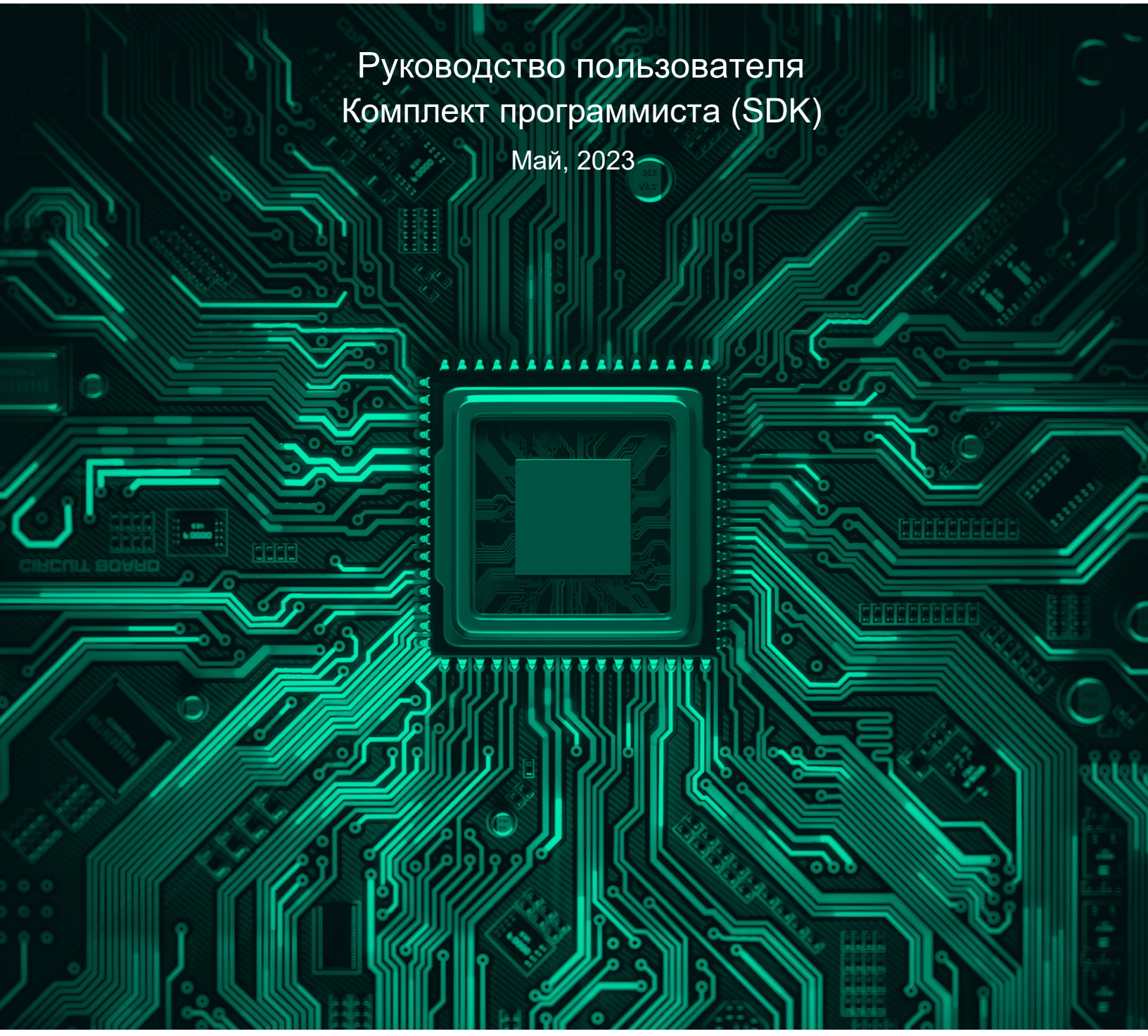
DeltaDesign

Комплексная среда сквозного проектирования электронных устройств

Руководство пользователя
Комплект программиста (SDK)

Май, 2023



EREMEX

Руководство пользователя

Внимание!

Данный документ предназначен для продвинутого пользователя ПК, знакомого с поведением и механизмами операционной системы Windows, уверенно владеющего инструментарием операционной системы.
Последнюю версию документа можно получить в сети Интернет по ссылке:

www.eremex.ru/knowleage-base/delta-design/docs

В случае возникновения вопросов по использованию программ Delta Design, TopoR, SimOne, пожалуйста, обращайтесь:

Форум компании «ЭРЕМЕКС»: www.eremex.ru/society/forum

Техническая поддержка
E-mail: support@eremex.ru
Skype: supporteremex

Отдел продаж
Тел. +7 (495) 232-18-64
E-mail: info@eremex.ru
E-mail: sales@eremex.ru

Руководство пользователя

Добро пожаловать!

Компания «ЭРЕМЕКС» благодарит Вас за приобретение системы Delta Design и надеется, что она будет удобным и полезным инструментом в Вашей проектной деятельности.

Система Delta Design является интегрированной средой, обеспечивающей средствами автоматизации сквозной цикл проектирования электронных устройств, включая:

- Формирование базы данных радиоэлектронных компонентов, ее сопровождение и поддержание в актуальном состоянии;
- Проектирование принципиальных электрических схем;
- SPICE - моделирование работы аналоговых устройств;
- Разработка конструкций печатных плат;
- Размещение электронных компонентов на наружных слоях печатной платы и проектирование сети электрических соединений (печатных проводников, межслойных переходов) в соответствии с заданной электрической схемой и правилами проектирования структуры печатного монтажа;
- Выпуск конструкторской документации в соответствии с ГОСТ;
- Выпуск производственной документации, в том числе необходимой для автоматизированных производственных линий;
- Подготовка данных для составления перечня закупаемых изделий и материалов, необходимых для изготовления изделия.

Руководство пользователя

Требования к аппаратным и программным средствам

Система Delta Design предназначена для использования на персональных компьютерах, работающих под управлением следующих версий операционных систем:

- Microsoft Windows 7 SP1+ Patch (KB976932), Windows 8.1, Windows 10.

На компьютере также должны быть установлены следующие программные средства:

- Platform Update Patch (KB2670838) для Windows 7.

Конфигурация рабочего места для использования Delta Design 3.0 и выше

Минимальные требования:

- Поддерживается только 64-разрядная версия ОС.
- Процессор от 2 ядер и выше тактовой частотой от 2.5 Ггц.
- Оперативная память от 8 Гб.
- Монитор с разрешением FullHD (1920x1080) и размером диагонали 24” с IPS или VA матрицей.

Для комфортной работы рекомендуется:

- 4-х или 8-и ядерный процессор с тактовой частотой от 3.5 Ггц.
- Требуемый размер оперативной памяти зависит от размера проектов, размера библиотек и числа одновременно открытых проектов. Рекомендуется от 16 Гб оперативной памяти. Для построения реалистичных 3D моделей больших печатных плат может потребоваться 32 Гб и более оперативной памяти. Не рекомендуется использование файла подкачки, поскольку это существенно снижает производительность системы.
- Для быстрого открытия и сохранения проектов рекомендуется SSD диск с объёмом, достаточным для хранения системы Delta Design и всех данных. Рекомендуется выделенный SSD диск от 256 Гб (для версий Standard и Professional).
- Желательно дискретная видеокарта с объёмом видеопамяти от 3Гб.
- 2 монитора с разрешением 1920x1080 и размером диагонали 24” или 1 монитор с разрешением WQHD (2560x1440) с размером диагонали 32”. Матрица с IPS или VA. Размер монитора должен соответствовать его разрешению, чтобы комфортно работать без масштабирования изображения, т.е. в режиме 100% (96DPI). Delta Design не поддерживает масштабирование интерфейса.

Примечание! В минимальной конфигурации возможность построения реалистичной 3D модели большой печатной платы не гарантируется!

Примечание! Совместная работа в варианте поставки «Delta Design Workgroup» поддерживает одновременную работу с одной базой данных не более 10 клиент-приложений.

Конфигурация рабочего места должна быть сбалансированной, поэтому применение 4K монитора требует лучшей видеокарты, большего объёма оперативной памяти и более мощного процессора.

Руководство пользователя

Техническая поддержка и сопровождение

Примечание! Техническая поддержка оказывается только пользователям, прошедшим курс обучения. Подробные сведения о курсе обучения могут быть получены по адресу в интернете

www.eremex.ru/learning-center

При возникновении вопросов, связанных с использованием Delta Design, рекомендуем:

- Ознакомиться с документацией (руководством пользователя);

  www.eremex.ru/knowleage-base/delta-design/docs

- Ознакомиться с информацией на сайте в разделе «База знаний», содержащей ответы на часто задаваемые вопросы;

  www.eremex.ru/knowleage-base

- Ознакомиться с существующими разделами форума. Также имеется возможность задать вопрос на форуме www.eremex.ru/society/forum если интересующая Вас тема ранее не освещалась.

Примечание! Если вышеперечисленные источники не содержат рекомендаций по разрешению возникшей проблемы, обратитесь в техническую поддержку. Подробную информацию о проблеме, действиях пользователя, приведших к ней, и информацию о программно-аппаратной конфигурации используемого компьютера, направить по адресу support@eremex.ru

Содержание

Комплект программиста (SDK)

1 Общие сведения

Система Delta Design поддерживает реализацию программных скриптов для автоматизации выполнения различных операций. Ниже описаны интерфейсы прикладного программирования (API) для таких скриптов.

Для написания скриптов используется язык C#.

Скрипт может содержать несколько классов. Все используемые классы должны находиться в пространстве имён Prosoft.ECAD.Script.

Хотя бы один класс скрипта – главный класс – должен являться наследником класса ScriptBase. Если таких классов в скрипте несколько, то главным классом считается тот, который имеет атрибут ScriptClass. Главный класс скрипта реализует функцию с именем Main и сигнатурой

```
public async Task Main()
```

которая определяет точку входа скрипта.

Важно! Наименование скрипта должно состоять строго из буквенно-цифровых символов. При этом имя создаваемого/запускаемого в Delta Design скрипта должно начинаться ТОЛЬКО с буквенных символов (кириллица/латиница). В имени скрипта не допускается использование символов (к примеру, @, ?, $ и т.д.).

2 Создание скрипта

Для создания нового скрипта необходимо из выпадающего списка раздела «Файл» главного меню вызвать команду «Создать», далее выбрать команду «Скрипт», Рис. 1.

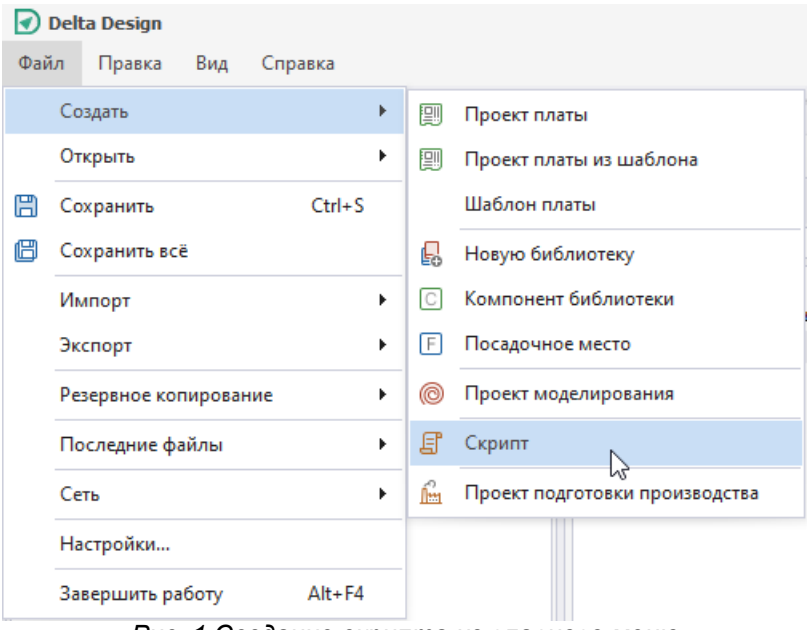


Рис. 1 Создание скрипта из главного меню

В рабочей области окна текстового редактора отображается шаблон скрипта, на основе которого создается пользовательский скрипт, Рис. 2.

Скрипт.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Threading.Tasks;

namespace Prosoft.ECAD.Script
{
    [ScriptClass]
    public class Script : ScriptBase
    {
        public async Task Main()
        {
            // Здесь будет код скрипта
        }
    }
}
```

Рис. 2 Шаблонная структура скрипта

Встроенный текстовый редактор существенно упрощает работу по созданию скрипта. Технология автодополнения IntelliSense™ дописывает название функции при вводе начальных букв, Рис. 3.

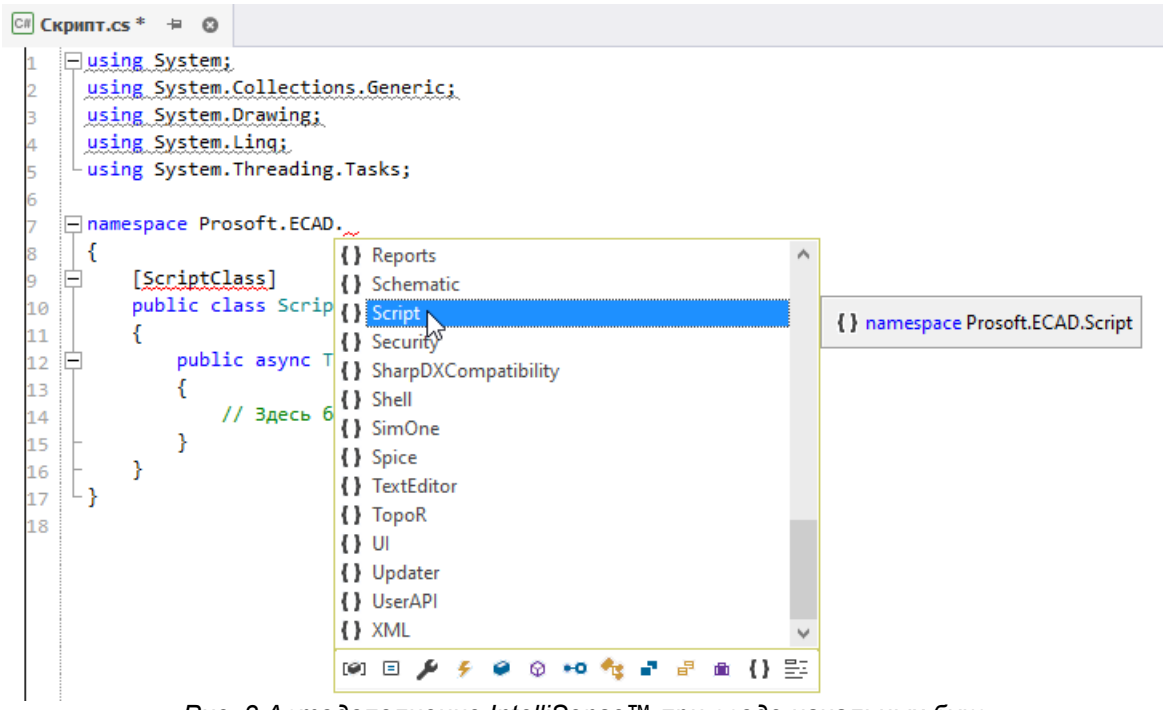


Рис. 3 Автодополнение IntelliSense™ при вводе начальных букв

Работа со скриптом выполняется из панели инструментов «Скрипты», Рис. 4.

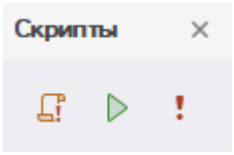


Рис. 4 Панель инструментов для работы со скриптами

Панель содержит следующие инструменты, предназначенные для работы со скриптами:

- Инструмент «Откомпилировать», обозначенный значком , – предназначен для проверки кода скрипта на имеющиеся ошибки;
- Инструмент «Старт», обозначенный значком , – предназначен для запуска кода скрипта;
- Инструмент «Отладить», обозначенный значком !, – предназначен для отладки кода скрипта.

Пример! Главный класс скрипта, выводящий строку «Hello, World!».

```
namespace Prosoft.ECAD.Script
{
    [ScriptClass]
    public class NetlistScript : ScriptBase
    {
        public async Task Main()
        {
            Log.WriteLine("Hello, World!");
        }
    }
}
```

3 Описание класса ScriptBase

Классы скриптов с описанием приведены в Табл. 1.

Таблица 1 Классы скриптов:

Класс скрипта	Описание
`int DelayTime`	Задержка выполнения асинхронной операции в миллисекундах. Использование задержки необходимо для правильной и своевременной реакции системы при выполнении асинхронных функций. Использование значения 0 и очень малых значений не рекомендуется. Значение по умолчанию 100 мс.
`object[] Args`	Аргументы, переданные при запуске скрипта. Только для чтения.
`string ScriptDirectory`	Каталог, из которого запущен скрипт. Только чтение.
`string ScriptFile`	Полный путь к файлу скрипта. Только чтение.
`object Result`	Результат выполнения скрипта. Значение null, если нет результата. Только чтение.
`LogProvider Log`	Журнал. Позволяет выводить сообщения в журнал Delta Design. Только чтение.

Класс скрипта	Описание
`async Task ExecuteScript(string text, string[] args = null)`	Выполняет пользовательский скрипт. Аргумент text – исходный текст скрипта, args – аргументы этого скрипта.
`async Task ExecuteScriptFromFile(string filePath, string[] args = null)`	Выполняет скрипт, содержащийся в файле. Аргумент filePath – файл с исходным текстом скрипта, args – аргументы этого скрипта.
`string SelectProjectDialogOpen()`	Открывает диалоговое окно выбора проекта. Возвращает имя выбранного проекта, либо значение null, если никакой проект не выбран.
`string project SelectBoardDialogOpen()`	Открывает диалоговое окно выбора платы. Возвращает имя выбранного проекта и имя платы в проекте в виде кортежа.
`async Task<Schematic> OpenSchematic(string projectName)`	Открывает схему заданного проекта, аргумент projectName – имя проекта. Возвращает объект для работы со схемой.
`async Task<Pcb> OpenPcb(string projectName, string boardName = null)`	Открывает плату заданного проекта, projectName – имя проекта, boardName – имя платы в проекте. Если имя платы не задано, то берётся плата с именем проекта. Возвращает объект для работы с платой.
`ExportProvider Export(string logFilePath = null)`	Экспортирует данные в различные форматы. Аргумент logFilePath - путь к файлу журнала для вывода сообщений о ходе экспорта. Возвращает объект для экспорта данных.
`ImportProvider Import(string logFilePath = null)`	Импортирует данные из различных форматов. Аргумент logFilePath - путь к файлу журнала для вывода сообщений о ходе импорта. Возвращает объект для импорта данных.
`Script.Comparer Comparer(string logFilePath = null)`	Сравнивает файлы. Аргумент logFilePath - путь к файлу журнала для вывода сообщений о сравнении. Возвращает объект для сравнения файлов.

Класс скрипта	Описание
`Logger GetLogger(string logFilePath)`	Получает содержание журнала. Аргумент logFilePath - путь к файлу журнала. Если значение logFilePath не задано, возвращается журнал Delta Design.
`ScriptClass`	Атрибут обозначает главный класс скрипта, где присутствует точка входа – функция Main(). Данный атрибут не наследуемый.

4 Функции работы со схемой

Для работы со схемой используется объект класса Schematic. Получить объект класса Schematic можно с помощью функции

async Task<Schematic> OpenSchematic(string projectName)

описанной выше в классе ScriptBase.

Пример! Скрипт, открывающий схему проекта

```
namespace Prosoft.ECAD.Script
{
    [ScriptClass]
    public class SchematicOpenScript : ScriptBase
{
    public async Task Main()
    {
        // Открыть диалог выбора проекта
        var projectName = SelectProjectDialogOpen();
        if (string.IsNullOrEmpty(projectName))
        {
            Log.WriteLine("Проект не выбран!");
            return;
        }
        // Открыть схему проекта
        var sch = await OpenSchematic(projectName);
        if (sch == null)
        {
Log.WriteLine(string.Format("Проект '{0}' не найден!",
projectName));
            return;
        }
        Log.WriteLine("Выполнено");
    }
  }
}
```

4.1 Свойства и методы работы со схемой

string Name

название схемы.

string GetAttribute(string name)

Получает значение атрибута схемы, где в данном случае

- name – имя атрибута.

void SetAttribute(string name, string value)

Устанавливает значение атрибута схемы, где в данном случае:

- name – имя атрибута;
- value – значение атрибута.

4.2 Свойства и методы работы с листами схемы

string CurrentPage

название текущего листа схемы.

string[] Pages

массив имён всех листов схемы.

bool IsPageExist(string name)

Определяет, существует ли лист схемы с заданным названием, где:

- name – название листа схемы.

async Task ShowPage(string name)

Показывает заданный лист схемы. Делает заданный лист схемы текущим.

async Task CreatePage(string name, string borderTemplate = null)

Создаёт лист схемы, где:

- name – название листа;
- borderTemplate – название шаблона форматного листа.

```
async Task RenamePage(string oldName, string newName)
```

Переименовывает лист схемы, где:

- oldName – старое (текущее) название листа;
- newName – новое название листа.

```
async Task DeletePage(string name)
```

Удаляет лист схемы.

4.3 Свойства и методы работы с нетлистом (списком соединений)

```
string[] Components
```

массив позиционных обозначений всех компонентов схемы.

```
string[] Nets
```

массив имён цепей.

```
string[] Buses
```

массив имён шин.

```
string[] NetClasses
```

массив имён классов цепей.

```
string[] DiffPairs
```

массив имён дифференциальных пар.

```
ComponentInstanceXO GetComponentInfo(string designator)
```

Получает объект с информацией о компоненте схемы, где:

- designator – позиционное обозначение компонента.

```
NetXO GetNetInfo(string netName)
```

Получает объект с информацией о цепи схемы, где:

- netName – имя цепи.

```
BusXO GetBusInfo(string busName)
```

Получает объект с информацией о шине, где:

- busName – имя шины.

4.4 Методы размещения компонентов, проводников, шин и т.п. на схеме

```
async Task<string> PlaceComponent(string libraryName, string componentName, PointF location, string sheet = null, int gate = 1, string partName = null, string designator = null, int angle = 0, bool flipped = false)
```

Размещает компонент на схеме, где:

- libraryName – имя библиотеки;
- componentName – имя компонента;
- location – координаты точки размещения компонента на схеме;
- sheet – название листа (необязательный параметр). Если не указано, берётся текущий лист;
- gate – номер секции (необязательный параметр). Если не указано, берётся первый;
- partName – название радиодетали (необязательный параметр). Если не указано, берётся первая радиодеталь компонента;
- designator – позиционное обозначение (необязательный параметр). Если значение не указано, генерируется системой автоматически;
- angle – угол поворота, выраженный в градусах. Поддерживаются только значения, кратные 90 градусам. Если значение не указано, компонент размещается без поворота.

Важно! Функция возвращает позиционное обозначение размещённого компонента.

```
async Task<string> PlaceWire(IEnumerable<PointF> points, string net = null, string sheet = null)
```

Размещает проводник на схеме, где:

- points – список точек проводника;
- net – название цепи (необязательный параметр). Если значение не указано, генерируется новое уникальное название цепи;
- sheet - Название листа (необязательный параметр). Если имя листа не указано, берётся текущий лист.

Важно! Функция возвращает название цепи, которой принадлежит размещённый проводник.

```
async Task PlacePowerPort(PointF location, string symbol, string sheet = null, string netName = null)
```

Размещает на схеме порт питания, где:

- location – координаты порта на схеме;
- symbol – название УГО порта;
- sheet – название листа схемы (необязательный параметр). Если название не указано, берётся текущий лист схемы;
- netName – имя цепи (необязательный параметр).

```
async Task PlaceConnectionPort(PointF location, string symbol, string sheet = null, string netName = null)
```

Размещает на схеме порт-соединитель, где:

- location – координаты размещения порта на схеме;

- symbol – название УГО порта;
- sheet – название листа схемы (необязательный параметр). Если название не указано, берётся текущий лист схемы;
- netName – имя цепи (необязательный параметр).

```
async Task<string> PlaceBus(IEnumerable<PointF> points, string name = null, string nets = null, string sheet = null)
```

Размещает шину на схеме, где:

- points – набор координат точек шины на схеме;
- name – название шины (необязательный параметр). Если название не указано, генерируется новое уникальное название шины;
- nets – список/диапазон цепей в шине (необязательный параметр). Если значение не указано, создаётся шина с произвольным набором цепей;
- sheet – название листа схемы (необязательный параметр). Если значение не указано, берётся текущий лист схемы.

4.5 Функции выбора объектов на схеме

```
async Task SelectComponent(string designator, int gate = 1)
```

Выбирает компонент на схеме, где:

- designator – позиционное обозначение компонента на схеме;
- gate – номер секции. Если значение не указано, берётся первая секция компонента.

4.6 Функции прокладки проводника на схеме

```
PointF[] FindWirePath(PointF startPoint, PointF endPoint)
```

Находит путь проводника из точки в точку, где:

- startPoint – начальная точка;
- endPoint – конечная точка.

Важно! Возвращает массив точек, определяющий путь проводника.

```
PointF[] FindWirePath(string pin1, string pin2)
```

Находит путь проводника от вывода компонента до другого вывода, где:

- pin1 – начальная точка (вывод компонента в формате [поз. обозначение компонента]:[номер вывода], например, «DD2:5»).
- pin2 – конечная точка (вывод компонента в формате [поз. обозначение компонента]:[номер вывода], например, «R10:1»).

5 Функции работы с платой

Для работы с платой используется объект класса Pcb. Получить объект класса Pcb можно с помощью функции

async Task<Pcb> OpenPcb(string projectName, string boardName = null),

описанной в классе ScriptBase.

Пример! Скрипт, открывающий плату проекта

```
namespace Prosoft.ECAD.Script
{
    [ScriptClass]
public class PcbOpenScript : ScriptBase
{
    public async Task Main()
    {
        // Открыть диалог выбора проекта
        var projectName = SelectProjectDialogOpen();
        if (string.IsNullOrEmpty(projectName))
        {
            Log.WriteLine("Проект не выбран!");
            return;
        }
        // Открыть плату проекта
        var pcb = await OpenPcb(projectName);
        if (pcb == null)
        {
Log.WriteLine(string.Format("Проект '{0}' не найден!",
projectName));
            return;
        }
        Log.WriteLine("Выполнено");
    }
  }
}
```

5.1 Методы размещения (компонентов, треков, переходных отверстий и т.п.)

async Task PlaceComponent(string designator, PointF location, MountSide mountSide, int angle = 0, Technology technology = Technology.Default)

Размещает компонент на плате, где:

- designator – позиционное обозначение компонента;
- location – координаты точки размещения;

- mountSide – сторона размещения (MountSide.Top – верх или MountSide.Bottom – низ);
- angle – угол поворота (необязательный параметр). Если не указан, берётся значение 0 (без поворота). Угол отсчитывается против часовой стрелки и должен быть кратный 90°;
- technology – технология (плотность) монтажа (необязательный параметр). Если не указана, берётся плотность, заданная для платы.

`async Task PlaceComponentsFromFile()`

Групповое размещение компонентов на плате из файла формата CSV. Функция вызывает диалоговое окно выбора файла.

Пример!

Пример CSV файла

RefDes; X; Y; Mount; Angle

R1; 0; 5; Top; 0

R2; 5; 0; Top; 90

`async Task PlaceTrack(PointF start, IEnumerable<TrackSegment> segments, string net,string layer = null)`

Размещает дорожку на плате, где:

- start – начальная точка;
- segments – набор сегментов дорожки;
- net – имя цепи;
- layer – имя слоя.

`async Task PlaceDiffPair(IEnumerable<DiffPairPart> parts, string diffpair, string layer = null)`

Размещает дорожку дифференциальной пары на плате, где:

- parts – набор участков дорожки дифференциальной пары;
- diffpair – имя диффпары, объединяющей пару цепей;

- layer – имя слоя.

async Task PlaceVia(PointF location, string style = null, string net = null)

Размещает переходное отверстие на плате, где:

- location – координаты отверстия;
- style – стиль переходного отверстия;
- net – имя цепи.

async Task<int> CreateFanouts(string[] args)

Строит фанауты на печатной плате. Параметры фанаутов назначаются через управляющую таблицу в вызывающем скрипте.

Args[0] – ?

Args[1] – ?

Args[2] – назначение BGA класса компонентов (yes/no)

Args[3] – назначение количества пропускаемых внешних рядов BGA матрицы

Args[4] – назначение SMD класса компонентов (yes/no)

Args[5] – назначение направления вывода фанаутов (both/in/out)

Args[6] – назначение монтажного слоя (both/top/bottom)

Args[7] – назначение фильтров имен компонентов. Значение задаётся регулярным выражением. Разрешаются следующие метасимволы:

* – ноль и более любых символов

? – строго один символ

^ – начало строки

$ – конец строки

Значению «**all**» – соответствуют все компоненты.

Args[8] – назначение фильтров имен выводов компонентов. Задаётся аналогично значению args[7].

Args[9] – назначение фильтров имен цепей для выводов компонентов. Задаётся аналогично значению args[7].

Args[10] – назначение типов обрабатываемых цепей («signal» - сигнальные / «powerground» - силовые).

Args[11] – назначение ширины проводника (Nominal/Minimal/Neck или десятичное число в формате A.B)

Args[12] – назначение правила максимального удаления фанаута от соответствующего вывода (десятичное число).

Args[13] – ограничение числа выводов, подключенных к одному фанауту (целое число)

Args[14] – назначение правила фиксации фанаутов (yes/no)

Args[15] – ? (create/delete).

5.2 Функции трассировки

```
async Task<int> ConvertIntoSystemValueAsync(double value)
```

Конвертирует значение, заданное в пользовательских единицах, в системные единицы.

```
async Task<Point> CreatePointAsync(double x, double y)
```

По заданным координатам (в системе координат пользователя) создает точку в системе координат внутренней базы данных проекта. Возвращает сформированную точку.

```
async Task<List<string>> GetNetListAsync(string filter = "*")
```

Извлекает и возвращает описания цепей в символьном формате. Фильтрация цепей выполняется по заданному фильтру.

```
async Task<int> CleanupLayoutAsync(string netsFilter = "*",
string fixedObjects = "")
```

Удаляет дорожки и переходные отверстия в составе цепей, имена которых соответствуют установленному фильтру. Непустая строка второго параметра указывает на необходимость удаления и фиксированных объектов (дорожек, переходных отверстий) в составе обрабатываемых цепей. Возвращает нулевой код возврата в случае успешного выполнения, и положительное число - в случае аварийного завершения.

```
async Task<int> AutoRouteAsync(string[] args)
```

Автоматическая трассировка соединений на печатной плате. Интерпретация семантики передаваемых параметров выполняется в вызываемом методе.

Автоматическая трассировка выполняется последовательного между парами объектов {args[0],args[1]} на слое args[2].

Каждый трассируемый объект задан в строковом формате: component_designator.pad_name

Входными аргументами могут быть также следующие параметры:

- args[3] – разрешение на трассировку проводников под углом 90° (on | off);
- args[4] – разрешенные направления подключения к контактным площадкам (any | axes);
- args[5] – разрешение на расталкивание проложенных проводников при трассировке (on | off);
- args[6] – дотягивается ли проводник до центра контактной площадки (on | off).

Функция возвращает нулевое значение кода возврата в случае успешного завершения трассировки и положительное число - в противном случае.

```
async Task<int> AutoRouteDaisyChainAsync(Point fromPad, Point toPad, Point[] points, string layerName, double width = 0.0, string netName = "UNDEFINED", string rulesChecking = "on", string connDirections = "any", string pushMode = "off", string route90 = "off", string connectToCenter  = "off")
```

Автоматическая трассировка соединений выполняется последовательно для заданного множества точек на печатной плате.

Автоматическая трассировка соединений выполняется последовательно между объектами: fromPad, toPad, где:

- fromPad – точка начала трека. Начальной точкой может быть не обязательно контактная площадка компонента, а, например, via, монтажное отверстие, другой трек и т.д.
- toPad – точка конца трека. Способ задания аналогичен точке fromPad, т.е это не обязательно КП компонента.
- points[ ] – промежуточные точки следования трека (точки, определяющие геометрию трека). Если массив points не задан, трек прокладывается от точки fromPad до toPad.

- netName – имя цепи прокладываемого трека.
- layerName – имя слоя трека.
- width – ширина трека. Если этот параметр не задан, ширина трека определяется автоматически.
- rulesChecking – включена ли проверка нарушений при прокладке трека (on/off). connectionDirections – тип подключения трека (any | [axes | !45 | !wide]).
- pushMode – режим толкания (on/off).
- route90 – разрешен ли поворот треков под 90° (on/off).
- connectToCenter – доводить ли треки до центра КП (on/off).

Важно! Функция возвращает нулевое значение в случае успешного завершения трассировки и положительное число - в противном случае.

5.3 Функции выбора объектов на плате

```
async Task SelectComponent(string designator)
```

Выбирает компонент на плате, где:

- designator – позиционное обозначение компонента.

```
async Task SelectPad(string componentDesignator, string padNumber)
```

Выбирает контактную площадку компонента, где:

- componentDesignator – позиционное обозначение компонента;
- padNumber – номер контактной площадки.

5.4 Функции отображения слоев

```
async Task ShowLayers(params string[] layers)
```

Отображает слои на плате. Имена слоёв отделяются запятыми. Без параметров – все слои.

```
async Task HideLayers(params string[] layers)
```

Скрывает слои на плате. Имена слоёв отделяются запятыми. Без параметров – все слои.

5.5 Функции удаления объектов

```
async Task RemoveComponent(string designator)
```

Удаляет компонент с платы, при этом компонент в нетлисте остаётся, где:

- designator – позиционное обозначение компонента.

5.6 Функции информирования об объектах платы

```
IEnumerable<PcbNetXO> GetNets()
```

Получает набор объектов, содержащих информацию о цепях платы. Каждый объект содержит информацию об одной цепи.

```
ComponentInfo GetComponentInfo(string designator)
```

Получает информацию о компоненте на плате, где:

- designator – позиционное обозначение компонента.

```
List<ComponentInfo> GetComponents()
```

Получает список объектов, содержащих информацию о компонентах платы. Каждый объект содержит информацию об одном компоненте.

```
PcbNetXO GetPcbNetInfo(string netName)
```

Получает информацию о цепи, где:

- netName – имя цепи.

```
PcbDiffPairXO GetPcbDiffPairInfo(string name)
```

Получает информацию о дифференциальной паре, где:

- name – имя дифференциальной пары.

6 Функции импорта данных

async Task<string> ImportDDL(string ddlFile)

Загружает библиотеку из файла .ddl.

- ddlFile – имя и адрес файла библиотеки.

Возвращает имя загруженной библиотеки.

async Task<string> ImportDDC(string ddcFile)

Загружает проект печатной платы из файла .ddc.

- ddcFile – имя и адрес (полный путь) файла проекта.

Возвращает имя загруженного проекта.

async Task<string> ImportStandards(string ddsFile)

Импортирует стандарты DeltaDesign из файла.

- ddsFile – файл стандартов.
-

async Task<string> ImportPCAD(string schFile, string pcbFile, string projectName, string settingFile)

- schFile – имя ASCII файла схемы PCAD (.sch);
- pcbFile – имя ASCII файла платы PCAD (.pcb);
- projectName – имя проекта Delta Design;
- settingFile – файл настройки импорта (соответствия атрибутов, семейств и слоёв .ims).

Возвращает имя импортированного проекта.

7 Функции экспорта данных

async Task ExportTopoR(string projectName, string boardName, string fstFile)

Выгружает проект печатной платы в TopoR (создает файл .fst), где:

- projectName – имя выгружаемого проекта;
- boardName – имя платы;
- fstFile – имя и адрес файла fst.

async Task ExportBoardToDXF(string projectName, string boardName, string dxfFile)

Выгружает проект печатной платы в DXF (создает файл .dxf), где:

- projectName – имя проекта, плату из которого нужно выгружать;
- boardName – имя платы;
- dxfFile – имя и адрес файла .dxf.

async Task ExportDDC(string projectName, string ddcFile)

Выгружает проект печатной платы в XML (создает файл .ddc), где:

- projectName – имя выгружаемого проекта;
- ddcFile – имя и адрес файла .ddc.

async Task ExportStandards(string ddsFile)

Экспортирует стандарты DeltaDesign, где:

- ddsFile – имя файла экспорта стандартов.

async Task ExportIDF(string projectName, string boardName, string brdFile, string libFile)

Экспортирует проект в IDF (3d), где:

- projectName – имя выгружаемого проекта;
- boardName – имя платы;
- brdFile – имя и адрес файла платы (.brd);

- libFile – имя и адрес файла библиотеки (.lib).

`async Task ExportGerber(string projectName, string boardName, string folder, params string[] layerList)`

Создает файлы производства (Gerber и Drill) для проекта в указанной директории. Единицы измерения - мм., где:

- projectName – имя экспортируемого проекта;
- boardName – имя платы;
- folder – каталог, где будут созданы файлы Gerber и Drill;
- layerList – список слоёв.

`async Task ExportODBpp(string projectName, string boardName, string folder)`

Создает файлы производства (ODB++), где:

- projectName – имя экспортируемого проекта;
- boardName – имя платы;
- folder – каталог, в котором будут созданы файлы.

`async Task ExportDRC(string projectName, string boardName, string drcFile)`

Выполняет проверку нарушений данного проекта печатной платы, где:

- projectName – имя экспортируемого проекта;
- boardName – имя платы;
- drcFile – имя и адрес файла отчета о нарушениях.

Создаёт текстовый файл с перечислением ошибок.

Компания ЭРЕМЕКС поставила своей задачей создать точную и удобную систему, предназначенную для создания комплексной среды сквозного проектирования электронных устройств, которой и стала система Delta Design.

Мы постарались учесть все возможные алгоритмы и пути решения задач, которые может поставить перед собой наш пользователь, заложив в систему Delta Design наибольшее количество опций, логических ходов, надстроек, расширенный функционал и т.д.

Компания ЭРЕМЕКС благодарит Вас за приобретение системы Delta Design и надеется, что она станет удобным и полезным инструментом в Вашей деятельности.