Melda 2

РУКОВОДСТВО РАЗРАБОТЧИКА

Быстрый старт

Для запуска приложения на Melda необходимо либо скачать шалон типового проекта Melda, либо произвести создание проекта самостоятельно по следующей инструкции:

1) Скачать и подключить к своему проекту библиотеку melda2vxlib.jar

2) Скачать и установить прототип базы данных meldadb.sql

3) Создать пользователя meldaroot для базы данных. Предоставить ему полные права для управления БД.

4) Внести пользователя meldaroot в таблицу bms_users

#	1 id	2 ulogin	3 upass	4 uname	5 usecname	6 usurname	7 position	8 ugroup
1	1	meldaroot	Password	Владимир	Александрович	Пупкин	(NULL)	1

5) Создать проект со следующей структурой пакетов:

--app:
----components
----injects
----menu
----models
----res

6) В пакете Menu создаем класс USRMenuBuilder, наследующий Melda-класс MenuBuilder:

```
import MELDAVX.annotations.MeldaMenuBuilder;
import MELDAVX.windowmodel.menus.MenuBuilder;

@MeldaMenuBuilder
public class USRMenuBuilder extends MenuBuilder {

  @Override
  public void build() {

  }
}
```

7) В пакете Models создаем класс USRDataModelsInit, наследующий Melda-класс DataModelInit:

```
import MELDAVX.annotations.MeldaUserDMInit;
import MELDAVX.datamodelinit.DataModelInit;
import MELDAVX.datamodelinit.DataModelsInit;

@MeldaUserDMInit
public class USRDataModelsInit extends DataModelsInit implements DataModelInit {
      @Override
      public void init() {

     }
}
```

8) В пакете app создаем класс FakeMain, этот опционльаный класс будет использоваться для старта Main класса Melda приложения:

```
import sample.app.Main;


public class FakeMain {
  public static void main(String[] args) {
    Main.main(args);
  }
}
```

9) В пакете app создаем класс Main, это главный класс инициализирующий Melda приложение:

```
public class Main extends Application {

  @Override
  public void start(Stage stage) throws Exception{
    // Задание основных параметров приложения
    Melda.i().setAppName("Имя приложения");
    Melda.i().setAppPackage("имя_родительского_пакета.app");
    Melda.i().createDBSyncDefault("ip/url БД", 3306, "имя БД", DBSync.typeMySQL);
    Melda.i().setUserAndPass("meldaroot", "пароль");

    // Включение режима отладки Melda (по умолчанию отключено)
    // Melda.i().setOptDBDebugMode(false);

    // Включение режима коррекции БД Melda (по умолчанию отключено)
    // Melda.i().setOptMode(OptDB.optFullUpdate);

    Melda.i().init(stage);
  }

  public static void main(String[] args) {
    launch(args);
  }

}
```

10. Запускаем приложение через стартовый класс FakeMain.

N/B: В случае создания проекта через типовой шаблон, необходимо произвести только пункт 4 настройки проекта

1. Архитектура Melda-приложения

Любое приложение на базе Melda состоит из экранов с меню и компонентами, форм, модели данных и инъекций компонентов.

Экран это базовый UI элемент содержащий компоненты – таблицы, списки, кнопки и т. д.

Все компоненты в Melda делятся на две группы:

- **Встроенные компоненты Melda**. Данные компоненты можно добавлять на экран непосредственно в конструкторе BMS Setup (пт. BMS Setup Components). Поведение компонентов можно отчасти менять при помощи инъекций (пт. Инъекции).

- **Пользовательские компоненты**. Эти компоненты полностью разрабатываются пользователем. Классы данных компонентов размещаются строго в пакете app.components. Подробнее о создании пользовательских компонентов в Melda в пт, Пользовательские компоненты.

Каждый компонент может быть связан с данными. Для представления данных в Melda используются - Melda-списки (пт. 1 Модель данных). Эти списки связывают данные между собой.

Модель данных это набор классов, каждый из которых представляет структуру Melda-списка. Все такие классы должны находиться в пакете app.models. Каждый melda-список соответствует строго одной таблице из БД. Списки состоят из полей, также как таблицы БД также включают поля, но типы полей в Melda могут не соответствовать типам полей в БД. Melda автоматически следить за типами полей в БД и производит перевод значений.

Форма — это специальный экран, предназначенный для ввода или изменения данных в Melda-списках. Связанные данные в БД изменяются автоматически. Форма состоит элементов формы, которые добавляются в конструкторе BMS Setup (пт. BMS Setup Forms).

Меню — это элемент главного окна, который позволяет открывать экраны в главном окне или запускать методы-инъекции. Melda поддерживает лишь два уровня меню. Также для пользователя meldaroot открыто специальное меню BMS Setup предоставляющее доступ к конструктору экранов и форм. Добавление новых пунктов меню производится строго в классе UsrMenuBuilder в пакете app.menu Подробнее об управлении в пт. Меню.

Инъекция — специальные методы, обернутые в классы-инъекций расширяющие интерфейс компонентов экранов. Все классы инъекций должны располагаться в пакете app.injects

2. Модель данных

Каждая таблица базы данных связана с классом, представляющим соответствующую структуру данных - список в Melda. Все такие классы наследуют класс DataModel, хранятся в пакете models и инициализируются перегрузкой метода init() данного класса:

```
public class SimulatorCatsModel extends DataModel {
  @Override
  public void init() {

     this.add(DFB.dfString("title", "Надпись категории", 200));
  }
}
```

Каждое поле таблицы инициализируется фабрикой DFB. Melda поддерживает следующие типы полей:

ПРОСТЫЕ ПОЛЯ:

DFString – строковое поле
Добавление:
this.add(DFB.dfString([имя поля в DB:str], [надпись в Melda:str], [max симв.:int]));

Максимальное количество символов также допускает специальные значения:
DFString.charsCountTiny
DFString.charsCountMedium

DFBoolean – логическое поле
Добавление:
this.add(DFB.dfBoolean([имя поля в DB:str], [надпись в Melda:str]));

DFInteger – целочисленное поле
Добавление:
this.add(DFB.dfInteger([имя поля в DB:str], [надпись в Melda:str]));

DFMoney – поле денежной суммы
Добавление:
this.add(DFB.dfMoney([имя поля в DB:str], [надпись в Melda:str], [валюта]));

Поддерживаются след. виды валют:
DFMoney.mRUB – рубли
DFMoney.mEURO – евро

DFDate – поле даты
Добавление:
this.add(DFB.dfDate([имя поля в DB:str], [надпись в Melda:str]));

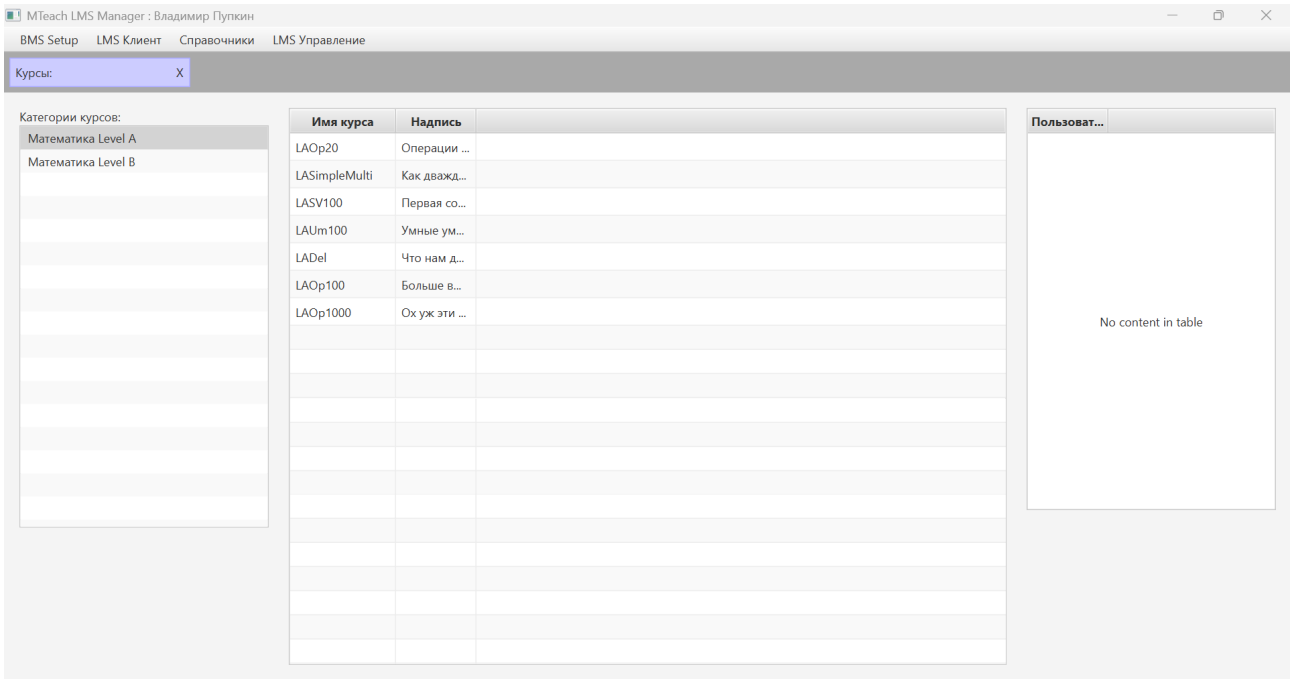
DFTime – поле времени
Добавление:
this.add(DFB.dfTime([имя поля в DB:str], [надпись в Melda:str]));

СПЕЦИАЛЬНЫЕ ПОЛЯ:

DFData – поле связанных данных. Указывает на строку в другом списке Melda.
Добавление:
this.add(DFB.dfData([имя поля в DB:str], [надпись в Melda:str], [имя источника данных - списка в Melda:str]));

Обычно связанный объект при отображении его в таблицах или полях показывается не всеми своими полями, а выбранными.
Для выбора отображаемых полей связанного объекта сразу после добавления поля DFData используем метод:
DFB.addShowedFields([fieldName : string])

Например, здесь каждому курсу соответствует единственная категория курсов:



Модель данных категории курсов задается классом, описывающим поля соответствующего Melda-списка:

```
public class CourseCatsModel extends DataModel {
  @Override
  public void init() {

     this.add(DFB.dfString("name", "Имя категории", 100));
     this.add(DFB.dfString("title", "Надпись", 300));
     this.add(DFB.dfString("descshort", "Краткое описание",
DFString.charsCountTiny));
     this.add(DFB.dfString("desclong", "Полное описание",
DFString.charsCountMedium));...
```

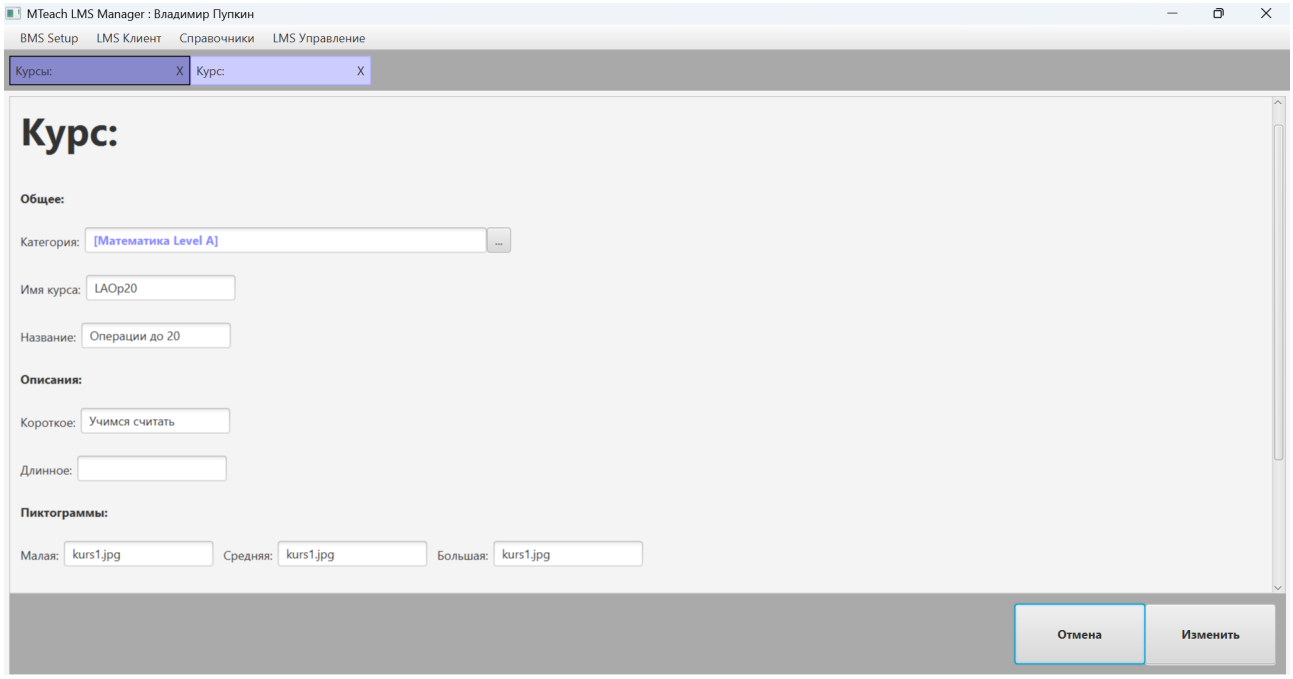
Модель данных курса задается аналогичным классом:

```
public class CoursesModel extends DataModel {
  @Override
  public void init() {

     this.add(DFB.dfString("coursename", "Имя курса", 200));

     this.add(DFB.dfData("coursecat", "Категория", "coursecats"));
     DFB.addShowedFields("title");
```

Как мы видим в качестве выбранного поля для отображения категории курса, например в формах, используется поле title категории курсов:



DFFile – поле, указывающее на файл.

Добавление:

this.add(DFB.dfData([имя поля в DB:str], [надпись в Melda:str], [имя хоста :str]));

В качестве имени хоста может выступать один из зарезервированных хостов melda-списка bms_remote_hosts, тогда указывается одно из значений поля fhostname. Либо можно использовать ключевое слово localhost для выбора файла на локальных носителях.

Для уточнения директории, в которой может осуществляться выбор файла на выбранном хосте используется метод сразу после добавления поля:

```
DFB.setFilePath([filePath : string])
```

Также для уточнения расширений файлов для выбора может использоваться метод:

```
DFB.addFileExt([fileExt : string])
```

Последний метод можно использовать несколько раз для добавления множества расширений. Если расширения не добавлены, то по умолчанию используется выбор любого расширения.

Например:

```
this.add(DFB.dfFile("imagesmall", "Маленький рисунок", "mtClientHost"));
 DFB.setFilePath("/imagescourses/");
 DFB.addFileExt("png"); DFB.addFileExt("jpg"); DFB.addFileExt("jpeg");
DFB.addFileExt("gif");
```

Инициализация модели данных

После создания модели данных Melda списка его необходимо инициализировать. Все списки инициализируются в файле USRDataModelsInit в пакете app.models Вот пример инициализации списков:

```
import MELDAVX.annotations.MeldaUserDMInit;
import MELDAVX.datamodelinit.DataModelInit;
import MELDAVX.datamodelinit.DataModelsInit;

@MeldaUserDMInit
public class USRDataModelsInit extends DataModelsInit implements DataModelInit {
      @Override
      public void init() {
        listInit(new LandsModel(), "lands", "default");
        listInit(new RegionsModel(), "regions", "default");
        listInit(new CitiesModel(), "cities", "default");
        listInit(new UsersModel(), "users", "default");
     }
}
```

Для инициализации используется метод:

listInit([listModelObject : DataModel], [listName : str], [dbSourceName : str])

где:

listModelObject – объект модели списка
listName – имя списка (совпадает с именем таблицы в БД)
dbSourceName – имя источника данных (главный источник всегда - Default).

Автосоздание структуры данных в БД

Melda поддерживает синхронизацию не только данных, но и структуры данных в БД. Это значит, что после задания структуры Melda-списков, соответствующие таблицы могут быть созданы автоматически. Для этого необходимо включить режим отладки при старте Melda с опцией редактирования структуры данных в БД:

```
public class Main extends Application {
  @Override
  public void start(Stage stage) throws Exception{
    … Прочий код ...
    // Включение режима коррекции отладки Melda (по умолчанию отключено)
    Melda.i().setOptDBDebugMode(false);
    // Включение режима коррекции БД Melda (по умолчанию отключено)
    Melda.i().setOptMode(OptDB.optFullUpdate);
    … Прочий код …
```

При запуске Melda в отладочном режиме запуск графического режима не производится. Вместо этого производится верификация и обновление структуры данных в БД и отчет выводится в консоль.

N/B: Для редактирования структуры данных БД пользователю должны быть предоставлены соответствующие права на уровне БД

В случае успешного создания таблицы в консоле вы увидите похожее сообщение:

```
Melda opt: Debug mode started.
Melda opt: FULL UPDATE MODE ACTIVE!!!
Melda opt: db table testdata was builded in DBS: default
Melda opt: all tables was builded.
Melda opt: all fields was checked.
```

Это значит, что Melda-список testdata не был ранее представлен в базе данных и Melda создала соответствующую таблицу в базе данных. При изменении типов полей в melda-списке структура таблицы в БД также будет изменена с ВОЗМОЖНЫМ сохранением данных (если это поддерживается на уровне БД).

N/B: Мы рекомендуем создавать резервную копию БД перед запуском режима отладки с опцией обновления данных!

При запуске Melda в режиме отладки без опции обновления структуры данных будет произведена только верификация структуры данных БД без обновления и соответствующий отчет будет выведен в консоль. Структура данных и сами данные в данном режиме остаются в безопасности.

3. Меню

В пакете app.menu хранится файл Buidler меню экрана:

```
package sample.app.menu;


import MELDAVX.annotations.MeldaMenuBuilder;
import MELDAVX.windowmodel.menus.MenuBuilder;

@MeldaMenuBuilder
public class USRMenuBuilder extends MenuBuilder {

  @Override
  public void build() {
    addMenu("LMS Клиент");
    addMenuInjectEvent("LMS Клиент", "Data Sync", "dataSync");
    addMenuInjectEvent("LMS Клиент", "Sessions Update", "sessionsUpdate");
    addMenuInjectEvent("LMS Клиент", "Clear slots and sessions",
"clearSlotsSessions");
}
```

Melda поддерживает два уровня меню: родительское и дочернее.

Для добавления родительского меню в перегрузку метода build класса USRMenuBuilder добавляется метод:

addMenu([menuName:str]);

Имя меню должно быть уникальным и используется как идентификатор для добавления дочерних меню.

Дочерние меню разделяются на два вида:

1. Меню запуска экрана. Служат для запуска экрана из melda-списка экранов bms_screens. Сами экраны создаются при помощи конструктора BMS Setup (см. пт. BMS Setup – Экраны). Для добавления такого пункта меню используется метод:

   addMenuScreen([parentMenuName:str], [menuName:str], [screenName:str]);

2. Меню события-инъекции. Данные меню служат для запуска специальных методов-инъекций, поведение которых определяется самим пользователем через написание классов-инъекций. Для добавления такого пункта меню используется метод:

   addMenuInjectEvent([parentMenuName:str], [menuName:str], [injectName:str]);

   Имя инъекции задается в конструкторе класса инъекции. А при клике по меню вызывается перегрузка метода callMethod класса инъекции.

   Например при добавлении меню:

addMenuInjectEvent("LMS Клиент", "Data Sync", "dataSync");

при клике по меню будет вызываться метод callMethod класса InjDataSync:

```
@MeldaInjectMethod
public class InjDataSync extends InjectMethod {

  public InjDataSync() {
    super("dataSync");
  }

  @Override
  public void callMethod(Object onObject, ArrayList<Object> pars) {
    accessDataRegistry.i().getDbSync("default").updateData();
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("MTeach Manager");
    alert.setHeaderText("Data Sync");
    alert.setContentText("All data sync succesfull!");
    alert.showAndWait();
    Melda.i().getMeldaUI().getCollector().update();
  }
}
```

Более подробно об инъекциях читайте в пт. Инъекции.

N/B: После добавления меню, чтобы оно появилось в главном окне нужно предварительно открыть к нему доступ к текущей группе пользователей. Подробнее о правах пользователей читайте в пт. Пользователи и доступ

3. Пользовательские компоненты

Помимо базовых Melda-компонентов, которые можно добавлять на экраны и настраивать в конструкторе BMS Setup, пользователь может разработать и добавить свои компоненты.

Все пользовательские компоненты должны храниться в пакете app.components и быть унаследованными от базового класса для Melda-компонентов Component. При этом можно использовать в качестве базового класса и дочерние классы Melda-компонентов, например NavigatorComponent. Все они наследуют структуру класса Component и расширяют родительский функционал.

Melda автоматически регистрирует в качестве пользовательских компонентов все классы в пакете app.components, наследующие класс Component.

Рассмотрим примерную структуру пользовательского класса:

```
public class SMEditorComponent extends NavigatorComponent {

  public SMEditorComponent(String name, Component sourceComponent, DataList
dataSource, int align) {
    super(name, sourceComponent, dataSource, null, "Редактор тренажеров", align);
  }

  @Override
  protected void buildPane() {
    //Построение интерфейса компонента
 }

  @Override
  public void update() {
   //Действия при поступлнении внешнего сигнала обновления контента компонента,
например при обновлении данных
  }

}
```

Данный компонент наследует структуру Melda-класса Navigator-component. Все Melda-компоненты, отображающие наборы данных, такие как список, таблица, выпадающий список и т.д. наследуют структуру данного класса.

Конструктор супер-класса Navigator-Component включает в себя следующие поля:

```
NavigatorComponent(String name, Component sourceComponent, DataList dataSource,
ArrayList<String> selectedFields, String title, int align)
```

name – имя компонента, поступает из конструктора Setup BMS, в котором настраивается компонент экрана

sourceComponent – родительский компонент, обычно это компонент самого Melda-экрана и автоматически устанавливается Melda.

dataSource – источник данных, Melda-список. Эти данные обычно и отображает компонент полностью или частично (фильтровано).

selectedFields – список выбранных полей в источнике данных для отображения в компоненте. Настраивается в конструкторе Melda Setup.

title – надпись компонента. Настраивается в конструкторе Melda Setup.

align – расположение компонента на родительском экране. Настраивается в конструкторе Melda Setup. Полностью управляется Melda.

При создании пользовательского компонента на базе Navigator-component часто приходится обращаться к следующим параметрам родительского класса:

super.pane – панель, на которой располагаются элементы UI компонента, при активации метода buildPane()