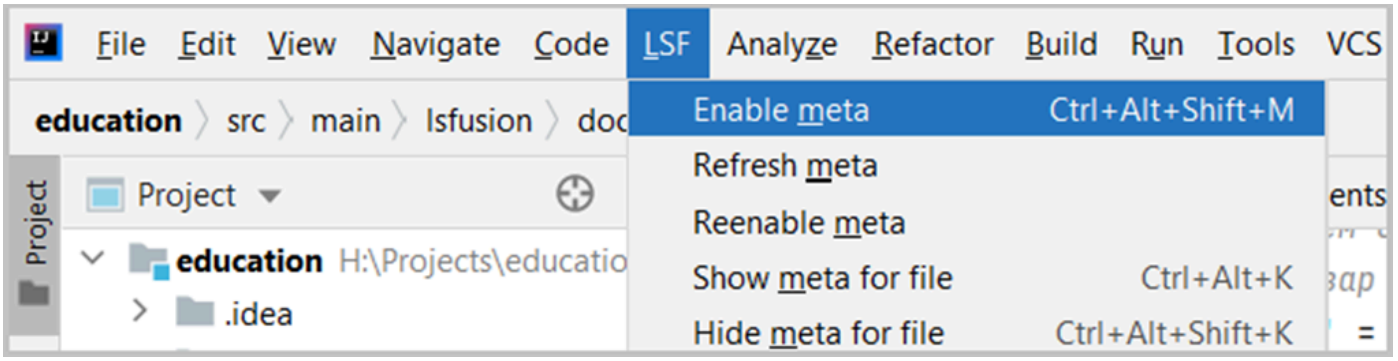
```
// 6. Формы и 7. Навигатор
@forms(Reception, 'Приход', 'Приходы', 'Поставщик', 'Фирма');
```

Код в модуле Expenses с использованием метакода записывается так:

```
// 6. Формы и 7. Навигатор
@forms(Expenses, 'Расход', 'Расходы', 'Покупатель', 'Магазин');
```

Отображение метакода в свернутом состоянии (*@имя*) не всегда может быть удобным для восприятия, когда необходимо понять логику работы, увидеть реальные свойства и формы, например в связи с предстоящими доработками.

В редакторе IDEA, благодаря плагину lsFusion, можно управлять видимостью метакода.



Могут быть два состояния выделенного пункта меню:

- Enable meta - метакоды не развернуты, выполнение разворачивает метакоды.
- Disabled meta - метакоды развернуты, выполнение сворачивает метакоды.

Описание пунктов меню:

- Enable meta/Disabled meta – действует глобально для всех модулей проекта: разворачивает/сворачивает метакоды. *При этом, если проект очень большой (тысячи модулей), то это может занять некоторое время.*
- Refresh meta – обновляет внутренние содержание метакодов.
- Reenable meta – запускает механизм развертывание метакодов, которые по каким-либо причинам оказались неразвернутыми. *Например, произошла вставка из буфера обмена свернутого* метакода. *При этом возможна ситуация, при которой* метакод *не развернется автоматически. В этой ситуации можно либо свернуть все* метакоды *и потом развернуть, либо выполнить пункт Reenable meta. Пункт меню сработает, если перед выполнением* метакоды *были развернуты, иначе будет предупреждение о невозможности выполнения.*
- Show meta for file – раскрывает метакод для текущего модуля.
- Hide meta for file – скрывает метакод для текущего модуля.

Если метакод развернут, то содержание метакода высвечивается на сером фоне:

Expenses.lsf

```
@forms(Expenses, 'Расход', 'Расходы', 'Покупатель', 'Магазин'){
    // форма редактирования
    FORM editExpenses 'Расход'
        OBJECTS o = Expenses PANEL
        PROPERTIES (o) number, date, organizationName, name, quantity, price
        EDIT Expenses OBJECT o
    ;
    // форма отображения
    FORM viewExpenses 'Расходы'
        OBJECTS o = Expenses
        PROPERTIES (o) number, date, organizationName, name, quantity, price, userName, userDate
        PROPERTIES (o) NEWSESSION NEW, EDIT, DELETE
        ORDERS date(o), number(o)
    ;
    // дизайн формы
    DESIGN editExpenses {
        OBJECTS {
            NEW cntDoc {
                horizontal = TRUE; // распределяем элементы слева-направо-сверху-вниз
                caption = 'Накладная';
                height = 60;
                MOVE PROPERTY (number(o));
                MOVE PROPERTY (date(o));
            }
            NEW cntOrg {
                horizontal = TRUE;
                caption = 'Покупатель';
                height = 60;
                MOVE PROPERTY (organizationName(o)) { caption = 'Магазин'; }
            }
            NEW cntRow {
                horizontal = FALSE ;
                caption = 'Строки накладной';
                MOVE PROPERTY (name(o));
                MOVE PROPERTY (quantity(o));
                MOVE PROPERTY (price(o));
            }
        }
    }
    NAVIGATOR {
        documents {
            NEW FORM viewExpenses;
        }
    }
};
```

Хорошо просматривается содержимое метакода: форма редактирования, форма отображения, дизайн формы редактирование, элементы навигатора.

При использовании метапрограммирования есть некоторые особенности, связанные с отображением элементов как в самом метакоде, так и элементов кода, ссылающихся на объявление в метакоде.

Замечание 1
Рассмотрим отображение некоторых элементов в блоке метакода, отмеченных как ошибки:

```
META forms(object, title1, title2, title3, title4)
    // форма редактирования
    FORM edit##object title1
        OBJECTS o = object PANEL
        PROPERTIES (o) number, date, organizationName, name, quantity, price
        EDIT object OBJECT o
    ;
```

Property or action 'organizationName' not found

Некоторые свойства могут выделяться или подчеркиваться, так как для метакода они могут быть неизвестны, но это не является ошибкой. Например, имя класса object выделяется оранжевым цветом, так как оно для модуля неизвестно. Свойства organizationName и price в блоке формы PROPERTIES также выделены цветом, так как они не определены в текущем модуле. Свойство name, под которым понимается наименование товара, подчеркнуто, так как текущий модуль не может определится с их принадлежность. Если что-либо вызывает сомнение, то можно навести указатель мыши на элемент, выделенный цветом или подчеркиванием, через секунду отобразится подсказка с описанием проблемы.

Замечание 2
Отображение элементов кода, ссылающихся на объявления в метакоде, как ошибки. Рассмотрим пример. В модуле Expenses определим свойство allExpenses в блоке метакода:

```
// Вычисление текущего остатка
META calcAll(pp)
    allExpenses  (Reception r) = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
END
```

Если метакод свернут, то:

- то свойство allExpenses в выражении вычисления остатка balance будет выделено красным цветом, как ошибка;
- если навести курсор мыши на свойство allExpenses, то всплывающая подсказка укажет на то, что свойство не найдено.

```
META calcAll(pp)
    allExpenses  (Reception r) -> NUMERIC[10,3] = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
END

@calcAll(NULL);
balance 'Остаток' (Reception r) -> NUMERIC[10,3] = quantity(r) (-) allExpenses(r);
CONSTRAINT balance(reception(Expenses o)) < 0 MESSAGE 'Расход превышает ос
```

Property 'allExpenses' not found

При этом программно все будет работать без ошибок.

Если метакод будет развернут, то ошибка фиксироваться не будет.

```
META calcAll(pp)
    allExpenses  (Reception r) -> NUMERIC[10,3] = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
END

@calcAll(NULL){
    allExpenses  (Reception r) -> NUMERIC[10,3] = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
};
balance 'Остаток' (Reception r) -> NUMERIC[10,3] = quantity(r) (-) allExpenses(r);
CONSTRAINT balance(reception(Expenses o)) < 0 MESSAGE 'Расход превышает остаток';
```

Замечание 3
Существующий метакод можно преобразовать в простой код.
Для этого надо добавить еще один знак "@" перед вызовом метакода, например:

@@calcAll(NULL), вместо @calcAll(NULL).

В этом случае метакод будет преобразован (*мгновенно*) плагином lsFusion в обычный код, при этом объявление метакода останется без изменений:

```
META calcAll(pp)
    allExpenses  (Reception r) -> NUMERIC[10,3] = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
END

allExpenses  (Reception r) -> NUMERIC[10,3] = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
balance 'Остаток' (Reception r) -> NUMERIC[10,3] = quantity(r) (-) allExpenses(r);
CONSTRAINT balance(reception(Expenses o)) < 0 MESSAGE 'Расход превышает остаток';
```

Отменить преобразование можно по Ctrl+Z (Edit-Undo).

См. также
Документация, Инструкция META

Пример кода:
examples\t211_meta_программирование\src

Краткие итоги

После рассмотрения вопроса наследования, построения форм и вычислений, практическую часть задачи "учета товародвижения" в соответствии со своей постановкой, можно считать полностью решенной.

При этом за "кадром" осталось много вопросов, которые не были рассмотрены намеренно, чтобы не усложнять изложение:

- документы были выбраны плоскими, весь интерфейс свелся к двум основным табличным формам;
- не рассматривались дополнительные интерфейсы;
- была проигнорирована полноценная работа со справочниками;
- опущена работа с отчетами;
- и другое.

Тем не менее, можно предположить, что упрощенный вариант решения отразил основные моменты написания программного кода на платформе IsFusion, продемонстрировал некоторые возможности и простоту решения поставленной задачи. В ходе изучения материала были рассмотрены вопросы:

- Создание модулей
- Создание классов, определение их свойств, композиции
- Создание меню
- Создание форм отображения, редактирования и выбора
- Управление дизайном форм
- Контроль ввода и использование событий
- Наследование классов
- Метапрограммирование
- Вычисления

Дальнейшее изложение материала будет отталкиваться от первоначальной постановки задачи, но при этом к решению задачи будут применены традиционные подходы построения интерфейсов, рассмотрены вопросы создания отчетов, работы с внешними данными.

Тема 3. Делаем все правильно

Рассматриваемые вопросы:

- Постановка задачи
- Документ типа "шапка-строки"
  - Обновленный модуль Document
  - Обновленный модуль Reception
  - Обновленный модуль Expenses
- Справочники
  - Предварительные действия
  - Справочник товарных групп
  - Справочник товаров
  - Справочник единиц измерения
  - Обновленный модуль Organization
  - Обновленный модуль Documents
  - Обновленный модуль Reception
  - Обновленный модуль Expenses
- Заполнение справочников
  - Заполнение товарных групп
  - Миграция. Действия. Кнопки. Интерпретатор
- Деревья. Журнал остатков

Постановка задачи

Надо создать программу для учета товародвижения на оптовой базе.
Принцип работы базы с товарами заключается в приеме товаров от поставщиков и продаже товаров розничным торговым объектам.
Документы прихода и расхода должны быть представлены документами типа "шапка-строки".
В разработке отталкиваемся от партионного учета, то есть позиции для расхода списываются с позиций прихода, которые являются остатком.
Должен быть предусмотрен контроль списания и заполнения данных.
Для заполнения строк прихода используется справочник товаров, связанный с товарными группами и справочником единиц измерения. Иными словами, данные должны быть представлены в нормализованном виде.
Должен быть предусмотрен журнал остатков в разрезе товарных групп.
Для расходной накладной свойства название товара и цена берутся из строки приходной накладной, с которой товар списывается.
При списании товара действует оптовая надбавка в 10%.
Товары учитываются в ценах поставщика.

Что надо сделать, можно представить в виде некоторой иерархии документов и справочников:

1. Документы
- Заголовок (шапка) документа
    - Справочник организаций
- Строки документа
    - Справочник товаров
    - Справочник единиц измерения
2. Остатки товара
- Справочник товарных групп
    - Остатки товара
        - Справочник товаров
        - Справочник единиц измерения
3. Справочник организаций
4. Справочник товарных групп
5. Справочник товаров
- Справочник товарных групп
    - Справочник товаров
6. Справочник единиц измерения

Отталкиваясь от ранее рассмотренных технологий, для решения текущих задач, будут использоваться:

- построение форм
- наследование классов
- метапрограммирование
- композиции
- использование форм выбора

Особенности постановки задачи:

- По своей сути новая постановка задачи совпадает с постановкой задачи из темы 2. Та же оптовая база, партионный учет, списание с приходных позиций. В старой постановке задачи рассматривался плоский документ, без выделения заголовка документа и его строк. Это было сделано намеренно, чтобы упростить изложение. В новой постановке

задачи необходимо создать целостный документ, состоящий из заголовка (шапки) документа и связанных с ним строк.
- Добавление новых справочников.
- Появление древовидного справочника товаров.
- Расширение вычислений и обработок.

После прочтения новой постановки задачи и того, что предстоит сделать в новой редакции разрабатываемой программы, может создаться впечатление, что разработка завязнет серьезно и надолго. Но это ошибочное впечатление. Используя технологии платформы, задача решается быстро и с минимальными усилиями.

См. также
Что такое нормализация данных?

Документ типа "шапка-строки"

Необходимо создать единый документ, который состоит из 2-х взаимосвязанных частей: заголовок документа и строки документа. Задача может показаться сложной в реализации. На самом деле она решается достаточно просто, так как:

- все необходимые технологии были рассмотрены ранее
- нужные "кубики" уже созданы и могут быть повторно использованы в новом приложении.

Для решения задачи необходимо выполнить рефакторинг приложения с небольшими дополнениями.

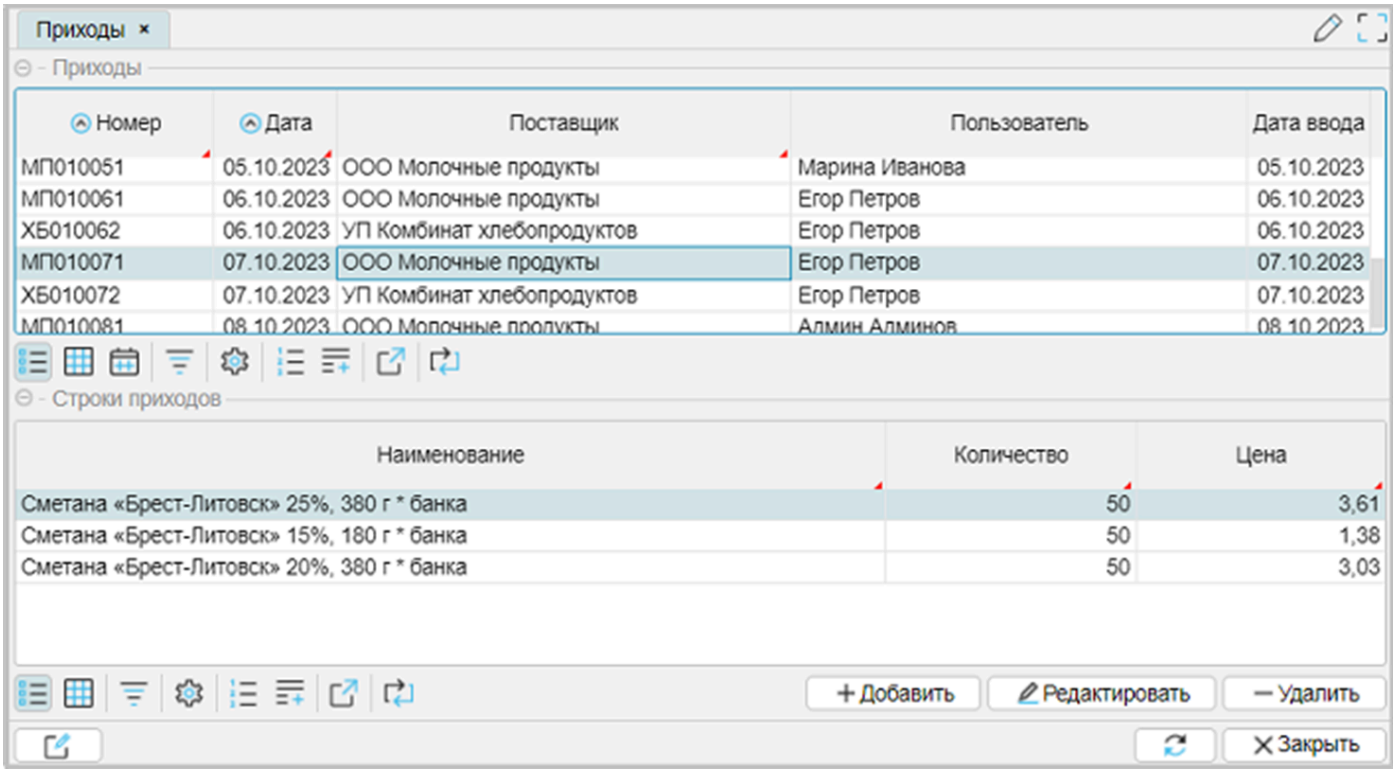
Алгоритм действий прост:

- Изменить структуру модуля Document:
  - создать два родительских класса (вместо одного): заголовок документа (Header) и строки документа (Row),
  - исправить метакод форм отображения и редактирования;
- Внести изменения в модуль Reception;
- Внести изменения в модуль Expenses.

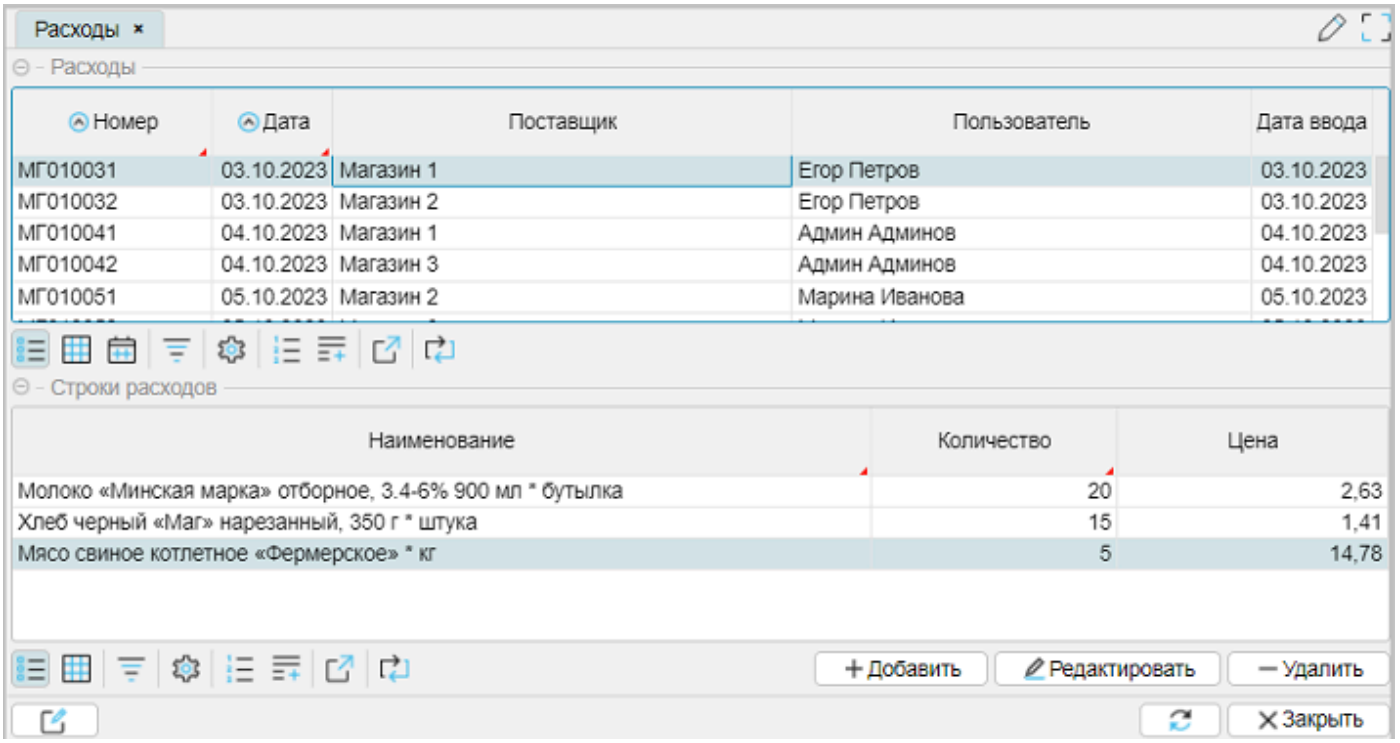
Примечание:

- Так как меняется структура документа, все данные по приходу и расходу будут потеряны. Если бы данные были *важны*, то можно было бы рассмотреть вопрос их сохранения с последующим восстановлением в новой структуре.

После выполнения обновления кода модулей, старта сервера приложений и внесения данных, форма отображения Приходов примет вид:



После внесения приходов было выполнено списание товаров:



Одной из тем, рассматриваемых далее, станет более полноценная работа со справочниками. При этом одной из важных задач, которая будет рассматриваться в разделе Справочники, станет сохранение введенных данных.

См. также

Пример кода:
examples\t301_документ_шапка_строки\src

Архив SQL:
examples\t301_документ_шапка_строки\sql

Список внесенных товаров:
examples\doc\список_товаров_групп.xlsx

Обновленный модуль Document

Из класса Document создаем два класса:

- Заголовок документа - Header
  - номер документа - number
  - дата документа - date
  - название организации - organizationName (композиция)
  - пользователь - userName
  - дата ввода - userDate
- Строки документа - Row
  - количество товара - quantity

Определяем уникальность документа: документы не повторяются в течении дня по своему номеру.

```
// новое: определяем уникальность документа
unique = GROUP AGGR Header o BY number(o), date(o);
```

Связываем строки документа с его заголовком созданием нового свойства header для класса Row, у которого класс типа данных Header. При этом добавляем ограничение определенности NONULL DELETE, которое будет работать таким образом, что при удалении шапки документа строки, связанные с шапкой, будут удаляться автоматически:

```
// новое: связываем строки с шапкой
header 'ID шапки' = DATA Header (Row) NONULL DELETE;
```

Часть 1. Код модуля (без метакода).

```
MODULE Documents;

REQUIRE Time, Authentication, Organization;

CLASS ABSTRACT Header 'Заголовки документа';

number 'Номер'  = DATA STRING[10] (Header) NONULL;
date 'Дата' = DATA DATE (Header) NONULL;
userDate 'Дата ввода' = DATA DATE (Header);
userName 'Пользователь' = DATA STRING[50] (Header);

// автозаполнение
WHEN SET (Header o IS Header) AND NOT userDate(o) DO userDate(o) ← currentDate();
WHEN SET (Header o IS Header) AND NOT userName(o)
                                          DO userName(o) ← STRING[50](currentUserName());

// новое: определяем уникальность документа
unique = GROUP AGGR Header o BY number(o), date(o);

CLASS ABSTRACT Row 'Строки документа';

quantity 'Количество' = DATA NUMERIC[10,3] (Row) NONULL;

// новое: связываем строки с шапкой
header 'ID шапки' = DATA Header (Row) NONULL DELETE;

// контроль данных
CONSTRAINT quantity(Row o) > 1000 OR quantity(o) ⩽ 0 MESSAGE 'Недопустимое количество';
```

Так как для форм редактирования и отображения приходных и расходных документов будет использоваться метакод, то рассмотрим построение форм на приводимых примерах:

Форма отображения, пример:
Если необходимо связать на отображении заголовки со строками так, чтобы свой заголовок "видел" свои строки, на форме используется фиксированный фильтр:

```
FORM view 'Документы'
    OBJECTS h = Header LAST
    PROPERTIES (h) READONLY number, date, organizationName, userName, userDate
    PROPERTIES (h) NEWSESSION NEW, EDIT, DELETE

    OBJECTS r = Row
    PROPERTIES (r) READONLY name, quantity, price
    FILTERS header(r) = h
;
```

Примечание:

- порядок следования объектов при описании формы определяет порядок следования их на форме;
- количество объектов на форме не ограничено;
- опция LAST используется для того, чтобы указать, что при открытии формы отображения указатель должен стоять на последней записи.

Форма редактирования, пример:
Документ типа "шапка-строки" может быть отредактирован одной формой, включающей панель редактирования заголовка и табличной части для ввода строк документа:

```
FORM edit 'Документ'
    OBJECTS h = Header PANEL // редактирование заголовка
    PROPERTIES (h) number, date, organizationName

    OBJECTS r = Row // редактирование строк
    PROPERTIES (r) name, quantity, price
    EDIT Header OBJECT h
    FILTERS header(r) = h
;
```

Примечание:

- первый объект (класса Header) отображается с опцией PANEL - то есть текущая редактируемая запись, а объект строк (класса Row) отображается как табличная часть (GRID) набора записей строк, связанных с шапкой документа;
- на форме редактирования наличие фиксированного фильтра "подскажет" форме, что при сохранении данных свойство header всех строк должно получить ссылку на заголовок редактируемого документа.

На основании приведенных примеров, а также понимания того, что формы приходных и расходных документов будут выглядеть примерно одинаково, можно оформить создание форм с помощью метакода. Шаблон метакода отражает основные элементы и свойства форм. По мере необходимости отсутствующие свойства в конкретных формах и модулях можно будет добавить, используя расширение форм EXTEND FORM.

Часть 2. Код модуля (метакод).

```
META forms(object, title1, title2, title3)
   // форма редактирования
   FORM edit##object title1
       OBJECTS h = Header##object PANEL // редактирование заголовка
       PROPERTIES (h) number, date, organizationName

       OBJECTS r = object // редактирование строк
       PROPERTIES (r) name, quantity, price
       PROPERTIES (r) NEW, DELETE
       EDIT Header##object OBJECT h
       FILTERS header(r) = h
   ;
   // подправляем заголовок свойства для разных форм редактирования
   DESIGN edit##object { PROPERTY (organizationName(h)) {caption = title3;} }
   // форма отображения
   FORM view##object title2
       OBJECTS h = Header##object LAST
       PROPERTIES (h) READONLY number, date, organizationName, userName, userDate
       PROPERTIES (h) NEWSESSION NEW, EDIT, DELETE

       OBJECTS r = object
       PROPERTIES (r) READONLY name, quantity, price
       FILTERS header(r) = h
       ORDERS date(h), number(h)
   ;
   // панель кнопок редактирования шапки передвигаем в панель редактирования строк
   DESIGN view##object { TOOLBARRIGHT (r) { MOVE TOOLBARRIGHT (h); }}
   NAVIGATOR {
       documents {
           NEW FORM view##object;
       }
   }
END
```

Примечание:

- Так как панель редактирования всегда располагается по умолчанию под своим объектом, для того чтобы сделать отображение элементов редактирования формы более эстетичным, – расположить внизу формы – перемещаем панель (контейнер) кнопок редактирования шапки в область панели редактирования строк используя инструкцию DESIGN. То есть кнопки редактирования документа будут располагаться не под верхним GRID, а под нижним.
- Если бы возникла ситуация редактирования шапки и строк документа отдельными формами, то для редактирования строк накладной должна была бы быть использована форма выбора из списка всех существующих накладных для привязки строк к заголовку накладной, примерно, как происходит списание или выбор значения из справочника.
- DESIGN: подправляем в форме редактирования заголовок свойства Организации так, чтобы для приходных накладных это был "Поставщик", а для расходных – "Покупатель".

См. также

Документация. Инструкция FORM. Блоки формы
Документация. Блоки фильтров и сортировок. Блок фиксированных фильтров
Документация. Опции свойства
Документация. Простые ограничения

Обновленный модуль Reception

Выполним доработки модуля:

```
MODULE Reception;

// 1. Необходимые для работы модули
REQUIRE Documents, AppMenu, Organization;

// 2. Задаем общее пространство имен для всех документов
NAMESPACE Documents;

// 3. Наследуем класс заголовка прихода от класса Header
CLASS HeaderReception 'Приходы':Header;

// 3. Наследуем класс строки прихода от класса Row
CLASS Reception 'Строки приходов':Row;

// 4. Добавляем свойства, которых нет в базовом классе
name 'Наименование' = DATA STRING[100] (Reception) NONULL;
price 'Цена' = DATA NUMERIC[10,2] (Reception) NONULL;

// 4. Композиция
supplier = DATA Organization (HeaderReception);
organizationName 'Поставщик' (HeaderReception o) = name(supplier(o));

// 5. Контроль данных
CONSTRAINT SETCHANGED(supplier(HeaderReception o)) AND
    NOT organizationType(supplier(o)) = OrganizationType.supplier
    CHECKED BY supplier[HeaderReception]
    MESSAGE 'Для приходной накладной организация должна быть поставщиком';

// 6. Формы редактирования и отображения, дизайн, 7. меню
@forms(Reception, 'Приход', 'Приходы', 'Поставщик');
```

Все изменения, коснувшиеся модуля, это пере наследование классов заголовка документа и строк.

Обновленный модуль Expenses

Выполним доработки модуля:

```
MODULE Expenses;

// 1. Необходимые для работы модули
REQUIRE Documents, AppMenu, Organization, Reception, Utils;

// 2. Задаем общее пространство имен для всех документов
NAMESPACE Documents;

// 3. Наследуем класс заголовка расхода от класса Header
CLASS HeaderExpenses 'Расходы':Header;

// 3. Наследуем класс строки расхода от класса Row
CLASS Expenses 'Строки расходов':Row;

// 4. Добавляем свойства, которых нет в базовом классе
// так как товар и цена берутся из прихода, то нужна композиция
reception 'ID' = DATA Reception (Expenses) NONULL;
name 'Наименование' (Expenses o) = name(reception(o));
price 'Цена' (Expenses o) = round(1.1 * price(reception(o)), 2); // 1.1 - оптовая надбавка

// 4. Композиция
customer = DATA Organization (HeaderExpenses);
organizationName 'Покупатель' (HeaderExpenses o) = name(customer(o));

// 5. Контроль данных и вычисления остатка
allExpenses  (Reception r) = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
balance 'Остаток' (Reception r) = quantity(r) (-) allExpenses(r);
CONSTRAINT balance(reception(Expenses o)) < 0 MESSAGE 'Расход превышает остаток';
CONSTRAINT SETCHANGED(customer(HeaderExpenses o)) AND
   NOT organizationType(customer(o)) = OrganizationType.customer
   CHECKED BY customer[HeaderExpenses]
   MESSAGE 'Для расходной накладной организация должна быть покупателем';

// 6. Формы редактирования и отображения, дизайн, 7. меню
@forms(Expenses, 'Расход', 'Расходы', 'Покупатель');

// 6. форма выбора остатков
FORM listReception 'Остатки'
   OBJECTS r = Reception
   PROPERTIES (r) READONLY name, price, quantity, balance
   LIST Reception OBJECT r
   FILTERS balance(r)
;
```

Все изменения, коснувшиеся модуля, это пере наследование классов заголовка документа и строк.
Созданная ранее логика списания и форма выбора не изменились, так как по-прежнему списание отталкивается от строк прихода.

Справочники

В соответствии с постановкой задачи, можно выделить 4 подзадачи:

1. Справочник товарных групп в виде дерева
2. Справочник товаров, связанный со справочником товарных групп
3. Справочник единиц измерения
4. Справочник организаций: он хоть и есть, будет требовать доработки

Можно предположить, что все справочники похожи:

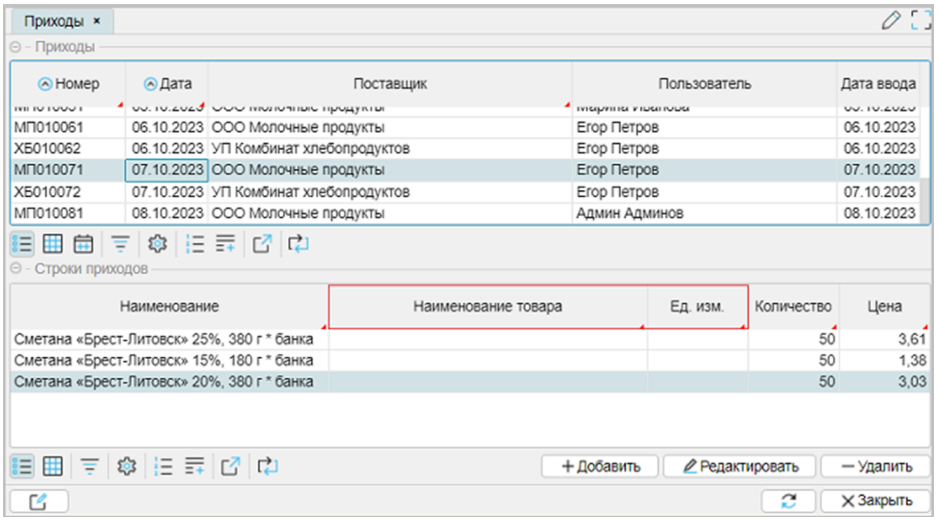
- Наличием свойства name - наименование
- Наличием 3-х форм: отображения, редактирования и выбора
- Наличием отдельного пункта меню
- Контролем уникальности вводимых значение

Поэтому логично было бы использовать возможности Метапрограммирования.

Что надо сделать:

- Определить метакод для:
  - определения свойства name (наименование в справочнике: товаров, групп, организаций, единиц измерения) с контролем уникальности и ограничениями
  - описания форм отображения, редактирования и выбора
- Создать справочник товарных групп в виде древовидного списка
- Создать справочник товаров
- Создать справочник единиц измерения
- Внести изменения для справочника Организаций
- Доработать модуль Documents:
  - внести в блок PROPERTIES свойства name свойства nameProduct, nameUnit для форм отображения и редактирования. Свойства будут определены позже через композицию в модулях Reception и Expenses.
- Доработать модули Reception и Expenses:
  - задействовать композицию выбора из справочника товаров
  - задействовать композицию выбора единиц измерения
  - свойство Reception.name (старое наименование товара) в ходе доработок остается

После выполненных доработок и запуска сервера приложений, форма отображения "Приходы" примет следующий вид:



Примечание:

- На форме в нижней табличной части видна заполненная колонка "Наименование", при этом колонки "Наименование товара" и "Ед. изм." пусты. Так и должно быть на данном этапе. Следующие вопросы, которые будут рассмотрены в "Заполнение справочников", будут решать задачи переноса данных.
- Форма отображение "Расходы" будет иметь такой же вид.

См. также

Пример кода:
examples\t302_справочники\src

Архив SQL:
examples\t302_справочники\sql

Предварительные действия

Необходимо выполнить некоторые предварительные действия:

1. Внутри "Package references" создадим каталог "product", для создания модулей справочников, связанных с товарами.
2. В меню "Справочники" (references) отдельно выделим пункт "Товары" (products) для лучшей навигации, внутри которого будут вызываться формы справочников, связанные с товарами.

С этой целью исправим модуль AppMenu:

```
MODULE AppMenu;

NAVIGATOR {
   NEW FOLDER move 'Движение' WINDOW toolbar FIRST {
       NEW FOLDER documents 'Документы движения';
       NEW FOLDER references 'Справочники' {
           NEW FOLDER products 'Товары';
       }
   }
}
```

3. Так как все справочники будут предполагать наличие формы редактирования, отображения и выбора, а также наличие своего пункта меню, то участки кода можно оформить с использованием метакода. Недостающие элементы форм в отдельных модулях можно будет добавить потом с использованием инструкции расширения форм EXTEND FORM.

Для решения этой задачи на уровне каталога references создадим модуль RForms с метакодом для использования при работе со справочниками:

```
MODULE RForms;

REQUIRE Utils, AppMenu;

// объект, пункт меню, заголовок
META rforms(object, menu, title)
    name 'Наименование' = DATA STRING[100] (object) NONULL; // общее свойство
    unique = GROUP AGGR object o BY upper(name(o)); // контроль уникальности
    FORM edit##object 'Редактирование'
        OBJECTS o = object PANEL
        PROPERTIES (o) name
        EDIT object OBJECT o
    ;
    FORM list##object 'Поиск'
        OBJECTS o = object
        PROPERTIES (o) READONLY name
        LIST object OBJECT o
    ;
    FORM view##object title
        OBJECTS o = object
        PROPERTIES (o) READONLY name
        PROPERTIES (o) NEWSESSION NEW, EDIT, DELETE
    ;
    NAVIGATOR {
        menu {
            NEW FORM view##object;
        }
    }
END
```

Справочник товарных групп

Создадим в "Package product" модуль "Group":

```
MODULE Group;

REQUIRE RForms;

CLASS Group 'Товарные группы';

@rforms(Group, products, 'Справочник товарных групп');

parent = DATA Group (Group);
nameParent 'Родительская группа' (Group g) = name(parent(g));
level 'Уровень' (Group child, Group parent) =
    RECURSION 1l IF child IS Group AND parent = child STEP 2l
    IF parent = parent($parent) MATERIALIZED;
canonicalName 'Каноническое имя' (Group group) = GROUP CONCAT name(Group parent),
    ' / ' ORDER DESC level(group, parent) CHARWIDTH 100;

EXTEND FORM editGroup PROPERTIES (o) nameParent;
EXTEND FORM listGroup PROPERTIES (o) READONLY nameParent;
EXTEND FORM viewGroup PROPERTIES (o) READONLY nameParent, canonicalName ORDERS
canonicalName(o);
```

Примечание:

- Создание справочника товарных групп выполнено в полном соответствии со статьей "Работа с иерархиями в lsFusion".
- Отсутствующие элементы в формах добавлены с использованием инструкции EXTEND FORM.
- Если метакоды свернуты, то можно получить отображение некоторых элементов красным (цвет ошибки).
  Однако, это не является ошибкой. В данном случае при свернутых метакодах редактор "не видит" сделанных в метакоде объявлений.
  Ситуацию легко проверить, выполнив пункты меню "LSF - Enable meta" для всех модулей проекта или для текущего "LSF-Show meta for file". Ситуация рассматривалась ранее при рассмотрении вопроса "Метапрограммирование".

См. также
Статья. Работа с иерархиями в lsFusion
Документация. Оператор RECURSION

Справочник товаров

Создадим в "Package product" модуль "Product":

```
MODULE Product;

REQUIRE RForms, Group;

CLASS Product 'Товары';

@rforms(Product, products, 'Справочник товаров');

group 'ID группы' = DATA Group (Product) NONULL;
groupName 'Группа' (Product o) = name(group(o));

EXTEND FORM editProduct PROPERTIES (o) groupName;
EXTEND FORM viewProduct PROPERTIES (o) READONLY groupName ORDERS groupName(o), name(o);
EXTEND FORM listProduct PROPERTIES (o) NEWSESSION NEW;
```

Примечание:

- Справочник товаров должен быть связан с товарными группами, поэтому создаем свойства для композиции group, groupName.
- Инструкция EXTEND FORM используется для отображения новых свойств и сортировки (ORDERS).

Справочник единиц измерения

Создадим в "Package product" модуль "Unit":

```
MODULE Unit;

REQUIRE RForms;

CLASS Unit 'Единицы измерения';

@rforms(Unit, products, 'Справочник ЕИ');
```

Обновленный модуль Organization

Изменим содержание модуля "Organization":

```
MODULE Organization;

REQUIRE AppMenu, RForms;

CLASS OrganizationType 'Тип организации' {
   supplier 'Поставщик', customer 'Покупатель'
}

CLASS Organization 'Организации';

@rforms(Organization, references, 'Справочник организаций');

organizationType 'ID' = DATA OrganizationType (Organization) NONULL;
organizationTypeName 'Тип организации' (Organization o) = staticCaption(organizationType(o));

EXTEND FORM editOrganization PROPERTIES (o) organizationTypeName;
EXTEND FORM viewOrganization PROPERTIES (o) READONLY organizationTypeName;
EXTEND FORM listOrganization PROPERTIES (o) READONLY organizationTypeName;
```

Примечание:

- Инструкция EXTEND FORM используется для отображения новых свойств, которые не предусматривает метакод rforms.

Обновленный модуль Documents

Как указывалось выше (см. "Что надо сделать") в блок метакода необходимо внести изменения, добавлением новых свойств:

- nameProduct, Наименование товара;
- nameUnit, Единица измерения;
- ограничиваем ширину колонки единиц измерения до 10 символов для формы отображения. Это не обязательно, но так как свойство name для всех справочников было выбрано 100 символов, форма при просмотре выделит для колонки единиц измерения столько же места, как и для наименования товара, уменьшив ширину колонки в названии товара, что визуально не очень хорошо.

```
META forms(object, title1, title2, title3)
   // форма редактирования
   FORM edit##object title1
       OBJECTS h = Header##object PANEL // редактирование заголовка
       PROPERTIES (h) number, date, organizationName
       OBJECTS r = object // редактирование строк
       PROPERTIES (r) name, nameProduct, nameUnit, quantity, price
       PROPERTIES (r) NEW, DELETE
       EDIT Header##object OBJECT h
       FILTERS header(r) = h
   ;
   // подправляем заголовок свойства для разных форм редактирования
   DESIGN edit##object { PROPERTY (organizationName(h)) {caption = title3;} }
   // форма отображения
   FORM view##object title2
       OBJECTS h = Header##object LAST
       PROPERTIES (h) READONLY number, date, organizationName, userName, userDate
       PROPERTIES (h) NEWSESSION NEW, EDIT, DELETE
       OBJECTS r = object
       PROPERTIES (r) READONLY name, nameProduct, nameUnit, quantity, price
       FILTERS header(r) = h
       ORDERS date(h), number(h)
   ;
   // панель кнопок редактирования шапки передвигаем в панель редактирования строк
   // и ограничиваем ширину колонки единиц измерения
   DESIGN view##object {
       TOOLBARRIGHT (r) { MOVE TOOLBARRIGHT (h); }
       PROPERTY (nameUnit(r)) {charWidth=10;}
   }
   NAVIGATOR {
       documents {
           NEW FORM view##object;
       }
   }
END
```

Примечание:

- свойство name должно быть удалено (выделено красным) после перемещения данных.

Обновленный модуль Reception

Создаем новые свойства для создания композиций названия товара и единицы измерения:

```
MODULE Reception;

// 1. Необходимые для работы модули
REQUIRE Documents, AppMenu, Organization, Product, Unit;

// 2. Задаем общее пространство имен для всех документов
NAMESPACE Documents;

// 3. Наследуем класс заголовка прихода от класса Header
CLASS HeaderReception 'Приходы':Header;

// 3. Наследуем класс строки прихода от класса Row
CLASS Reception 'Строки приходов':Row;

// 4. Добавляем свойства, которых нет в базовом классе
name 'Наименование' = DATA STRING[100] (Reception) NONULL;
price 'Цена' = DATA NUMERIC[10,2] (Reception) NONULL;

// 4. Композиция
supplier = DATA Organization (HeaderReception);
organizationName 'Поставщик' (HeaderReception o) = name(supplier(o));

// 4. Новые свойства:
product 'ID товара' = DATA Product (Reception) NONULL;
unit 'ID единиц измерения' = DATA Unit (Reception) NONULL;
nameProduct 'Наименование товара' (Reception o) = name(product(o));
nameUnit 'Ед. изм.' (Reception o) = name(unit(o));

// 5. Контроль данных
CONSTRAINT SETCHANGED(supplier(HeaderReception o)) AND
    NOT organizationType(supplier(o)) = OrganizationType.supplier
    CHECKED BY supplier[HeaderReception]
    MESSAGE 'Для приходной накладной организация должна быть поставщиком';

// 6. Формы редактирования и отображения, дизайн, 7. меню
@forms(Reception, 'Приход', 'Приходы', 'Поставщик');
```

Примечание:

- свойство name должно быть удалено (выделено красным) после перемещения данных.

Обновленный модуль Expenses

Создаем новые свойства и композиции для отображения названия товара и единиц измерения:

```
MODULE Expenses;

// 1. Необходимые для работы модули
REQUIRE Documents, AppMenu, Organization, Reception, Utils;

// 2. Задаем общее пространство имен для всех документов
NAMESPACE Documents;

// 3. Наследуем класс заголовка расхода от класса Header
CLASS HeaderExpenses 'Расходы':Header;

// 3. Наследуем класс строки расхода от класса Row
CLASS Expenses 'Строки расходов':Row;

// 4. Добавляем свойства, которых нет в базовом классе
// так как товар и цена берутся из прихода, то нужна композиция
reception 'ID' = DATA Reception (Expenses) NONULL ;
name 'Наименование' (Expenses o) = name(reception(o));
price 'Цена' (Expenses o) = round(1.1 * price(reception(o)),2); // 1.1 - оптовая надбавка

// 4. Композиция
customer = DATA Organization (HeaderExpenses);
organizationName 'Покупатель' (HeaderExpenses o) = name(customer(o));

// 4. Новые свойства
nameProduct 'Наименование' (Expenses o) = nameProduct(reception(o));
nameUnit 'Ед. изм.' (Expenses o) = nameUnit(reception(o));

// 5. Контроль данных и вычисление остатка
allExpenses  (Reception r) = GROUP SUM quantity(Expenses e) BY reception(e) MATERIALIZED;
balance 'Остаток' (Reception r) = quantity(r) (-) allExpenses(r);
CONSTRAINT balance(reception(Expenses o)) < 0 MESSAGE 'Расход превышает остаток';
CONSTRAINT SETCHANGED(customer(HeaderExpenses o)) AND NOT organizationType(customer(o)) =
OrganizationType.customer
   CHECKED BY customer[HeaderExpenses] MESSAGE 'Для расходной накладной организация должна быть
покупателем';

// 6. Формы редактирования и отображения, дизайн, 7. меню
@forms(Expenses, 'Расход', 'Расходы', 'Покупатель');

// 6. форма выбора остатков
FORM listReception 'Остатки'
   OBJECTS r = Reception
   PROPERTIES (r) READONLY name, nameProduct, nameUnit, price, quantity, balance
   LIST Reception OBJECT r
   FILTERS balance(r)
;

// ограничиваем ширину колонки ед. изм.
DESIGN listReception {PROPERTY (nameUnit(r)) {charWidth=10;} }
```

Примечание:

- свойство name должно быть удалено (выделено красным) после перемещения данных.

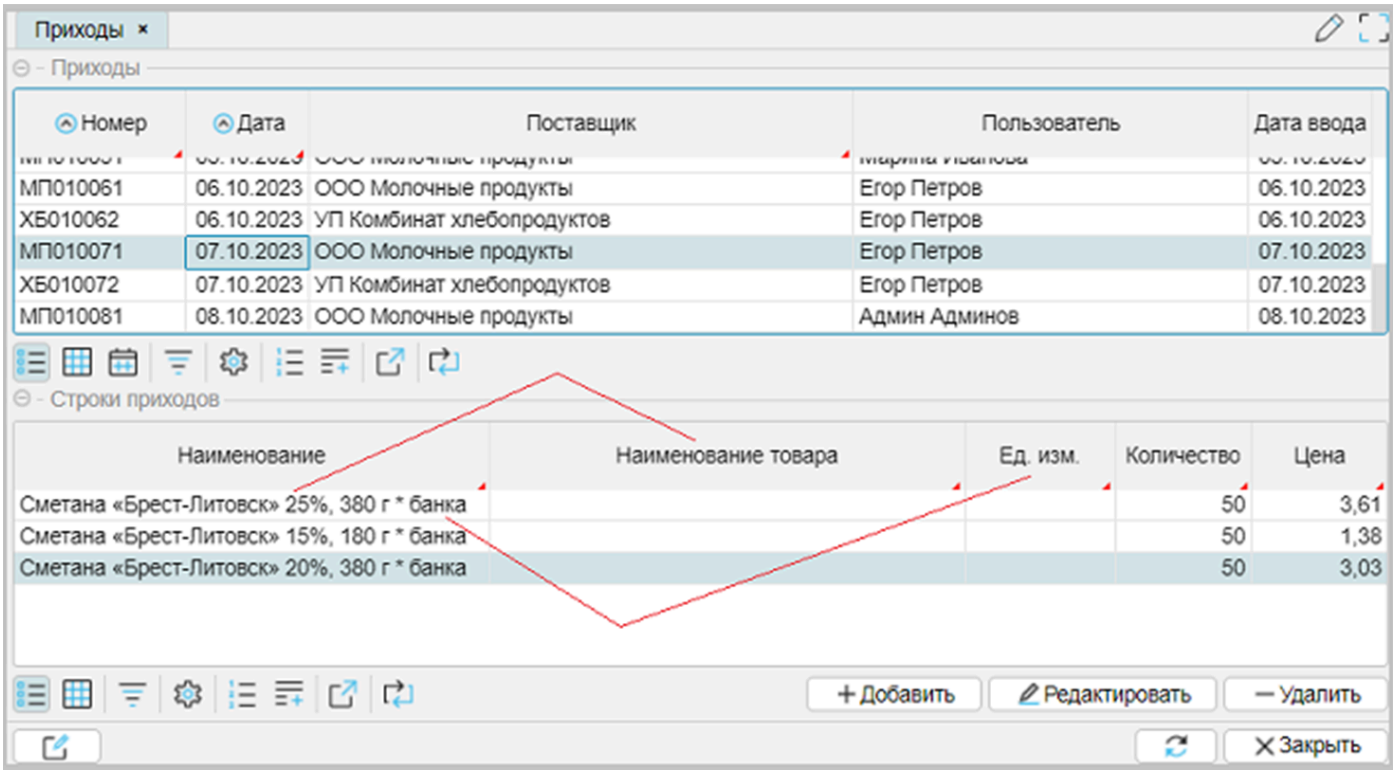
Заполнение справочников

Если бы данные вносились в изначально *правильно спроектированное приложение*, то рассматриваемых ниже вопросов не возникло бы:

- кто-то бы заполнил справочник товарных групп, исходя из специфики работы товарной базы,
- кто-то бы заполнил справочник единиц измерения возможными вариантами,
- кто-то бы по мере внесения приходных накладных добавлял информацию в справочник товаров, связывая товары с товарными группами.

Как указывалось выше, структура приложения при рассмотрении работы на платформе IsFusion, для упрощения изложения материала, намеренно была выбрана денормализованной (плоские таблицы, минимум справочников). Это позволило рассмотреть многие вопросы создания приложений не особо "погружаясь" в архитектуру самого приложения. По мере развития приложения, денормализованная структура базы данных должна быть приведена к нормализованному виду использованием справочников и документов типа "шапка-строки".

Но возникла некоторая проблема – это наличие уже введенных данных, которые необходимо сохранить.



Вносить данные заново – плохое решение. Это при том, что информации, которая присутствует в документах, почти достаточно, чтобы заполнить новые справочники.

Некоторый анализ характера введенных данных дает представление:

- о возможных вариантах товарных групп,
- об использовании символа "*" в наименовании товара для разделения наименования товара и единиц измерения.

Что нужно сделать:

- заполнить самостоятельно справочник товарных групп,

- из свойства name можно извлечь:
    - наименование товара для заполнения справочника товаров,
    - наименование единиц измерения для заполнения справочника единиц измерения;

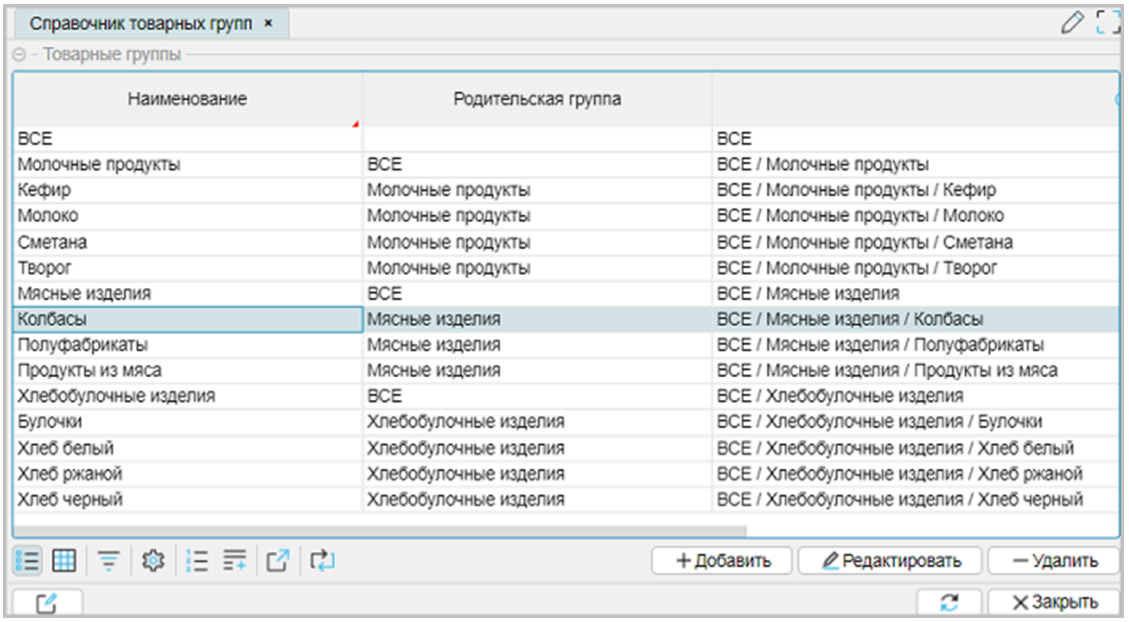
  поэтому, можно автоматизировать процесс заполнения справочников товаров и единиц измерения
- связать справочник товаров и единиц измерения со строками прихода,
- связать справочник товаров с товарными группами,
- завершить работу удалением свойства name из приходных и расходных документов как утратившего свою актуальность.

Часть работ можно автоматизировать, что можно рассматривать как миграцию (перенос) данных.
Автоматизация предполагает выполнение обработки данных, используя программный код.

См. также
Нормальная форма
Денормализация

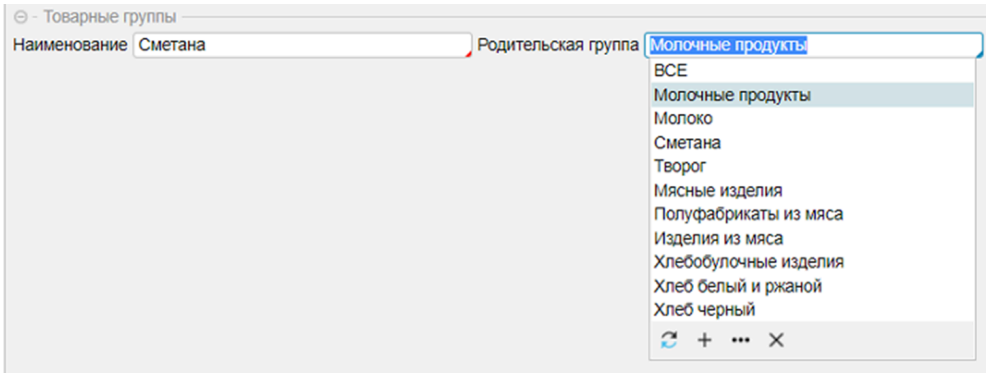
Заполнение товарных групп

На основе анализа введенных товарных строк можно определить список существующих товарных групп. При этом вводимые группы должны отражать вложенность. После ввода данных форма отображения должна содержать следующую информацию:



При этом последняя колонка табличной части формы отображения через слэш (символ /) отражает фактическую вложенность групп одна в другую. Отображение групп в виде дерева будет рассмотрено чуть позже.

Форма редактирования справочника товарных групп имеет вид:



Примечание:

- для наименования "ВСЕ" родительская группа не задается (это вершина дерева).

См. также

Статья. Работа с иерархиями в lsFusion

Архив SQL:
examples\t303_заполнение_товарных_групп\sql

Список товарных групп:
examples\список_товаров_групп.xlsx

Миграция. Действия. Кнопки. Интерпретатор

Необходимо выполнить некоторые действия по обработке существующих данных и программно развести их по новым справочникам. Для того, чтобы выполнить обработку, необходимо определить:

- набор действий, которые будут ответственны за решение этой задачи,
- место в интерфейсе, откуда действие будет доступно.

Платформа lsFusion не оперирует понятиями процедур и функций, вместо этого используется понятие "действие". Любое действие может быть представлено выражением вида:

Имя_действия 'Название_действия' (список_параметров) { *строки_программного_кода: действия_и_операторы*}

Навскидку, действие можно определить в уже существующем меню "Движение" или добавить кнопку, вызывающую действие на какой-нибудь форме отображения. Но это не совсем верно, учитывая технологический характер операции, к тому же исполняемой однократно. Вообще, на платформе lsFusion за выполнение, в том числе технологических операций, отвечает раздел "Администрирование". Как правило, раздел "Администрирование" скрыт от других пользователей, что исключает его использование неопытными людьми. Для разовых операций подходит использование интерпретатора (меню: "Администрирование - Система - Интерпретатор").

Рассмотрим два способа выполнения задачи:

1. кнопкой на форме "Миграция" (меню "Администрирование - Приложение - Миграция"),
2. используя интерпретатор.

Алгоритм обработки:
Строки товаров (класс "Reception") имеют свойство name, в котором наименование товара и единица измерения отделены друг от друга через символ "*":

Наименование	Наименование товара	Ед. изм.
Сметана «Брест-Литовск» 25%, 380 г * банка		
Сметана «Брест-Литовск» 15%, 180 г * банка		
Сметана «Брест-Литовск» 20%, 380 г * банка		

Поэтому, если организовать цикл FOR по всем строкам прихода, можно извлечь из старого наименования товара (свойство name) значения соответствующие названию товара и названию единиц измерения. Используя GROUP MAX по условию < свойство name нового справочника = выделенное (getWord) название товара (единицы измерения) >, можно получить ссылку на позицию в новом справочнике. Если позиция не существует, то используя оператор NEW или блок NEW оператора FOR, позицию можно добавить в соответствующий справочник. После получения ссылки на позицию в справочнике заполнить свойства product и unit в строке прихода. Аналогично можно связать товарные позиции в справочнике товаров со справочником товарных групп (Group). В конце обязательно необходимо использовать оператор APPLY, для того чтобы сделанные изменения обрели силу.

1. Форма "Миграция"

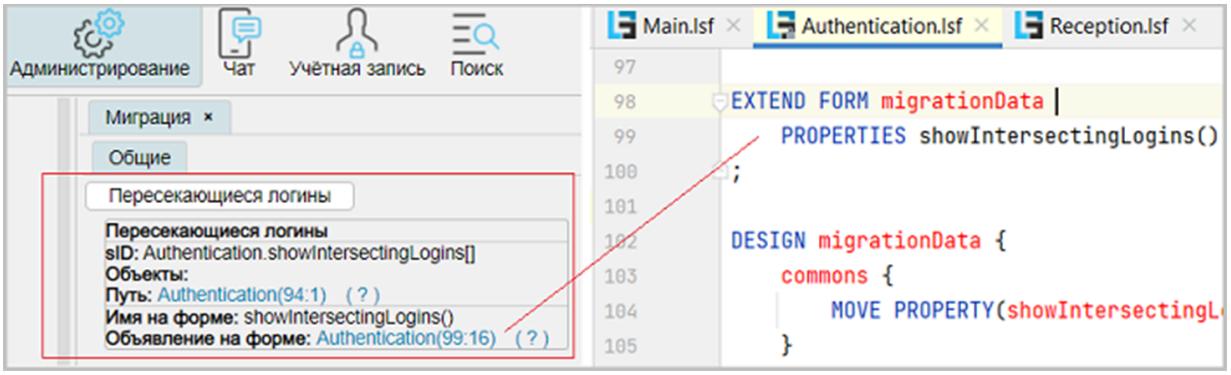
Новый модуль не создаем, обновляем пустое содержимое модуля "Main".

```
MODULE Main;

REQUIRE Reception, Utils;

migrate 'Перенос данных' () {
   ASK 'Вы действительно хотите выполнить Перенос данных?' DO {
       FOR [GROUP MAX Reception r BY trim(getWord(name(r), '*', 2))](STRING[100] v) NEW u =
Unit DO {
           name(u) ← v;
       }
       FOR [GROUP MAX Reception r BY getWord(name(r), '*', 1)](STRING[100] v) NEW p = Product
DO {
           name(p) ← v;
       }

       group(Product p) ← (
           CASE
               WHEN istartsWith(name(p), 'Булочка') THEN Group.unique('БУЛОЧКИ')
               WHEN istartsWith(name(p), 'Мясо') THEN Group.unique('ПОЛУФАБРИКАТЫ')
               WHEN istartsWith(name(p), 'Из мяса') THEN Group.unique('ПРОДУКТЫ ИЗ МЯСА')
               WHEN istartsWith(name(p), 'Колбаса') THEN Group.unique('КОЛБАСЫ')
               ELSE OVERRIDE (GROUP MAX Group g IF istartsWith(name(p), name(g))),
Group.unique('ВСЕ')
           ) WHERE p IS Product AND NOT group(p);
       APPLY;

       unit(Reception r) ← Unit.unique(upper(trim(getWord(name(r), '*', 2)))) WHERE r IS
Reception AND NOT unit(r);
       product(Reception r) ← Product.unique(upper(getWord(name(r), '*', 1))) WHERE r IS
Reception AND NOT product(r);

       APPLY;
       IF NOT canceled() THEN
           MESSAGE 'Операция выполнена';
       ELSE
           MESSAGE 'Операция не выполнена\nИз-за существующих ограничений\nВозможно, миграция
выполняется повторно';
   } ELSE {
       MESSAGE 'Выполнение отменено';
   }
}

EXTEND FORM migrationData PROPERTIES migrate();

DESIGN migrationData {
   commons {
       MOVE PROPERTY(migrate());
   }
}
```

Важные замечания:

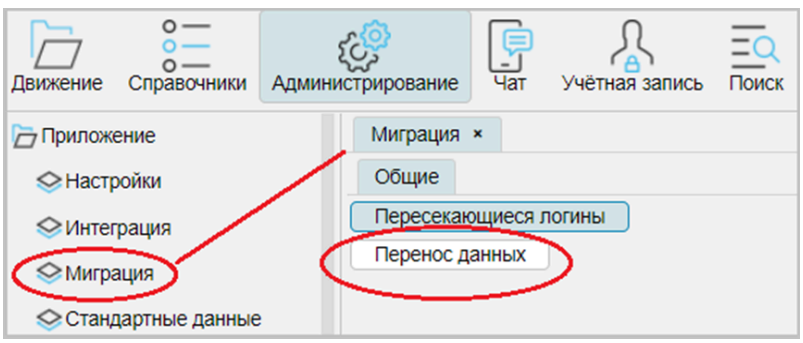
- Поместив действие migrate() в блок формы PROPERTIES создали кнопку на форме. Синтаксис такой же, как и для свойства. Платформа lsFusion "сама понимает", где определено свойство или действие. При этом блок PROPERTIES может быть определен с параметрами или без, например, < PROPERTIES(r) name > или < PROPERTIES name(r) >. Если параметры не определены в блоке PROPERTIES, то они задаются в свойстве или действии.

- Место вызова действия migrate() на форме "Миграция" определили с помощью подсветки свойств формы:



  Используя подсветку, узнали имя модуля, имя формы – migrationData, добавили свое действие onRemoveData и поправили дизайн формы по аналогии с кнопкой "Пересекающиеся логины".
- Количество уникальных наименований товаров и единиц измерения меньше, чем количество строк приходов. Поэтому используется два FOR для заполнения уникальными значениями справочников единиц измерения и товаров соответственно. На примере заполнения справочника единиц измерения: инструкция < [GROUP MAX Reception r BY trim(getWord(name(r), '*', 2))](STRING[100] v >.
- Выражение < group(Product p) > фактически изменит значение свойства, указанного в блоке WHERE. Аналогично работают заполнение ID единиц измерения и справочника товаров для строк прихода (подробнее здесь).
- Системное свойство canceled() будет NULL, если изменения удалось применить в базу данных. При отмене операции применения изменений в базу данных (APPLY) в результате выполнения оператора CANCEL или нарушения какого-либо ограничения, в локальное свойство canceled() записывается TRUE. Это возможно, если пытаться выполнять действие migrate() повторно. В этом случае циклы FOR будут пытаться записать значения в справочники единиц измерений (товаров), при этом будет нарушаться уникальность наименований. Варианты для многократного использования, *хотя это не логично для одноразовой операции*, могут заключаться в дополнительном условии для FOR или предварительной очистке справочников единиц измерения и товаров перед началом операции.

После перезапуска сервера приложений в пункте меню "Администрирование - Приложение - Миграция" на форме появилась кнопка "Перенос данных":



После выполнения операции строки прихода примут вид:

Наименование	Наименование товара	Ед. изм.
Сметана «Брест-Литовск» 25%, 380 г * банка	Сметана «Брест-Литовск» 25%, 380 г	банка
Сметана «Брест-Литовск» 15%, 180 г * банка	Сметана «Брест-Литовск» 15%, 180 г	банка
Сметана «Брест-Литовск» 20%, 380 г * банка	Сметана «Брест-Литовск» 20%, 380 г	банка

Несложно заметить, что наименование товара и единицы измерения легли туда, куда надо. Стоит обратить внимание на справочник единиц измерения, который содержит неповторяющиеся значения всех вариантов единиц измерения, встречающихся в строках прихода. Справочник товаров связан со своими товарными группами.

2. Используя интерпретатор

Как пользоваться и пример использования интерпретатора для выполнения операции описан в разделе "Интерпретатор" (см. Выполнить скрипт).

Можно отметить, что содержимое действия migate() при вызове формы и использовании интерпретатора один-в-один совпадают. Отсюда следует, что можно всегда отладить скрипт для интерпретатора, как программный код на кнопке или пункте меню, и выполнить отлаженный код на рабочей базе клиента, используя интерпретатор.

См. также
Документация. Действия
Документация. Оператор FOR
Документация. Оператор GROUP
Документация. Оператор MAX
Документация. Оператор CHANGE
Документация. Оператор IF
Документация. Оператор CASE
Документация. Оператор ASK
Документация. Оператор NEW
Документация. Оператор APPLY
Документация. Оператор CANCEL
Документация. Оператор RETURN
Интерфейс. Табличный процессор
Модуль Utils
Примеры. Оператор FOR

Пример кода:

examples\t304_действия_перенос_данных\src_name
код содержит старое наименование name и код переноса данных

examples\t304_действия_перенос_данных\src
код не содержит старое наименование name и код переноса данных

Архив SQL:
examples\t304_действия_перенос_данных\sql
содержит результат переноса данных

Список товарных групп:
examples\список_товаров_групп.xlsx

Деревья. Журнал остатков

В соответствии с постановкой задачи необходимо отобразить журнал текущих остатков в виде древовидного списка. Некоторые действия для создания древовидного списка уже были выполнены при создании справочника товарных групп.

Фактически все готово:

- создан и заполнен справочник товарных групп,
- создан и заполнен справочник товаров, связанный со справочником товарных групп,
- существуют документы приходов, связанные со справочником товаров – они же строки остатков в идеологии партионного учета,
- присутствует расчет остатков.

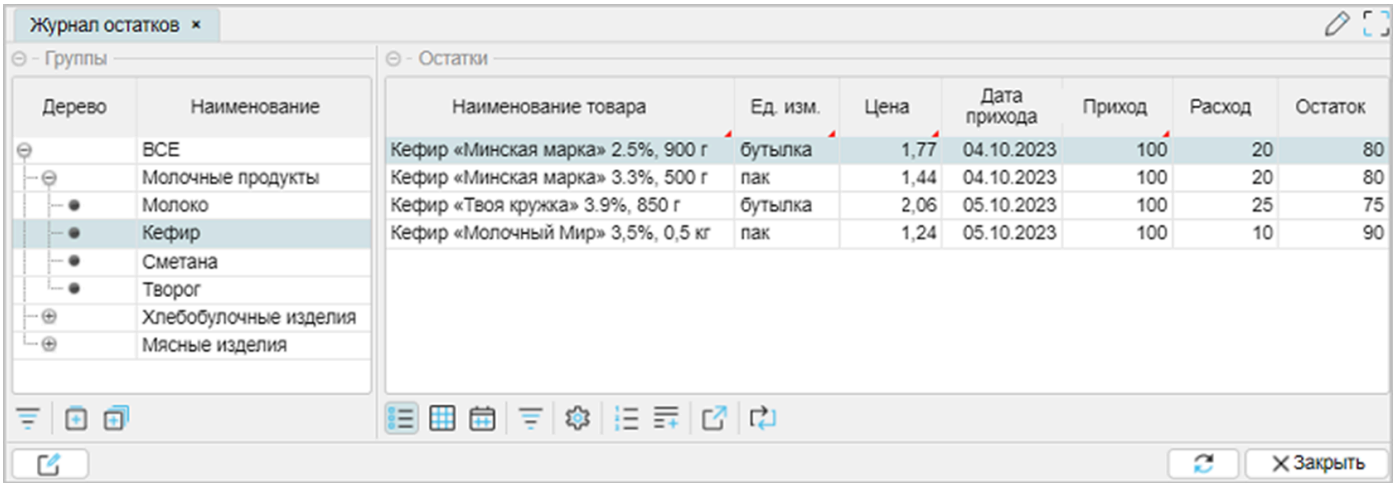
Осталось визуализировать имеющиеся данные в виде:

- дерева товарных групп, которое управляет подмножеством отображаемых товаров,
- формы остатков в составе свойств: название и единицы измерения товара, цена, дата и количество прихода, общий расход, остаток.

Для этого создадим в папке "Package documents" модуль "Balance":

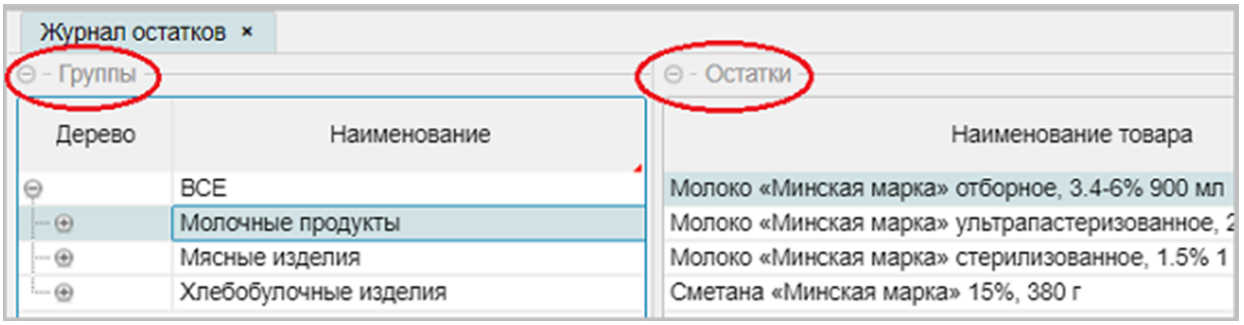
```
MODULE Balance;

REQUIRE Expenses, Group, AppMenu;

date 'Дата прихода' (Reception o) = date(header(o)) CHARWIDTH 10;

FORM viewBalance 'Журнал остатков'
    TREE groups g = Group PARENT parent(g)
    PROPERTIES READONLY name(g)
    OBJECTS r = Reception
    PROPERTIES (r) READONLY nameProduct, nameUnit, price, date, quantity, allExpenses, balance
    FILTERS level(group(product(r)),g)
;

DESIGN viewBalance {
    OBJECTS {
        NEW pane {
            fill = 1;
            horizontal = TRUE ;
            MOVE BOX(TREE groups) { fill=1; caption='Группы'; }
            MOVE BOX(r) { fill=20; caption='Остатки'; }
        }
    }
    PROPERTY (nameUnit(r)) { charWidth=10; }
    PROPERTY (quantity(r)) { caption='Приход'; charWidth=10; }
    PROPERTY (allExpenses(r)) { caption = 'Расход'; charWidth=10; }
    PROPERTY (balance(r)) { charWidth=10; }
}

NAVIGATOR {
    documents {
        NEW FORM viewBalance;
    }
}
```

После старта сервера приложений журнал остатков должен принять вид:



Особенности:

- На форме отображения строка < TREE groups g = Group PARENT parent(g) > определяет отображение дерева, < name(g) > – название группы, связанное с деревом.
- Фильтр < FILTERS level(group(product(r)),g) > связывает дерево с товарами. При этом подмножество товаров зависит от уровня группы: чем выше уровень, тем большее подмножество отображается. Рассмотрим на примере: можно было бы определить фильтр, как < FILTERS group(r) = g >, по аналогии с другими, ранее рассмотренными, формами. В этом случае, если в группу "ВСЕ" входит группа "Молочные продукты", а в "Молочные продукты" входит группа "Молоко", то только для группы "Молоко" будут высвечиваться товары, во всех остальных случаях список будет пуст. При использовании < FILTERS level... > для группы "ВСЕ" будут высвечиваться все товары, для группы "Молочные продукты" будут высвечиваться все молочные продукты, а для группы "Молоко" только товары типа молоко.
- DESIGN При наличии нескольких объектов на форме, по умолчанию табличные (панельные) части располагаются друг за другом вертикально, в порядке описания объектов. Поэтому, чтобы расположить контейнеры объектов горизонтально, необходимо исправить дизайн формы: создать новый контейнер (pane), задать для свойства horizontal = TRUE, переместить туда контейнеры BOX(TREE groups) и BOX(r).
- DESIGN Изменим отводимую площадь для контейнеров BOX для дерева и для табличной части остатков, используя свойство fill (см. также flex).
- DESIGN Изменили заголовки контейнеров BOX для дерева и для табличной части, вместо тех, которые бы выводились по умолчанию, на "Группа" и "Остатки".
-



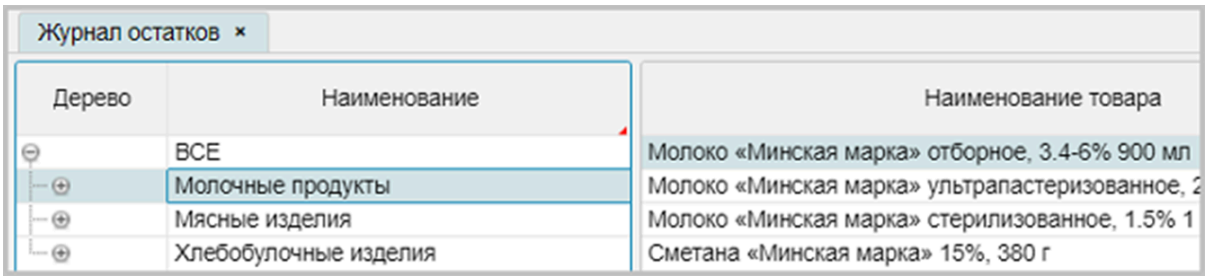
- DESIGN Для свойство quantity (количество) изменили заголовок на "Приход", чтобы все количественные колонки выглядели одинаково: "Приход", "Расход", "Остаток". Для свойства nameUnit ограничили ширину отображаемой колонки.

- Свойство date (дата прихода) выразили через композицию. При этом определили ширину отображения в 10 символов, поэтому заголовок колонки автоматически отображается в две строки.

Важное замечание 1

- DESIGN Если заголовки контейнеров не нужны, то их можно скрыть в DESIGN формы, определив их значение, как NULL:
-

```
MOVE BOX(TREE groups) { fill=1; caption=NULL; }
MOVE BOX(r) { fill=20; caption=NULL; }
```



Важное замечание 2

- DESIGN Если возникла необходимость вывести заголовок колонки с расширенным описанием, то максимально возможное видимое значение будет 3 строчки, если не менять размер шрифта.

```
PROPERTY (date(r)) { caption = 'Дата прихода из шапки приходной накладной';charWidth=10;
}
```

Чтобы высота колонки изменилась автоматически под весь текст надо изменить для GRID значение < headerHeight = -1 >:

```
GRID(r) { headerHeight=-1; }
```

См. также

Статья. Работа с иерархиями в lsFusion
Документация. Инструкция FORM. Блоки формы
Документация. Инструкция DESIGN. Свойство компонентов

Пример кода:
examples\t305_деревья_журнал_остатков\src

Краткие итоги

После рассмотрения вопроса "Деревья. Журнал остатков" все вопросы, связанные с постановкой задачи темы 3, можно считать рассмотренными полностью. Анализ рассмотренных вопросов темы 3 позволяет утверждать, что значительная часть программного кода базировалась на знаниях, полученных при рассмотрении темы 2.

В ходе изучения материала были рассмотрены вопросы:

- документы типа "шапка-строки",
- расширение работы со справочниками,
- работа с деревьями,
- понятие действий,
- кнопки на форме,
- доработка форм раздела "Администрирование",
- работа с интерпретатором.

Хотя постановку задачи можно считать выполненной, тем не менее, как это реально бывает в жизни любой программы, приложение будет требовать улучшений со стороны пользователей, работающих с ним.

Тема 4. Расширяем возможности

Рассматриваемые вопросы:

- Вкладки на форме
- Постановка задачи: Расчет итогов
  - Расчет итогов. Использование PARTITION, SUM, MAX
  - Расчет скидок. Использование PARTITION
  - Краткие итоги
- Фильтры
  - Фиксированные фильтры
  - Фильтр по отмеченным
  - Группа фильтров
- События свойств формы. Оператор DIALOG
- Постановка задачи: Картинки на форме
  - Доработка справочника товаров
  - Доработка документов

Вкладки на форме

Предположим, что, посмотрев на журнал остатков, клиент попросил доработать журнал так, чтобы видеть аналогичную информацию по остаткам, но не в разрезе товарных групп, а в разрезе поставщиков товаров. Можно, конечно, создать еще одну форму и еще один пункт меню, так чтобы получилось:

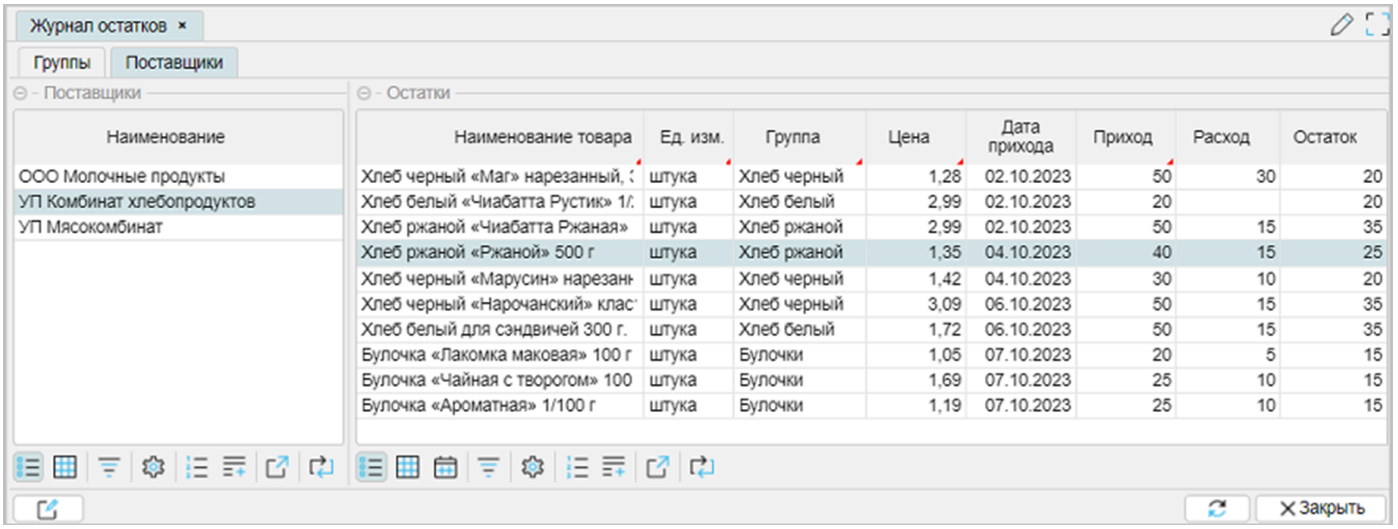
- Остатки по товарным группам,
- Остатки по поставщикам.

Но это неудобно и не функционально. Намного лучше, когда схожая информация находится рядом. Поэтому, можно организовать на форме вкладки, каждая из которых будет отвечать за свой разрез представления данных.

Делается это достаточно просто:
Необходимо доработать форму viewBalance модуля Balance, добавив туда пару объектов: справочник организаций (Organization) и табличную часть, отражающую остатки. Используя фильтр привязать остатки к организациям. В дизайнере формы создать новый контейнер, для которого установить свойство < tabbed = TRUE > – все контейнеры, входящие в этот контейнер, будут отражаться на форме как отдельные вкладки. Исправления добавим в конец модуля Balance:

```
nameGroup 'Группа' (Reception o) = name(group(product(o)));
organization 'ID организации' (Reception o) = supplier(header(o));

EXTEND FORM viewBalance
    OBJECTS o = Organization
    PROPERTIES (o) READONLY name
    OBJECTS r2 = Reception
    PROPERTIES (r2) READONLY nameProduct, nameUnit, nameGroup, price, date, quantity,
                                                                      allExpenses, balance
    FILTERS organizationType(o) = OrganizationType.supplier
    FILTERS organization(r2) = o
;

DESIGN viewBalance {
    pane { caption = 'Группы'; }
    OBJECTS {
        NEW tabbed {
            tabbed = TRUE; // Задает вкладки
            fill = 1;
            MOVE pane; // перемещаем контейнер дерева и остатков
            NEW  pane2 { // контейнер организаций и остатков
                horizontal = TRUE;
                caption = 'Поставщики';
                MOVE BOX(o) { fill=1; caption='Поставщики'; }
                MOVE BOX(r2) { fill=20; caption='Остатки'; }
            }
        }
    }
    PROPERTY (nameUnit(r2)) { charWidth=10; }
    PROPERTY (quantity(r2)) { caption='Приход'; charWidth=10; }
    PROPERTY (allExpenses(r2)) { caption = 'Расход'; charWidth=10; }
    PROPERTY (balance(r2)) { charWidth=10; }
}
```

После старта сервера приложений журнал остатков должен выглядеть следующим образом:



Краткие итоги:

- Вкладки создаются с помощью дизайна форм DESIGN, назначением "главного" контейнера tabbed (*имя любое*) с опцией < tabbed = TRUE >. Все последующие контейнеры, входящие в "главный" контейнер, будут образовывать вкладки.
- DESIGN Контейнер pane (дерево и остатки) просто переместили инструкцией MOVE в контейнер tabbed.
- DESIGN Создали новый контейнер pane2 для связанной пары классов организаций и остатков внутри "главного" контейнера tabbed, в который переместили табличные части контейнеров BOX(o) и BOX(r2). Изменили отводимую площадь для контейнеров BOX используя свойство fill. Изменили заголовки контейнеров BOX на "Поставщики" и "Остатки".
- DESIGN Для свойства quantity(r2) (количество) изменили заголовок на "Приход", чтобы все количественные колонки выглядели одинаково: "Приход", "Расход", "Остаток". Для свойства nameUnit(r2) ограничили ширину отображаемой колонки.
- Используется композиция nameGroup – название товарной группы в строках остатка.
- Композиция organization необходима, чтобы связать строки остатков условием фильтра с организациями.

См. также

Документация. Инструкция FORM. Блоки формы

Документация. Инструкция DESIGN. Свойство компонентов

Пример кода:

examples\t401_вкладки_на_форме\src

Постановка задачи: Расчет итогов

Как это обычно бывает в жизни любой программы, заказчика все устраивает, но всегда есть новые пожелания.

Для всех документов:

1. Нужны итоги по документу: количество, сумма, количество строк документа.
2. Нумерация строк в документе.
3. Нужна информационная панель "Итоги по документу" для отражения итогов по документу.
4. Нужна информационная панель "Итоги за день" для отражения суммового итога за день.

Для документов расхода:

1. При отпуске товаров покупателям (магазинам) возможна фиксированная скидка до 5% на сумму документа. Размер скидки для покупателей разный.
2. В информационных панелях должны быть отражены значения сумм со скидкой по документу и за день.
3. Отпуск со скидкой должен быть отражен в строках документа, как значение стоимости со скидкой по каждой позиции документа.
4. В табличной части заголовков расхода желательно выделить документы со скидкой.

Решение:

- для всех документов: Расчет итогов
- для документов расхода: Расчет скидки

Расчет итогов

Определим вычисляемые свойства:

- index – номер по порядку для строк накладной;
- cost – стоимость как количество (quantity) * цену (price);
- amount – итоговая сумма по накладной;
- count – общее количество натуральных единиц по накладной;
- cntpos – количество позиций в накладной;
- amountDay – итоговая сумма за день по всем накладным.

Создание новых свойств будем производить, используя операторы GROUP и PARTITION.

- Оператор GROUP создает свойство, которое разбивает наборы объектов на группы, при этом *для каждой группы вычисляется значение агрегирующей функции*. Если разбиение на группы (блок BY) не используется, то результат вычисляется для всех наборов объектов. Для решения поставленных задач будет использоваться агрегирующие функции:
  - SUM для вычисления общей суммы по накладной – свойство amount.
  - SUM для вычисления общего количество натуральных единиц по накладной – свойство count.
  - SUM для вычисления максимального количества позиций в накладной – свойство cntpos.
- Оператор PARTITION создает свойство, которое разбивает наборы объектов на группы, при этом *для каждого набора объектов в группе вычисляется значение агрегирующей функции*. Важным является использование блока ORDER, которое влияет на результат. При использовании блока ORDER для каждой группы результат рассчитывается нарастающим итогом. Если блок ORDER не используется, то результат для каждого набора объектов будет общим. Для решения поставленных задач будет использоваться агрегирующие функции:
  - SUM с блоком ORDER для вычисления порядкового номера строк документа – свойство index. Порядковый номер для каждого набора рассчитывается нарастающим итогом по строкам накладной (1,2,..n).
  - SUM без использования блока ORDER для вычисления общего итога по всем накладным за день – свойство amountDay. Так как блок ORDER не используется, то результат для каждой накладной за одну и ту же дату будет одним и тем же значением свойства amountDay.

Так как требования по итогам общие для документов прихода и расхода, выполняем доработку на уровне метакода модуля Documents. Чтобы не перегружать основной метакод forms, касающийся работы с формами, создадим новый метакод totalValues:

```
META totalValues(object)
   index 'Nпп' (object r) = PARTITION SUM 1 ORDER r BY header(r) CHARWIDTH 3;
   cost 'Стоимость' (object r) = NUMERIC[10,2](round(quantity(r) * price(r),2)); // по строке
   amount 'Сумма' (Header##object h) = GROUP SUM cost(object r) IF header(r) = h;
   count 'Количество' (Header##object h) = GROUP SUM quantity(object r) IF header(r) = h;
   cntpos 'Всего позиций' (Header##object h) = GROUP SUM 1 IF header(object r) = h;
   amountDay 'Сумма' (Header##object h) = PARTITION SUM(amount(h)) BY date(h);

  // форма отображения
   EXTEND FORM view##object
       PROPERTIES (h) READONLY amount AFTER organizationName(h)
```

```
        PROPERTIES (r) READONLY index FIRST, cost
        OBJECTS h2 = Header##object PANEL // информационное табло
        PROPERTIES(h2) READONLY amount, count, cntpos, amountDay, date
        FILTERS h2 = h;

    // исправляем дизайн: итоги по документу, итоги за день
    DESIGN view##object {
        PANEL {
            NEW pane {
                horizontal = TRUE;
                fill = 1;
                NEW totalCurrent {
                    horizontal = TRUE;
                    caption = 'Итоги по документу';
                    MOVE PROPERTY (amount(h2)) { background = RGB(204, 204, 255); }
                    MOVE PROPERTY (count(h2)) { background = RGB(204, 204, 255); }
                    MOVE PROPERTY (cntpos(h2)) { background = RGB(204, 204, 255); }
                }
                NEW totalDay {
                    horizontal = TRUE;
                    caption = 'Итоги за день';
                    MOVE PROPERTY (date(h2)) { background = RGB(204, 255, 204); }
                    MOVE PROPERTY (amountDay(h2)) {
                        background = RGB(204, 255, 204);
                        marginLeft = 20;
                    }
                }
            }
        }
    }
END;
```

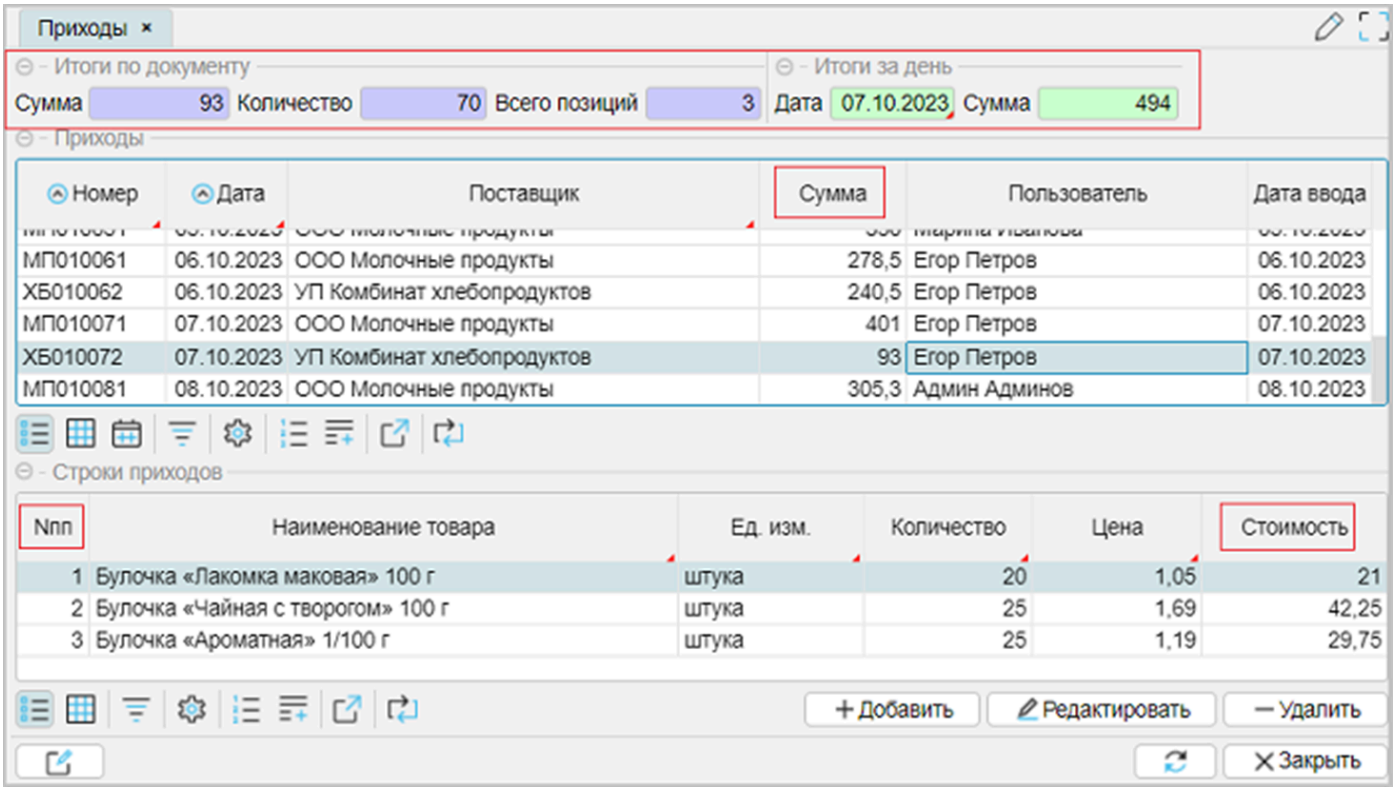
Соответственно доработаем модуль Reception – вызовем метакод:

```
// Расчет итогов
@totalValues(Reception)
```

Доработаем также модуль Expenses – вызовем метакод:

```
// Расчет итогов
@totalValues(Reception)
```

После старта сервера приложений формы отображения в приходах (расходах) примут следующий вид (на примере формы отображения приходов):



Особенности:

- Вычисляемые свойства могут быть определены на основе других вычисляемых свойств, например:

```
cost 'Стоимость' (object r) = NUMERIC[10,2](round(quantity(r) * price(r),2)); //по строке
amount 'Сумма' (Header##object h) = GROUP SUM cost(object r) IF header(r) = h;
```

- EXTEND FORM Отображение новых свойств в нужных местах с использованием ключевых слов AFTER, FIRST:

```
    PROPERTIES (h) READONLY amount AFTER organizationName(h)
    PROPERTIES (o) READONLY index FIRST, cost
```

- EXTEND FORM Создание общей информационной панели с итогами, после чего в DESIGN панель будет разбиваться на "Итоги по документу" и "Итоги за день":

```
   OBJECTS h2 = Header##object PANEL // информационная панель
   PROPERTIES(h2) READONLY amount, count, cntpos, amountDay, date
```

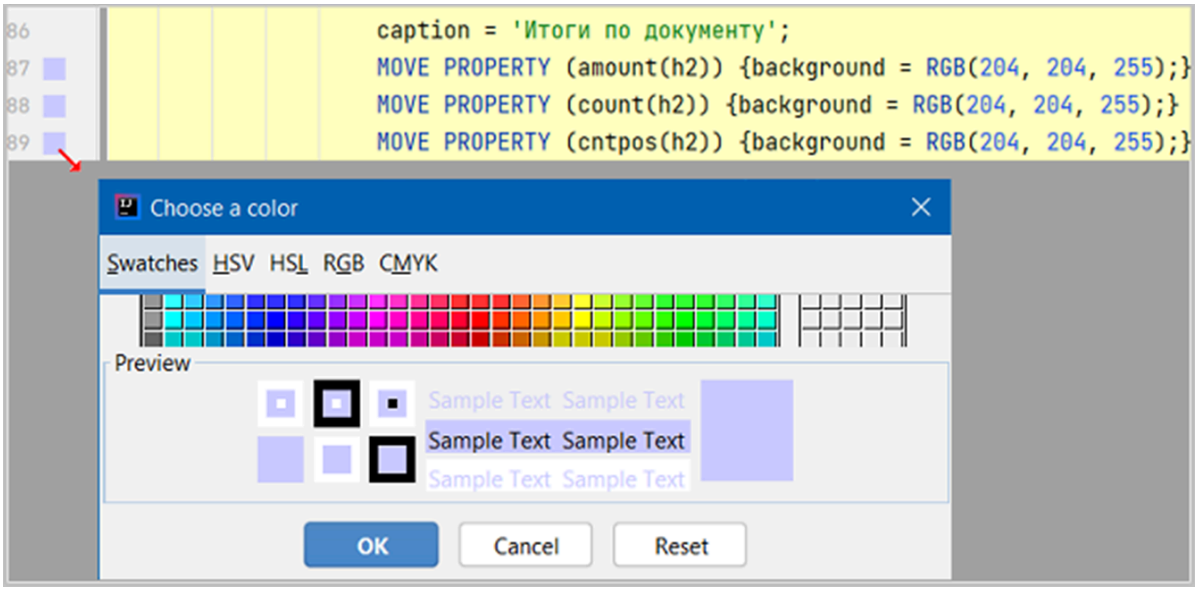
- DESIGN Изменение порядка следования свойств в контейнере информационной панели:

```
   MOVE PROPERTY (date(h2)) {background = RGB(204, 255, 204);}
   MOVE PROPERTY (amountDay(h2)) {background = RGB(204, 255, 204);marginLeft = 20;}
```

  Хотя в PROPERTIES определен порядок < amountDay, date >, но использование MOVE PROPERTY задает новый порядок < date, amountDate >.

- DESIGN Помощь в выборе цвета. Для свойств информационной панели задан фон отображения с использованием свойств компонента < background >. Вообще, когда идет упоминание о цвете (свойство, дизайн, форма) в редакторе IDEA, на служебном поле,

где отображается нумерация строк, высвечивается квадратик выбранного цвета. Двойной клик на квадратике вызовет форму выбора цвета, которая автоматически после выбора, выполнит изменения в программном коде - заполнит RGB. Квадратик всегда отображается выбранным цветом RGB.



См. также

Документация. Оператор PARTITION
Документация. How-to: PARTITION
Документация. How-to: GROUP SUM
Документация. How-to: GROUP MAX/MIN/AGGR

Расчет скидок

Особенности постановки задачи:

- Скидка дается на документ в целом. При этом в строках документа должно отражаться значение стоимости со скидкой. Естественно, суммы должны совпадать: сумма со скидкой на документ и сумма всех стоимостей со скидкой по строкам документа.
- Значение скидки должно быть учтено в информационной панели "Итоги по документу" как "Сумма со скидкой".
- Значение скидки должно быть учтено в информационной панели "Итоги за день" как "Сумма со скидкой".

Может сложиться мнение, что реализация потребует много затрат.
На самом деле все решается достаточно просто, в том числе с использованием оператора PARTITION.

Что надо сделать:

- Определить место ввода скидки – пусть это будет справочник организаций; добавим свойство discount – процент скидки.
- Связать расчет итогов по документу (amountDiscount) со значением discount справочника организаций.
- Рассчитать итоги за день со скидкой (amountDiscountDay).
- Исправить дизайн формы.

В модуле Organization добавим свойство discount, создадим ограничение CONSTRAINT, исправим формы редактирования и отображения:

```
// доработка под скидку
discount 'Скидка' = DATA NUMERIC[6,2] (Organization);
CONSTRAINT discount(Organization o) > 5
   MESSAGE 'Недопустимый размер скидки\nРазмер скидки не более 5%';

EXTEND FORM editOrganization PROPERTIES (o) discount;
EXTEND FORM viewOrganization PROPERTIES (o) READONLY discount;
```

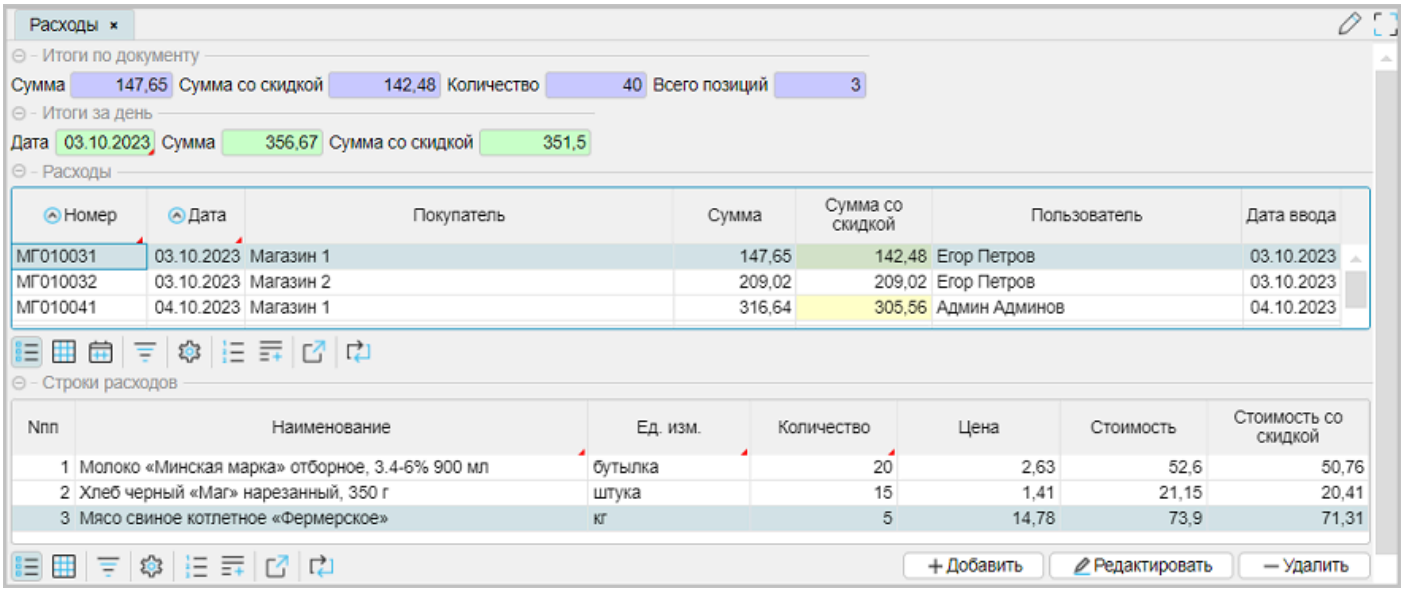
Для Магазина 1 (*после старта сервера приложений*) установим скидку 3.5 %.

Добавим в конец модуля Expenses (документы расхода) следующие изменения:

```
// Расчет итогов со скидкой
costDiscount 'Стоимость со скидкой' (Expenses o) =
                      round(cost(o) (-) (discount(customer(header(o)))/100) * cost(o),2);

amountDiscount 'Сумма со скидкой' (HeaderExpenses h) =
                      GROUP SUM costDiscount(Expenses r) BY header(r);

amountDiscountDay 'Сумма со скидкой' (HeaderExpenses h) =
                      PARTITION SUM(amountDiscount(h)) BY date(h);


EXTEND FORM viewExpenses
   PROPERTIES (h) READONLY amountDiscount AFTER amount(h)
   BACKGROUND RGB(255, 255, 204) IF discount(customer(h))
   PROPERTIES (h2) READONLY amountDiscount, amountDiscountDay
   PROPERTIES (r) costDiscount
;
```

```
DESIGN viewExpenses {
   totalCurrent {
       MOVE PROPERTY (amountDiscount(h2))
       AFTER PROPERTY (amount(h2)) { background = RGB(204, 204, 255); }
   }
   totalDay {
       MOVE PROPERTY (amountDiscountDay(h2))
       AFTER PROPERTY (amountDay(h2)) { background= RGB(204, 255, 204); }
   }
}
```

После старта сервера приложений форма отображения расходов примет вид:



Особенности:

- EXTEND FORM Использование BACKGROUND по условию < если скидка определена >.
- DESIGN Использование ключевых слов AFTER для перемещения свойств в нужную позицию в информационных панелях.
- DESIGN Из-за особенности выбранного размера окна браузера (не на весь экран), итоги в расходах отразились в две строки.

См. также

Документация. Оператор PARTITION

Документация. How-to: PARTITION

Краткие итоги

Не смотря на 8 пунктов в постановке задачи, реализация заняла немного времени, а ее решение заняло практически "ничего" за счет продвинутых возможностей платформы. Условно, вся реализация свелась к расчету отдельных свойств однострочными выражениями и небольшим изменениям в дизайнере форм.

При этом важно отметить:

- Эффективное использование метапрограммирования, которое убирает много лишней работы, выступая шаблоном при решении схожих задач с одной стороны, с другой стороны, совершенно не мешает дорабатывать отдельные подзадачи, которые не являются общими для всех.
- Эффективное использование оператора PARTITION для решения части поставленных задач.
- Использование GROUP SUM для расчета итоговых сумм.
- Управление цветами на форме как в описании формы, так и с помощью дизайнера форм.
- Управление порядком отображения свойств на форме с использованием ключевых слов AFTER, FIRST.

См. также

Пример кода:

examples\t402_итоги_скидки\src

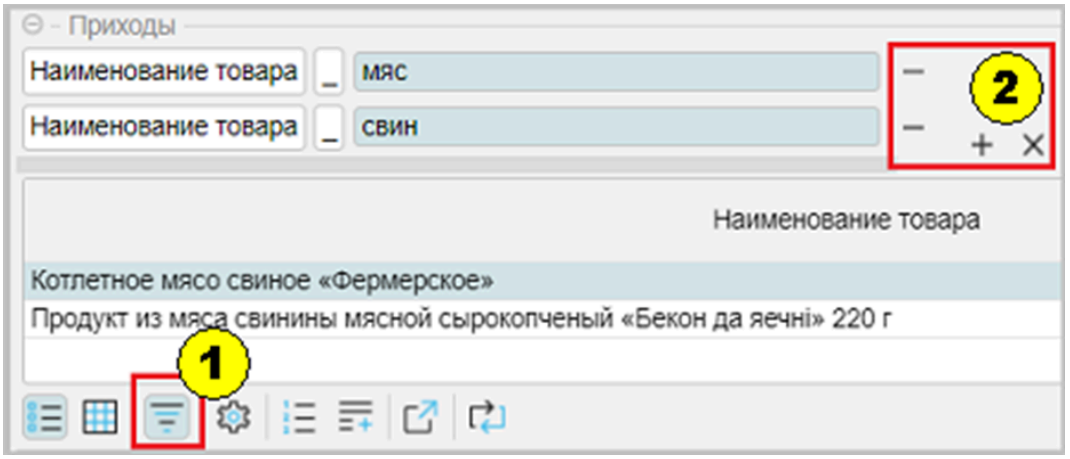
Архив SQL:

examples\t402_итоги_скидки\sql

Фильтры на форме

В платформе присутствует несколько типов фильтров:

- Пользовательские фильтры, предоставляемые табличным процессором. Табличная часть формы имеет фильтры [1] по любой отображаемой колонке со своими элементами управления [2].



При этом фильтров может быть несколько.
При использовании нескольких фильтров отбор строится с участием оператора AND для каждого нового фильтра.

- Использование на форме блока фиксированных фильтров FILTERS:
  - для связывания между собой двух (или более) сущностей, отображаемых на форме. Например, связывали выражением фильтра между собой шапку и строки документа:

```
FILTERS header(o) = h
```

  - для фильтрации по определенному значению. Например, при вызове справочника организаций, отдельно фильтровали Поставщиков или Покупателей:

```
FILTERS organizationType(o) = OrganizationType.supplier
```

Если продолжить рассмотрение блока фильтров FILTERS, то можно прийти к рассмотрению:

- Фиксированные фильтры
- Фильтр по отмеченным
- Группы фильтров

Важное замечание:

- Все фильтры: и фиксированные, и пользовательские (GRID), и группы фильтров – всегда работают совместно. То есть наличие одного типа фильтра не отменяет действие других типов фильтров.

Фиксированные фильтры

В платформе lsFusion кроме пользовательских фильтров, вызываемых из панели управления GRID, возможно использование фиксированных фильтров. Из ранее приводимых примеров фиксированные фильтры использовались для связывания сущностей или присвоения предопределенного значения. При этом, содержание фильтров определялось программно, и пользователь не мог влиять на их содержание. Можно расширить область применения фиксированных фильтров: предоставить пользователю визуальный интерфейс, который позволит управлять отбором данных. Иногда это может стать хорошей альтернативой пользовательским фильтрам GRID: отображение, управление, логика работы.

Постановка задачи:

- Создать группу фиксированных фильтров по следующим реквизитам:
  - Номер документа, number.
  - Дата документа, date.
  - Организация, organization.
  - Окончательная сумма документа: amount (приходы) и amountDiscount (расходы). При этом вводимое значение для фильтра вносится целочисленное (без дробной части).
- Необходимо визуально отобразить на форме группу фильтров ниже итогов в информационной панели.
- Необходимо добавить действие по очистке значений фильтров.
- Организовать фильтрацию сумм без копеек.

Что надо сделать:

- Определить новые локальные свойства и отобразить их на форме.
- Добавить действие, очищающее значения всех фиксированных фильтров.
- Доработать формы отображения (приходы/расходы), добавив фиксированные фильтры.
- Доработать дизайн форм.

Особенности реализации:

- Так как есть схожие документы (приходы/расходы), то в модуле Documents будет создан новый метакод для формирования пользовательских фильтров (filters).
- Так как окончательные суммы разнятся для приходов и расходов, то потребуется отдельная небольшая доработка модуля Expense.

Реализация:

- В модуле Documents создадим новый метакод filters:

```
META filters(object1, object2, title1)
   f##object1##Number   'Номер' = DATA LOCAL STRING[10];
   f##object1##Date     'Дата'  = DATA LOCAL DATE;
   f##object1##Amount   'Сумма' = DATA LOCAL INTEGER; // суммы без копеек
   object2 'ID организации' = DATA LOCAL Organization ();
   f##object2##Name title1 = name(object2());

   CONSTRAINT SETCHANGED(object2()) AND NOT organizationType(object2()) =
       OrganizationType.object2
       CHECKED BY object2[] MESSAGE 'Ограничение на тип организации';

   on##object1##Clear 'Очистить' () {
       f##object1##Number() ← NULL;
```

```
        f##object1##Date() ← NULL;
        f##object1##Amount() ← NULL;
        object2() ← NULL;
    }

    EXTEND FORM view##object1
        PROPERTIES() on##object1##Clear, f##object1##Number,
            f##object1##Date, f##object1##Amount, f##object2##Name
        FILTERS NOT f##object1##Number() OR number(h) = f##object1##Number(),
            NOT f##object1##Date() OR date(h) = f##object1##Date(),
            NOT f##object1##Amount() OR floor(amount(h)) = f##object1##Amount(),
            NOT f##object2##Name() OR organizationName(h) = f##object2##Name()
    ;

    DESIGN view##object1 {
        PANEL {
            NEW allPane {
                horizontal = FALSE ;
                MOVE pane;
                NEW filters {
                    caption = 'Фиксированные фильтры';
                    horizontal = TRUE;
                    MOVE PROPERTY (on##object1##Clear()) {background= RGB(204, 204, 255);};
                    MOVE PROPERTY (f##object1##Number());
                    MOVE PROPERTY (f##object1##Date());
                    MOVE PROPERTY (f##object2##Name());
                    MOVE PROPERTY (f##object1##Amount());
                }
            }
        }
    }
END
```

Примечание:

- Наличие CONSTRAINT … CHECKED BY используемого в качестве фильтра для выбора организаций, соответствующих приходным или расходным документам.
- Наличие действия (будет отображаться как кнопка в панели фильтров) on..Clear для очистки ранее заполненных значений фильтров.
- FORM Для преобразования суммы amount в целочисленное значение использовалось свойство floor модуля Utils.
- DESIGN Так как дизайн форм с информационной панелью итогов уже сложился, а постановка требует вывести поля фильтров под информационной панелью, то создаем отдельный контейнер allPane, в него переносим контейнер с итогами pane, создаем новый контейнер filters и в него переносим новые действия и свойства формы.

Соответственно, выполним доработку модуля Reception (добавим изменения в конец модуля):

```
// Пользовательские фильтры
@filters(Reception, supplier, 'Поставщик');
```

Аналогичным образом выполним доработку модуля Expenses (добавим изменения в конец модуля):

```
// Пользовательские фильтры
@filters(Expenses, customer, 'Покупатель');
```

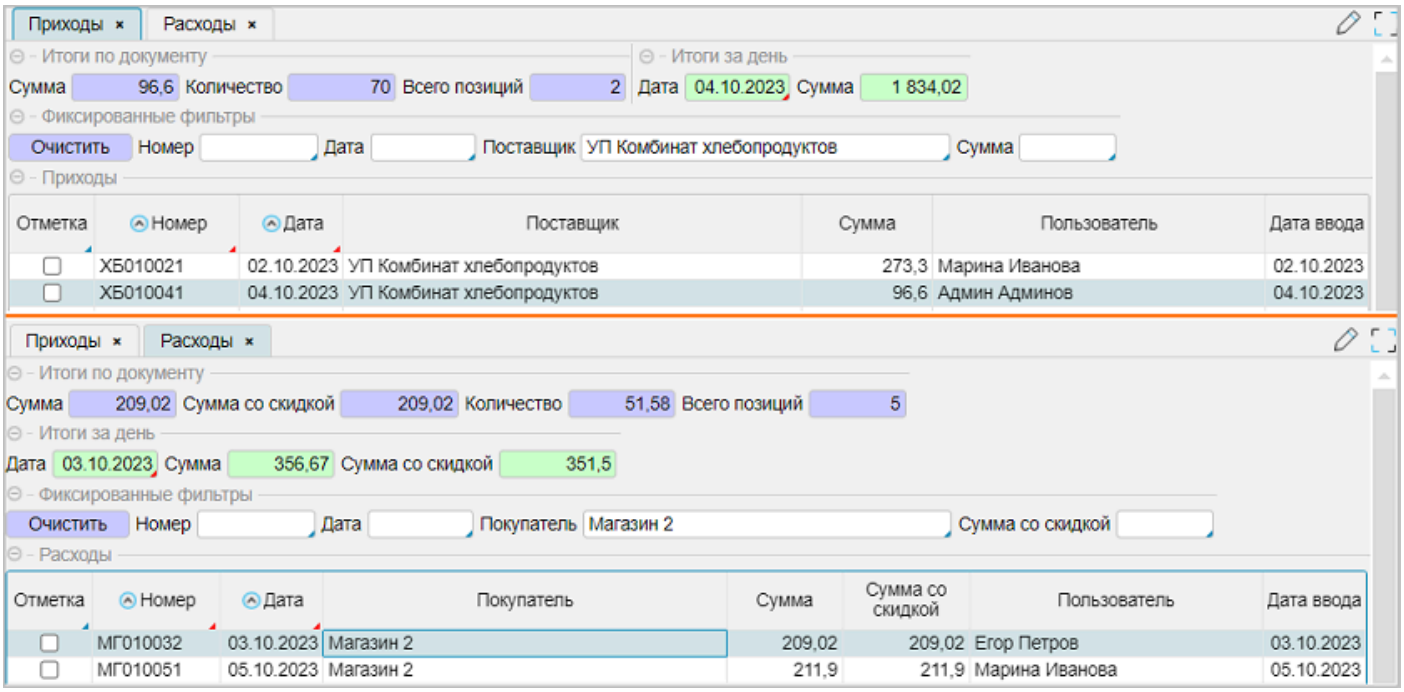
Так как для расходов по условиям задачи сумма в отпускных ценах не интересна (свойство amount), а интересует именно окончательная сумма со скидкой (amountDiscount), то необходимо выполнить небольшую доработку в конце модуля:

```
// ДОРАБОТКА: фильтр сумма со скидкой
fAmountDiscount 'Сумма со скидкой' = DATA LOCAL INTEGER;

AFTER onExpensesClear() DO { fAmountDiscount() ← NULL; }

EXTEND FORM viewExpenses
   PROPERTIES fAmountDiscount()
   FILTERS NOT fAmountDiscount() OR floor(amountDiscount(h)) = fAmountDiscount();

DESIGN viewExpenses {
   REMOVE PROPERTY (fExpensesAmount());
   filters {
       MOVE PROPERTY (fAmountDiscount());
   }
}
```

Примечание:

- Доработка создает новое свойство fAmountDiscount, размещает его на форме и устанавливает пользовательский фильтр. Используя DESIGN формы, старое свойство fExpensesAmount, созданное метакодом, удаляется (REMOVE PROPERTY) и в конец панели фильтров добавляется новое свойство fAmountDiscount (MOVE PROPERTY). Используя инструкцию AFTER, добавляем очистку значения фильтра fAmountDiscount. Действие, указанное в инструкции AFTER, выполнится после основного действия onExpensesClear.

После старта сервера приложений внешний вид информационных панелей приходов и расходов примет вид:



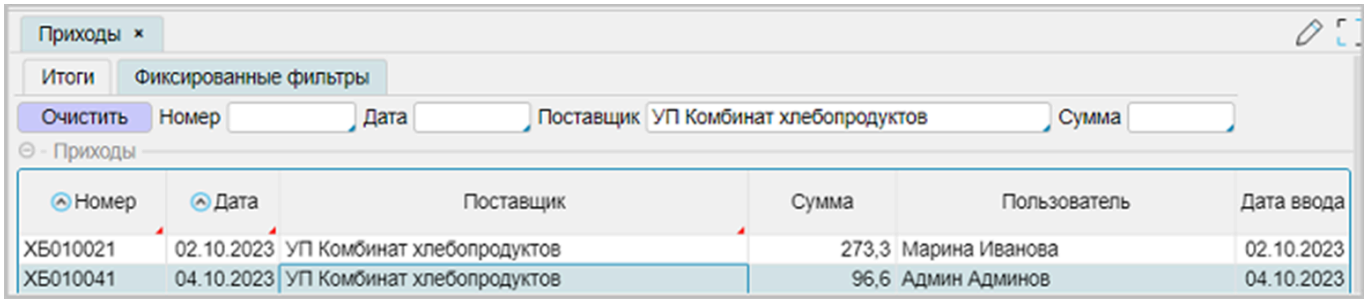
Примечание:

- Если немного доработать DESIGN в блоке метакода filters модуля Documents, то можно получить вкладки на форме, разделив на разные вкладки итоги и фильтры.

```
DESIGN view##object1 {
   PANEL {
       NEW allPane {
```

```
            horizontal = FALSE ;
            tabbed = TRUE;
            MOVE pane {caption = 'Итоги';}
            NEW filters {
                caption = 'Фиксированные фильтры';
                horizontal = TRUE;
                MOVE PROPERTY (on##object1##Clear()) {background= RGB(204, 204, 255);};
                MOVE PROPERTY (f##object1##Number());
                MOVE PROPERTY (f##object1##Date());
                MOVE PROPERTY (f##object2##Name());
                MOVE PROPERTY (f##object1##Amount());
            }
        }
    }
}
```

Напоминаем, что вкладки получаются применением свойства < tabbed = TRUE >. Плюсом такого подхода может быть дополнительная экономия места.
После старта сервера приложений внешний вид формы, на примере приходов, будет такой:



Примечание:

- DESIGN Так как изначально контейнер pane не имел заголовка, в связи с тем, что был определен как общий контейнер для итогов, то при использовании его как "вкладки", ему присваивается заголовок, который станет заголовком вкладки.

См. также

Документация. Оператор DATA. Параметр LOCAL
Документация. Первичные свойства (DATA). Локальные первичные свойства
Документация. Инструкция FORM. Блоки формы
Документация. Блоки фильтров и сортировок. Блок фиксированных фильтров
Документация. Инструкция AFTER

Пример кода:
examples\t403_фиксированные_фильтры\src
без использования вкладок в приходах и расходах

examples\t403_фиксированные_фильтры\src_как_панель_фильтров
панель фильтров, как отдельная вкладка на форме

Фильтр по отмеченным

Действие фильтров может распространятся не только на отображаемый диапазон данных, но и на экспорт, и на выполнение каких-либо действий по расчету, и на отчеты, и так далее. При этом не всегда возможно подобрать условие фильтра, при котором выбранный диапазон будет удовлетворять поставленной задаче, или подбор такого условия будет сложен. Лучше всего для решения таких задач подходит отметка записей на форме.

Что надо сделать:

- Создать свойство, которое будет отвечать за отметку записи.

Так как предполагается отмечать записи и в приходах, и в расходах, то вынесем создание свойства на уровень модуля Documents, для чего:

- перед блоком метакода forms доработаем общий класс заголовков документов Header

```
// ДОРАБОТКА: отметка записи
mark 'Отметка' = DATA LOCAL BOOLEAN (Header) CHARWIDTH 8;
```

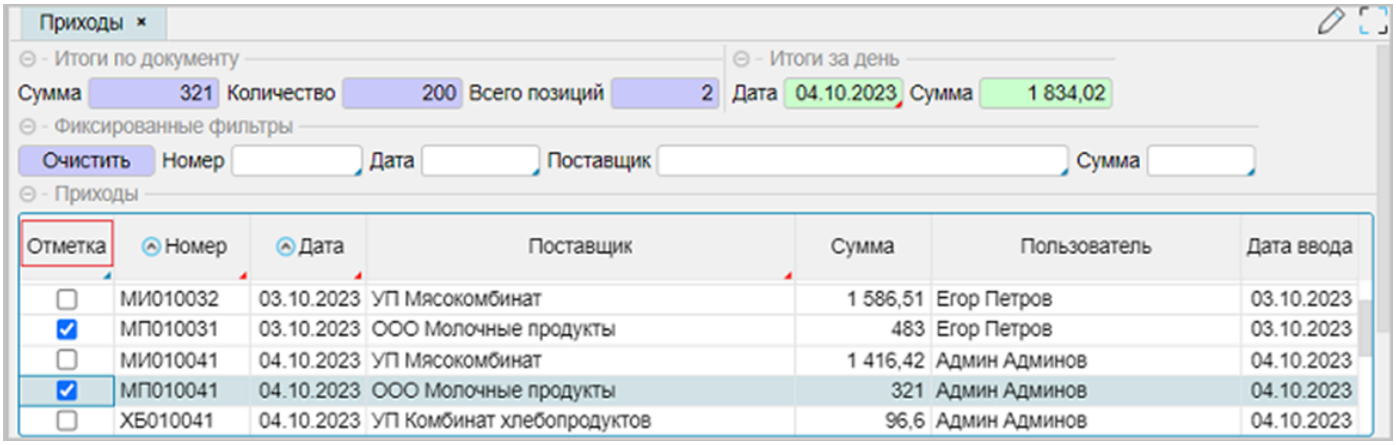
- доработаем код блока метакода filters (в конце):

```
// ФИЛЬТР по отмеченным
EXTEND FORM view##object1 PROPERTIES(h) mark FIRST;
AFTER on##object1##Clear() DO mark(Header##object1 o) ← NULL;
```

Примечание:

- EXTEND FORM Добавили в форму отображения новое свойство.
- AFTER Добавили в выполнение действия по очистке фильтров: ранее отмеченных записей.

После старта сервера приложений получим следующий вид (на примере приходов) формы отображения:

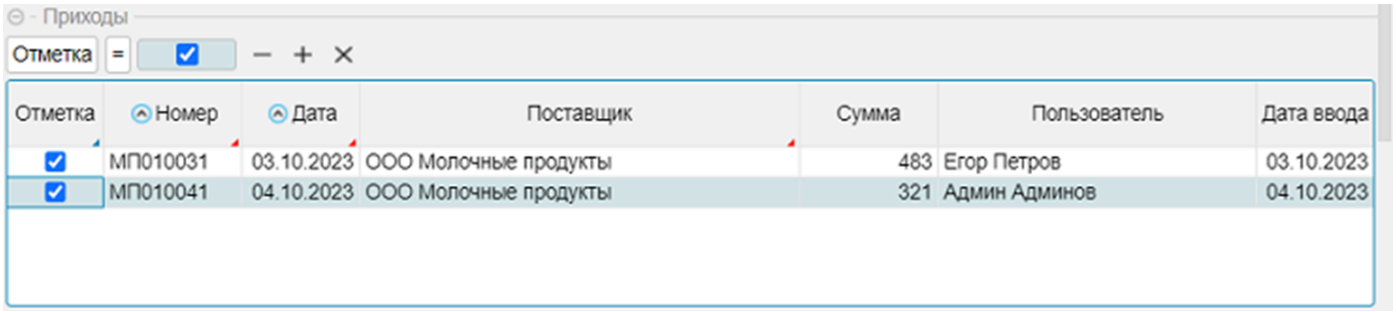


Примечание:

- DATA Создаваемое свойство mark связано с классом Header (шапка документа) и отмечено, как LOCAL – локальное. Локальное свойство действует в рамках текущей сессии и его изменения не доступны другим пользователям. **Важно:** изменение локального свойства не вызывает появление на форме кнопок "Сохранить"/"Отменить".

- FORM В отличие от всех добавляемых свойств, свойство mark не READONLY, то есть редактируется прямо на форме мышью или по F2.

Для высвечивания только отмеченных (или всех не отмеченных (инверсия)) записей можно воспользоваться пользовательским фильтром:



Отметка	Номер	Дата	Поставщик	Сумма	Пользователь	Дата ввода
☑	МП010031	03.10.2023	ООО Молочные продукты	483	Егор Петров	03.10.2023
☑	МП010041	04.10.2023	ООО Молочные продукты	321	Админ Админов	04.10.2023

Например, после выделения нужного диапазона записей, их можно экспортировать в Excel, используя кнопку  стандартной панели инструментов табличной части:

Отметка	Номер	Дата	Поставщик	Сумма	Пользователь	Дата ввода
ИСТИНА	МП01003	03.10.23	ООО Молочные продукты	483	Егор Петров	03.10.23
ИСТИНА	МП01004	04.10.23	ООО Молочные продукты	321	Админ Админов	04.10.23

См. также

Пример кода:
examples\t405_Фильтр_по_отмеченным\src

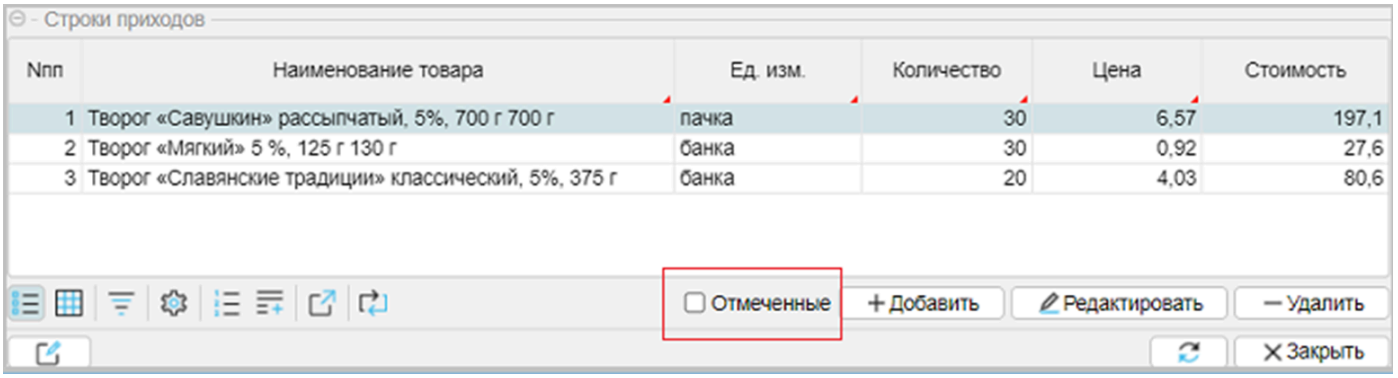
Группа фильтров

Еще одной разновидностью фильтров на форме является группа фильтров. Названы они так, потому что в одну группу может входить от одного и более разных фильтров.

Например, можно создать в рамках текущего приложения группу фильтров, связанных с отмеченными записями в документах (приходы/расходы). Доработку выполним (*уже становится традицией борьбы с "копипастом"*) в блоке метакода filters модуля Documents (добавим в конец):

```
// Группа фильтров
EXTEND FORM view##object1 FILTERGROUP group FILTER 'Отмеченные' mark(h);
```

После старта сервера приложений на форме отображения (на примере приходов) появится чекбокс 'Отмеченные' внизу табличной части "Строки приходов":



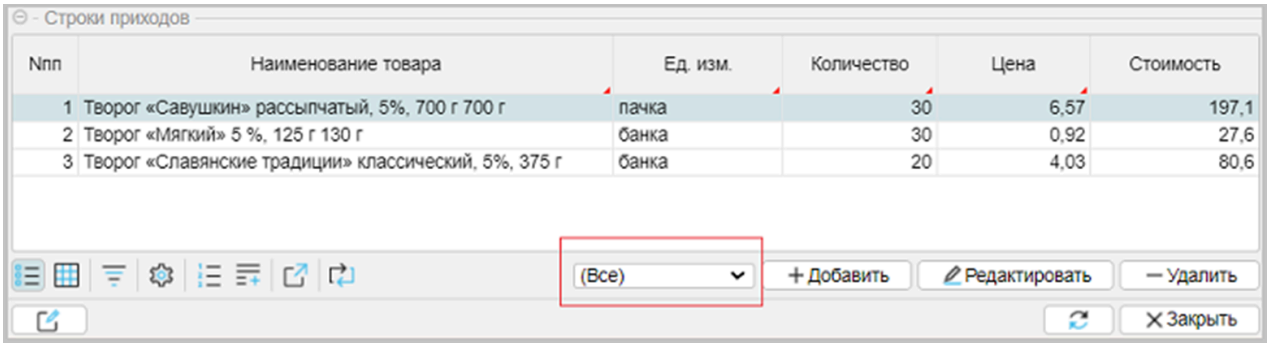
Примечание:

- После старта сервера чекбокс "Отмеченные" не выбран. Если существует необходимость, чтобы фильтр был установлен, на момент открытия формы, то используется ключевое слово DEFAULT, добавляемое в конец выражения фильтра. *Для текущего примера это неудобно, так как при открытии формы отмеченных записей не будет и форма отобразиться пустой.*

Добавим в группу новый фильтр: сумма (amount) больше 300 рублей – отсечем "мелочевку". Можно подправить выражение группы фильтров group, добавив еще один FILTER. Но поступим по-другому. Группа фильтров поддерживает EXTEND (примерно, как и формы), которое позволяет расширить область фильтров группы:

```
EXTEND FORM view##object1 EXTEND FILTERGROUP group FILTER 'Больше 300 руб.' amount(h) > 300;
```

После старта сервера приложений на форме отображения (на примере приходов) появится выпадающий список внизу табличной части "Строки приходов" из 3-х элементов: "Все" – ничего не выбрано, "Отмеченные", "Больше 300 руб.".

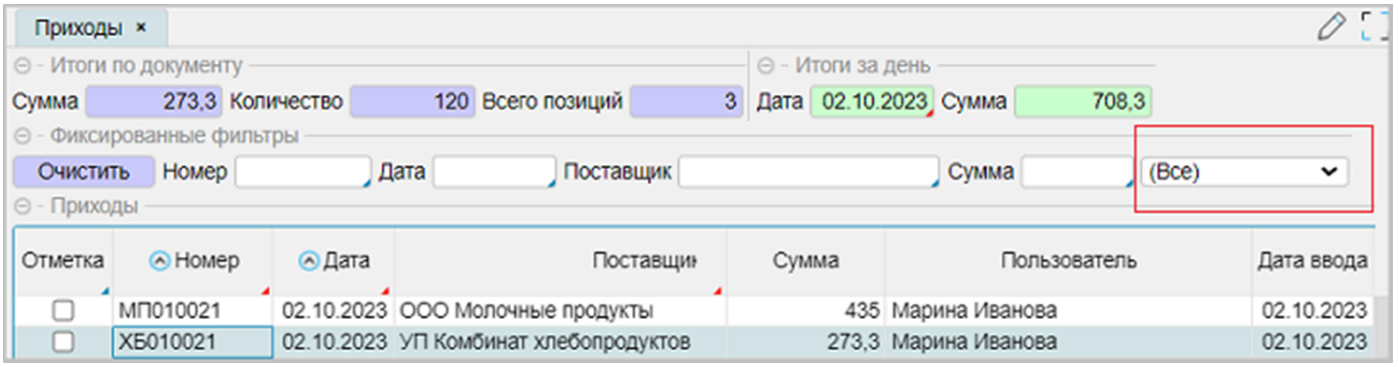


Примечание:

- При необходимости можно перенести группу фильтров, например, к фиксированным фильтрам информационного табло, используя ключевое слово LAST:

```
DESIGN view##object1 { filters { MOVE FILTERGROUP (group) LAST; } }
```

После старта сервера приложений форма отображения (на примере приходов) примет следующий вид:



См. также

Документация. Инструкция FORM. Блоки формы
Документация. Блоки фильтров и сортировок. Блок группы фильтров

Пример кода:
examples\t406_группа_фильтров\src

examples\t406_группа_фильтров\src_перенос
перенос группового фильтра в панель filters на форме

События свойств формы. Оператор DIALOG

Кроме событий форм могут обрабатываться события свойств.
Для примера изменим обработку при выборе организации на форме редактирования документов, чтобы:

- всегда вместо выпадающего списка вызывалась форма справочника выбора организаций,
- при выборе покупателей на форме выбора присутствовало значение скидки, предоставляемой покупателю.

Для решения этой задачи доработаем форму выбора организаций, модуль Organization, добавлением в форму выбора свойства discount по условию:

```
// отображение discount по условию
EXTEND FORM listOrganization
   PROPERTIES (o) READONLY discount SHOWIF organizationType(o) = OrganizationType.customer
;
```

Примечание:

- использование опции SHOWIF позволяет управлять видимостью свойства discount:
  - приход – свойство не отображается
  - расход – свойство отображается
- важно то, что ограничение всегда является глобальным по отношению к приложению, и использование конструкции CONSTRAINT … CHECKED BY распространяется на отображение в фильтрах (используются локальные свойства) и в формах редактирования (используются свойства классов), поэтому становится возможным определить условие SHOWIF. Скидка отображается в журнале расходов, но не отображается в журнале приходов.

Выполним также доработку в блоке метакода модуля Documents – редактирование свойства organizationName, используя события ON CHANGE и оператор DIALOG в блоке метакода forms, доработаем форму редактирования и изменим количество передаваемых параметров:

```
META forms(object, object2, title1, title2, title3)
   // форма редактирования
   FORM edit##object title1
       OBJECTS h = Header##object PANEL // редактирование заголовка
       PROPERTIES (h) number, date
       PROPERTIES (h) organizationName ON CHANGE {
           DIALOG listOrganization OBJECTS o = object2(h) CHANGE DO object2(h) ← o;
       }
       OBJECTS r = object // редактирование строк
       PROPERTIES (r) nameProduct, nameUnit, quantity, price
       PROPERTIES (r) NEW, DELETE
       EDIT Header##object OBJECT h
       FILTERS header(r) = h
   ;
```

Примечание:

- ON CHANGE событие сработает при попытке отредактировать свойство, в данном случае organizationName. Сам по себе блок может содержать множественные действия, хотя в примере вызывается только одно: оператор DIALOG форма выбора организаций.
- DIALOG рассмотрим содержание на примере приходов (для расходов аналогично, вместо supplier будет customer):

- listOrganization - имя формы, которую надо отразить.
- OBJECTS o = supplier(h) – если такой объект (supplier(h)) существует, то при открытии формы выбора указатель установится на соответствующую запись, иначе на первую.
- CHANGE – это опция, которая позволяет учитывать установленные ограничения в системе. Поэтому по-прежнему работает CONSTRAINT … CHECKED BY, который устанавливает фильтр на то, какие организации должны отображаться.
- DO – означает, что в случае изменений надо выполнить какое-то действие. В данном случае меняем значение supplier(h) <- o, где supplier это свойство шапки приходов, o – ссылка на выбранный объект из справочника.

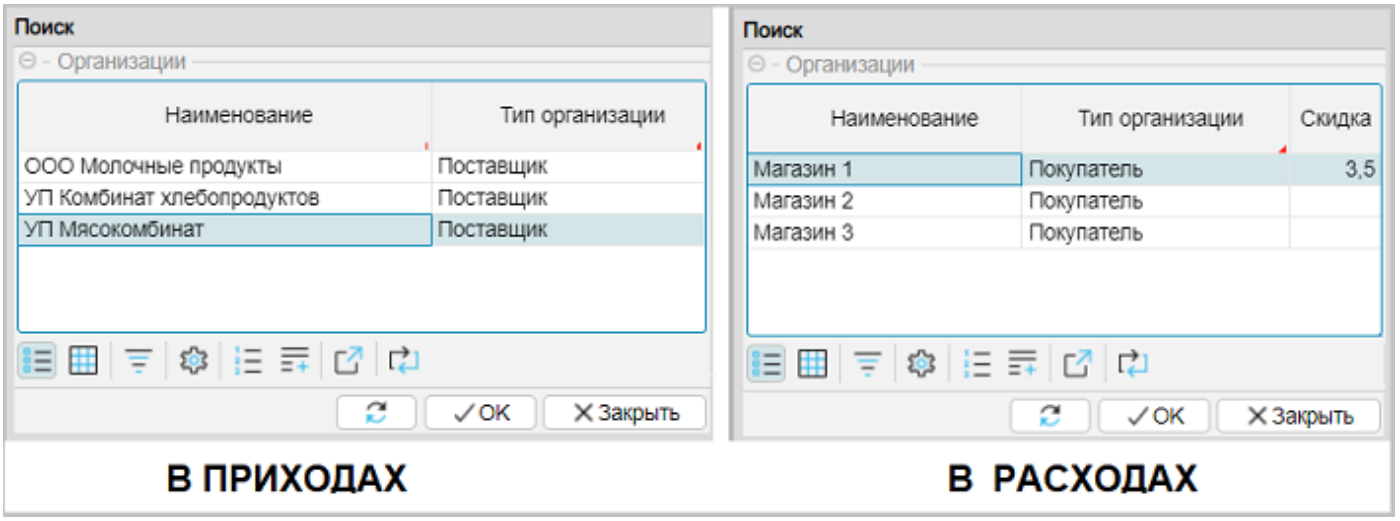
Так как метакод forms изменился по количеству передаваемых параметров, необходимо в модуле Reception выполнить изменения:

```
// 6. Формы редактирования и отображения, дизайн, 7. меню
@forms(Reception, supplier, 'Приход', 'Приходы', 'Поставщик');
```

Аналогично для модуля Expenses:

```
// 6. Формы редактирования и отображения, дизайн, 7. меню
@forms(Expenses, customer, 'Расход', 'Расходы', 'Покупатель');
```

После старта сервера приложений форма выбора организаций в форме редактирования в приходах и расходах будет иметь следующий вид:



См. также

Документация. События формы. События свойств
Документация. Блок свойств и действий. Опции свойства или действия
Документация. Оператор ASK
Документация. Оператор DIALOG
Примеры. Оператор DIALOG

Пример кода:
examples\t407_события_свойств\src

Постановка задачи: Картинки на форме

Есть новое пожелание от заказчика: выводить визуальную информацию о товаре в строках приходно-расходных документов.

Что надо сделать:

- наверное, самое сложное – это найти сами изображения,
- доработать справочник товаров, модуль Product, для привязки изображений:
    - добавить новое свойство, например image, тип IMAGEFILE
    - доработать формы отображения и редактирования для работы с image,
- так как image – это свойство справочника товаров Product, необходимо, используя композицию, создать свойство image для приходных и расходных документов,
- в блоке метакода модуля Documents доработать форму отображения.

Решение:

- Доработка справочника товаров
- Доработка документов

Доработка справочника товаров

Для привязки картинок к товарам необходимо доработать справочник товаров, модуль Product:

- создать новое свойство, которое будет хранить изображения, image

```
image 'Фото' = DATA IMAGEFILE (Product);
```

- создать действия по загрузке изображений и их очистке, в случае отмены:

```
onLoadImage 'Загрузить' (Product o) { INPUT img = IMAGEFILE DO image(o) ← img; }
onClearImage 'Очистить' (Product o) { image(o) ← NULL;}
```

- доработать форму редактирования и исправить ее дизайн:

```
EXTEND FORM editProduct
    PROPERTIES (o) onLoadImage, onClearImage
    PROPERTIES (o) READONLY image
;

DESIGN editProduct {
    OBJECTS {
        NEW pane {
            horizontal = FALSE ;
            MOVE PROPERTY (image(o)) { caption = ''; width=250; height=250; }
            NEW command {
                horizontal = TRUE;
                MOVE PROPERTY (onLoadImage(o));
                MOVE PROPERTY (onClearImage(o));
            }
        }
    }
}
```

Примечание:

- FORM свойство image и действия можно было отобразить без READONLY в одну строку, как < `PROPERTIES (o) image, onLoadImage, onClearImage` >. В этом случае клик мышью по картинке вызывал бы диалог открытия файла для поиска изображений. Если image выводится с ключевым словом READONLY, то клик вызовет просмотрщик файлов изображений, определенный в операционной системе.
- DESIGN создаем общий контейнер pane, в который перемещаем image с указанием размеров области изображения, и создаем в pane контейнер command для вызова действий по загрузке и очистке изображения.
- При выводе изображения, оно форматируется пропорционально по меньшей из сторон и выводится внутри области размера изображения, поэтому могут быть области, не заполненные изображением. *Для того, чтобы сделать все красиво при выводе изображений, необходимо воспользоваться графическим редактором: изменить размер и установить прозрачный или белый фон.*

- исправить форму отображения и ее дизайн:

```
EXTEND FORM viewProduct
    OBJECTS p = Product PANEL
    PROPERTIES (p) READONLY image
    FILTERS p=o
```

```
;

DESIGN viewProduct {
    OBJECTS {
        NEW pane {
            horizontal = TRUE ;
            fill = 1;
            MOVE BOX(o);
            MOVE PROPERTY (image(p)) { caption = ''; width = 100; height = 100; };
        }
    }
}
```

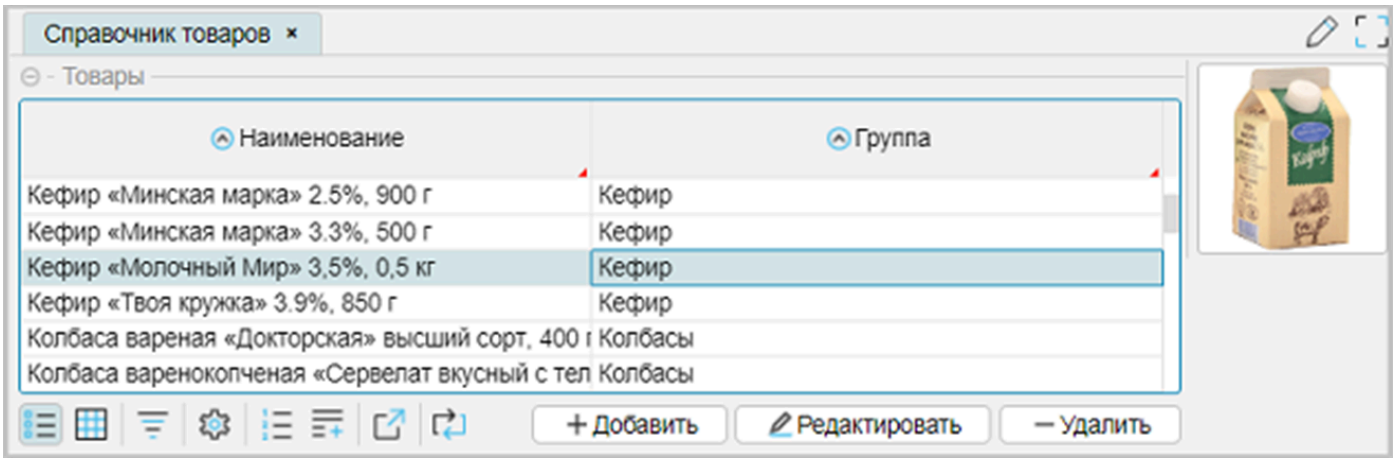
Примечание:

- DESIGN для того, чтобы разместить на одном уровне (горизонтально) контейнер табличной части справочника (BOX(o)) и изображение, создаем отдельный контейнер pane, в который перемещаем BOX(o) и свойство image.

Примечание:

- все доработки выполнены в конце модуля.

После восстановления базы данных SQL и запуска сервера приложений, внешний вид формы отображения:



Доработка документов

Перед использованием метакода forms в модулях приходных (Reception) и расходных (Expenses) документов необходимо создать композицию image :

- ·модуль Reception:

```
image 'Фото' (Reception o) = image(product(o));
```

- модуль Expenses:

```
image 'Фото' (Expenses o) = image(reception(o));
```

Примечание:

- В соответствии с выбранной организацией построения приложения, расходы связаны со справочником товаров (Product), поэтому композиция строится относительно product. Расходные документы со справочником товаров напрямую не связаны, а связаны со строками приходов (Reception), с которых списываются остатки по поступлениям товаров. Поэтому основная связь через reception.

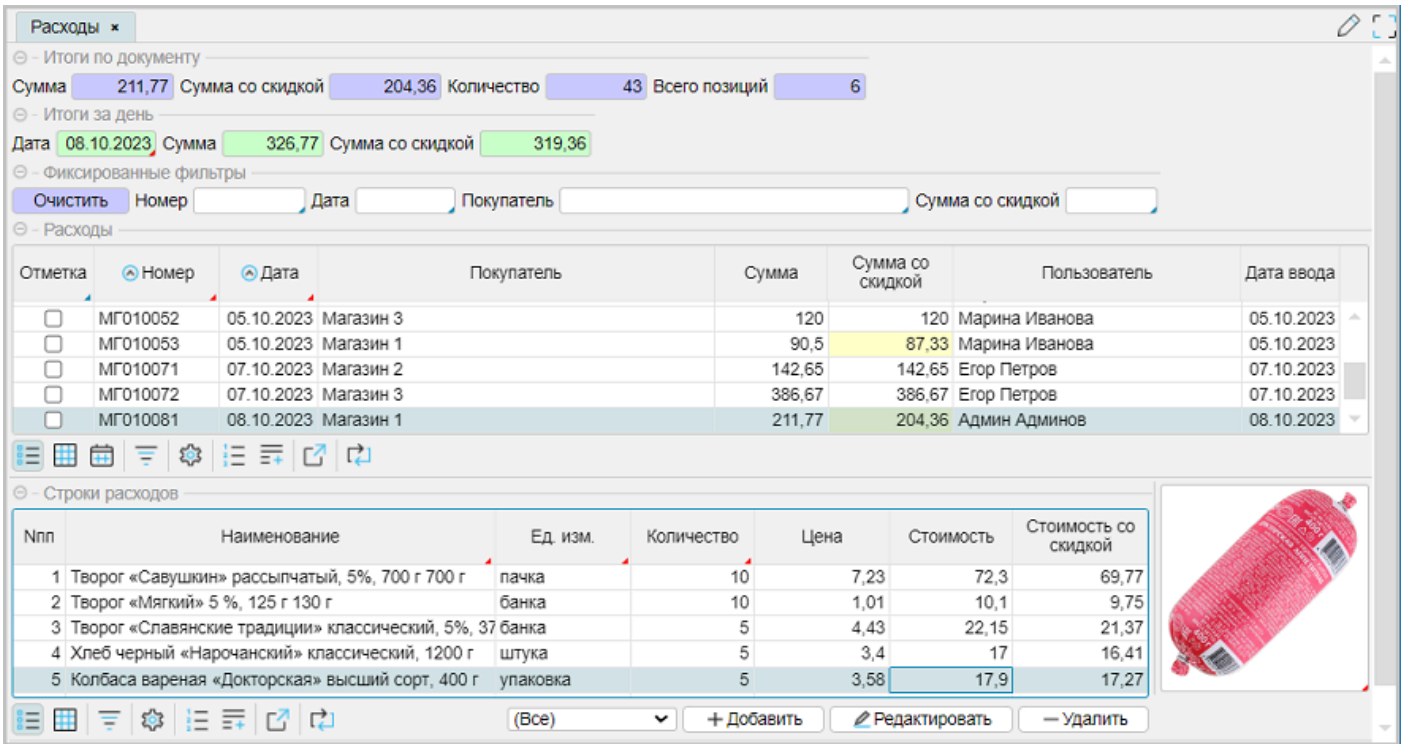
Уже *традиционно* дорабатываем блок метакода forms модуля Documents для формы отображения и исправим ее дизайн (исправления добавляем в конец блока):

```
// ДОРАБОТКА: картинки на форме
EXTEND FORM view##object
   OBJECTS o2 = object PANEL
   PROPERTIES (o2) READONLY image
   FILTERS o2 = r
;
DESIGN view##object {
   OBJECTS {
       NEW image {
           horizontal = TRUE;
           fill = 1;
           MOVE BOX(r);
           MOVE PROPERTY (image(o2)) {caption = ''; width = 150; height = 150;}
       }
   }
}
```

Примечание:

- FORM создаем отдельную панель, для отображения свойства image, и по фильтру связываем со строками документа.
- DESIGN для того, чтобы расположить горизонтально строки документа и изображение, создаем контейнер image, в который перемещаем контейнер табличной части строк документа BOX(r) и свойство image(o2).

После восстановления базы данных SQL и запуска сервера приложений, внешний вид формы отображения на примере расходов:



См. также

Пример кода:
examples\t408_картинки_на_форме\src

Архив SQL:
examples\t408_картинки_на_форме\sql

Картинки товаров:
examples\image

Тема 5. Отчеты

Рассматриваемые вопросы:

- Общие сведения
- Постановка задачи
- Предварительные действия
- Создание форм и вывод на экран
- Создание шаблона отчета
- Редактирование шаблона отчета
- Реестр расходов. Копирование шаблона
- Общий шаблон для реестров
- Использование фильтров формы отображения
- Подотчеты. Товарный отчет

Общие сведения

Под отчетами понимается структурированное отображение информации, формируемое на основе данных, хранящихся в информационной системе, предоставляемое по запросу пользователя.

Отсутствие отчетов сильно занижает функциональные возможности программы, а в некоторых случаях делает ее эксплуатацию невозможной, так как наличие отчетных документов в соответствии с учетной политикой обязательно, а их вид и наполнение могут быть определены законодательно.

Технически любая форма может быть использована в качестве простого отчета, данные для которого могут быть сформированы через экспорт в Excel, используя кнопку экспорта данных панели инструментов табличной части:

При этом диапазон экспортированных данных может быть ограничен при необходимости фильтрами. Но так можно сделать только для одной табличной части. Внешний вид такого отчета будет примитивным, только название колонок и строки детализации:

	A	B	C	D	E	F	G	H
1								
2		Отметка	Номер	Дата	Поставщик	Сумма	Пользователь	Дата ввода
3			МП010021	02.10.23	ООО Молочные продукты	435	Марина Иванова	02.10.23
4			ХБ010021	02.10.23	УП Комбинат хлебопродуктов	273.3	Марина Иванова	02.10.23
5			МИ010032	03.10.23	УП Мясокомбинат	1586.51	Егор Петров	03.10.23

Говорить о специализированной форме отчета не приходиться.

Платформа lsFusion поддерживает механизмы создания и формирования сложных отчетов, состоящих из многих разделов (bands), включающих в рамках одного отчета информацию от разных классов и свойств, с возможностью экспорта отчетов в разные форматы. Построение отчетов на платформе lsFusion связано с формой, которая выступает в качестве источника данных для отчета. Технически уже существующая форма, например форма отображения, может являться источником данных для отчета. Но как правило, под отчет создается отдельная форма.

Любое построение отчетов можно представить как:

- создание формы, ответственной за вывод данных в отчет;
- управление дизайном отчета, в том числе с помощью популярного дизайнера отчетов JasperSoft Studio, который включен в инсталляционный пакет. Дизайн отчета или шаблон представляет собой xml структуру с расширением jrxml.
- назначение действия для вызова отчета.

См. также

Краткое руководство по JasperReports

Постановка задачи

Необходимо создать отчеты:

1. Реестр приходных документов в диапазоне выбранных дат по всем поставщикам или выбранному поставщику.
2. Реестр расходных документов в диапазоне выбранных дат по всем поставщикам или выбранному поставщику.
3. Товарный отчет в диапазоне выбранных дат.

Общие условия:

- Отчеты вызываются из форм отображения приходных и расходных документов.
- Отчеты содержат заглавную страницу с названием отчета и диапазоном выбранных дат.
- Округления сумм в отчетах до 2-х знаков после запятой.
- Отчеты формируются в ценах поставщика.
- Все отчеты должны по выбору быть сформированы в форматах Excel, PDF или MS Word.
- Отдельно для реестров документов:
    - выполняется группировка по реквизитам накладной (номер, дата, организация),
    - подбивается итоговая сумма за отчетный период.
- Отдельно для товарного отчета выводится:
    - значение суммы остатка на начало, которая рассчитывается как сумма всех приходов - сумма всех расходов до начала отчетного периода;
    - для приходных документов: номер, дата, организация, сумма за отчетный период;
    - итоговая сумма по всем приходам за отчетный период;
    - для расходных документов: номер, дата, организация, сумма за отчетный период;
    - итоговая сумма по всем расходам за отчетный период;
    - значение остатка на конец отчетного периода, которая рассчитывается как сумма всех приходов - сумма всех расходов до конца выбранного диапазона включительно.

Что надо сделать:

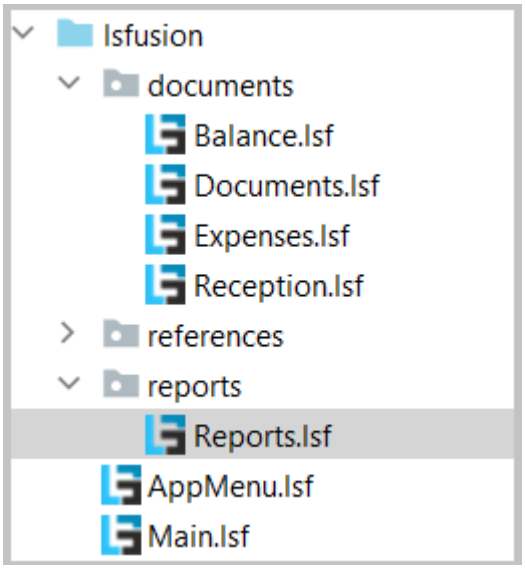
- Так как для отчетов необходимо выбирать диапазон, поставщиков и тип выходного документа, то необходима отдельная форма для ввода параметров.
- Создать формы реестров для приходных и расходных документов.
- Создать форму для товарного отчета.
- Назначить действия для вызова печатных форм.
- Исправить дизайн отчетов, используя дизайнер отчетов JasperSoft Studio.

Предварительные действия

Так как предполагается перед вызовом любого из отчетов выбирать :

- отчетный период (диапазон дат)
- организацию (только для реестров)
- формат выходного документа (Excel, PDF, MS Word или предпросмотр на экран),

то необходимо создать форму, которая позволит выбрать и установить свойства, которые будут использованы при формировании отчета. Также необходимо наличие форм, свойств и действий, связанных с формированием печатных документов. С этой целью создадим отдельный модуль Reports в папке reports:



Код модуля Reports:

```
MODULE Reports;

REQUIRE Expenses;

CLASS OutPrint 'Формат печати' {
   pdf '1. Вывод отчета в PDF',
   xls '2. Вывод отчета в Excel',
   doc '3. Вывод отчета в MS Word'
}

FORM viewOutPrint
   OBJECTS op = OutPrint
   PROPERTIES (op) READONLY staticCaption
   LIST OutPrint OBJECT op
   ORDERS staticCaption(op)
;

DESIGN viewOutPrint {
   PROPERTY (staticCaption(op)) {
       caption='Формат печати';
       charWidth=12;
   }
}

dateFrom 'Дата начала' = DATA LOCAL DATE;
dateTo   'Дата конец' = DATA LOCAL DATE;
formType 'Реестр или т/отчет' = DATA LOCAL BOOLEAN;
```

```
oid      'ID организации' = DATA LOCAL Organization;
otype    'Тип организации' = DATA LOCAL OrganizationType;
total1   'Итоги' = DATA LOCAL NUMERIC[10,2];
title1   'Надписи' = DATA LOCAL STRING;

FORM prnParameters 'Параметры отчета'
   OBJECTS interval = (dateFrom = DATE, dateTo = DATE) PANEL
   PROPERTIES dateFrom 'Дата, с:' = VALUE(dateFrom), dateTo 'Дата, по:' = VALUE(dateTo)

   OBJECTS o = Organization PANEL NULL
   PROPERTIES name(o) SELECTOR SHOWIF formType()
   FILTERS organizationType(o) = otype()

   OBJECTS p = OutPrint PANEL NULL
   PROPERTIES (p) staticCaption SELECTOR
   EVENTS ON INIT {
       SEEK prnParameters.p = OutPrint.pdf;
   }
;

DESIGN prnParameters {
   size = (400, 120);
   BOX(interval) { caption=NULL; }
   BOX(o) { caption=NULL; }
   PROPERTY (name(o)) { caption = 'Выбор организации:'; }
   BOX(p) {caption=NULL;}
   PROPERTY (staticCaption(p)) { caption = 'Формат печати:'; }
}
```

Примечание:

- Класс OutPrint нужен для управления выводом отчета в разные форматы: Excel, PDF, DOC и предпросмотр отчета, соответственно для класса определены статические объекты.
- В событии формы INIT определено значение по умолчанию формата вывода PDF, присвоением значения статического объекта.
- Для различия организаций в приходах и расходах при выборе организации используется фильтр < FILTERS organizationType(o) = otype() >, значение которого будет определено в метакоде.
- Локальное свойство formType() необходимо для управления отображением выбора организации:
    - для реестров, в соответствии с постановкой задачи, кроме диапазона дат, можно выбрать организацию;
    - для товарного отчета можно выбрать только диапазон дат, поэтому для товарного отчета выбор организации будет скрыт.

См. также

Документация. Блок свойств и действий. SELECTOR. VALUE. Пример

Документация. Оператор SEEK

Создание форм и вывод на экран

Определение печатной формы реестров документов мало чем отличается от форм отображения документов типа "шапка-строки", сформированных в модуле Documents, за исключением количества отображаемых свойств и используемых фильтров.

При этом печатные формы реестров накладных и для приходов, и для расходов технически одинаковы, за исключением имен классов. Поэтому, чтобы не заниматься "копипастом", создание печатной формы и действия по ее вызову оформим в блоке метакода модуля Reports:

```
META registry(object,object2)
   // печатная форма
   FORM registry##object 'Реестр документов'
       PROPERTIES title1(), total1()
       OBJECTS h = Header##object
       PROPERTIES (h) number, date, organizationName, amount
       OBJECTS o = object
       PROPERTIES (o) index, nameProduct, nameUnit, quantity, price, cost
       FILTERS header(o) = h
       FILTERS date(h) ⩾ dateFrom() AND date(h) ⩽ dateTo()
       FILTERS object2(h) = oid() OR NOT oid()
       ORDERS date(h), number(h), organizationName(h)
   ;
   // действие на форме для вызова отчета
   onRegistry##object 'Реестр документов' (Header##object h) {
       formType() ← TRUE; otype() ← OrganizationType.object2;
       DIALOG prnParameters OBJECTS dateFrom=date(h) INPUT, dateTo=date(h) INPUT,
           o INPUT, p INPUT DO {
           dateFrom() ← dateFrom; dateTo() ← dateTo;
           oid() ← o;
           title1() ← 'за период дат с ' + toDateDDMMYY(dateFrom()) + ' по ' +
               toDateDDMMYY(dateTo()) + 'г.';
           // вычисление итога по всему отчету
           total1() ← GROUP SUM amount(Header##object ob)
               IF date(ob) ⩾ dateFrom() AND date(ob) ⩽ dateTo();
           CASE
               WHEN OutPrint.xls = p THEN
                   PRINT registry##object XLSX;
               WHEN OutPrint.doc = p THEN
                   PRINT registry##object DOC;
               ELSE
                   PRINT registry##object PDF;
       }
   }
   EXTEND FORM view##object PROPERTIES(h) onRegistry##object;
   DESIGN view##object {
       TOOLBARLEFT(h) {
           MOVE PROPERTY (onRegistry##object(h)) { marginLeft = 100; };
       }
   }
END
```

Примечание:

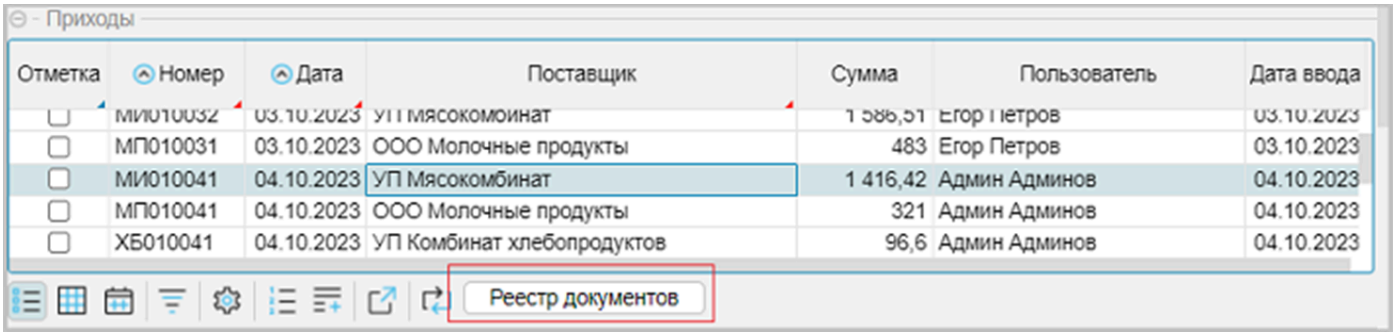
- Печатная форма вызывается оператором PRINT. Для оператора PRINT указывается имя печатной формы и формат вывода. Если формат не указан, то отчет выводится в режиме предпросмотра.

- При вызове оператора DIALOG задаем начальные значения для объектов формы prnParameters.
- Используя оператор CASE, выбираем формат вывода отчета
- В режиме Desktop клиента, когда вызывается окно предпросмотра, доступны кнопки управления шаблонами отчета (создание, редактирование, удаление). Для редактирования отчетов вызывается приложение JasperSoft Studio
- Вызов печатной формы производится в диалоге оператора DIALOG, если на форме prnParameters была нажата кнопка "Ок"

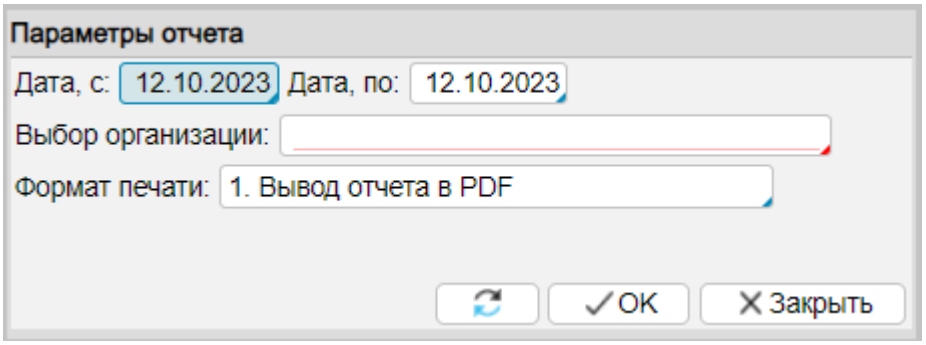
В конце модуля Reports вызовем метакод для форм отображения приходов и расходов:

```
// вызов реестра документов из форм приходов/расходов
@registry(Reception, supplier);

@registry(Expenses, customer);
```

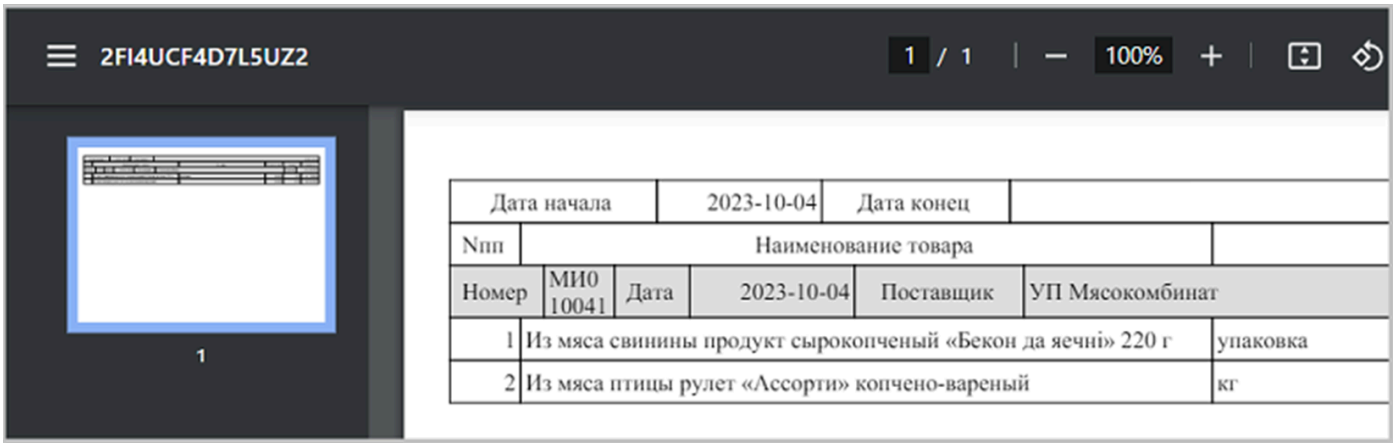
После старта сервера приложений для форм отображения приходов (и расходов), под верхним GRID появится кнопка "Реестр документов":



При нажатии на кнопку вызовется форма с параметрами, например:



После нажатия на кнопку "Ok", на экране появится автоматически сформированный отчет:



Дата начала		2023-10-04	Дата конец		
Nпп	Наименование товара				
Номер	МИО 10041	Дата	2023-10-04	Поставщик	УП Мясокомбинат
1	Из мяса свинины продукт сырокопченый «Бекон да яечні» 220 г				упаковка
2	Из мяса птицы рулет «Ассорти» копчено-вареный				кг

Примечание:

- вид отчета получен в WEB интерфейсе

Хотя выводимая информация понятна, но отчет нуждается в доработке для придания ему законченного вида. Изменения формы отчета выполняется в дизайнере отчетов JasperSoft Studio.

См. также

Документация. Оператор PRINT

Пример кода:

examples\t500_создание_форм\src

Создание шаблона отчета

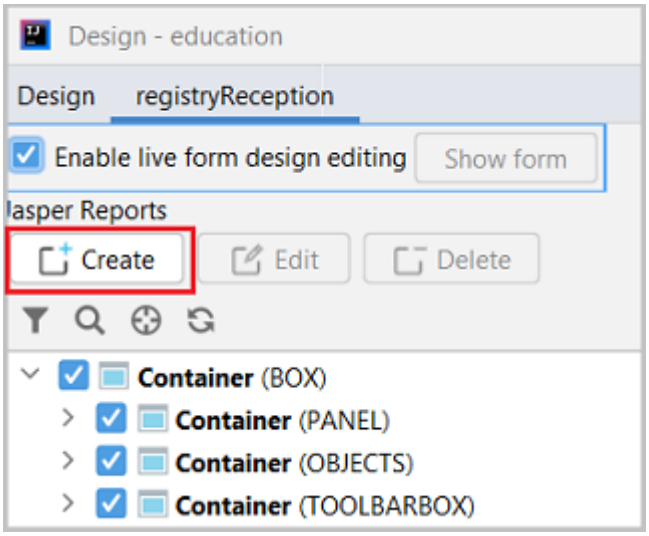
Для того чтобы редактировать шаблон отчета в JasperSoft Studio его необходимо создать. Процесс создания шаблонов автоматизирован на уровне редактора IDEA благодаря разработанному плагину lsFusion. При этом должны быть соблюдены некоторые условия:

- должна быть запущена служба служба web-клиента lsFusion (*должна работать после установки программы*),
- приложение, для которого разрабатывается отчет, должно быть запущено в режиме Debug,
- печатная форма должна быть описана своими свойствами и объектами формы.

Далее необходимо нажать на значок формы на служебном поле редактора IDEA:

```
96  // вызов реестра документов из форм приходов / расходов
97  @registry(Reception,OrganizationType.supplier){
98      // печатная форма
99      FORM registryReception 'Реестр документов'
100         PROPERTIES title1(), total1()
101         OBJECTS h = HeaderReception
102         PROPERTIES (h) number, date, organizationName, amount
103         OBJECTS o = Reception
104         PROPERTIES (o) index, nameProduct, nameUnit, quantity, price, cost
105         FILTERS header(o) = h
106         FILTERS date(h) >= dateFrom() AND date(h) <= dateTo()
107         FILTERS IF oid() THEN organizationName(h) = name(oid()) ELSE TRUE
108         ORDERS date(h), number(h), organizationName(h)
109     ;
```

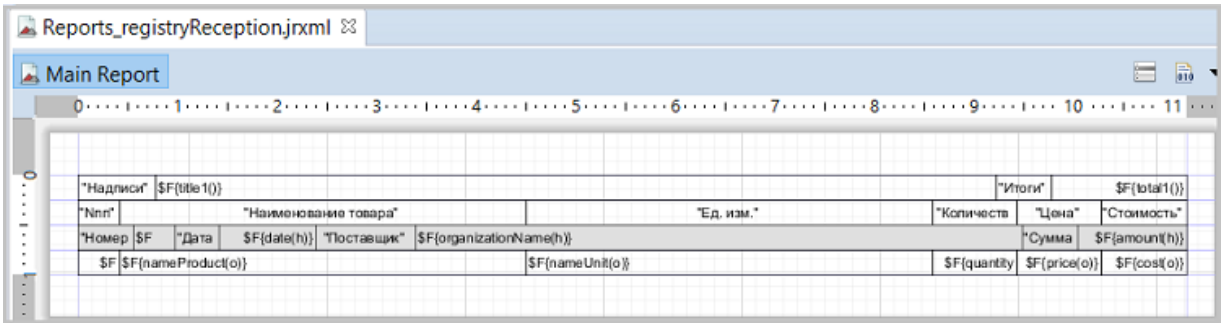
На экране откроется дизайнер редактируемой формы:



На форме необходимо нажать на кнопку "Create", при этом произойдут некоторые события:

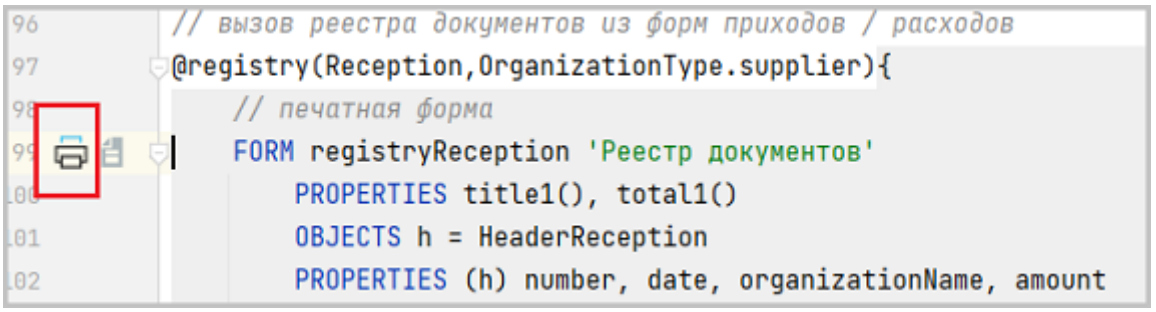
- в корневой директории проекта появится файл с именем и расширением имяПространстваИмен_имяФормы.jrxml. Соответственно для разных форм отчетов будут сформированы разные файлы:

- для приходных документов: Reports_registryReception.jrxml;
- для расходных документов: Reports_registryExpenses.jrxml, *(но пока это не делаем*);
- будет вызвано приложение JasperSoft Studio для редактирования шаблона.



Примечание:

- В рассматриваемом примере печатные формы создавались с использованием метакода registry и созданием шаблона отчета в JasperSoft Studio из развернутой части метакода. Шаблоны отчетов могут создаваться как из обычного описания формы, так и (*как в примере*) из развернутой части метакода.
- После создания шаблона отчета, справа от формы на служебном поле редактора IDEA появился значок принтера.

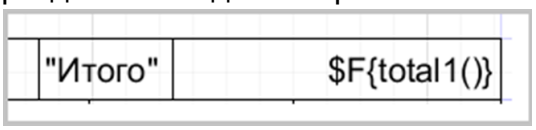


Повторный вызов JasperSoft Studio (*если он был закрыт*) вызывается нажатием на отмеченную кнопку, при этом запуск сервера приложений специально не требуется.

Редактирование шаблона отчета

Что надо сделать:

- Добавить новый раздел Summary и перенести в него поле total1(), которое определено в разделе Fields дизайнера отчетов JasperSoft Studio.

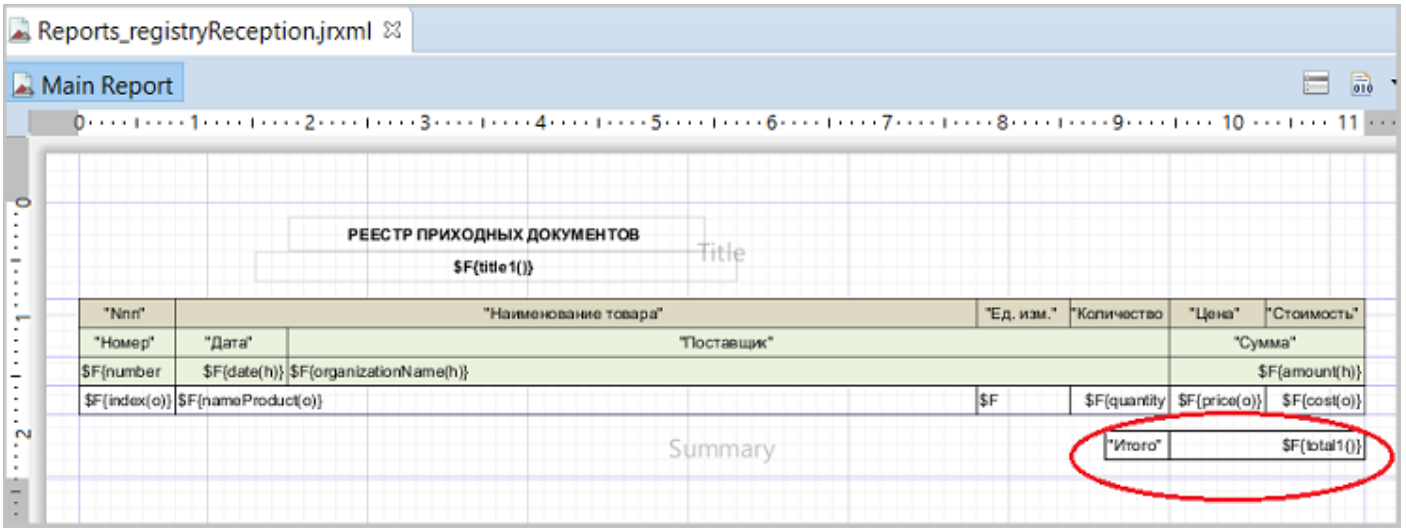


Примечание:

  - Итог по отчету total1 рассчитан через GROUP SUM перед вызовом отчета, и передан как свойство формы. Итоги можно рассчитать средствами JasperSoft Studio или передать как свойство формы.

- Преобразовать внешний вид.

Форма шаблона в JasperSoft Studio после внесенных изменений примет вид:



Форма отчета с обновленным шаблоном примет такой вид:

РЕЕСТР ПРИХОДНЫХ ДОКУМЕНТОВ

за период дат с 06.10.23 по 06.10.23г.

Nпп	Наименование товара	Ед. изм.	Количество	Цена	Стоимость
Номер	Дата	Поставщик		Сумма	
МП010061	06.10.2023	ООО Молочные продукты			278.5
1	Сметана «Минская марка» 15%, 380 г	банка	50	2.55	127.5
2	Сметана «Минская марка» 18%, 380 г	банка	50	3.02	151
ХБ010062	06.10.2023	УП Комбинат хлебопродуктов			240.5
1	Хлеб черный «Нарочанский» классический, 1200 г	штука	50	3.09	154.5
2	Хлеб белый для сэндвичей 300 г.	штука	50	1.72	86
				Итого	519

Примечание:

- Для реестра расходов пока ничего не делаем.

См. также

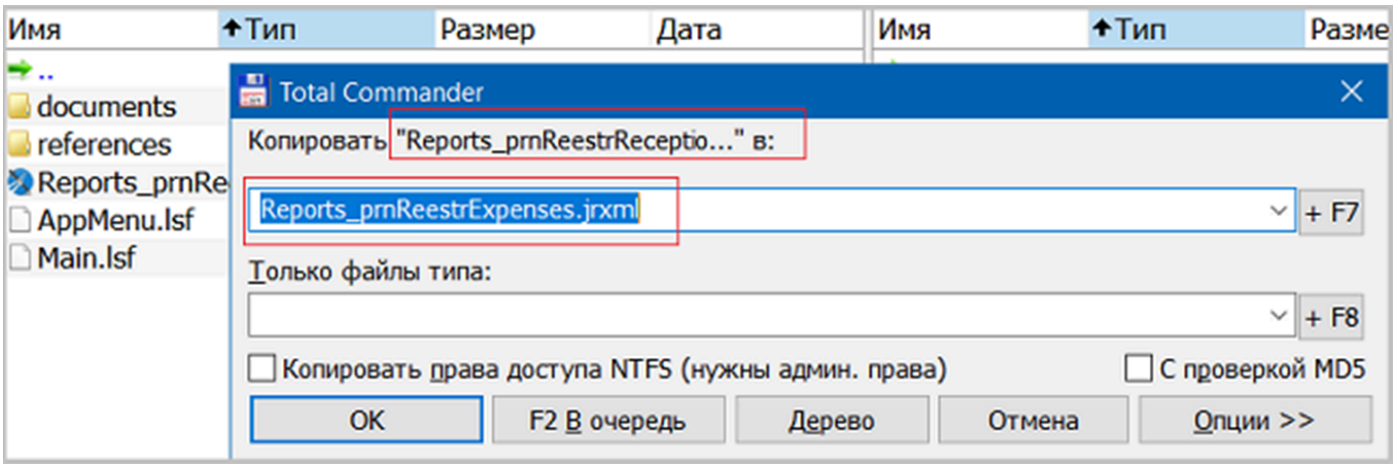
Учебник по JasperReports

Пример кода:
examples\t501_реестр_приходов\src

Реестр расходов. Копирование шаблона

Очевидным представляется создание шаблона реестра расходных документов таким же путем, как и создание шаблона реестра приходных документов. Для этого надо затратить какое-то время, и все будет работать.

Но можно поступить несколько иначе.
Если открыть шаблон Reports_registryReception.jrxml на просмотр, то окажется, что это обычный документ типа xml, в котором нигде внутри шаблона не встречается упоминаний об HeaderReception (шапка документа) и Reception (строки документа). Наоборот, везде свойства отчета представлены с использованием имен объектов печатной формы h и o, например, date(h) или nameProduct(o) и так далее. Ранее говорилось, что реестры приходных и расходных документов практически схожи и для их формирования использовалось метапрограммирование. Собственно, можно скопировать существующий шаблон Reports_registryReception.jrxml в Reports_registryExpenses.jrxml (ожидаемое по умолчанию имя отчета из формы расходов).



После выполнения копирования и вызова отчет реестра расходных документов, получим отчет следующего вида:

РЕЕСТР ПРИХОДНЫХ ДОКУМЕНТОВ

за период дат с 03.10.23 по 03.10.23г.

Nпп	Наименование товара	Ед. изм.	Количество	Цена	Стоимость
Номер	Дата	Поставщик		Сумма	
МГ010031	03.10.2023	Магазин 1			147.65
1	Молоко «Минская марка» отборное, 3.4-6% 900 мл	бутылка	20.000	2.630	52.60
2	Хлеб черный «Маг» нарезанный, 350 г	штука	15.000	1.410	21.15
3	Мясо свиное котлетное «Фермерское»	кг	5.000	14.780	73.90
МГ010032	03.10.2023	Магазин 2			209.02
1	Молоко «Минская марка» ультрапастеризованное, 2.5% 900 мл	бутылка	20.000	2.160	43.20
2	Хлеб ржаной «Чиабатта Ржаная» 1/240 г	штука	10.000	3.290	32.90
3	Хлеб черный «Маг» нарезанный, 350 г	штука	10.000	1.410	14.10
4	Мясо птицы «Голень цыпленка-бройлера» глубокозамороженная	кг	7.800	8.070	62.95
5	Мясо свиное котлетное «Фермерское»	кг	3.780	14.780	55.87
			Итого		356.67

Примечание:

- Как несложно заметить, отражены документы расхода, правда в заголовке документа значится "РЕЕСТР ПРИХОДНЫХ ДОКУМЕНТОВ", что неверно. Чтобы убрать это несоответствие, надо отредактировать шаблон.
- Иногда бывает удобно скопировать схожие шаблоны документов и потом отредактировать, при необходимости добавить новые поля и удалить ненужные. Можно сэкономить время на создании дизайна отчета.

См. также

Учебник по JasperReports

Пример кода:

examples\t502_реестр_расходов\src

Общий шаблон для реестров

Можно вообще обойтись одной формой для схожих отчетов. Для этого надо назначить принудительно для формы шаблон с заданным именем, используя блок формы REPORT, в котором указать имя шаблона в кавычках.

Для это надо:

- так как шаблона Reports_prnReestrExpenses.jrxml уже не нужен, то можем его удалить
- переименовываем Reports_prnReestrReception.jrxm (*например*) в registry.jrxml
- создаем новое локальное свойство в модуле Reports

```
title2   'Надписи' = DATA LOCAL STRING;
```

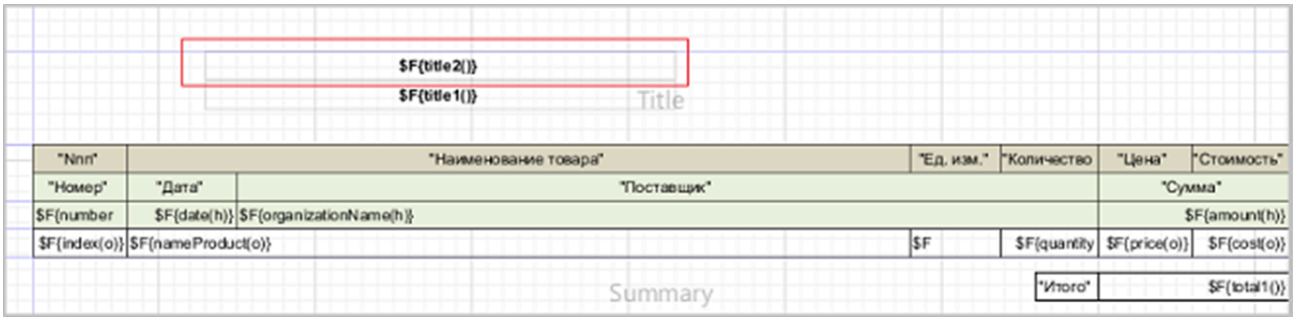
- в блоке метакода модуля Reports задаем новый параметр (имя отчета). Для печатной формы в блок PROPERTIES добавляем новое свойство title2 и блок формы REPORT с именем отчета ‘registry.jrxml’:

```
META registry(object,object2, nameReport)
   // печатная форма
   FORM registry##object 'Реестр документов'
       PROPERTIES title1(), total1(), title2()
       OBJECTS h = Header##object
       PROPERTIES (h) number, date, organizationName, amount
       OBJECTS o = object
       PROPERTIES (o) index, nameProduct, nameUnit, quantity, price, cost
       FILTERS header(o) = h
       FILTERS date(h) ⩾ dateFrom() AND date(h) ⩽ dateTo()
       FILTERS object2(h) = oid() OR NOT oid()
       REPORT 'registry.jrxml'
       ORDERS date(h), number(h), organizationName(h)
   ;
```

- Внутри действия onRegistry##object блока метакода модуля Reports в блоке кода оператора DIALOGS определяем значение локального свойства title2, которое отвечает за заголовок отчета - присваиваем значение входного параметра:

```
title2() ← nameReport;
```

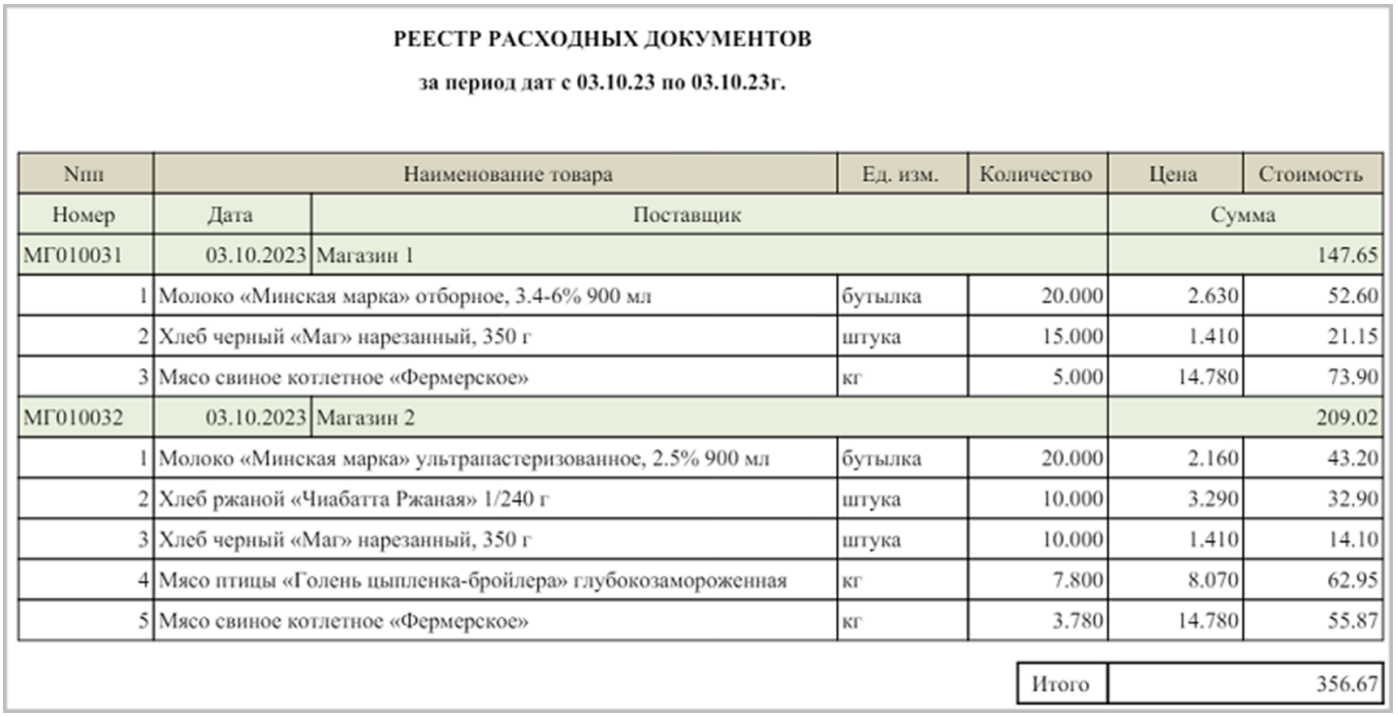
- В JasperSoft Studio на форме шаблона удаляем старый заголовок отчета "РЕЕСТР ПРИХОДНЫХ ДОКУМЕНТОВ", на его место добавляем поле title2:



- в модуле Reports исправляем вызов метакода:

```
// вызов реестра документов из форм приходов/расходов
@registry(Reception,supplier,'РЕЕСТР ПРИХОДНЫХ ДОКУМЕНТОВ');
@registry(Expenses,customer,'РЕЕСТР РАСХОДНЫХ ДОКУМЕНТОВ');
```

После старта сервера приложений, все работает как для приходных, так и для расходных документов:



РЕЕСТР РАСХОДНЫХ ДОКУМЕНТОВ

за период дат с 03.10.23 по 03.10.23г.

Nпп	Наименование товара		Ед. изм.	Количество	Цена	Стоимость
Номер	Дата	Поставщик			Сумма	
МГ010031	03.10.2023	Магазин 1				147.65
1	Молоко «Минская марка» отборное, 3.4-6% 900 мл		бутылка	20.000	2.630	52.60
2	Хлеб черный «Маг» нарезанный, 350 г		штука	15.000	1.410	21.15
3	Мясо свиное котлетное «Фермерское»		кг	5.000	14.780	73.90
МГ010032	03.10.2023	Магазин 2				209.02
1	Молоко «Минская марка» ультрапастеризованное, 2.5% 900 мл		бутылка	20.000	2.160	43.20
2	Хлеб ржаной «Чиабатта Ржаная» 1/240 г		штука	10.000	3.290	32.90
3	Хлеб черный «Маг» нарезанный, 350 г		штука	10.000	1.410	14.10
4	Мясо птицы «Голень цыпленка-бройлера» глубокозамороженная		кг	7.800	8.070	62.95
5	Мясо свиное котлетное «Фермерское»		кг	3.780	14.780	55.87
				Итого		356.67

Примечание:

- Принудительное указание имени шаблона может быть использовано не только для схожих шаблонов, но также для случаев, когда одна и та же печатная форма использует по каким-либо причинам разные шаблоны.
- Также, при использовании подотчетов, могут замещаться шаблоны отдельных подотчетов, используя блок формы REPORTFILES. В этом случае имена файлов подотчетов указываются через запятую.
- После переименования шаблона файла отчета текущие механизмы для автоматического вызова JasperSoft Studio из редактора IDEA не работают. Для редактирования шаблона можно вызвать на редактирование файл jrxml в редакторе IDEA (*для опытных*) или отдельно запустить приложение JasperSoft Studio для редактирования шаблона.

См. также

Учебник по JasperReports

Пример кода:

examples\t503_общий_шаблон\src

Использование фильтров формы отображения

Для того, чтобы для печатной формы использовать фильтры, примененные на форме отображения, можно воспользоваться оператором FILTER, который создает свойство, возвращающее TRUE, если объект входит в установленный фильтр.

```
// печатная форма
FORM registry##object 'Реестр документов'
   PROPERTIES title1(), total1(), title2()
   OBJECTS h = Header##object
   PROPERTIES (h) number, date, organizationName, amount
   OBJECTS o = object
   PROPERTIES (o) index, nameProduct, nameUnit, quantity, price, cost
   FILTERS header(o) = h
   FILTERS date(h) ⩾ dateFrom() AND date(h) ⩽ dateTo()
   FILTERS object2(h) = oid() OR NOT oid()
   FILTERS [FILTER view##object.h](h)
   REPORT 'registry.jrxml'
   ORDERS date(h), number(h), organizationName(h)
;
```

Для проверки можно задать диапазон дат отчета с 01.10.23 по 30.10.23, а в табличной части фильтром выделить какого-либо поставщика и выполнить отчет.

См. также

Пример кода:
examples\t504_использование_фильтров\src

Подотчеты. Товарный отчет

Товарный отчет – это документ, подтверждающий количество товара, числящегося за конкретным материально ответственным лицом на начало нового отчетного периода, и отражающий движение товаров за прошедший отчетный период.

В товарный отчет включаются расчеты сумм:

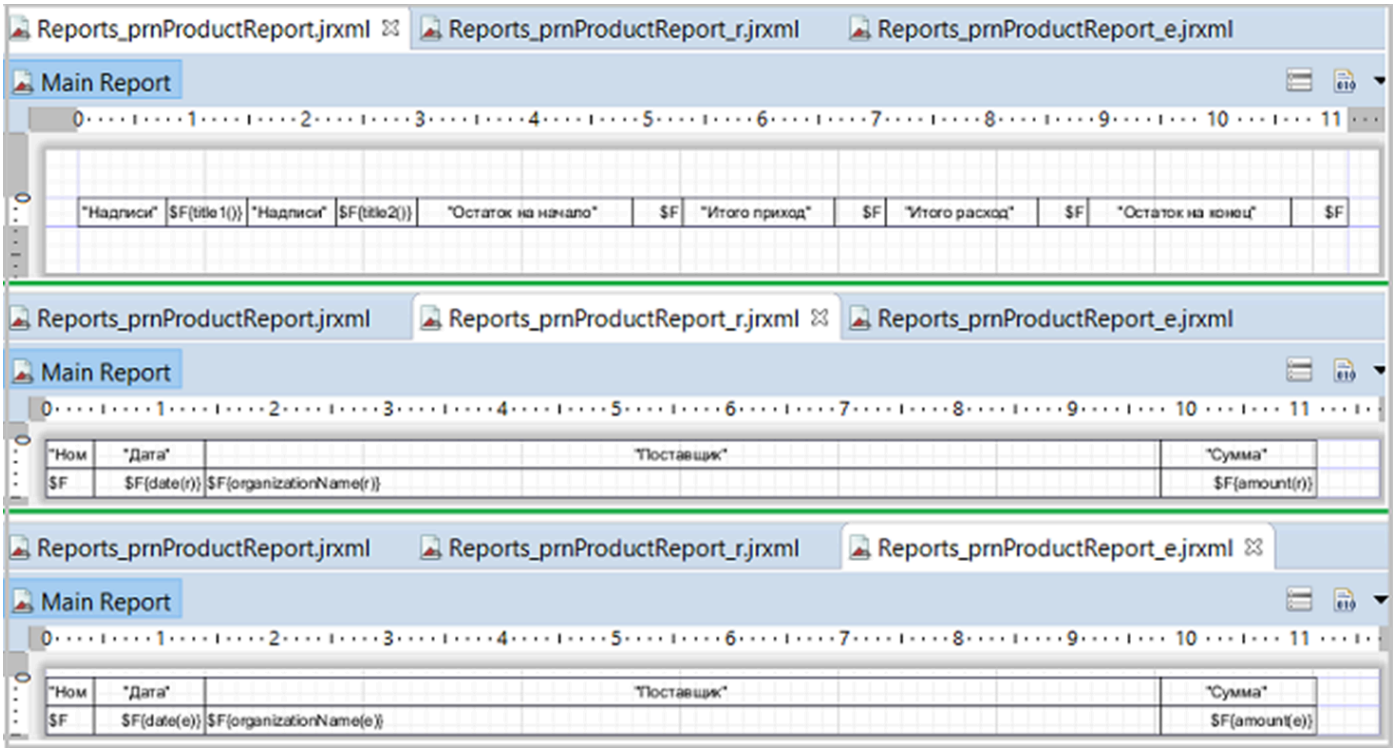
1. остаток на начало, который рассчитывается как все приходы минус все расходы с самого начала до даты отчетного периода;
2. движение по приходам и расходам и их итоги за отчетный период;
3. остаток на конец расчетного периода, рассчитываемый как сумма остатка на начало + приходы за отчетный период - расходы за отчетный период.

Все доработки выполним в конце модуля Reports

```
// ТОВАРНЫЙ ОТЧЕТ
num2num (NUMERIC n1) = NUMERIC[10,2](OVERRIDE n1, 0.00); // если null, возвращает 0.00
balanceReceptionStart 'Остаток на начало, приходы' =
   GROUP SUM amount(HeaderReception o) IF date(o) < dateFrom();
balanceExpensesStart  'Остаток на начало, расходы' =
   GROUP SUM amount(HeaderExpenses o) IF date(o) < dateFrom();
balanceStart 'Остаток на начало' = num2num(balanceReceptionStart() (-) balanceExpensesStart());

totalReception 'Итого приход' =
   GROUP SUM amount(HeaderReception o) IF date(o) ⩾ dateFrom() AND
       date(o) ⩽ dateTo();
totalExpenses 'Итого расход' =
   GROUP SUM amount(HeaderExpenses o) IF date(o) ⩾ dateFrom() AND
       date(o) ⩽ dateTo();

balanceEnd 'Остаток на конец' = num2num(balanceStart() (+)
   totalReception() (-) totalExpenses());

FORM prnProductReport 'Товарный отчет'
   PROPERTIES() title1, title2, balanceStart, totalReception, totalExpenses, balanceEnd

   OBJECTS r = HeaderReception
   PROPERTIES (r) number, date, organizationName, amount
   FILTERS date(r) ⩾ dateFrom() AND date(r) ⩽ dateTo()
   ORDERS date(r), number(r)

   OBJECTS e = HeaderExpenses
   PROPERTIES (e) number, date, organizationName, amount
   FILTERS date(e) ⩾ dateFrom() AND date(e) ⩽ dateTo()
   ORDERS date(e), number(e)
;

onProductReport 'Товарный отчет' (Header h) {
   formType() ← NULL;
   DIALOG prnParameters OBJECTS dateFrom=date(h) INPUT, dateTo=date(h) INPUT, p INPUT DO {
       dateFrom() ← dateFrom; dateTo() ← dateTo;
       title1() ← 'ТОВАРНЫЙ ОТЧЕТ';
       title2() ← 'за период дат с ' + toDateDDMMYY(dateFrom()) +
           ' по ' + toDateDDMMYY(dateTo()) + 'г.';


      CASE
           WHEN OutPrint.xls = p THEN
               PRINT prnProductReport XLSX;
```

```
            WHEN OutPrint.doc = p THEN
                PRINT prnProductReport DOC;
            ELSE
                PRINT prnProductReport PDF;
    }
}


EXTEND FORM viewReception PROPERTIES (h) onProductReport;
DESIGN viewReception {
    TOOLBARLEFT(h) {
        MOVE PROPERTY (onProductReport(h)) { marginLeft=100; };
    }
}

EXTEND FORM viewExpenses PROPERTIES (h) onProductReport;
DESIGN viewExpenses {
    TOOLBARLEFT(h) {
        MOVE PROPERTY (onProductReport(h)) { marginLeft=100; };
    }
}
```

После запуска сервера приложений и создания шаблона отчетов, будет создано 3 jrxml файла:



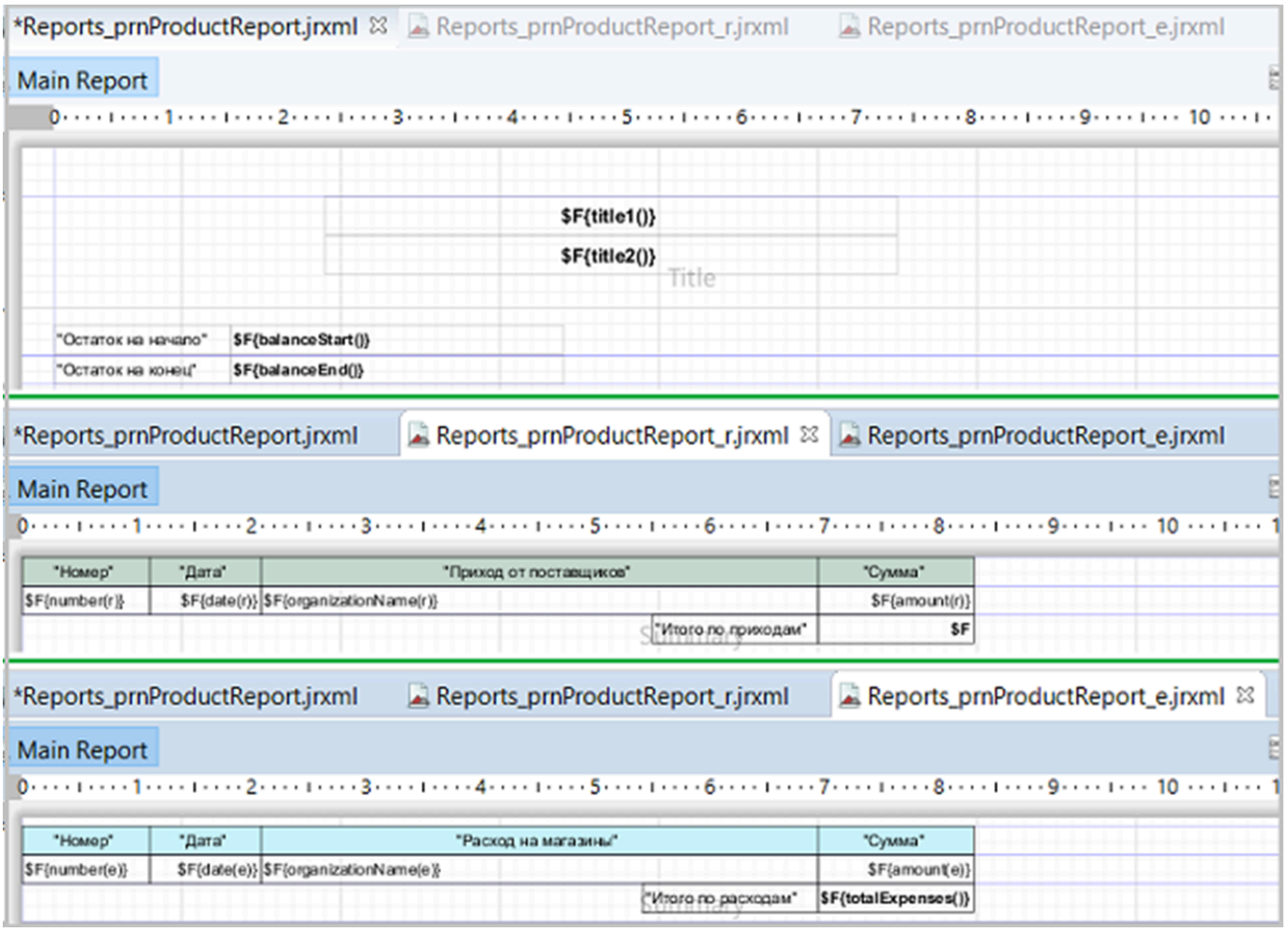
Почему создалось 3 три файла на один отчет? В данном случае в печатной форме присутствуют два не связанных между собой объекта и отдельные локальные свойства, поэтому платформа автоматически формирует:

1. основной шаблон - контейнер подотчетов, содержащий локальные свойства, раздел Fields
2. шаблон, связанный с приходами, являющийся подотчетом (subreport)
3. шаблон, связанный с расходами, являющийся подотчетом

Что надо сделать:

- в основном шаблоне добавить раздел Summary (резюме) и перенести туда поле balanceEnd() - Остаток на конец
- из основного шаблона убрать поля totalReception(), totalExpenses();
- в шаблон [2] в раздел Fields добавить новое поле totalReception(), Class java.math.BigDecimal, добавить раздел Summary, в который поместить поле totalReception() - Итого по приходам;
- в шаблон [3] в раздел Fields добавить новое поле totalExpenses(), Class java.math.BigDecimal, добавить раздел Summary, в который поместить поле totalReception() - Итого по приходам;
- для того чтобы отображаемое значение дат выводились в привычном виде необходимо изменить текущий класс для полей date(e) и date(r) на java.util.Date. На форме Properties, вкладка Text Field в поле Pattern помещаем значение паттерна dd.MM.yyyy;
- навести общий порядок с колонками.

После доработки шаблонов, они примут следующий вид:



Товарный отчет примет вид на предпросмотре:

ТОВАРНЫЙ ОТЧЕТ

за период дат с 02.10.23 по 03.10.23г.

Остаток на начало **0.00**

Номер	Дата	Приход от поставщиков	Сумма
МП010021	02.10.2023	ООО Молочные продукты	435.00
ХБ010021	02.10.2023	УП Комбинат хлебопродуктов	273.30
МИ010032	03.10.2023	УП Мясокомбинат	1586.51
МП010031	03.10.2023	ООО Молочные продукты	483.00
		Итого по приходам	**2777.81**

Номер	Дата	Расход на магазины	Сумма
МГ010031	03.10.2023	Магазин 1	147.65
МГ010032	03.10.2023	Магазин 2	209.02
		Итого по расходам	**356.67**

Остаток на конец **2421.14**

См. также

Документация. How-to: Отчеты. Пример 3

Здесь. Добавление разделов

Здесь. Добавление полей в отчет

Пример кода:

examples\t505_товарный_отчет\src

Тема 6. Импорт и Экспорт

Рассматриваемые вопросы:

- Общие сведения
- Импорт данных. Постановка задачи
  - Импорт из DBF и XLS
  - Импорт из XML
- Экспорт данных. Постановка задачи
  - Предварительные действия
  - Экспорт DBF, XLS, XML

Общие сведения

Вопросы импорта и экспорта достаточно важны для программ, так как позволяют осуществлять интеграцию с другими приложениями. При этом направление взаимодействия может быть обоюдным, то есть относительно текущей программы может осуществляться как импорт, так и экспорт данных.

Простой пример, но достаточно жизненный.
Есть приложение складского учета (*наше приложение*). Поставщики товара предлагают кроме бумажной накладной ее электронный аналог в файлах разных форматов, а также предоставлять им сведения, также в разных форматах, о реализации их товаров для планирования будущих отгрузок. Достигаемая при этом цель: возросшая скорость обработки информации, а также точность принимаемых данных – исключаются ошибки ввода.

Платформа lsFusion обладает продвинутыми механизмами для импорта и экспорта данных в разных форматах.

Импорт данных. Постановка задачи

Необходимо организовать от поставщиков импорт товарных накладных. Так как программное обеспечение у поставщиков разное, то данные будут представлены в разных форматах:

- УП Комбинат хлебопродуктов, DBF
- ООО Молочные продукты, Excel
- УП Мясокомбинат, XML

Предполагается, что при доставке товара на оптовую базу, вместе с товаром и накладными будет присутствовать содержание накладных в электронном виде на USB-флеш накопителях ("флэшка"). Задача оператора по приемке товара, кроме принятия товара, провести электронный документ и сверить итоги.

Импортируемая на "флэшках" информация может содержать несколько товарных накладных, что обусловлено спецификой отгрузки и формирования документов на складах поставщиков.

Интерфейсно по окончании процесса импорта данных указатель в шапке приходов должен становиться на последнюю добавленную запись, а импортированные накладные должны получить отметку, чтобы можно было выделить групповым фильтром "Отмеченные" все импортированные накладные для сверки. Ранее проведенные накладные игнорируются по условию номер и дата накладной.

Примечание:

- для имитации съемных накопителей выступает каталог: Examples\data\import, содержащий импортируемые данные.

См. также

Архив SQL:
examples\t601_импорт_данных\sql
версия БД до начала выполнения всех импортов

Импорт из DBF и XLS

В основном каталоге lsFusion создадим каталог **exchange** (обмен) – это будет каталог для модулей, связанных так или иначе с чтением данных. В каталоге exchange надо создать новый модуль Import.

Перед тем как продолжить создание модуля, необходимо определить некоторые условия:

- По условиям задачи имеется 3 поставщика, каждый из которых представляет данные в своем формате. В предлагаемом примере, чтобы упростить демонстрацию и не строить сложных схем, расширение файла импорта связывается со своим поставщиком. *В реальной жизни можно было поступить иначе, например, за каждой организацией закрепить свой процесс импорта данных, договорится с поставщиками о формате передаваемых данных, наименовании файлов и так далее.*
- При импорте данных необходимо учесть, что работа затронет 5 классов: шапки и строки приходов, а также 3 справочника. При этом надо учесть, что импортируемые документы не должны повторятся, поэтому должна быть организована проверка на дату и номер накладной для шапки приходов.
- При импорте наименований справочников товаров, единиц измерения, товарных групп не должны создаваться новые наименования, если они уже присутствуют в справочной системе. Стоит оговорится, что существующее ограничение на уникальность не даст это сделать. Если появляются новые наименования, то они должны быть добавлены. И соответственно, найденные или добавленные наименования должны быть связаны со строками приходов.
- Импорт новых товарных групп привязывается к родительской группе "ВСЕ".
- Если при импорте товаров отсутствуют товарные группы, то товары привязываются к товарной группе "Прочие", и могут быть переназначены потом. Для этой цели в справочник товарных групп добавлена группа "Прочие" (см. examples\t601_импорт_данных\sql).

Выполним анализ импортируемых документов. Приведем в виде таблицы структуру и натурное (визуальное) содержание информации импортируемых файлов.

Структура DBF файла, поставщик "УП Комбинат хлебопродуктов"; в файле может содержаться несколько накладных:

Поле	Тип	Размер	Описание	Соответствие
npp	C	4	Номер по порядку	нет
tn	C	10	Номер накладной	HeaderReception.number
dn	D	8	Дата накладной	HeaderReception.date
name	C	100	Наименование товара	Product.name
ei	C	10	Единица измерения	Unit.name
gruppa	C	300	Товарная группа	Group.name
q	C	10	Количество	Reception.quantity
cena	C	10	Цена	Reception.price
suma	C	10	Стоимость	нет
operator	C	15	Кладовщик	нет
sclad	C	4	Склад отгрузки	нет

Фактическое содержание данных в файле x20231010.dbf:

X20231010

	Npp	Tn	Dn	Name	Ei	Grupa	Q	Cena	Suma	Operator	Sclad
▶	1	ХБ600001	10.10.2023	Хлеб черный «Маг» нарезанны	нарезка	Хлеб черный	25.000	1.28	32.00	Гришина	12
	2	ХБ600001	10.10.2023	Хлеб черный «Марусин» нарез	штука	Хлеб черный	20.000	1.42	28.40	Гришина	12
	3	ХБ600001	10.10.2023	Хлеб черный «Нарочанский» к	штука	Хлеб черный	20.000	3.09	61.80	Гришина	12
	1	ХБ600012	10.10.2023	Булочка «Лакомка маковая» 1	штука	Булочки	20.000	1.05	21.00	Зотова	6
	2	ХБ600012	10.10.2023	Булочка «Ароматная» 1/100 г	штука	Булочки	20.000	1.19	23.80	Зотова	6

Структура XLS файла, поставщик "ООО Молочные продукты"; в файле может содержаться несколько накладных:

Колонка	Описание	Соответствие
A	Номер накладной	HeaderReception.number
B	Дата накладной	HeaderReception.date
C	Товарная группа	Group.name
D	Название товара	Product.name
E	Единицы измерения	Unit.name
F	Количество	Reception.quantity
G	Цена	Reception.price
H	Стоимость	нет
I	Размер	нет
J	Единицы	нет

Фактическое содержание данных в файле e20231011.xlsx:

	A	B	C	D	E	F	G	H	I	J
1	Номер	Дата	Группа	именован	Ед. изм	Количество	Цена	Стоимость	Размер	Единицы
2	МП1000001	11.10.2023	Молоко	Молоко «М	бутылка	50	2.39	119.50	900	мл
3	МП1000001	11.10.2023	Молоко	Молоко «М	бутылка	50	1.96	98.00	900	мл
4	МП1000102	11.10.2023	Молоко	Молоко «М	бутылка	20	2.39	47.80	900	мл
5	МП1000102	11.10.2023	Сметана	Сметана «С	банка	20	2.55	51.00	380	грамм
6	МП1000102	11.10.2023	Сметана	Сметана «С	банка	20	3.02	60.40	380	грамм
7	МП1000103	11.10.2023	Кефир	Кефир «Ми	бутылка	50	1.77	88.50	900	мл
8	МП1000103	11.10.2023	Кефир	Кефир «Ми	ультра-п	30	1.44	43.20	500	мл
9	МП1000103	11.10.2023	Творог	Творог «М	бутылка	40	0.92	88.50	130	грамм

Анализ информации, позволяет судить о том, что в целом структуры содержат требуемые для импорта сведения: дата и номер накладной, наименование товара, единицы измерения, товарная группа (кроме XML файла), количество, цена. Прочие, отражаемые в структурах данные, являются избыточными и участия в импорте не принимают, в том числе стоимость и итоги по накладной, которые рассчитываются автоматически.

Технически импорт из плоских документов DBF и XLS мало чем отличается друг от друга. В одном случае мы оперируем импортируемыми полями, в другом - импортируемыми колонками. Поэтому импорт можно сделать общим для этих структур:

```
MODULE Import;

REQUIRE Reception;

number = DATA LOCAL STRING[10] (INTEGER);
date = DATA LOCAL DATE (INTEGER);
group = DATA LOCAL STRING (INTEGER);
product = DATA LOCAL STRING[100] (INTEGER);
unit = DATA LOCAL STRING[100] (INTEGER);
price = DATA LOCAL NUMERIC[10,2] (INTEGER);
quantity = DATA LOCAL NUMERIC[10,3] (INTEGER);


reset 'Сбросить локальные свойства' () {
   number(INTEGER i) ← NULL;
   date(INTEGER i) ← NULL;
   group(INTEGER i) ← NULL;
   product(INTEGER i) ← NULL;
   unit(INTEGER i) ← NULL;
   price(INTEGER i) ← NULL;
   quantity(INTEGER i) ← NULL;
}

import 'Импорт'(FILE file, Organization organization) {
   reset();
   CASE
       WHEN istartsWith(extension(file), 'XLS') THEN
           IMPORT XLS HEADER FROM file TO
               number = 'Номер', product = 'Наименование', date = 'Дата',
               quantity = 'Количество', price = 'Цена', unit = 'Ед. изм', group = 'Группа'
       WHEN upper(extension(file)) = 'DBF' THEN
           IMPORT DBF CHARSET 'CP1251' FROM file TO
               number = tn, product = name, date = dn, quantity = q, price = cena,
               unit = ei, group = grupa
       ELSE {
           MESSAGE 'Неизвестный формат';
           RETURN;
   }

   // Создаем новые единицы измерений, которых нет в справочнике
   FOR [GROUP MAX INTEGER i BY unit(i)](STRING unitName) AND
       NOT Unit.unique(upper(trim(unitName)) AS STRING) NEW unit = Unit DO {
       name(unit) ← STRING[100](unitName);
   }

   // Создаем группы товаров, если они есть в файле импорта, но их нет в справочнике групп
   FOR [GROUP MAX INTEGER i BY group(i)](STRING groupName) AND
       NOT Group.unique(upper(trim(groupName)) AS STRING) NEW group = Group DO {
       parent(group) ← Group.unique('ВСЕ');
       name(group) ← STRING[100](groupName);
   }

   // Создаем товары, которых нет в справочнике
   FOR INTEGER index = [GROUP MAX INTEGER i BY product(i)](STRING productName) AND
       NOT Product.unique(upper(trim(productName)) AS STRING) NEW product = Product DO {
       group(product) ← OVERRIDE Group.unique(upper(trim(group(index)))),
                                                                   Group.unique('ПРОЧИЕ');
       name(product) ← STRING[100](productName);
   }
```

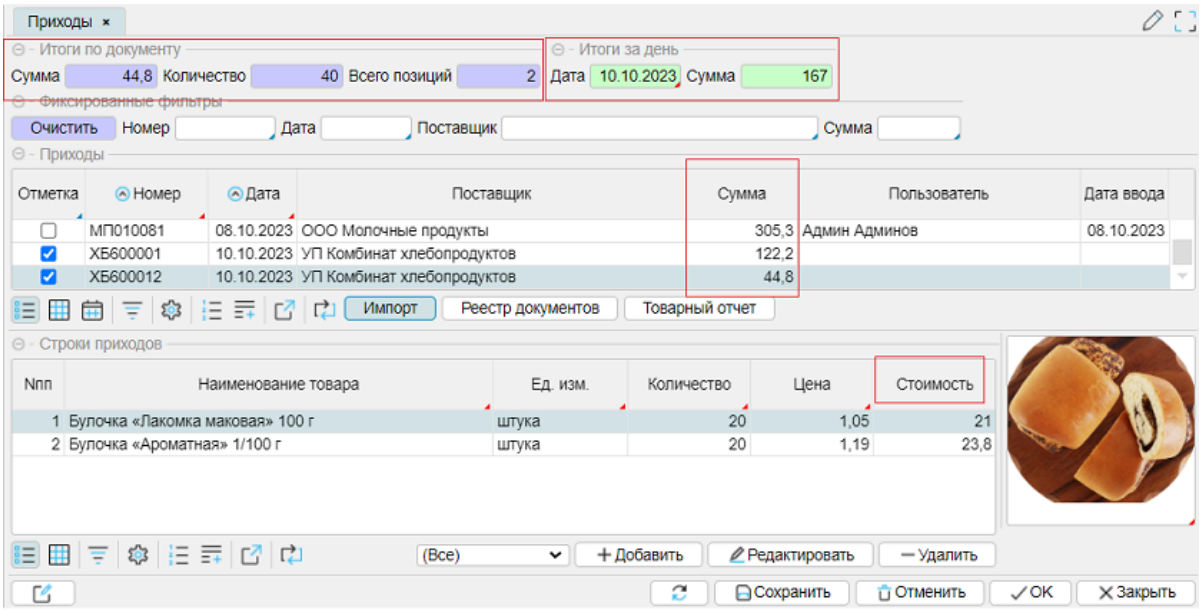
```
// Заполним список документов, которых еще нет в базе
   LOCAL create = INTEGER (STRING[10], DATE);
   create(STRING[10] number, DATE date) ←
       [GROUP MAX INTEGER i BY number(i), date(i)](number, date)
       IF NOT Documents.unique(number, date);

   IF NOT GROUP MAX create(STRING[10] number, DATE date) THEN {
       canceled() ← TRUE;
       MESSAGE 'Импорт данных не выполнен\Возможно данные были импортированы ранее';
       RETURN;
   }

   // Создаем шапки документов, которые не были импортированы ранее
   FOR create(STRING[10] number, DATE date) NEW document = HeaderReception DO {
       number(document) ← number;
       date(document) ← date;
       supplier(document) ← organization;
       mark(document) ← TRUE;
   }

   // Создаем строки импортированных документов:
   FOR STRING number = number(INTEGER i) AND DATE date = date(i) AND
       Header document = Documents.unique(number, date) AND create(number, date)
       NEW reception = Reception DO {
           header(reception) ← document;
           product(reception) ← Product.unique(upper(product(i)));
           price(reception) ← price(i);
           quantity(reception) ← quantity(i);
           unit(reception) ← Unit.unique(upper(unit(i)));
   }
   MESSAGE 'Данные импортированы!';
}

import 'Импорт' () {
   INPUT dbfile = FILE DO {
       CASE
           WHEN upper(extension(dbfile)) = 'DBF'  THEN
               import(dbfile, Organization.unique('УП КОМБИНАТ ХЛЕБОПРОДУКТОВ'));
           WHEN upper(extension(dbfile)) = 'XLSX' THEN
               import(dbfile, Organization.unique('ООО МОЛОЧНЫЕ ПРОДУКТЫ'));
           WHEN upper(extension(dbfile)) = 'XML'  THEN;
               // Обработка не назначена
           ELSE
               MESSAGE 'Не ожидаемое расширение в имени файла\Выполнение прервано!';
   }
   IF NOT canceled() THEN {
       SEEK LAST viewReception.h; // указатель на последнюю запись
   }
}

EXTEND FORM viewReception PROPERTIES import();
DESIGN viewReception { TOOLBARLEFT (h) { MOVE PROPERTY (import()); } }
```

Примечание:

- В созданном модуле присутствует особенность: существуют два действия с именем "import". В платформе lsFusion два действия или свойства могут иметь одинаковое имя внутри одного пространства имен при условии, что их параметры различаются либо по количеству, либо по классам. Это не влияет на импорт, а приведено в качестве примера, как один из способов наименования действий. Нижнее действие import является, как бы главным: оно считывает импортируемый файл, связывает его по

расширению с организацией и вызывает действие import, которое, собственно, и реализует алгоритм импорта.

- При импорте из XLS используется ключевое слово HEADER, поэтому в качестве соответствия имен колонок выбрана 1-строка Excel файла. Если не использовать ключевое слово HEADER или использовать NOHEADER, то соответствие импортируемым колонкам задается относительно их действительных имен (A..ZZ), например: number = A, product = D, date = B, quantity = F, price = G, unit = E, group = C.
- В модуле назначены общие локальные свойства для разных импортов и действие по их очистке reset, вызываемое перед каждым импортом.
- Циклы FOR для импорта наименований справочников (единиц измерения, товарных групп и товаров) эффективны, так как обрабатывают только отсутствующие в справочниках наименования.
- Формирование шапки и строк документа учитывают ранее не проведенные документы, используя композицию create в условии FOR.
- Свойства userName (Пользователь) и userDate (Дата ввода) заполнятся автоматически после нажатия кнопки "Сохранить" внизу формы.
- До операции сохранения данных после выполненного импорта, все фильтры и расчеты действуют по отношению к ним – это приятная особенность платформы lsFusion. Потому можно, например, сверить проводимые данные до выполнения финальной записи.
- После импорта из DBF появится новая единица измерения: нарезка. Все прочие импортируемые товары вносились ранее, будут соответствовать своим товарным группам и иметь соответствующие изображения.



См. также

Документация. Структурированное представление
Документация. Оператор IMPORT
Документация. How-to: Импорт данных

Пример кода:
examples\t602_импорт_dbf_xls\src
Архив SQL:
examples\t602_импорт_dbf_xls\sql

Импорт из XML

Структура XML файла "УП Мясокомбинат" предлагает информацию об отгруженных накладных всегда в файле с именем "data.xml". В файле может содержаться несколько накладных:

Уровень	Тег	Описание	Соответствие
0	V8Exch:Data	Корень	нет
1	v8:Документ	Узел, шапка документа	класс HeaderReception
2	v8:Операция	Название операции	нет
2	v8:Номер	Номер накладной	HeaderReception.number
2	v8:Дата	Дата накладной	HeaderReception.date
2	v8:Сумма	Сумма по документу	нет
2	v8:СтрокиДокумента	Узел	класс Reception
3	v8:НазваниеТовара	Название товара	Product.name
3	v8:ЕдиницаИзмерения	Единицы измерения	Unit.name
3	v8:ОтпускнаяЦена	Цена	Reception.price
3	v8:Количество	Количество	Reception.quantity
3	v8:Стоимость	Стоимость	нет

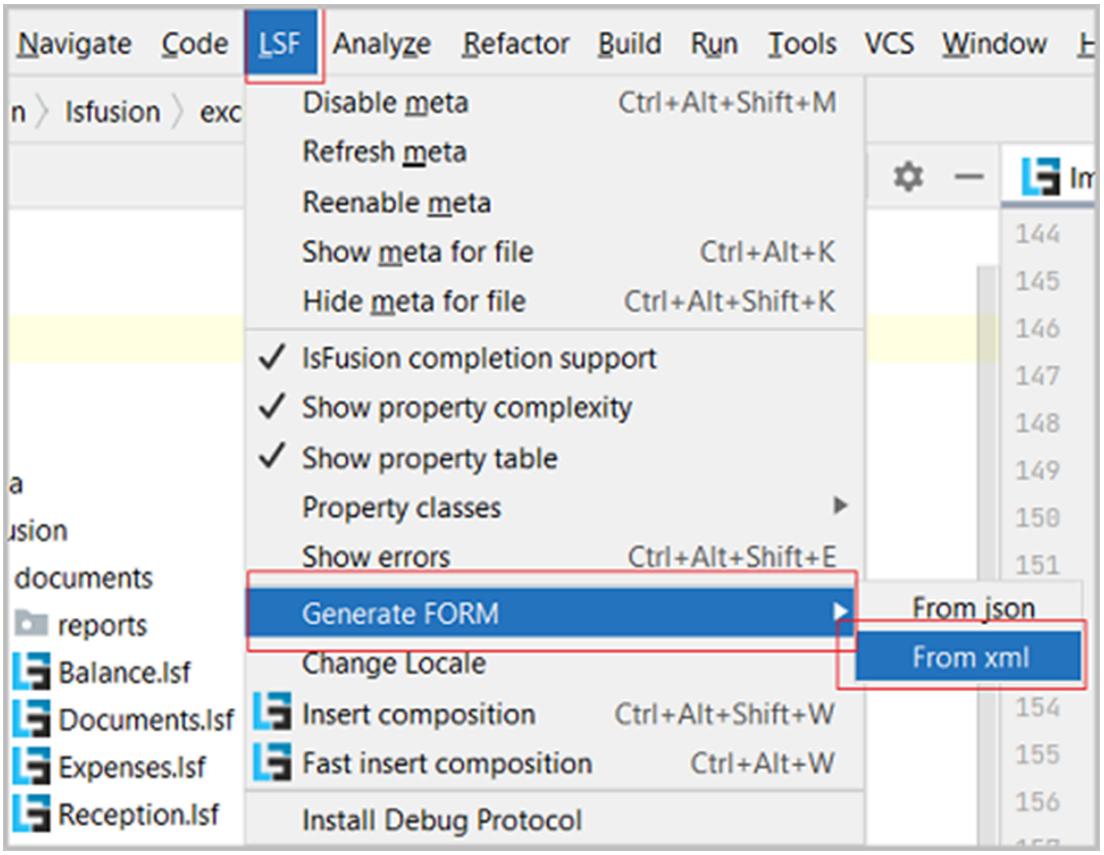
Фрагмент содержания информации в файле data.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <V8Exch:_1CV8DtUD xmlns:V8Exch="http://www.1c.ru/V8/1CV8DtUD/" xmlns:core="http://v8.1c.ru/data
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  - <V8Exch:Data>
    - <v8:Документ>
        <v8:Операция>Расход</v8:Операция>
        <v8:Номер>МИ100001</v8:Номер>
        <v8:Дата>09.10.2023</v8:Дата>
        <v8:Сумма>232.50</v8:Сумма>
      - <v8:СтрокиДокумента>
          <v8:НазваниеТовара>Колбаса вареная «Докторская» высший сорт, 400 г</v8:НазваниеТовара>
          <v8:ЕдиницаИзмерения>упаковка</v8:ЕдиницаИзмерения>
          <v8:ОтпускнаяЦена>3.25</v8:ОтпускнаяЦена>
          <v8:Количество>20.000</v8:Количество>
          <v8:Стоимость>65.00</v8:Стоимость>
        </v8:СтрокиДокумента>
```

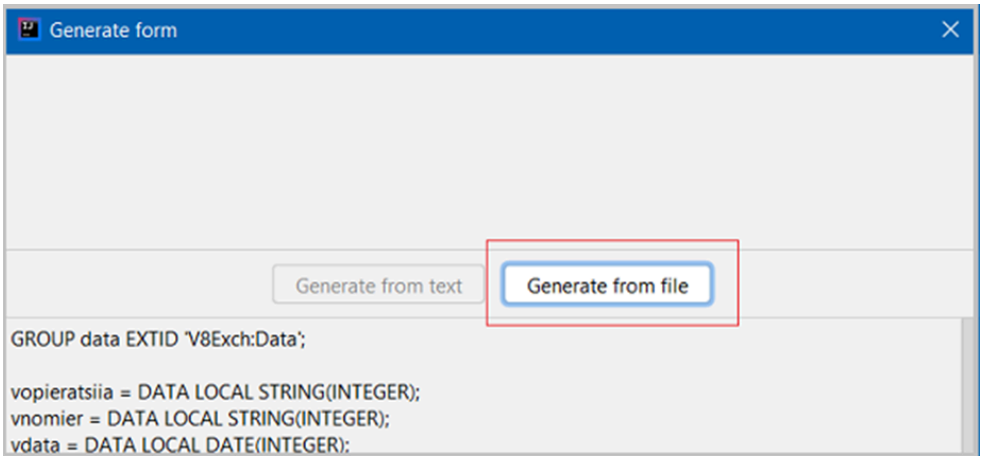
В отличии от плоских документов, представленных в формате DBF и XLS, структура XML является древовидной и может отражать вложенные друг в друга данные. В конкретном примере это накладные. Каждая накладная содержит несколько товаров (наименование, единицы измерения, количество, цена).

Отличительная особенность содержимого XML файла от файлов XLS и DBF – это отсутствие реквизита, соответствующего товарной группе. Это сделано намеренно, для примера, но не помешает выполнить импорт данных.

Импорт из XML выполним, используя форму.
Создание формы для импорта из XML может вызвать затруднение. В редакторе IDEA плагин lsFusion предоставляет механизм для формирования форм при импорте из XML и Json. Расположение в меню "LSF - Generate FORM - From xml" или "LSF - Generate FORM - From json".



На экране откроется форма, в которую нужно ввести текст XML (*верхнее окно формы*) или добавить через диалог открытия файлов:



В нижнем окне будет сгенерирован код "как правильно".

Особенность сгенерированного кода в том, что все теги, представленные в импортированном документе, записаны кириллицей, поэтому для наименования свойств использовалась транслитерация (*выражение символов кириллицы символами латиницы)*.

Можно написать отдельный модуль для импорта из XML, но он практически повторит существующий модуль для импорта из DBF и XLS. Поэтому совместим сгенерированный код формы и созданные свойства с тем, что уже имеется в модуле Import. При этом сгенерированные свойства (они несколько избыточны, так как взяты все теги) заменим существующими свойствами. Для этого в модуле Import проведем небольшие доработки:

- добавим новые свойства и форму (после создания локальных свойств):

```
GROUP data EXTID 'V8Exch:Data';
document = DATA LOCAL INTEGER (INTEGER);

FORM importXml FORMEXTID 'V8Exch=http://www.1c.ru/V8/1CV8DtUD/:_1CV8DtUD'
   OBJECTS header = INTEGER EXTID 'v8:Документ' IN data
   PROPERTIES (header) number EXTID 'v8:Номер', date EXTID 'v8:Дата'
   FILTERS imported(header)

   OBJECTS line = INTEGER EXTID 'v8:СтрокиДокумента'
   PROPERTIES (line) product EXTID 'v8:НазваниеТовара',
       unit EXTID 'v8:ЕдиницаИзмерения',
       price EXTID 'v8:ОтпускнаяЦена', quantity EXTID 'v8:Количество'
   FILTERS document(line) == header
;
```

- добавим очистку локального свойства document в конец действия reset:

```
document(INTEGER i) ← NULL;
```

- для действия import (верхний) в структуру оператора CASE добавим импорт из XML:

```
WHEN upper(extension(file)) = 'XML' THEN
   IMPORT importXml XML FROM file;
```

- учтем изменения при проведении строк накладной, как для DBF и XLS, так и для XML, заменой существующего блока оператора FOR действия import (верхний):

```
// Создаем строки импортированных документов:
// пробуем для структурированных форматов, а если не вышло - для плоских
FOR INTEGER i = document(INTEGER j) AND i = create(number(i), date(i)) AND
   Header document = Documents.unique(number(i), date(i))
   NEW reception = Reception DO {
   header(reception) ← document;
   product(reception) ← Product.unique(upper(product(j)));
   price(reception) ← price(j);
   quantity(reception) ← quantity(j);
   unit(reception) ← Unit.unique(upper(unit(j)));
} ELSE FOR STRING number = number(INTEGER i) AND DATE date = date(i) AND
   Header document = Documents.unique(number, date) AND create(number, date)
   NEW reception = Reception DO {
       header(reception) ← document;
       product(reception) ← Product.unique(upper(product(i)));
       price(reception) ← price(i);
       quantity(reception) ← quantity(i);
       unit(reception) ← Unit.unique(upper(unit(i)));
}
```

- в нижнем действии import, там, где:

```
WHEN upper(extension(dbfile)) = 'XML'  THEN;
   // Обработка не назначена
```

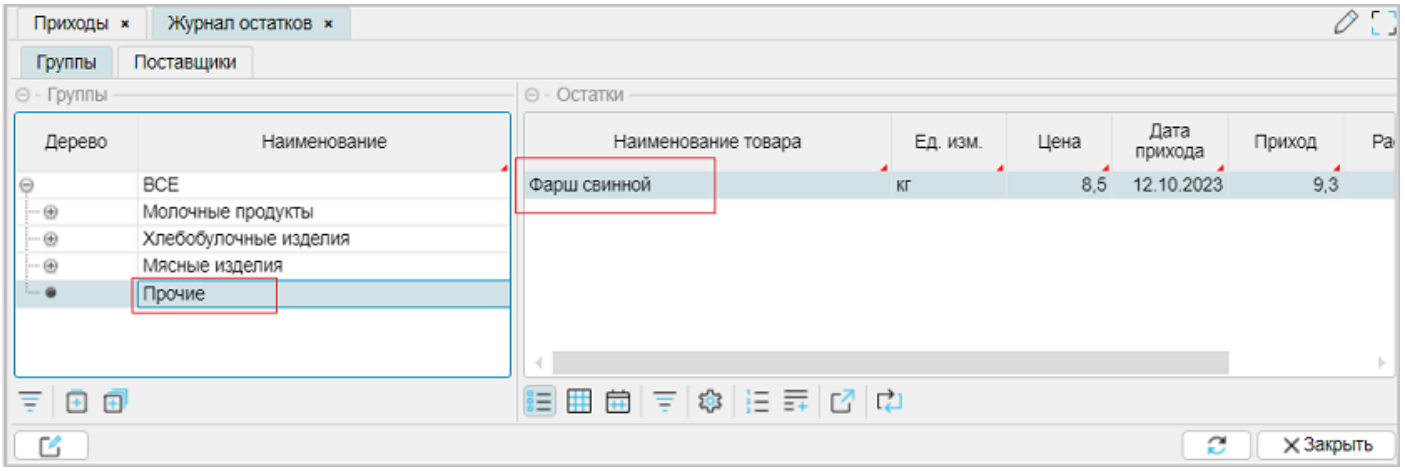
заменим на:

```
WHEN upper(extension(dbfile)) = 'XML'  THEN
```

```
import(dbfile, Organization.unique('УП МЯСОКОМБИНАТ'));
```

Примечание:

- Особенность использования оператора FOR при записи импортируемых строк накладной заключается в использовании конструкции FOR .. ELSE .. FOR. Если импортируется DBF (XLS), то свойство document будет NULL и выполнится FOR после ELSE. Соответственно, если импортируются данные из XML, будет выполнен первый FOR.
- После импорта из XML новый товар "Фарш мясной" будет добавлен в товарную группу "Прочие" – это можно увидеть в журнале остатков по группам. Все остальные товары вносились ранее, будут соответствовать своим товарным группам и иметь соответствующие изображения.



См. также

Документация. Структурированное представление
Документация. How-to: Импорт данных
Документация. Оператор IMPORT
Документация. Инструкция GROUP
Учебник по XML для начинающих

Пример кода:
examples\t603_импорт_xml\src

Архив SQL:
examples\t603_импорт_xml\sql

Экспорт данных. Постановка задачи

Возникла необходимость экспорта данных для поставщиков, работающих с оптовой базой.
После внедрения электронных копий накладных, поставщики попросили передавать им реализацию их товаров за выбранный период. Свое решение они мотивировали необходимостью вести статистический учет продаж их продукции для того, чтобы управлять своими складскими запасами: на основании расхода по заявке от базы и отгрузке с базы, они могут предполагать будущие потребности оптовой базы на приобретение товаров от них.

Структура представляемых сведений:

- дата отгрузки
- номер связанного расходного документа
- получатель товара (магазин)
- наименование товара
- единица измерений
- количество
- цена
- стоимость

Технически сведения поставщики будут забирать на "флэшках" при очередной поставке товара. Каталог формирования сведений Examples\data\export. Формат файлов и их наименование для поставщиков:

- УП Комбинат хлебопродуктов, DBF, xleb_ггггммdd.dbf
- ООО Молочные продукты, Excel, milk_ггггммdd.xls
- УП Мясокомбинат, XML, myso_ггггммdd.xml

В наименовании файла присутствует конечная дата периода запроса данных.
Технически сведения будут забираться из каталога обмена.

Предварительные действия

В каталоге exchange создадим модуль Export.

В модуле Export создадим:

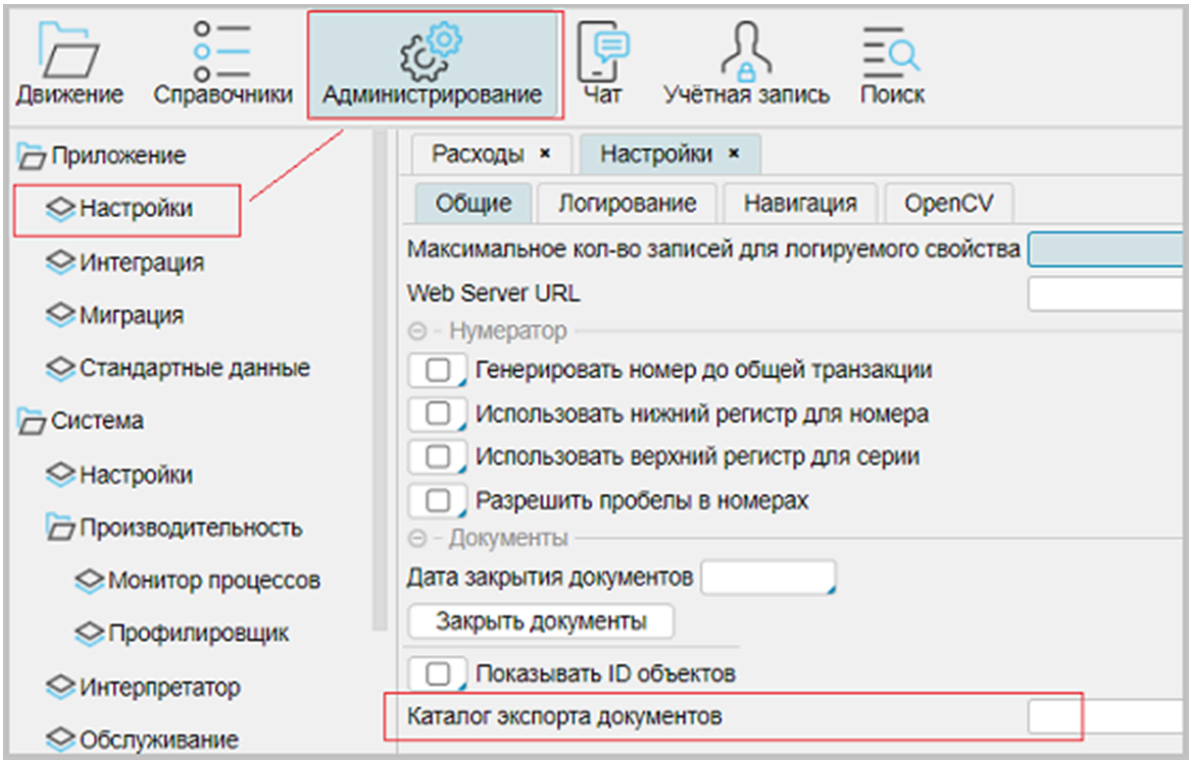
- локальные свойства для выбора диапазона дат, организации, каталога обмена;
- создадим глобальное свойство exportPath(), каталог экспорта, и свяжем его с формой Options, раздела "Администрирование - Настройки";
- форму для установки параметров экспорта: выбор дат, поставщика;
- действие по вызову формы установки параметров и обработки;
- свяжем созданное действие с формой отображения.

```
MODULE Export;

REQUIRE Expenses;

// локальные свойства
dateFrom 'Дата начала' = DATA LOCAL DATE;
dateTo   'Дата конец' = DATA LOCAL DATE;
organization 'Организация' = DATA LOCAL Organization;

dtos = FORMULA STRING[8] 'to_char(($1),\'YYYYMMDD\')';

// свойство: каталог экспорта документов, см. Администрирование - Настройки
exportPath 'Каталог экспорта документов' = DATA STRING[100];
EXTEND FORM options PROPERTIES exportPath();
DESIGN options { commons { MOVE PROPERTY (exportPath()); } }

// форма запроса параметров экспорта
FORM exportParameters 'Параметры экспорта'
   OBJECTS interval = (dateFrom = DATE, dateTo = DATE) PANEL
   PROPERTIES dateFrom 'Дата, с:' = VALUE(dateFrom), dateTo 'Дата, по:' = VALUE(dateTo)

   OBJECTS o = Organization PANEL NULL
   PROPERTIES name(o) SELECTOR FILTERS organizationType(o) = OrganizationType.supplier

   OBJECTS form = BOOLEAN PANEL NULL
   PROPERTIES form 'Экспорт с использованием формы' = VALUE(form)
;

DESIGN exportParameters {
   OBJECTS {
       BOX(interval) { caption = NULL; }
       BOX(o) { caption=NULL; }
       PROPERTY (name(o)) { caption = 'Поставщик:'; }
       BOX(form) { caption=NULL; }
   }
}

onExportDBF 'Экспорт DBF' (BOOLEAN form) { }
onExportXLS 'Экспорт XLS' (BOOLEAN form) { }
onExportXML 'Экспорт XML' (BOOLEAN form) { }

// главное действие по обработке экспорта
onExportMain 'Экспорт' (HeaderExpenses h) {
   IF NOT exportPath() THEN {
       MESSAGE 'Каталог экспорта не настроен\nСм. Администрирование - Настройки';
       RETURN;
   }
   DIALOG exportParameters OBJECTS dateFrom=date(h)
```

```
        INPUT, dateTo=date(h) INPUT, o INPUT, form INPUT DO {

        dateFrom() ← dateFrom;
        dateTo() ← dateTo;
        organization() ← o;

        CASE
            WHEN name(o) = 'УП Комбинат хлебопродуктов' THEN onExportDBF(form);
            WHEN name(o) = 'ООО Молочные продукты' THEN onExportXLS(form);
            WHEN name(o) = 'УП Мясокомбинат' THEN onExportXML(form);

        IF o THEN
            MESSAGE 'Операция выполнена';
        ELSE
            MESSAGE 'Поставщик не выбран';
    }
}

// связываем действие по обработке экспорта с формой расходов
EXTEND FORM viewExpenses PROPERTIES onExportMain(h);
DESIGN viewExpenses { TOOLBARLEFT (h) { MOVE PROPERTY (onExportMain(h)); } }
```

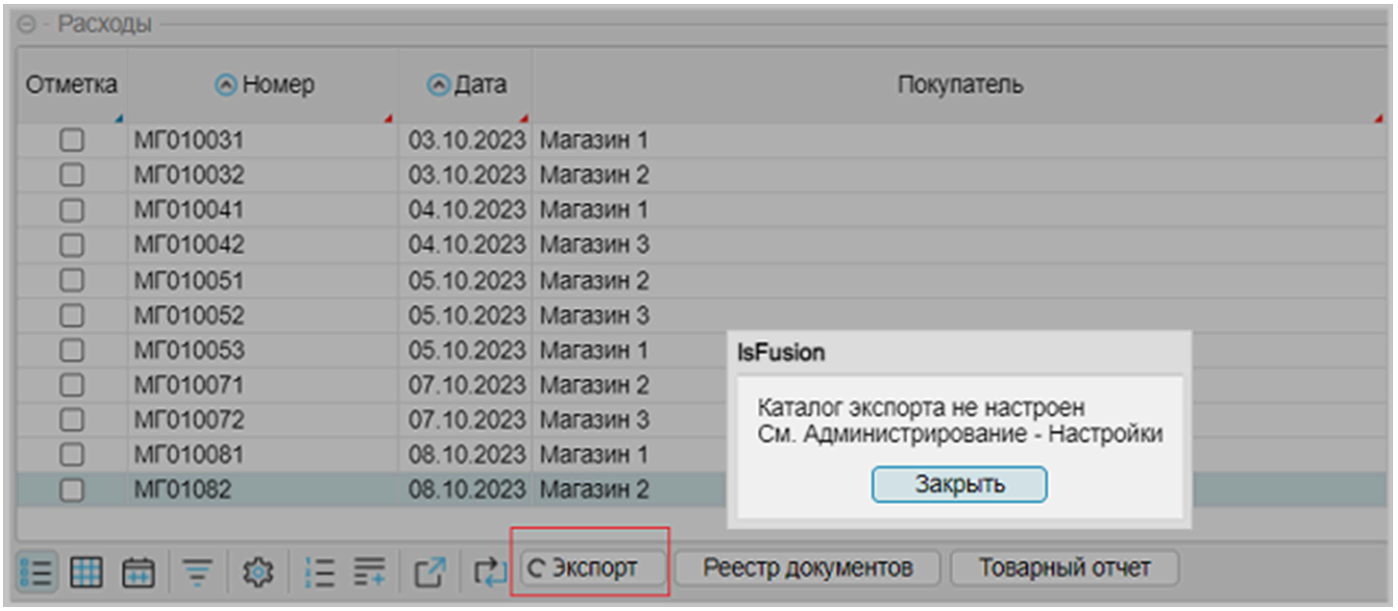
Примечание:

- PROPERTY exportPath() – свойство содержит значение каталога экспорта. Свойство добавляется на системную форму options.



- FORMULA свойство dtos, использующее оператор FORMULA, получает символьное представление даты, типа: ггггммдд.
- Действия onExportDBF, onExportXLS, onExportXML необходимы для экспорта данных по своему поставщику. Входной параметр form регулирует экспорт данных с использованием или без формы экспорта.
- onExportMain основное действие для экспорта данных, которое учитывает в структуре CASE экспорт для поставщиков.

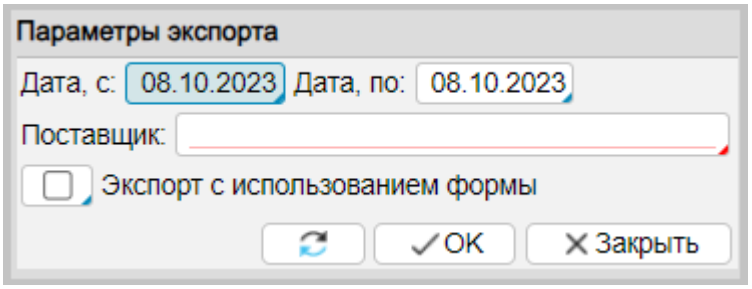
После старта сервера приложений, форма расходов получит кнопку "Экспорт", а нажатие на нее выведет сообщение о ненастроенном каталоге экспорта:



Примечание:

- для продолжения работы необходимо заполнить свойство exportPath(), например: d:/Examples/data/export/, но может быть и любое другое значение, соответствующее реальному каталогу;
- при заполнении каталога обмена стоит обратить внимание на:
  - путь к каталогу завершается слешем в конце.

Если каталог заполнен, то на экран будет выведена форма установки параметров экспорта:



См. также

Документация. Оператор FORMULA

Пример кода:

examples\t604_предварительные_действия_экспорт\src

Экспорт DBF, XLS, XML

Экспортировать данные, используя оператор EXPORT, можно несколькими способами:

Без использования формы

Экспортируемые свойства перечисляются в операторе EXPORT (*примерно также, как в операторе IMPORT*). Рассмотрим на примере экспорта в DBF формат, действие onExportDBF:

```
onExportDBF 'Экспорт DBF' (BOOLEAN form) {
   IF NOT form THEN {
       EXPORT DBF FROM date = date(header(Expenses e)),
       number = number(header(e)), shop = name(customer(header(e))),
       pname = nameProduct(e), nameUnit(e), quantity(e), price(e), cost(e)
           WHERE (date(header(e)) ⩾ dateFrom() AND date(header(e)) ⩽ dateTo()) AND
               supplier(header(reception(e))) = organization()
           ORDER date(header(e));
       WRITE exportFile() TO exportPath() + 'withoutform_xleb_' + dtos(dateTo());
   }
}
```

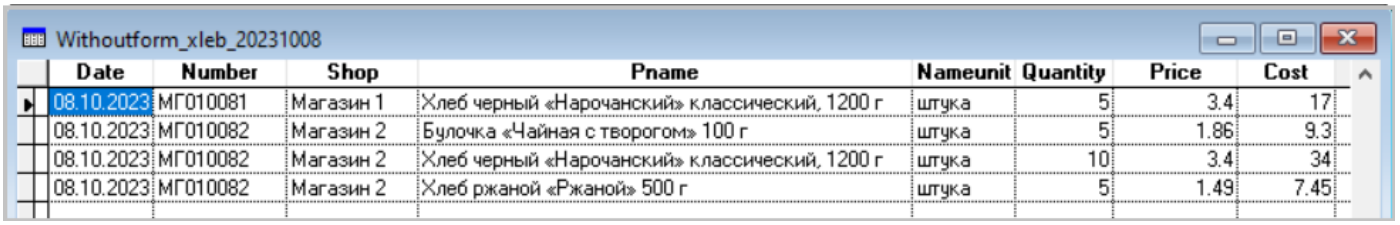
Примечание:

- Аналогично работает экспорт для формата в Excel и XML, меняется тип экспорта данных, указанный непосредственно после оператора EXPORT.
- Для экспорта в формат Excel используется опция HEADER для отражения в выходном файле первой строкой название колонок, хотя это не обязательное условие и приводится в качестве примера:

	A	B	C	D	E	F	G	H
1	date	number	shop	pname	nameUnit	quantity	price	cost
2	08.10.23	МГ010081	Магазин 1	Творог «Савушкин» рассыпчатый, 5%, 700 г 700 г	пачка	10	7.23	72.3
3	08.10.23	МГ010081	Магазин 1	Творог «Мягкий» 5 %, 125 г 130 г	банка	10	1.01	10.1
4	08.10.23	МГ010081	Магазин 1	Творог «Славянские традиции» классический, 5%, 375 г	банка	5	4.43	22.15

- Необходимо получить плоский DBF файл, включить значения свойств из шапки и строк. Часть имен полей при экспорте задано явно (date, number, shop, pname), а их значение определяется композицией, задаваемой справа от знака равно. Остальные имена полей по умолчанию определяются именами экспортируемых свойств.
- Имя поля pname для свойства nameProduct задано явно, так как имя поля по спецификации DBF не может быть больше 10 символов.
- Системное свойство exportFile() хранит содержимое экспорта, которое может быть записано в файл оператором WRITE.

После выполнения экспорта за 8.10.2023 для "УП Комбинат хлебопродуктов" структура и содержание файла (формат dbf), следующее:



Date	Number	Shop	Pname	Nameunit	Quantity	Price	Cost
08.10.2023	МГ010081	Магазин 1	Хлеб черный «Нарочанский» классический, 1200 г	штука	5	3.4	17
08.10.2023	МГ010082	Магазин 2	Булочка «Чайная с творогом» 100 г	штука	5	1.86	9.3
08.10.2023	МГ010082	Магазин 2	Хлеб черный «Нарочанский» классический, 1200 г	штука	10	3.4	34
08.10.2023	МГ010082	Магазин 2	Хлеб ржаной «Ржаной» 500 г	штука	5	1.49	7.45

С использованием формы

До определения действия onExportMain создадим форму и поместим ее в модуль Export, после инструкции DESIGN формы exportParameters, для того чтобы форма была доступна действиям по экспорту данных:

```
FORM exportData 'Форма экспорта'
   OBJECTS e = Expenses
   PROPERTIES READONLY date = date(header(e)),
       number = number(header(e)), shop = name(customer(header(e))),
       pname = nameProduct(e), nameUnit(e), quantity(e), price(e), cost(e)
   FILTERS date(header(e)) ⩾ dateFrom() AND date(header(e)) ⩽ dateTo()
   FILTERS supplier(header(reception(e))) = organization()
   ORDERS date
;
```

Примечание:

- по своей сути созданная форма exportData повторяет все, что было описано в операторе EXPORT (без использования формы): и композиции, и свойства, за исключением FILTERS вместо WHERE, и ORDERS вместо ORDER.

Доработаем также действия по экспорту данных в разные форматы (на примере onExportDBF):

```
onExportDBF 'Экспорт DBF' (BOOLEAN form) {
   IF NOT form THEN {
       EXPORT DBF FROM date = date(header(Expenses e)),
       number = number(header(e)), shop = name(customer(header(e))),
       pname = nameProduct(e), nameUnit(e), quantity(e), price(e), cost(e)
           WHERE (date(header(e)) ⩾ dateFrom() AND date(header(e)) ⩽ dateTo()) AND
               supplier(header(reception(e))) = organization()
           ORDER date(header(e));
       WRITE exportFile() TO exportPath() + 'withoutform_xleb_' + dtos(dateTo());
   } ELSE {
       EXPORT exportData DBF CHARSET 'CP1251' TO e = exportFile;
       WRITE exportFile() TO exportPath() + 'xleb_' + dtos(dateTo());
   }
}
```

Примечание:

- Добавляем условие ELSE в оператор IF.
- В операторе EXPORT добавляем название формы и тип экспорта (DBF, XLSX или XML).
- Для формата DBF можно указать кодировку с помощью опции CHARSET, например 'CP1251' - однобайтовую кириллическую кодировку для Windows. Возможный вариант 'CP866' (кодировка MS-DOS). Если опция CHARSET не указана, то при экспорте в dbf используется кодировка ‘CP1251’, а при экспорте в JSON и CSV - ‘UTF-8.

После старта сервера приложения и выполнения экспорта по трем поставщикам, получим:

- УП Комбинат хлебопродуктов, xleb_20231008.dbf

Xleb_20231008

Date	Number	Shop	Pname	Nameunit	Quantity	Price	Cost
10/08/2023	МГ010081	Магазин 1	Хлеб черный «Нарочанский» классический, 1200 г	штука	5	3.4	17
10/08/23	МГ01082	Магазин 2	Булочка «Чайная с творогом» 100 г	штука	5	1.86	9.3
10/08/23	МГ01082	Магазин 2	Хлеб черный «Нарочанский» классический, 1200 г	штука	10	3.4	34
10/08/23	МГ01082	Магазин 2	Хлеб ржаной «Ржаной» 500 г	штука	5	1.49	7.45

содержание поля Date (дата накладной) выводится в формате ММДДГГ

- ООО Молочные продукты, milk_20231008.xlsx

	A	B	C	D	E	F	G	H
1	date	number	shop	pname	nameUnit	quantity	price	cost
2	08.10.23	МГ010081	Магазин 1	Творог «Савушкин» рассыпчат	пачка	10	7.23	72.3
3	08.10.23	МГ010081	Магазин 1	Творог «Мягкий» 5 %, 125 г 130	банка	10	1.01	10.1
4	08.10.23	МГ010081	Магазин 1	Творог «Славянские традиции:	банка	5	4.43	22.15
5	08.10.23	МГ01082	Магазин 2	Творог «Савушкин» рассыпчат	пачка	5	7.23	36.15
6	08.10.23	МГ01082	Магазин 2	Сметана «Минская марка» 15%	банка	10	2.81	28.1

- УП Мясокомбинат, myso_20231008.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
- <exportData>
  - <e>
      <date>08.10.2023</date>
      <number>МГ010081</number>
      <shop>Магазин 1</shop>
      <pname>Колбаса вареная «Докторская» высший сорт, 400 г</pname>
      <nameUnit>упаковка</nameUnit>
      <quantity>5.000</quantity>
      <price>3.580</price>
      <cost>17.90</cost>
    </e>
  - <e>
      <date>08.10.2023</date>
      <number>МГ010081</number>
      <shop>Магазин 1</shop>
      <pname>Из мяса свинины продукт сырокопченый «Бекон да яечні» 220 г</pname>
      <nameUnit>упаковка</nameUnit>
      <quantity>8.000</quantity>
      <price>9.040</price>
      <cost>72.32</cost>
    </e>
</exportData>
```

Можно сравнить пары файлов с одинаковыми расширениями: начинающихся с префикса "withoutform_" и без префикса. Содержание файлов должно совпадать, что указывает на то, что оба подхода (с использованием формы и без ее использования) верны и могут одинаково использоваться при экспорте данных.

См. также

Документация. Оператор EXPORT
Документация. Оператор WRITE
Документация. Оператор FORMULA
Структура DBF файла

Пример кода:
examples\t605_экспорт_dbf_xls_xml\src

Файлы результатов экспорта:
examples\t605_экспорт_dbf_xls_xml\export

Тема 7. Внешние данные

Рассматриваемые вопросы:

- Работа с HTTP
- Работа с FTP
- Электронная почта
- Обращение к внешнему SQL

Работа с HTTP

Работа с протоколом HTTP на примере чтения курсов валют Национального Банка Республики Беларусь.

Постановка задачи:
Необходимо для расходных документов вывести значение курса USD на момент отгрузки, рассчитать сумму по накладной со скидкой в USD. Курс рассчитывается на дату относительно выписанных накладных. Должно быть предусмотрено два действия: 1. Обновить курс и 2. Очистить курс.

Предварительные действия:
Создадим отдельную папку external в папке lsfusion.

Что надо сделать:

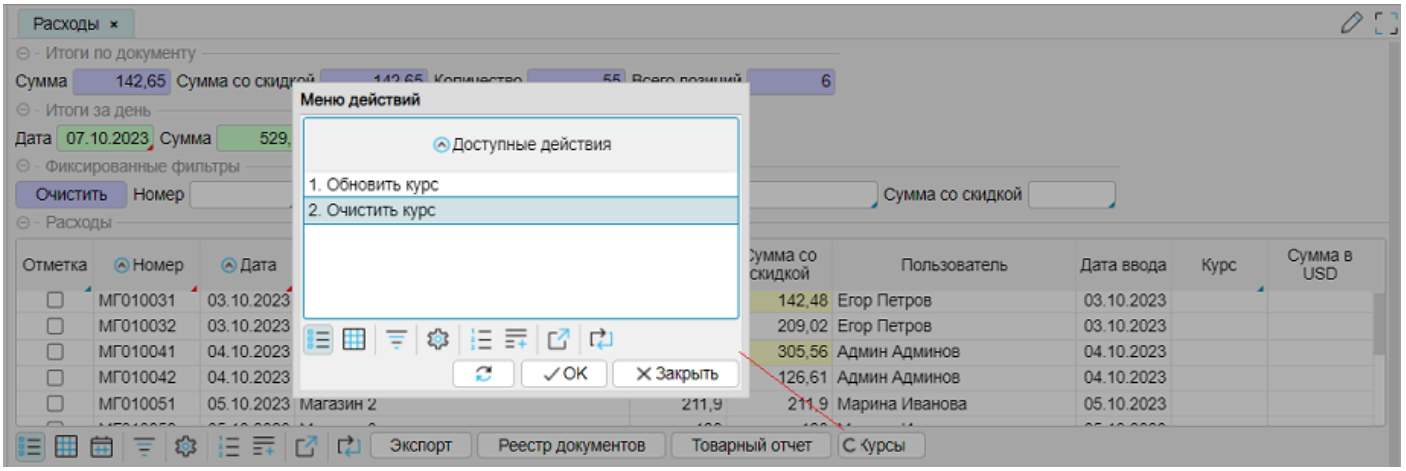
1. Создать в папке external новый модуль GetExchangeRate;
2. Добавить новые свойства rate (курс) и amountDiscountUSD (сумма со скидкой в USD);
3. Определить действия для чтения курса валют по HTTP и очистки курса;
4. Доработать форму отображения viewExpenses для отображения новых свойств.

```
MODULE GetExchangeRate;

REQUIRE Expenses, Utils;

// новые свойства
rate 'Курс' = DATA NUMERIC[10,4] (HeaderExpenses);
amountDiscountUSD 'Сумма в USD' (HeaderExpenses o) = round2(amountDiscount(o) / rate(o));

// меню для выбора действий на основе статических объектов
CLASS ActionHttp 'Выбор действий' {
   updateRate '1. Обновить курс',
   clearRate  '2. Очистить курс'
}

FORM selectAction 'Меню действий'
   OBJECTS a = ActionHttp
   PROPERTIES (a) READONLY staticCaption
   ORDERS staticCaption(a)
;

DESIGN selectAction {
   BOX(a) {caption=NULL;}
   PROPERTY (staticCaption(a)) { caption = 'Доступные действия'; }
}

// обработка действий для работы с курсами
onAction 'Курсы' (HeaderExpenses o) {
   DIALOG selectAction OBJECTS a INPUT DO {}
}

EXTEND FORM viewExpenses PROPERTIES (h) rate, amountDiscountUSD, onAction;
DESIGN viewExpenses {
   TOOLBARLEFT (h) { MOVE PROPERTY (onAction(h)) LAST; }
}
```

Примечание:

- Создаем новые свойства. Для свойства rate определим тип NUMERIC с дробной частью 4 знака (*курсы валют предоставляются банком с точностью до 4-х знаков после запятой*). Свойство amountDiscountUSD расчетное, результат его вычисления округляем до 2-х знаков, используя действия round2 модуля Utils (*вызывается в модуле Expenses*).
- Чтобы "не плодить" лишние кнопки, используя класс со статическими объектами, создадим меню действий для выбора операций по обновлению или очистки значения курса USD, которое будет вызываться из кнопки на форме.
- Действие onAction (кнопка на форме) будет вызывать меню действий. Входной параметр - ссылка на текущий объект шапки накладной, для получения даты. Относительно даты накладной будет выполняться запрос по HTTP для получения курса или очистка курса на дату выписки накладных.

После запуска сервера приложений форма расходов примет вид:



То есть в общем виде все работает. Осталось определиться с наполнением действия onAction. Так как было создано меню для выбора действий, то необходимо использовать оператор DIALOG для формы selectAction, внутри которого будет определяться обработка. На основании Описания API для работы с курсами валют и приводимых примеров URL строки запроса к Национальному банку РБ будет иметь вид:

https://api.nbrb.by/exrates/rates?ondate= + ДАТА + &periodicity=0'

где "ДАТА" - это дата курса, совпадающая с датой расходной накладной, в формате ГГГГ-ММ-ДД. Для преобразования даты в нужный вид потребуется соответствующее действия из модуля Time, этот модуль необходимо подключить инструкцией REQUIRE:

```
REQUIRE Expenses, Time;
```

При выполнении запроса к Национальному банку РБ ответ будет представлен в формате JSON как массив объектов валют. Для того, чтобы правильно обработать принятые данные, необходимо выполнить импорт принятого JSON, по аналогии, как это делалось со структурой XML при рассмотрении вопроса "Импорт из XML". Поэтому сам процесс получения курса валют должен быть разбит на две части:

1. создание формы для импорта,
2. собственно обработка импорта и выделение значения курса USD.

В целом, уже можно определить действия по чтению HTTP и по очистке значения курса, в обработке оператора DIALOG формы selectAction:

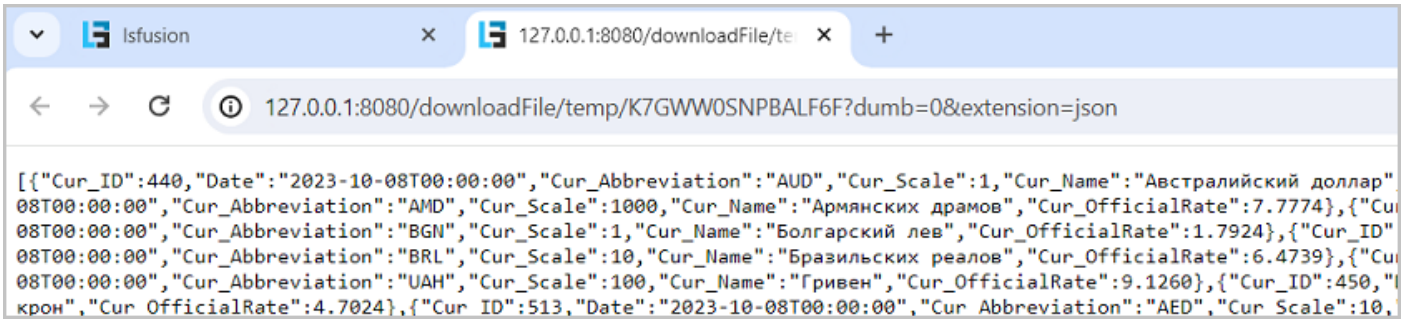
```
// обработка действий для работы с курсами
onAction 'Курсы' (HeaderExpenses o) {
   DIALOG selectAction OBJECTS a INPUT DO {
       IF ActionHttp.updateRate = a THEN {
           LOCAL url = STRING ();
           LOCAL result = FILE ();
           url() ← 'https://api.nbrb.by/exrates/rates?ondate=' +
                                                   toDateISO(date(o)) + '&periodicity=0';
           TRY {
               EXTERNAL HTTP GET url() TO result;
           } CATCH {
               MESSAGE 'Ошибка выполнения HTTP запроса\n' +
                                   messageCaughtException() + '\nURL: ' + url();
               RETURN;
           }
           open(result()); // только для создания формы экспорта
       } ELSE {
           ASK 'Вы уверены, что хотите\nочистить курс USD?' DO {
             rate(HeaderExpenses h) ← NULL WHERE date(h) = date(o);
             APPLY;
           }
       }
   }
}
```

Примечание:

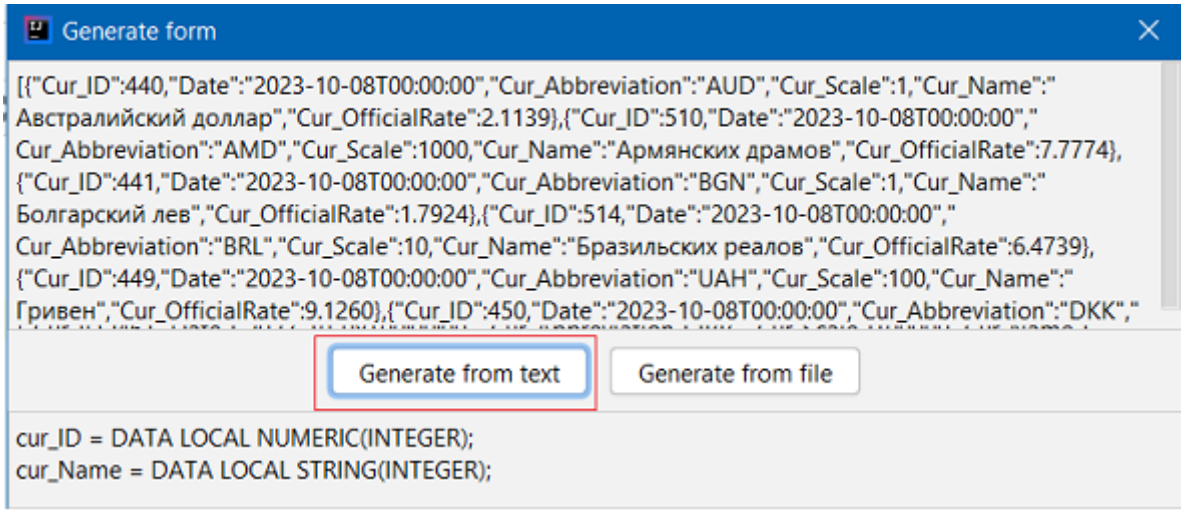
- Код достаточно прост: IF - получение курса на дату и ELSE - очистка курса на дату
- Чтение курса обернуто в конструкцию оператора TRY … CATCH. Зачем нужна такая конструкция? Иногда при выполнении программ могут возникать ошибки и непредвиденные ситуации, которые называются исключениями. В программе исключения могут возникать в результате, например, неправильных действий пользователя, отсутствия необходимого ресурса, или потери соединения с сервером и т.д. Оператор TRY позволяет выполнить действие, и, если в этом действии возникло исключение, выполнить действие, указанное в блоке CATCH.

- EXTERNAL HTTP GET url() TO result - собственно сам HTTP запрос, где:
  - EXTERNAL - оператор обращения к внешней системе
  - HTTP - ключевое слово, определяющее, что оператор выполняет http-запрос
  - GET - метод выполнения http-запроса (GET, POST, PUT, DELETE)
  - далее указан URL, по которому выполняется http-запрос
  - результат http-запроса направляется в свойство типа FILE.

Следующие системное действие open(result()) откроет результат чтения курса валют в окне браузера:



Далее необходимо создать локальные свойства и определить форму импорта. Для этого, как это уже делалось при рассмотрении вопроса "Импорт из XML", в редакторе IDEA вызвать пункты меню "LSF - Generate FORM - From json", скопировать из браузера текст JSON, перенести в верхнюю часть окна "Generate form" и нажать кнопку "Generate from text" на форме.



Следующие действия:

- Скопировать из нижней части окна определения локальных свойств и формы с именем generated и разместить чуть выше действия onAction. Из всех созданных свойств используем только cur_Abbreviation (аббревиатура валюты) и cur_OfficialRate (значение курса), которые действительно нужны в обработке (*хотя можно оставить, как есть*)

```
cur_ID = DATA LOCAL NUMERIC(INTEGER);
cur_Name = DATA LOCAL STRING(INTEGER);
cur_OfficialRate = DATA LOCAL NUMERIC[10,4](INTEGER); // исправить размер
cur_Abbreviation = DATA LOCAL STRING(INTEGER);
date = DATA LOCAL DATETIME(INTEGER);
cur_Scale = DATA LOCAL NUMERIC(INTEGER);

FORM generated
   OBJECTS value = INTEGER
   PROPERTIES(value) cur_ID EXTID 'Cur_ID', cur_Name EXTID 'Cur_Name',
       cur_OfficialRate EXTID 'Cur_OfficialRate', cur_Abbreviation EXTID
'Cur_Abbreviation',
       date EXTID 'Date', cur_Scale EXTID 'Cur_Scale'
   FILTERS imported(value);
```

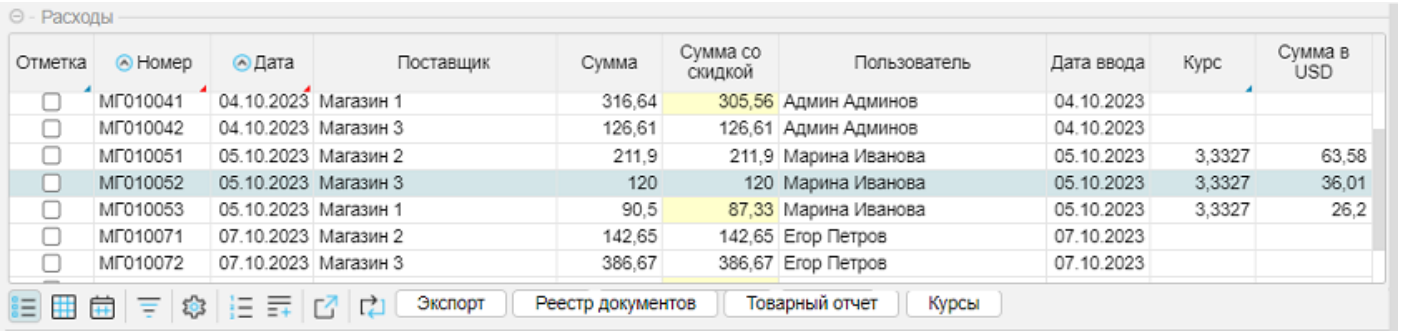
  для свойства cur_OfficialRate необходимо исправить размер на NUMERIC[10,4].
- Закомментировать (удалить) действие open(result()) - в его использовании нет больше необходимости.
- За действием open разместить импорт из формы и обработку шапок расхода:

```
IMPORT generated JSON FROM result();
rate(HeaderExpenses h) ←
   NUMERIC[10,4]([GROUP MAX cur_OfficialRate(INTEGER i) BY cur_Abbreviation(i)]('USD'))
WHERE date(h) = date(o);

APPLY;
```

  Примечание:

- Оператор IMPORT импортирует из результата http запроса информацию в массив локальных свойств.
- В конце вызываем оператор APPLY, чтобы применить сделанные изменения.

После запуска сервера приложений, получения курса на дату, форма расходов примет вид:



См. также

Описание API для работы с курсами валют
Коды ответов от WEB сервера
Методы HTTP
Документация. Оператор EXTERNAL
Документация. Оператор TRY

Пример кода:
examples\t701_работа_http\src

Архив SQL:
examples\t701_работа_http\sql

Работа с FTP

Постановка задачи:
От поставщиков товаров поступило предложение использовать протокол FTP для приема электронных накладных и выгрузки статистики по отгрузке их товаров покупателям за период, так как использование "флэшек" неудобно (забываются, теряются, ломаются).

Поставленная задача предполагает, что будут две операции:

1. Операция экспорт статистики и записи на FTP.
2. Операция импорта накладных с FTP (операция чтения с FTP)

В силу непонятных пока по содержанию задач конечное решение видится сложным и громоздким. Но, на самом деле, используя возможности платформы, все решается достаточно просто.

Внимание: для примера установки соединения используются вымышленные данные (логин - client, пароль - 12345, адрес ftp-сервера - myftp.by). Для реальной работы необходимо использовать действительные параметры подключения.

1. Экспорт статистики и запись на FTP

Т.к. в платформе lsFusion работа с ftp – это частный случай работы с файлами, то никакого дополнительного кода для экспорта на ftp-сервер писать не придется. Достаточно в настройке "Каталог экспорта документов" (*добавлена ранее*), заменить путь для экспорта на строку подключения к ftp вида: ftp://логин:пароль@адрес_ftp_сервера/папка_на_ftp/

Например:

Каталог экспорта документов	ftp://client:12345@myftp.by/education/export/

2. Импорт накладных

Можно достаточно просто организовать импорт накладных с FTP сервера. Фактически, основная часть импорта накладных написана (см. вопросы "Импорт из DBF и XLS", "Импорт из XML"). Рассматриваемый модуль импорта состоит из двух частей. 1-я часть (действие import без параметров) читает импортируемый файл с диска, связывает его по расширению с организацией и вызывает основное действие по импорту данных с параметрами: считанный файл и организация.

Поэтому задача по импорту с FTP решается достаточно просто. Необходимо прочитать файл с FTP, выделить по расширению файла организацию и передать это в качестве параметров в действие import. Чтобы не писать в коде модуля и сделать приложение настраиваемым, дополнительно создать свойство для пункта меню "Администрирование - Настройки", которое будет хранить настройки подключения к FTP вида:
ftp://логин:пароль@адрес_ftp_сервера/папка_на_ftp/ .

Например:

Каталог импорта документов	ftp://client:12345@myftp.by/education/import/

Создадим в папке external модуль ImportFtp:

```
MODULE ImportFtp;

REQUIRE Import;

// свойство: каталог импорта документов с FTP, см. Администрирования - Настройки
importPath 'Каталог импорта документов' = DATA STRING[100];

EXTEND FORM options PROPERTIES importPath();

DESIGN options {
   commons {
       MOVE PROPERTY (importPath());
   }
}

onImportFTP 'Прием ТН с FTP' () {

   IF NOT importPath() THEN {
       MESSAGE 'Каталог импорта документов не назначен\nВыполнение прервано';
       RETURN;
   }

   LOCAL dbfile = FILE();
   TRY {

       listFiles(importPath()); // получаем список для импорта
       FOR fileName(INTEGER i) AND NOT fileIsDirectory(i) DO {
           READ importPath() + fileName(i) TO dbfile; // читаем с FTP
           CASE
               WHEN upper(extension(dbfile())) = 'DBF'  THEN
                   import(dbfile(), Organization.unique('УП КОМБИНАТ ХЛЕБОПРОДУКТОВ'));
               WHEN upper(extension(dbfile())) = 'XLSX' THEN
                   import(dbfile(), Organization.unique('ООО МОЛОЧНЫЕ ПРОДУКТЫ'));
               WHEN upper(extension(dbfile())) = 'XML'  THEN
                   import(dbfile(), Organization.unique('УП МЯСОКОМБИНАТ'));
               ELSE
                   MESSAGE 'Не ожидаемое расширение в имени файла\nВыполнение прервано!';

           IF NOT canceled() THEN {
               SEEK LAST viewReception.h; // указатель на последнюю запись

           }
       }

   } CATCH {
       MESSAGE 'Ошибка чтения файлов с FTP\n' + messageCaughtException();
   }
}

// кнопка на форме
EXTEND FORM viewReception PROPERTIES onImportFTP();

DESIGN viewReception {
   TOOLBARLEFT (h) {
       MOVE PROPERTY (onImportFTP());
   }
```

```
}
```

Примечание:

- Системное действие listFiles выполняет чтение содержимого каталога на диске или FTP.
- После выполнения действия listFiles системное свойство fileName будет содержать имена считанных папок и файлов.
- Системное свойство fileIsDirectory возвращает TRUE, если считанный файл имеет атрибут папки. Поэтому в цикле FOR по условию NOT fileIsDirectory(i) обрабатываются только файлы.

См. также

Документация. Оператор READ

Пример кода:

examples\t702_работа_ftp\src

Файлы накладных для импорта с FTP:

examples\t702_работа_ftp\import_ftp

Электронная почта

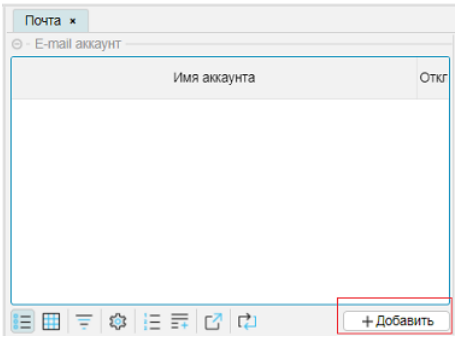
Постановка задачи:
Поставщик "УП Комбинат хлебопродуктов" попросил информацию по статистике, кроме отправки на FTP, отправлять также на почтовый адрес.

Что надо сделать:
1. Настроить почту
2. Создать действие для отправки сообщения

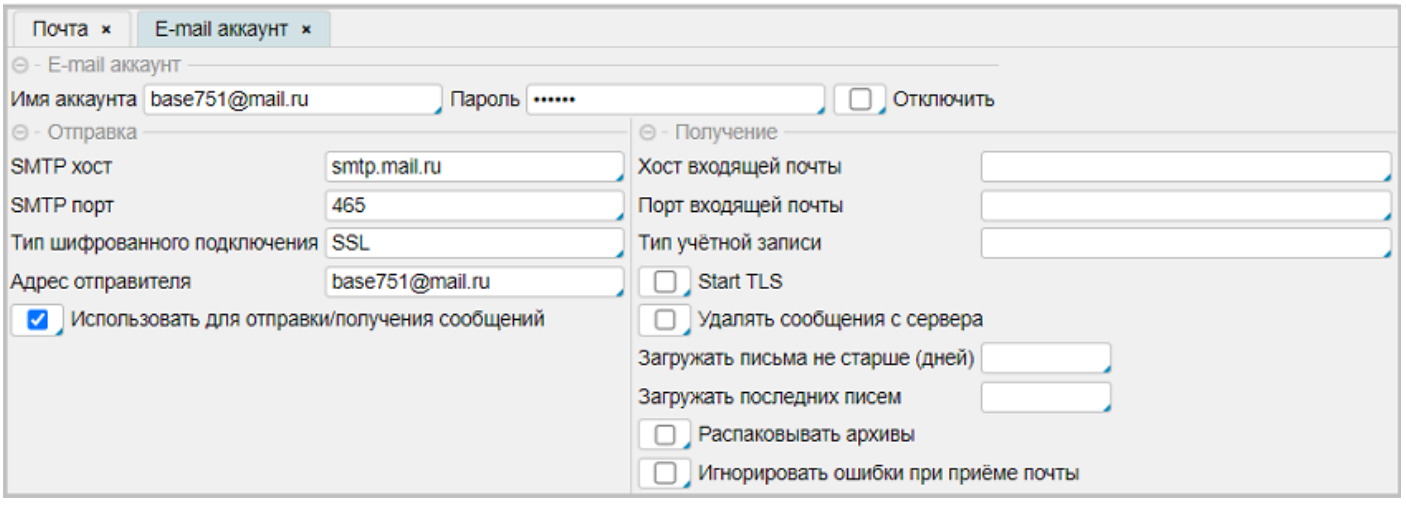
Внимание: для примера используются вымышленные почтовые данные для отправителя (base751@mail.ru) и получателя (office751@mail.ru). Для реальной работы необходимо использовать действительные параметры подключения.

1. Настроить почту

Для того, чтобы иметь возможность отправить почту, необходимо выполнить ряд настроек на уровне приложения. Выполним настройки, исходя из предположения, что почтовый ящик отправителя находится на mail.ru. Для выполнения этой задачи, необходимо перейти в меню "Администрирование - Система - Уведомления - Почта" и создать новый аккаунт.



Для создания нового аккаунта необходимо нажать на кнопку "Добавить". При этом на экране отобразится форма редактирования, в которой необходимо заполнить соответствующие поля, как показано на рисунке:



2. Действие для отправки сообщения

В папке external создадим новый модуль SendMail:

```
MODULE SendMail;

REQUIRE Export;

onSendMail 'Отправка e-mail' () {
   LOCAL from, to, subject, body = STRING ();
   from()      ← 'base751@mail.ru'; // почтовый адрес от кого
   to()        ← 'office751@mail.ru'; // почтовый адрес кому
   subject()   ← 'Статистика по движению';
   body()      ← 'Здравствуйте\n\nВысылаю в ваш адрес информацию о ' +
       'движении товаров, см. вложение\n\nЗав. складом';
   TRY {
       EMAIL FROM from() SUBJECT subject() TO to() BODY body() ATTACH exportFile();
       MESSAGE 'Отправлено по почте, Ok';
   } CATCH {
       MESSAGE 'Ошибка отправки почтового сообщения\n' +  messageCaughtException();
   }
}

AFTER onExportDBF(BOOLEAN form) DO onSendMail();
```

Примечание:

- Для получения экспортируемых данных используем существующее решение Экспорт DBF, XLS, XML. По окончанию действия onExportDBF (экспорт статистики в формате DBF) используя инструкцию AFTER, вызываем действие onSendMail для передачи почтового сообщения на почтовый ящик для "УП Комбинат хлебопродуктов".

См. также

Документация. Оператор EMAIL

Пример кода:

examples\t703_электронная_почта\src

Обращение к внешнему SQL

В процессе работы может возникнуть необходимость обращения к базе данных другого SQL сервера, например для импорта или экспорта каких-либо данных.

Для примера рассмотрим небольшую работу с базой данных mdata, которая будет располагаться на локальном сервере PostgreSQL.

Постановка задачи:

- Необходимо создать на сервере PostgreSQL базу данных mdata и таблицу goods. Таблица goods предназначена для хранения информации о текущих остатках.
- Должны быть предусмотрены механизмы контроля наличия базы данных mdata, таблицы goods и ее заполнения.
- Информация, хранимая в goods должна виде формы выводится на экран.
- Информация для таблицы goods берется из текущих остатков (модуль balance) по отдельной операции.
- В разделе навигатора "Движение" должен быть предусмотрен отдельный пункт "Внешний SQL" с входящими в него действиями: создание БД, заполнение данными, импорт из внешнего SQL, удаление БД.

Создадим в папке external модуль ExternalSQL:

```
MODULE ExternalSQL;


REQUIRE AppMenu, Balance, Time;


connection 'Строка подключения' = DATA LOCAL STRING();
emessage 'Сообщение об ошибке' = DATA LOCAL STRING();
return 'Возвращаемое значение' = DATA LOCAL FILE();
mdata 'Флаг существования БД mdata' = DATA LOCAL BOOLEAN ();


onExecSQL 'Выполнение запроса к SQL' (STRING dbname, text, BOOLEAN ret) {
   connection() ← 'jdbc:postgresql://127.0.0.1:5432/' + dbname +
                  '?user=postgres&password=11111';
   emessage() ← NULL;
   TRY {
       IF ret THEN
           EXTERNAL SQL connection() EXEC text TO return;
       ELSE
           EXTERNAL SQL connection() EXEC text;
   } CATCH {
       emessage() ← 'Ошибка выполнения SQL запроса\n' +
                    text + '\n' + messageCaughtException();
       MESSAGE emessage();
   }
}


chkDatabase 'Проверка существования БД' () {
   LOCAL dbname = STRING(INTEGER);

   onExecSQL('postgres', 'SELECT datname FROM pg_database WHERE datname = \'mdata\';', TRUE);
   IF emessage() THEN RETURN;
   IMPORT TABLE FROM return() TO dbname;
```

```
   mdata() ← dbname(0) = 'mdata';

}


onDeleteDatabase 'Удаление БД' () {
   chkDatabase();
   IF emessage() THEN RETURN;
   IF mdata() THEN {
       onExecSQL('postgres', 'DROP DATABASE mdata;', NULL);
       IF NOT emessage() THEN MESSAGE 'БД удалена!';
   } ELSE MESSAGE 'БД не существует';
}


onCreateDatabase 'Создание БД' () {
   chkDatabase();
   IF emessage() THEN RETURN;
   IF NOT mdata() THEN {
       onExecSQL('postgres', 'CREATE DATABASE mdata;', NULL);
       IF emessage() THEN RETURN;
       onExecSQL('mdata', 'CREATE TABLE goods (' +
           'key SERIAL PRIMARY KEY,date DATE, pname VARCHAR(100),' +
           'uname VARCHAR(10), gname VARCHAR(100), price NUMERIC(10,2), ' +
           'quantity NUMERIC(10,3))', NULL);
       IF NOT emessage() THEN MESSAGE 'БД создана';
   } ELSE MESSAGE 'БД уже существует';
}


onFillDatabase 'Заполнение данными' () {
   chkDatabase();
   IF emessage() THEN RETURN;
   IF NOT mdata() THEN {
       MESSAGE 'БД не существует';
       RETURN;
   }
   // проверка заполненности таблицы
   LOCAL date = DATE(INTEGER);
   onExecSQL('mdata', 'SELECT date FROM goods LIMIT 1;', TRUE);
   IF emessage() THEN RETURN;
   IMPORT TABLE FROM return() TO date;
   IF date(0) THEN {
       MESSAGE 'Таблица была заполнена ранее';
       RETURN;
   }
   // заполнение данными
   LOCAL text = TEXT();
   text() ← GROUP CONCAT
       'INSERT INTO goods (date,pname,uname,gname,price,quantity) VALUES (' +
       '\'' + toDateISO(date(Reception r)) + '\',' +
       '\'' + nameProduct(r) + '\',' +
       '\'' + nameUnit(r) + '\',' +
       '\'' + nameGroup(r) + '\',' +
       toChar(price(r),'9999999.99') + ',' +
       toChar(balance(r),'9999999.99') + ')', ';' ORDER r;

   onExecSQL('mdata', text(), NULL);
   IF emessage() THEN RETURN;
   MESSAGE 'Операция выполнена';
}
```

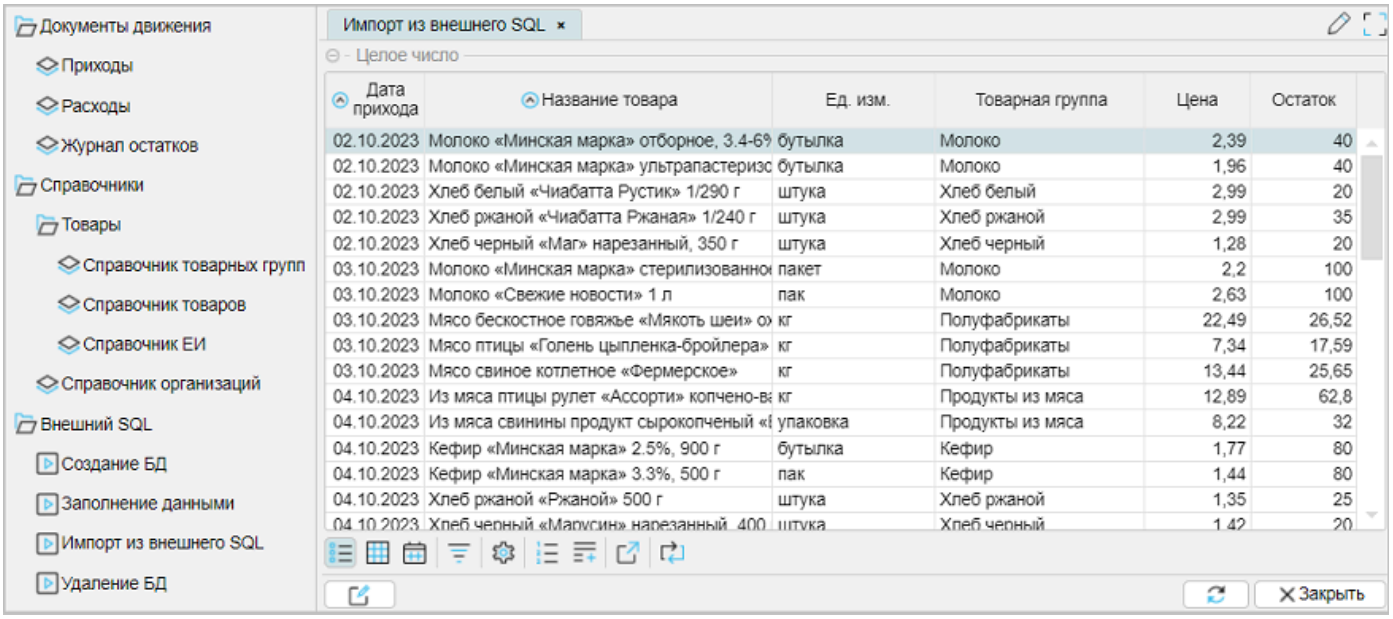
```
date = DATA LOCAL DATE (INTEGER);
pname = DATA LOCAL STRING[100] (INTEGER);
uname = DATA LOCAL STRING[100] (INTEGER);
gname = DATA LOCAL STRING[100] (INTEGER);
price = DATA LOCAL NUMERIC[10,2](INTEGER);
quantity = DATA LOCAL NUMERIC[10,3](INTEGER);


FORM mdata 'Импорт из внешнего SQL'
   OBJECTS i = INTEGER
   PROPERTIES (i) READONLY 'Дата\nприхода' = date, 'Название товара' = pname,
            'Ед. изм.' = uname, 'Товарная\nгруппа' = gname,
            'Цена' = price, 'Остаток' = quantity
   FILTERS imported(i)
   ORDERS date(i), pname(i)
;


onReadDatabase 'Чтение данных' () {
   onExecSQL('mdata', 'SELECT date,pname,uname,gname,price,quantity FROM goods;', TRUE);
   IF emessage() THEN RETURN;
   IMPORT TABLE FROM return() TO date, pname, uname, gname, price, quantity;
   SHOW mdata DOCKED NOWAIT;
}


NAVIGATOR {
   move {
       NEW FOLDER sql 'Внешний SQL' {
           NEW ACTION onCreateDatabase;
           NEW ACTION onFillDatabase;
           NEW ACTION onReadDatabase;
           NEW ACTION onDeleteDatabase;
       }
   }
}
```

Примечание:

- Действие onExecSQL общее для выполнения запросов к SQL серверу, принимает 3 входных параметра:
    1. Имя базы данных, от имени которой выполняется операция. При создании или удалении базы данных mdata имя передаваемой базы данных "postgres". При работе с базой данной mdata имя передаваемой базы данных "mdata".
    2. Текст SQL запроса
    3. Признак возвращать результат или нет. Значение TRUE устанавливается, если результат запроса должен возвращать результат.

  Внутри действия определено свойство connection с параметрами подключения к SQL, соответствующими параметрам при развертывании платформы.
- При заполнении таблицы goods остатками товаров используется экранирование значений, которые передаются как символьные (в одинарных кавычках).
- Скрипты на создание, удаление, проверки наличия базы, заполнение и чтения из нее, передаваемые в действие onExecSQL (2-й параметр) составлены в соответствии с синтаксисом языка SQL с учетом особенностей сервера PostgreSql. Более подробная информация может быть получена из документации на sql сервер, например: "PostgreSQL: Документация".

После старта сервера приложений, выполнения действий по созданию БД и заполнению данными, внешний вид формы "Импорт из внешнего SQL" примет вид:



См. также
Оператор EXTERNAL

Пример кода:
examples\t704_внешний_sql\src

Стандартные свойства и действия

На платформе lsFusion присутствует некоторое количество стандартных свойств и действий разного назначения, разбитых по области применения, по разным системным модулям платформы. Для подключения модулей используется инструкция REQUIRE, например:

```
1   MODULE Main;
2
3   REQUIRE Reception, AppMenu, Utils;
```

Для того, чтобы просмотреть код модуля, необходимо нажать в строке REQUIRE на соответствующем модуле комбинацию клавиш CTRL+B:

Примечание:

- Модуль System подключать отдельно необязательно – он всегда подключен по умолчанию.

Модуль Utils

Модуль Utils содержит действия и свойства общего назначения, в том числе для работы с файловой системой, числами и строками.

Наиболее употребительные действия и свойства:

Функция	Тип	Описание функции
СТРОКОВЫЕ		
trim	STRING	Удаляет начальные и конечные пробелы trim(' 12345 ') -> '12345'
left	STRING	Возвращает левую часть строки left('12345', 3) -> '123'
right	STRING	Возвращает правую часть строки right('12345', 3) -> '345'
lpad	STRING	Дополняет строку символом(и) слева на ширину области, определяемую 2-м параметром lpad('12345', 10, '.') -> '.....12345'
rpad	STRING	Дополняет строку символом(и) справа на ширину области, определяемую 2-м параметром rpad('12345', 10, '.') -> '12345.....'
length	INTEGER	Возвращает длину строки length('12345') -> 5
strpos	INTEGER	Возвращает позицию вхождения подстроки (2-й параметр) в строку: strpos('12345', '23') -> 2
replace	STRING	Заменяет в строке одно выражение другим replace('Привет Вася', 'Вася', 'Лена') -> 'Привет Лена'
substr	STRING	Возвращает из строки, начиная с позиции (2-й параметр) количество символов (3-й параметр) substr('123456789', 7, 2) -> '78'
isSubstring	STRING	Регистрозависимый поиск в строке подстроки isSubstring('Привет Вася', 'Вася') -> TRUE isSubstring('Привет Вася', 'вася') -> NULL isSubstring('Привет Вася', NULL) -> NULL
isISubstring	STRING	Регистронезависимый поиск в строке подстроки isISubstring('Привет Вася', 'Вася') -> TRUE isISubstring('Привет Вася', 'вася') -> TRUE
wordCount	INTEGER	Возвращает количество "слов" в строке, отделенных друг от друга разделителем (2-й параметр) wordCount('Сыр,12,2.56', ',') -> 3
getWord	STRING	Возвращает "слово" по номеру (3-й параметр) getWord('Сыр,12,2.56', ',', 1) -> 'Сыр' getWord('Сыр,12,2.56', ',', 3) -> '2.56' getWord('Сыр,12,2.56', ',', 8) -> NULL
МАТЕМАТИЧЕСКИЕ		
mod	INTEGER	Возвращает остаток от деления mod(10, 7) -> 3

Функция	Тип	Описание функции
toChar	STRING	Символьное представление числа с форматированием toChar(945670.29, '999 999.99') -> '945 670.29'
onlyDigits	BOOLEAN	Возвращает TRUE, если строка состоит только из цифр onlyDigits('945670') -> TRUE onlyDigits('945670.29') -> NULL
roundM1	NUMERIC	Округление до 10 (рублей) roundM1(124.445) -> 120.000 roundM1(126) -> 130
round0..round6	NUMERIC	Округление до 0..6 знаков после запятой round2(124.445) -> 124.450
floor	NUMERIC	Возвращает целую часть числа floor(10.4) -> 10.0 floor(10.8) -> 10.0
ceil	NUMERIC	Возвращает ближайшую большую целую часть числа ceil(10.4) -> 11.0 ceil(10.8) -> 11.0
РАБОТА С ФАЙЛАМИ		
delete	-	Удаление файла с диска delete(имя_файла) Перед выполнением необходимо проверить наличие файла и тогда удалять, иначе возникнет ошибка
fileExists		Проверяет наличие файла на диске Используется в паре: 1. выполняется действие проверки наличия файла 2. проверяется свойство - результат проверки Например: fileExists('currency.json'); IF fileExists() THEN delete('currency.json');

Модуль Time

Модуль Time отвечает за работу с датой и временем.

Наиболее употребительные свойства и действия:

Функция	Тип	Описание функции
currentDateTime()	DATETIME	Возвращает текущие дату и время
currentDate()	DATE	Возвращает текущую дату
currentTime()	TIME	Возвращает текущее время
toDateISO	STRING[10]	Возвращает символьное представление даты как гггг-мм-дд toDateISO(currentDate()) -> '2022-12-27'
toDateDDMMYY	STRING[8]	Возвращает символьное представление даты как дд.мм.гг toDateDDMMYY(CurrentDate()) -> '27.12.22'
toDateDDMMYYYY	STRING[10]	Возвращает символьное представление даты как дд.мм.гггг toDateDDMMYY(CurrentDate()) -> '27.12.2022'
firstDayOfMonth	DATE	Возвращает первый день месяца даты firstDayOfMonth(2022_02_14) -> 01.02.2022
lastDayOfMonth	DATE	Возвращает последний день месяца даты lastDayOfMonth(2022_02_14) -> 28.02.2022
subtract	DATE	Отнимает от даты количество дней (2-й параметр) subtract(2022_02_14, 10) -> 04.02.2022

Отдельные описания и примеры

В этом разделе приводятся отдельные описания и примеры использования операторов, которые не нашли место в основном описании.

Рассматриваемые вопросы:

- Операторы FOR и WHILE
- Оператор DIALOG

Операторы FOR и WHILE

Цикл по объектам класса:

```
onClick 'Цикл' () {
    FOR (Reception r IS Reception) DO {
        MESSAGE nameProduct(r);
    }
}
```

Цикл по объектам с условием:

```
onClick 'Цикл по условию' () {
    FOR price(Reception r) > 3.00 DO {
        MESSAGE nameProduct(r) + ', ' + toChar(price(r),'9999.99');
    }
}
```

Цикл: от значения 1 до значения 2, тип INTEGER:

```
onClick 'Цикл INTEGER' () {
    FOR iterate(INTEGER i, 1073741824-5, 1073741824) DO {
        MESSAGE toChar(i, '9999999999999');
    }
}
```

Цикл: от значения 1 до значения 2, тип DATE:

```
onClick 'Цикл DATE' () {
    FOR iterate(DATE dd, 2023_01_01, 2023_01_05) DO {
        MESSAGE dd;
    }
}
```

Цикл WHILE:

```
onClick 'Цикл INTEGER' () {
    LOCAL i = INTEGER ();
    i() ← 0;
    WHILE i() < 10 DO {
        i() ← i() + 1;
        IF i() > 5 THEN {
            MESSAGE 'i > 5 - выход';
            BREAK;
        }
        MESSAGE i();
    }
}
```

Оператор DIALOG

Форма редактирования документа, блок метакода модуля Documents.

Если выбор был сделан, свойство organization(h) получит ссылку на выбранный объект справочника организаций (o). Если выбор не был сделан, значение свойства organization(h) не изменится

Вызов формы выбора без условий:

```
PROPERTIES (h) organizationName ON CHANGE {
    DIALOG listOrganization OBJECTS o INPUT DO organization(h) ← o;
}
```

Вызов формы выбора с установкой указателя записи на ранее выбранную запись:

```
PROPERTIES (h) organizationName ON CHANGE {
    DIALOG listOrganization OBJECTS o = organization(h) INPUT DO organization(h) ← o;
}
```

Примечание:

- если ранее выбранной записи нет, указатель будет вверху формы (на первой записи)

Вызов формы выбора с условием фильтра:

```
DIALOG listOrganization OBJECTS o INPUT FILTERS left(name(o),2) = 'УП' DO organization(h) ← o;
```

Примечание:

- форма выбора отобразит все организации, начинающиеся с символов "УП"

Администрирование

Рассматриваемые вопросы:

- Создание архива базы данных
- Восстановление базы данных

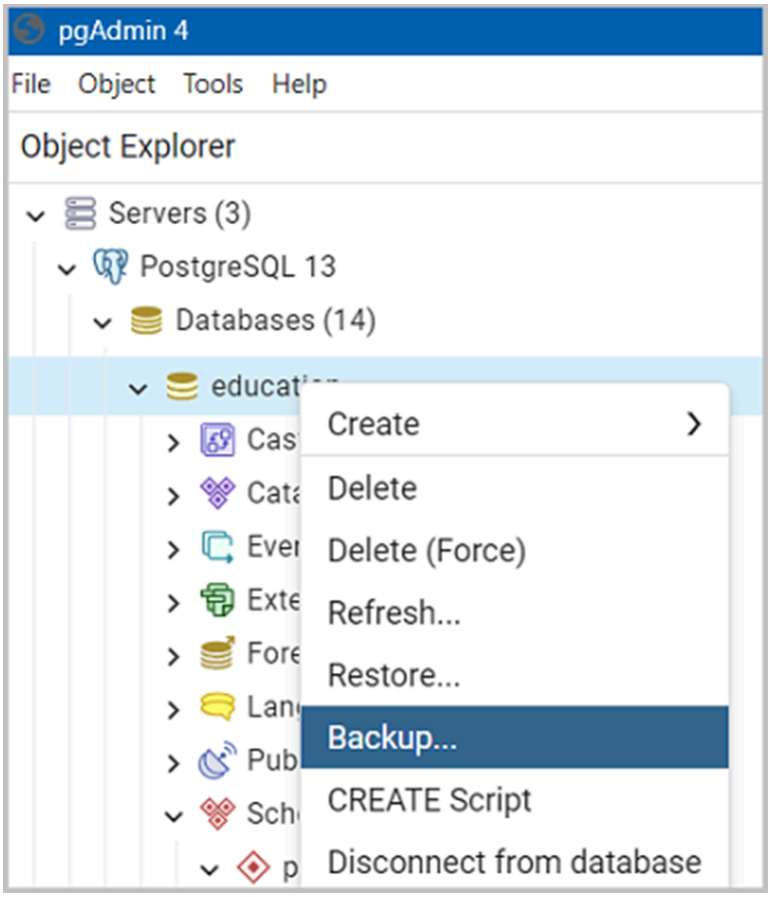
Работа с архивами БД

Рассматриваются вопросы создания архива БД и восстановления из архива средствами SQL и средствами платформы.

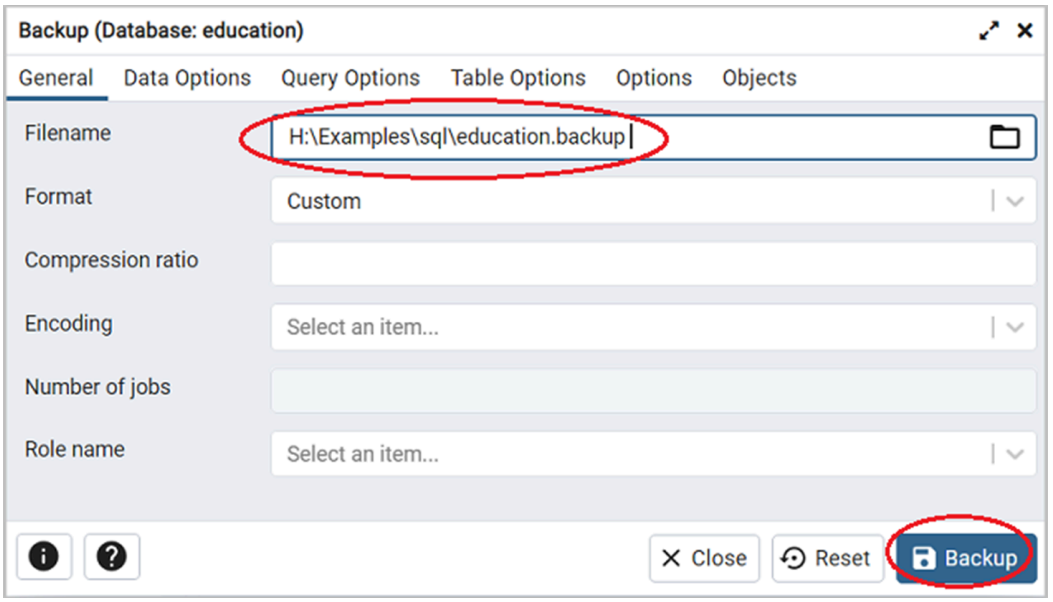
Создание архива, pgAdmin 4

pgAdmin 4 - это программа для администрирования и разработки БД PostgreSQL.

Для создания архива БД необходимо открыть список БД, установить указатель на выбранную базу данных, нажать правую кнопку мыши. Из выпадающего меню выбрать пункт "Backup…".



Откроется окно, в котором надо заполнить имя архива БД и нажать кнопку Backup.

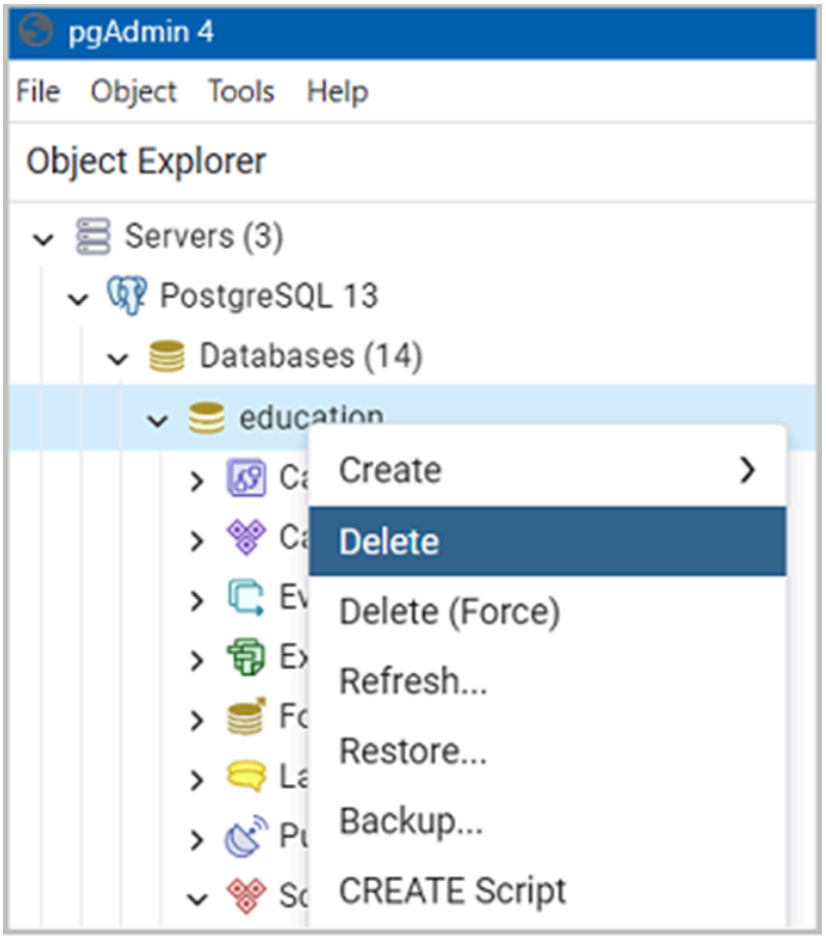


Восстановление из архива, pgAdmin 4

pgAdmin 4 - это программа для администрирования и разработки БД PostgreSQL.

Восстановление БД из архива не происходит поверх существующей БД. Есть определенный порядок:

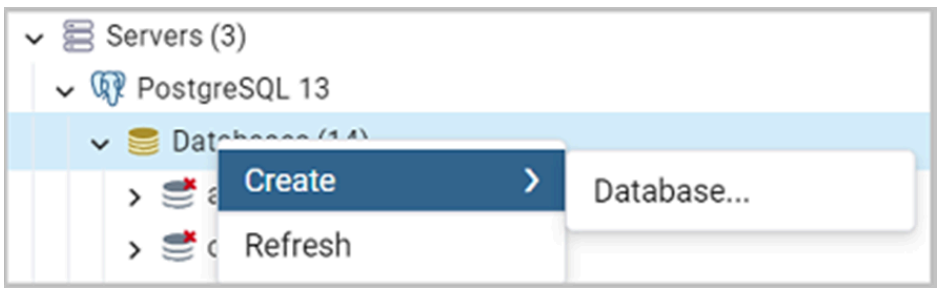
1. Существующую БД надо сначала удалить – она удалится вместе со своими таблицами (можно переименовать). Для этого надо стать в дереве объектов на удаляемую базу данных и нажать правую кнопку мыши. Из выпадающего меню выбрать пункт "Delete".



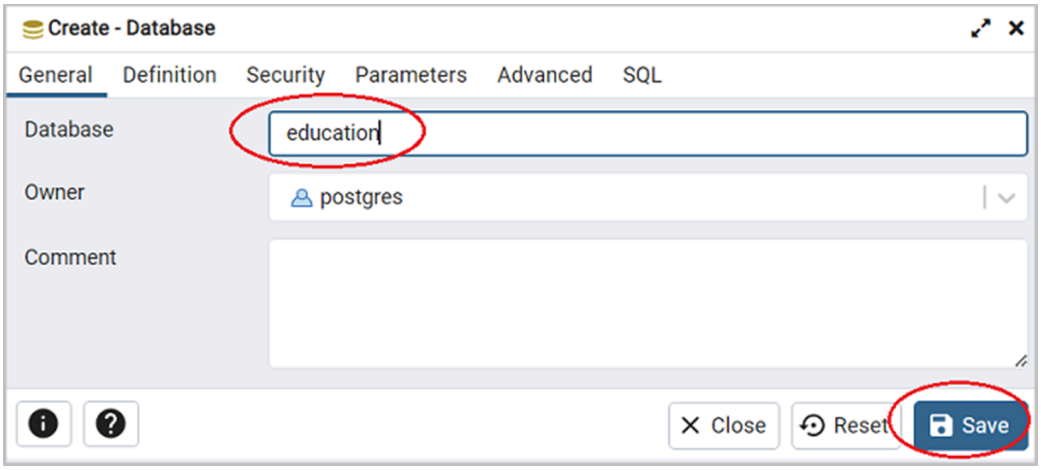
Примечание:

- БД не удалится, если она занята. Как минимум сервер приложений должен быть остановлен.

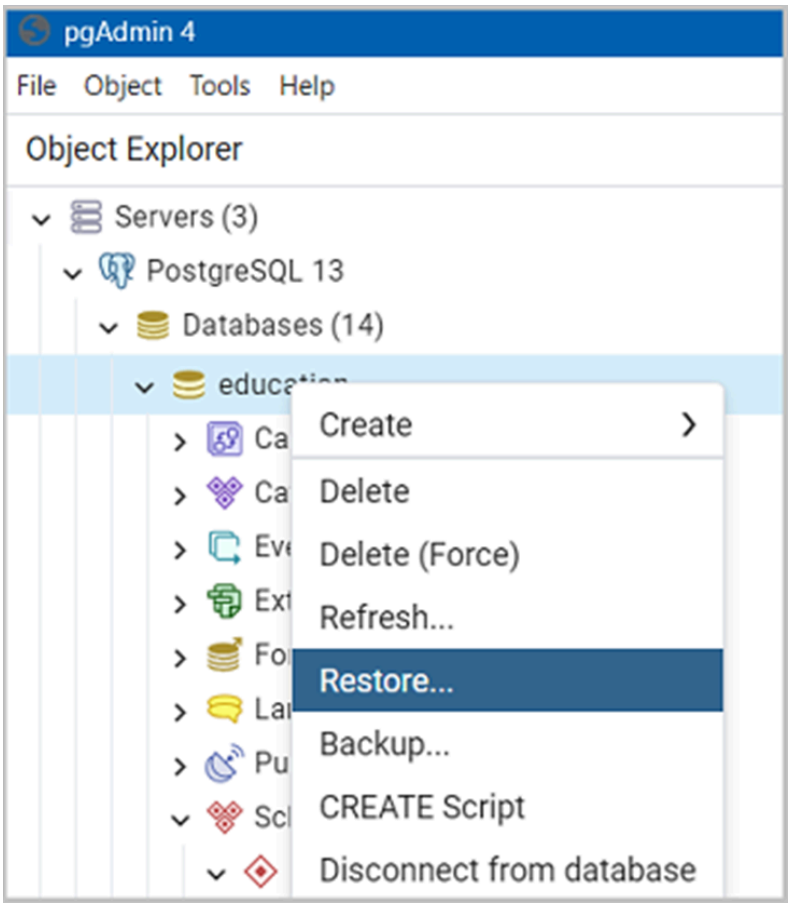
2. Создать БД с нужным именем. Для этого в дереве объектов стать на Databases и нажать правую кнопку мыши, из выпадающего меню выбрать пункты "Create - Database".



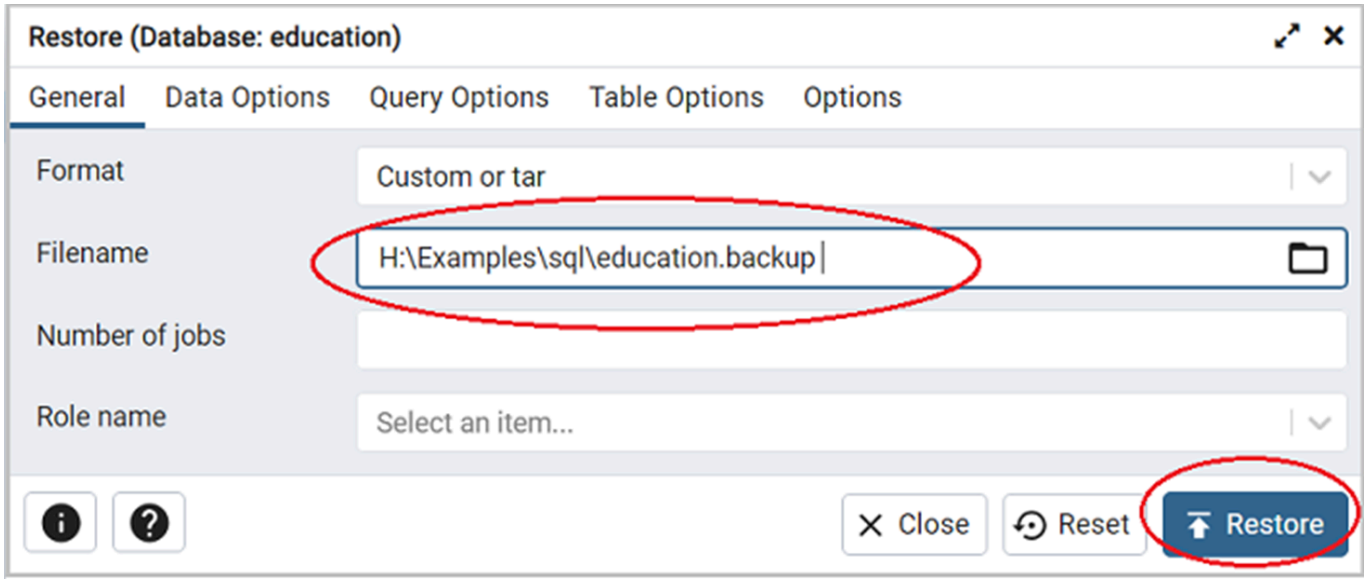
3. В появившемся окне вписать имя восстанавливаемой базы данных и нажать кнопку "Save".



4. В дереве объектов стать на имя созданной базы данных, нажать на правую кнопку мыши, из выпадающего меню выбрать пункт "Restore".

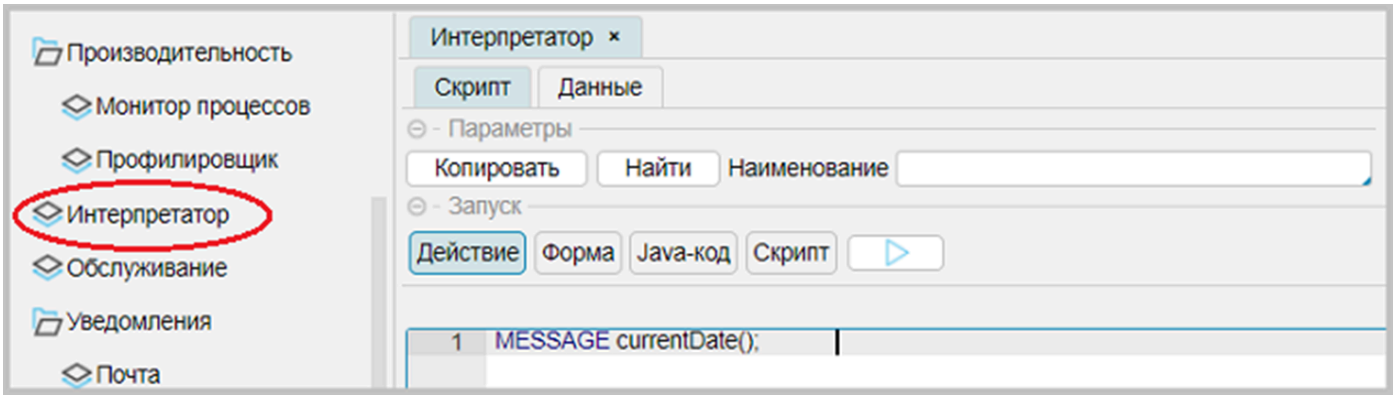


5. В появившемся окне выбрать архивный файл (backup) и нажать на кнопку "Restore".


Restore (Database: education)
General
Data Options
Query Options
Table Options
Options
Format
Custom or tar
Filename
H:\Examples\sql\education.backup
Number of jobs
Role name
Select an item...
Close
Reset
Restore

Интерпретатор

В настройках администрирования в разделе "Система" присутствует пункт "Интерпретатор". Интерпретатор позволяет выполнять скрипты на языке lsFusion.



Обычно интерпретатор используют для:

- выполнения действий разового характера, связанного с обслуживанием данных;
- выполнения действий для контроля данных, для которых отсутствуют нужные интерфейсы;
- тестирования участков кода;
- тестирования стандартных свойств и действий.

Все выполняемые в интерпретаторе действия могут быть сохранены и впоследствии воспроизведены.

По выполняемым действиям интерпретатора можно выделить:

- отображение форм, кнопка "Показать форму";
- выполнение действий, кнопка "Выполнить действие";
- выполнение скрипта, кнопка "Выполнить скрипт";
- кнопка "Выполнить Java-код" – выполняет тело void-метода java без входных параметров.

Чтение и запись скриптов

Каждый новый выполненный скрипт записывается на место предыдущего скрипта после нажатия на кнопку "сохранить" внизу формы

Для создания нового скрипта надо:

- нажать на кнопку "Копировать", внести изменения и нажать на кнопку "Сохранить" внизу формы;
- нажать на кнопку "Найти" и выбрать из формы "Скрипты" какой-либо скрипт, внести изменения и нажать на кнопку "Сохранить" внизу формы.

Поле "Наименование" используется для подписи скрипта, который будет сохранен.
При этом поле не является обязательным для заполнения.

Примеры использования интерпретатора

1. **Показать форму**

В этом варианте можно создать и отобразить форму с помощью инструкции FORM. При этом ключевое слово FORM вместе с указанием имени и заголовка опускается.
Для примера выведем форму отображения приходов:

```
OBJECTS h = HeaderReception
PROPERTIES (h) READONLY number, date, organizationName, userName, userDate
OBJECTS r = Reception
PROPERTIES (r) READONLY nameProduct, quantity, price
FILTERS header(r) = h
```

2. Выполнить действие

В этом варианте можно создать и выполнить произвольное действие с помощью оператора создания действия ({ … }). При этом внешние фигурные скобки не указываются.
Для примера отобразим существующую форму отображения приходных накладных:

```
ASK 'Вы хотите показать форму\nТоварных накладных' DO {
   SHOW viewReception;
}
```

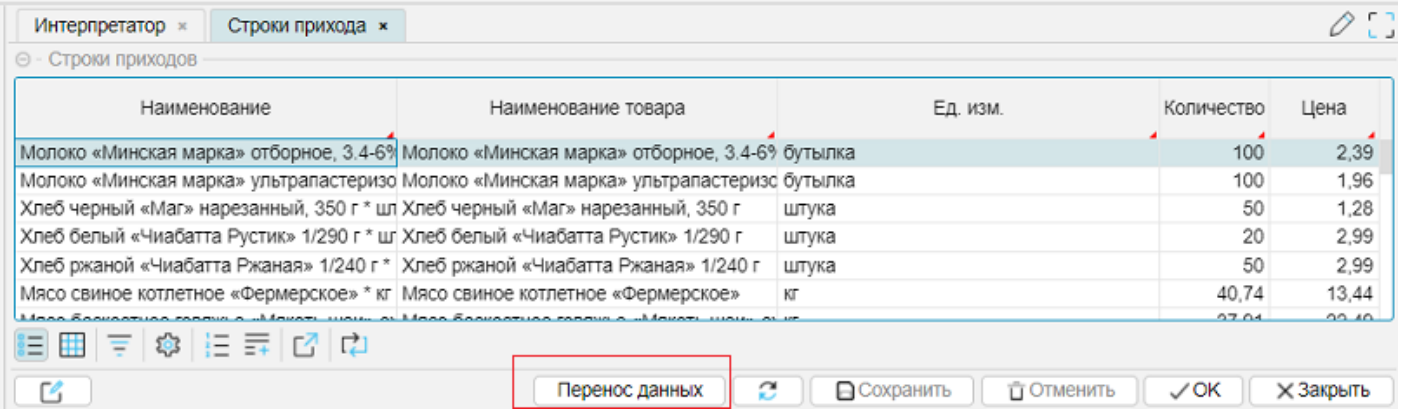
3. Выполнить скрипт

Выполняемый скрипт обладает широкими возможностями по отношению к "Показать форму" или "Выполнить действие", представляя собой небольшой модуль без заголовка. Этот модуль должен содержать в себе действие run(), которое и будет выполнено в результате. Для примера выполним заполнение справочников товаров и единиц измерения из свойства name строк прихода (см. Действие. Миграция. Интерпретатор). Перед выполнением необходимо восстановить БД SQL из архива: examples\t303_заполнение_товарных_групп\sql и восстановить программный код src из examples\t304_действия_миграция_данных\src_name. Код скрипта:

```
// обработка данных
migrate 'Перенос данных' () {
   ASK 'Вы действительно хотите\nвыполнить Перенос данных?' DO {
      FOR [GROUP MAX Reception r BY trim(getWord(name(r), '*', 2))](STRING[100] v) NEW u = Unit DO {
         name(u) <- v;
      }
      FOR [GROUP MAX Reception r BY getWord(name(r), '*', 1)](STRING[100] v) NEW p = Product DO {
         name(p) <- v;
      }

      group(Product p) <- (
         CASE
            WHEN istartsWith(name(p), 'Булочка') THEN Group.unique('БУЛОЧКИ')
            WHEN istartsWith(name(p), 'Мясо') THEN Group.unique('ПОЛУФАБРИКАТЫ')
            WHEN istartsWith(name(p), 'Из мяса') THEN Group.unique('ПРОДУКТЫ ИЗ МЯСА')
            WHEN istartsWith(name(p), 'Колбаса') THEN Group.unique('КОЛБАСЫ')
            ELSE OVERRIDE (GROUP MAX Group g IF istartsWith(name(p), name(g))), Group.unique('ВСЕ')
         ) WHERE p IS Product AND NOT group(p);
      APPLY;

      unit(Reception r) <- Unit.unique(upper(trim(getWord(name(r), '*', 2)))) WHERE r IS Reception AND NOT unit(r);
      product(Reception r) <- Product.unique(upper(getWord(name(r), '*', 1))) WHERE r IS Reception AND NOT product(r);

      APPLY;
      IF NOT canceled() THEN
         MESSAGE 'Операция выполнена';
```

```
    ELSE
        MESSAGE 'Операция не выполнена\nИз-за существующих ограничений\nВозможно, миграция
выполняется повторно';
  } ELSE {
    MESSAGE 'Выполнение отменено';
  }
}

FORM Reception 'Строки прихода'
OBJECTS r = Reception
PROPERTIES (r) READONLY name, nameProduct, nameUnit, quantity, price
PROPERTIES migrate() TOOLBAR;

// запуск скрипта
run () {
SHOW Reception DOCKED;
}
```

При выполнении скрипта на экране появится форма, в которой для выполнения операции переноса данных необходимо нажать на кнопку "Перенос данных":



Примечание:

- До выполнения операции колонки табличной части "Наименование товара" и "Ед. изм." будут пусты (так как справочники не заполнены). После выполнения операции колонки будут заполнены. Визуально можно сверить правильность заполнения.
- Скрипт действия migrate() используется без изменений так, как он был описан при рассмотрении вопроса Миграция. Действия. Кнопки. Интерпретатор.