Руководство для разработчиков DLL для Setun IDE

1. Требования к DLL

Бинарная совместимость

- Компилятор: Embarcadero C++ Builder (желательно та же версия, что и основное приложение)
- Calling Convention: __stdcall
- Выравнивание структур: одинаковое в EXE и DLL
- Настройки RTL: одинаковые (динамическая/статическая)

2. Экспортируемые функции

DLL ОБЯЗАНА экспортировать как минимум одну функцию:

```
// Обязательная функция регистрации
extern "C" {
   __declspec(dllexport) bool __stdcall RegisterLibrary(TLibraryManager* libraryManager);
}

// Опционально - функция выгрузки
extern "C" {
   __declspec(dllexport) void __stdcall UnregisterLibrary(TLibraryManager* libraryManager);
}
```

3. Структура проекта DLL

YourLibrary/

```
├── YourLibrary.dproj          // Проект DLL
├── Source/
│   ├── YourLibrary.h          // Главный заголовочный файл
│   ├── YourLibrary.cpp        // Реализация регистрации
│   ├── YourElements.h         // Ваши элементы
│   └── YourElements.cpp       // Реализация элементов
├── Headers/                   // Общие заголовки (должны быть идентичны EXE)
│   ├── CircuitElement.h
│   ├── CircuitElements.h
│   ├── ComponentLibrary.h
│   └── TernaryTypes.h
└── Resources/                 // Опционально - ресурсы
    └── Elements.res
```

Детальная реализация

1. Заголовочный файл DLL (YourLibrary.h)

```
#ifndef YourLibraryH
#define YourLibraryH

#include "ComponentLibrary.h"

// Макросы для экспорта
#ifdef BUILD_DLL
  #define DLL_EXPORT __declspec(dllexport)
#else
  #define DLL_EXPORT __declspec(dllimport)
#endif

// Версия библиотеки
#define YOUR_LIBRARY_VERSION "1.0.0"
#define YOUR_LIBRARY_NAME "YourLibraryName"
#define YOUR_LIBRARY_DESCRIPTION "Описание вашей библиотеки"

extern "C" {
  // Обязательные функции
  DLL_EXPORT bool __stdcall RegisterLibrary(TLibraryManager* libraryManager);
  DLL_EXPORT void __stdcall UnregisterLibrary(TLibraryManager* libraryManager);

  // Опциональные информационные функции
  DLL_EXPORT const char* __stdcall GetLibraryName();
  DLL_EXPORT const char* __stdcall GetLibraryVersion();
  DLL_EXPORT const char* __stdcall GetLibraryDescription();
  DLL_EXPORT int __stdcall GetElementCount();
}

// Глобальный указатель для управления библиотекой (опционально)
extern TComponentLibrary* GYourLibrary;

#endif
```

2. Реализация DLL (YourLibrary.cpp)

```
#define BUILD_DLL
#include "YourLibrary.h"
#include "YourElements.h"  // Ваши пользовательские элементы
#include <memory>
#include <stdexcept>

// Глобальный указатель на библиотеку
TComponentLibrary* GYourLibrary = nullptr;

// Ваши пользовательские элементы должны быть объявлены здесь
class TYourCustomElement : public TCircuitElement {
private:
   // Ваши поля
public:
   TYourCustomElement(int AId, int X, int Y);
   void Calculate() override;
   void Draw(TCanvas* Canvas) override;
};

// Реализация ваших элементов
TYourCustomElement::TYourCustomElement(int AId, int X, int Y)
   : TCircuitElement(AId, "Custom", X, Y) {
   // Настройка входов/выходов
   FInputs.push_back(TConnectionPoint(this, X-15, Y+20, TTernary::ZERO, true,
TLineStyle::POSITIVE_CONTROL));
   FOutputs.push_back(TConnectionPoint(this, X+95, Y+20, TTernary::ZERO, false,
TLineStyle::OUTPUT_LINE));
}

void TYourCustomElement::Calculate() {
   // Логика вашего элемента
   if (!FInputs.empty()) {
      FOutputs[0].Value = FInputs[0].Value;
   }
}

void TYourCustomElement::Draw(TCanvas* Canvas) {
   Canvas->Rectangle(FBounds.Left, FBounds.Top, FBounds.Right, FBounds.Bottom);
   Canvas->TextOut(FBounds.Left + 5, FBounds.Top + 5, "Custom");
   DrawConnectionPoints(Canvas);
}

extern "C" {

DLL_EXPORT bool __stdcall RegisterLibrary(TLibraryManager* libraryManager) {
   if (!libraryManager) {
      // Логирование ошибки (в продакшн можно использовать OutputDebugString)
      return false;
   }
```

```
try {
    // Создаем библиотеку с уникальным именем
    auto yourLib = std::make_unique<TComponentLibrary>(
        YOUR_LIBRARY_NAME,
        YOUR_LIBRARY_DESCRIPTION,
        YOUR_LIBRARY_VERSION
    );

    // РЕГИСТРАЦИЯ ЭЛЕМЕНТОВ - КЛЮЧЕВАЯ ЧАСТЬ

    // Базовый синтаксис:
    // yourLib->RegisterElement<ClassName>(
    //     "Имя для отображения",
    //     "Описание элемента",
    //     "Категория",
    //     Ширина, Высота
    // );

    // Примеры регистрации:
    yourLib->RegisterElement<TYourCustomElement>(
        "Пользовательский элемент",
        "Мой первый пользовательский элемент",
        "Пользовательские",
        80, 60
    );

    yourLib->RegisterElement<TYourCustomElement>(
        "Мощный пользовательский элемент",
        "Улучшенная версия с большим размером",
        "Пользовательские",
        120, 80
    );

    // ВАЖНО: Можно регистрировать ЛЮБОЙ класс, унаследованный от TCircuitElement
    // с конструктором (int, int, int) и реализацией Calculate() и Draw()

    // Сохраняем указатель для возможности выгрузки
    GYourLibrary = yourLib.get();

    // Регистрируем в менеджере
    libraryManager->RegisterLibrary(std::move(yourLib));

    return true;
}
catch (const std::exception& e) {
    // Обработка стандартных исключений
    // В реальной DLL стоит добавить логирование
    return false;
}
catch (...) {
    // Обработка любых других исключений
    return false;
```

```
    }
}

DLL_EXPORT void __stdcall UnregisterLibrary(TLibraryManager* libraryManager) {
    if (libraryManager && GYourLibrary) {
        libraryManager->UnregisterLibrary(GYourLibrary->Name);
        GYourLibrary = nullptr;
    }
}

// Опциональные информационные функции
DLL_EXPORT const char* __stdcall GetLibraryName() {
    return YOUR_LIBRARY_NAME;
}

DLL_EXPORT const char* __stdcall GetLibraryVersion() {
    return YOUR_LIBRARY_VERSION;
}

DLL_EXPORT const char* __stdcall GetLibraryDescription() {
    return YOUR_LIBRARY_DESCRIPTION;
}

DLL_EXPORT int __stdcall GetElementCount() {
    return GYourLibrary ? GYourLibrary->ElementCount : 0;
}

} // extern "C"

// Точка входа DLL (опционально, но рекомендуется)
#include <windows.h>

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved) {
    switch (fdwReason) {
        case DLL_PROCESS_ATTACH:
            // Инициализация, если нужна
            break;
        case DLL_PROCESS_DETACH:
            // Очистка, если нужна
            break;
    }
    return TRUE;
}
```

3. Настройки компиляции DLL

Обязательные настройки проекта:

1. Target: Package (DLL)
2. Calling Convention: __stdcall
3. Runtime Library: Dynamic (или такой же как в EXE)
4. Preprocessor Definitions: BUILD_DLL
5. Output: YourLibrary.dll

Рекомендуемые настройки:

// В опциях компилятора:
- Alignment: 8 bytes
- Stack Size: 16384
- Optimization: Speed
- RTTI: Enabled
- Exceptions: Enabled

Пример минимальной рабочей DLL

Минимальный вариант (только обязательные функции):

```
// MinimalLibrary.cpp
#define BUILD_DLL
#include "ComponentLibrary.h"
#include "CircuitElement.h"

class TMinimalElement : public TCircuitElement {
public:
    TMinimalElement(int AId, int X, int Y)
        : TCircuitElement(AId, "Minimal", X, Y) {
        FInputs.push_back(TConnectionPoint(this, X-15, Y+20, TTernary::ZERO, true,
TLineStyle::POSITIVE_CONTROL));
        FOutputs.push_back(TConnectionPoint(this, X+95, Y+20, TTernary::ZERO, false,
TLineStyle::OUTPUT_LINE));
    }

    void Calculate() override {
        if (!FInputs.empty()) {
            FOutputs[0].Value = FInputs[0].Value;
        }
    }

    void Draw(TCanvas* Canvas) override {
        Canvas->Rectangle(FBounds.Left, FBounds.Top, FBounds.Right, FBounds.Bottom);
        Canvas->TextOut(FBounds.Left + 5, FBounds.Top + 5, "Minimal");
        DrawConnectionPoints(Canvas);
    }
};

extern "C" {
__declspec(dllexport) bool __stdcall RegisterLibrary(TLibraryManager* libraryManager) {
    if (!libraryManager) return false;

    auto lib = std::make_unique<TComponentLibrary>("Minimal", "Минимальная библиотека");
    lib->RegisterElement<TMinimalElement>("Минимальный элемент", "Простой тестовый
элемент");

    libraryManager->RegisterLibrary(std::move(lib));
    return true;
}
} // extern "C"
```

Критически важные моменты

1. Совместимость заголовочных файлов

Заголовки должны быть ИДЕНТИЧНЫ в EXE и DLL
Одинаковые версии CircuitElement.h, ComponentLibrary.h и т.д.
Одинаковые настройки компиляции

2. Управление памятью

```
// ПРАВИЛЬНО - элементы создаются в DLL
return std::make_unique<TYourElement>(Id, X, Y);

// НЕПРАВИЛЬНО - не используйте сырые указатели
return new TYourElement(Id, X, Y);
```

3. Обработка ошибок

```
// Все исключения ДОЛЖНЫ быть перехвачены внутри DLL
try {
    // ваш код
}
catch (const std::exception& e) {
    return false; // или соответствующий код ошибки
}
catch (...) {
    return false;
}
```

Тестирование DLL

Процесс тестирования:

1. Скомпилируйте DLL
2. Положите в папку с EXE
3. Запустите Setun IDE
4. Проверьте, что библиотека появилась в выпадающем списке
5. Проверьте создание элементов
6. Проверьте симуляцию

Отладка проблем:

```
// Добавьте отладочный вывод в DLL
OutputDebugString(L"DLL: RegisterLibrary called\n");
```

Распространение DLL

Включите в дистрибутив:
- YourLibrary.dll
- README.txt с описанием элементов
- Лицензию (если требуется)

Структура README:

```
# YourLibrary для Setun IDE

## Элементы:
- **Элемент 1**: Описание, входы/выходы
- **Элемент 2**: Описание, входы/выходы

## Требования:
- Setun IDE версии 0.1+
- Совместимая версия Windows

## Установка:
Скопируйте YourLibrary.dll в папку с SetunIDE.exe
```

Эта структура гарантирует успешную загрузку и работу вашей DLL в Setun IDE!