

EDFM Simulator 开发者快速入门

1. 先明确当前主线

这个项目当前的正式 GUI 路线是：

- ◆ 界面文件： `gui_support/app/EDFM_Simulator_App.mlapp`
- ◆ 控制器： `gui_support/app/EDFMApController.m`
- ◆ 统一运行入口： `gui_support/runtime/run_case.m`

2. 当前架构怎么理解

建议把项目拆成三层理解。

第一层：界面层

正式界面在：

- ◆ `EDFM_Simulator_App.mlapp`

这一层主要负责：

- ◆ 组件摆放
- ◆ 组件命名
- ◆ 回调入口

第二层：控制器层

控制器文件：

- ◆ `gui_support/app/EDFMApController.m`

这一层主要负责：

- ◆ startup 初始化
- ◆ 模板加载
- ◆ 导入/导出配置
- ◆ 导入/导出结果
- ◆ UI 到 `config` 的写回
- ◆ `config` 到 UI 的回填
- ◆ 调用 `run_case(config)`
- ◆ 更新结果页
- ◆ 处理 wells 页表格逻辑

第三层：后端运行层

统一运行入口：

- ◆ `gui_support/runtime/run_case.m`

这一层负责：

- ◆ 根据 `config` 构造运行所需参数
- ◆ 调用网格、裂缝、流体、井、schedule 相关逻辑
- ◆ 最终调用原始求解器

3. 开发时应遵循的原则

以后开发尽量遵循下面的分工。

`.mlapp` 负责什么

- ◆ 负责界面布局
- ◆ 负责组件名称
- ◆ 负责简单回调入口

`EDFMApController.m` 负责什么

- ◆ 负责业务逻辑
- ◆ 负责数据同步
- ◆ 负责调用后端

`run_case.m` 负责什么

- ◆ 负责统一组织求解流程
- ◆ 负责把config 真正转换成后端求解器输入

4. 当前,mlapp 已经具备什么

根据你导出的 `edfm_simulator_app.txt` , 当前 `.mlapp` 已经不是空壳, 而是已经完成了较完整的绑定:

- ◆ 已有 `Controller EDFMAppController`
- ◆ 已有 `startupFcn`
- ◆ startup 时会执行:

```
app.Controller = EDFMAppController(app);  
app.Controller.startup();
```

- ◆ 顶部按钮已绑定到 controller
- ◆ results 页按钮已绑定到 controller
- ◆ wells 页表格增删与选中已绑定到 controller
- ◆ 各类绘图参数变化已绑定到 controller

5. App Designer 中最常见的组件

下面只讲和当前.mlapp 直接相关的组件。

uidropdown

下拉框，用于：

- ◆ 模板选择
- ◆ 模型选择
- ◆ 绘图选项选择

常见属性：

- ◆ Items
- ◆ Value
- ◆ ValueChangedFcn

uibutton

按钮，用于：

- ◆ 加载模板
- ◆ 导入导出
- ◆ 运行
- ◆ 绘图
- ◆ 表格行增删

常见属性：

- ◆ Text
- ◆ ButtonPushedFcn

uitable

表格控件，当前项目非常关键，尤其在well 页。

当前主要有：

- ◆ Well1Table
- ◆ Well2Table
- ◆ ScheduleTable
- ◆ FractureWellLocationTable

常见属性：

- ◆ `Data`
- ◆ `ColumnName`
- ◆ `RowName`
- ◆ `ColumnEditable`
- ◆ `CellSelectionCallback`

`uitextarea`

多行文本框，当前仍大量用于输入 MATLAB 表达式，如：

- ◆ 数组
- ◆ 矩阵
- ◆ `cell`
- ◆ 边界与裂缝文本数据

`uieditfield`

数值编辑框，用于：

- ◆ 求解器参数
- ◆ 初始条件
- ◆ 部分单值参数

`uiaxes`

绘图区，用于展示：

- ◆ 时间步图
- ◆ Newtons vs Time
- ◆ 部分结果图

6. 回调函数怎么理解

回调函数就是用户操作组件时触发的函数。

在当前 `mlapp` 主线下，推荐让回调函数保持很薄，只做“转发”。

例如按钮回调：

```
function onRun(app, event)
    app.Controller.runCurrentConfig();
```

```
end
```

这种写法的好处是：

- ◆ `.mlapp` 内代码简单
- ◆ 业务逻辑集中在 controller
- ◆ 后续换布局时不容易把逻辑打散

7. startup 机制

当前 `mlapp` 的关键启动逻辑是：

```
function startupFcn(app)
    app.Controller = EDFMAppController(app);
    app.Controller.startup();
end
```

App 创建完成后，`runStartupFcn(app, @startupFcn)` 会触发这段逻辑。

它的作用是：

1. 给 app 挂上 controller
2. 初始化下拉框
3. 设置表格
4. 加载默认模板
5. 刷新界面

如果启动后界面不正常，先查这条链路，而不是先去看求解器。

8. 组件名重要

`EDFMApController.m` 大量依赖组件名访问控件，例如：

```
obj.App.(propName)
```

这意味着：

- ◆ 组件名一旦变了
- ◆ controller 里对应逻辑就可能失效

所以在 App Designer 里改界面时，第一原则是：

- ◆ 不要随意修改已有组件名

如果确实要改名，必须同步修改 controller 中所有相关引用。

9. 修改界面的正确方式

9.1 纯布局修改

如果只是改：

- ◆ 位置
- ◆ 大小
- ◆ 标签文字
- ◆ 页签布局

那么直接在 App Designer 中改 `mlapp` 即可。

这种修改通常不需要动 controller。

9.2 新增一个控件

如果新增了一个输入控件，通常要做四件事。

第一步：在 `mlapp` 中添加控件

例如增加：

- ◆ 一个下拉框
- ◆ 一个按钮
- ◆ 一个数值框

第二步：给它起稳定名字

命名风格建议沿用现有项目，例如：

- ◆ `MyNewDropDown`
- ◆ `MyNewButton`
- ◆ `MyNewEditField`

第三步：添加回调

例如按钮：

```
function MyNewButtonPushed(app, event)
    app.Controller.doSomething();
end
```

第四步：在 controller 中补逻辑

这是关键步骤，不要把逻辑只写在.mlapp 里。

10. 新增一个参数字段时怎么改

假设新增一个参数，推荐按下面顺序。

1. 先决定它在config 中的位置

例如：

- ◆ `config.solver.xxx`
- ◆ `config.wells.xxx`
- ◆ `config.flow.multi_component.xxx`

2. 修改默认配置

编辑：

- ◆ `gui_support/config/create_empty_config.m`

3. 修改模板

编辑：

- ◆ `gui_support/templates/load_case_template_case01.m`
- ◆ 其他需要的 `caseXX`

4. 在.mlapp 中增加控件

并给出稳定组件名。

5. 在 controller 中补两处

必须同时补：

- ◆ `refreshUIFromConfig`

- ◆ `pushUIToConfig()`

这两个函数分别负责：

- ◆ `config -> UI`
- ◆ `UI -> config`

如果只改一边，参数就会出现“显示了但不会保存”或者“保存了但不会显示”的问题。

6. 如果运行要用到，再改 `run_case.m`

只有到这一步，才去改后端运行逻辑。

11. 如何实现一个完整功能

这里给一个适用于本项目的通用开发流程。

第一步：先明确功能边界

先回答这几个问题：

- ◆ 用户从哪个页面进入
- ◆ 输入是什么
- ◆ 点击哪个按钮
- ◆ 结果显示在哪里

第二步：落到统一 `config`

不要直接从控件把值硬塞到求解器里，优先让它进入统一`config`。

第三步：补齐 UI 和 config 的双向同步

需要改：

- ◆ `mlapp`
- ◆ `refreshUIFromConfig()`
- ◆ `pushUIToConfig()`

第四步：补齐运行逻辑

需要改：

- ◆ `run_case.m`
- ◆ 以及必要的后端函数

第五步：补齐结果显示

如果功能有结果输出，还需要改：

- ◆ `refreshResults`
- ◆ 结果页控件
- ◆ 可能的绘图逻辑

13. 开发者在 App Designer 中最常见的修改场景

场景一：改一个按钮文字

直接在 App Designer 中改 `Text` 即可。

场景二：改布局

直接在 App Designer 中调整控件位置、网格和页签。

场景三：加一个按钮

推荐做法：

1. 在 `mlapp` 里加按钮
2. 生成回调函数
3. 回调函数里调用 controller
4. 在 controller 实现业务逻辑

场景四：加一个新表字段

推荐做法：

1. 先改 `mlapp` 里的表展示
2. 再改 controller 表头
3. 再改默认行
4. 再改 config 和模板
5. 最后确认后端是否需要这个字段