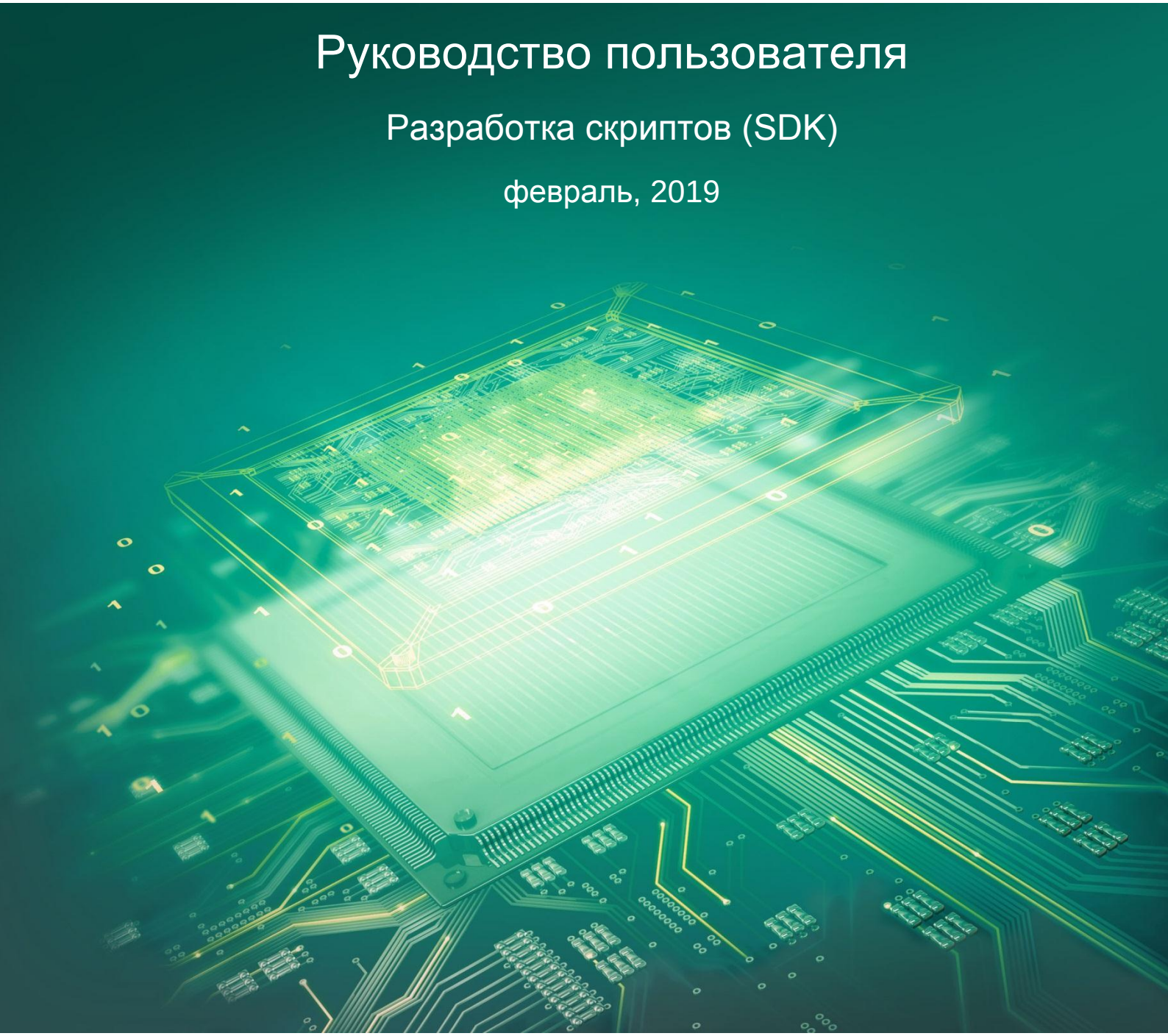
DeltaDesign

Комплексная среда сквозного проектирования
электронных устройств

Руководство пользователя

Разработка скриптов (SDK)

февраль, 2019



ЕREMEX

Внимание!

Последнюю версию документа можно получить в сети Интернет по адресу: https://www.eremex.ru/knowleage-base/delta-design/docs/.

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

Форум «ЭРЕМЕКС»: http://forum.eremex.ru/

Техническая поддержка

E-mail: support@eremex.ru

Skype: deltadesign_support (Delta Design), supporteremex (TopoR)

Отдел продаж

Тел. +7 (495) 232-18-64

E-mail: info@eremex.ru

E-mail: sales@eremex.ru

Содержание

1 Введение

1.1 Добро пожаловать

Система Delta Design является универсальным инструментом разработки электронных устройств, объединяющим различные средства автоматизированного проектирования.

Функционал системы Delta Design обеспечивает сквозной цикл проектирования электронных устройств:

- Формирование базы данных радиоэлектронных компонентов, ее сопровождение и поддержание в актуальном состоянии
- Разработка электрических схем
- SPICE-моделирование работы аналоговых устройств
- Разработка конструкции печатных плат
- Размещение компонентов и проведение полуавтоматической и автоматической трассировки печатных плат
- Выпуск конструкторской документации (в соответствии со стандартами)
- Выпуск производственной документации, в том числе для автоматизированных производственных линий
- Подготовка данных для составления перечня закупаемых изделий и материалов, необходимых для реализации проекта

1.2 Требования к аппаратным и программным средствам

Примечание. Требования к программным и аппаратным средствам, а также инструкции по установке и техническому обслуживанию Delta Design приводятся в документе «Администрирование системы», доступного по адресу:

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

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

- Microsoft Windows 7 SP1 (KB976932)
- Microsoft Windows 8.1
- Microsoft Windows 10

Также на компьютере должны быть установлены следующие программные средства:

- Platform Update Patch (KB2670838) (для Microsoft Windows 7)

- DirectX 11

  Минимальная конфигурация аппаратных средств:

- Процессор Core i5 - 2,4ГГц
- Оперативная память - 2ГБ
- Видеокарта с поддержкой DirectX 11

1.3 Техническая поддержка и сопровождение

ВАЖНО! Техническая поддержка оказывается только пользователям, прошедшим курс обучения. Подробные сведения о курсе обучения могут быть получены по адресу в интернете: https://www.eremex.ru/learning-center/

При возникновении каких-либо проблем с эксплуатацией Delta Design рекомендуем следующую последовательность действий.

- Обратитесь к документации по системе и попробуйте найти сведения о решении возникшей задачи.
- Ознакомьтесь с базой знаний продукта Delta Design, которая, в частности, содержит ответы на часто возникающие у пользователей вопросы.

  Адрес базы знаний продукта Delta Design:

  https://www.eremex.ru/knowleage-base/

- Вы также можете найти ответ или задать свой вопрос на форуме пользователей.

  Форум пользователей Delta Design располагается по адресу:

  http://forum.eremex.ru/forum/17-delta-design/

- Если указанные источники не содержат рекомендаций по возникшей проблеме, прибегните к услугам технического персонала вашего поставщика программных продуктов компании (дилера ЭРЕМЕКС).
- В том случае, если специалисты поставщика не смогли помочь в разрешении проблемы, свяжитесь непосредственно с офисом ЭРЕМЕКС.

Перед обращением, пожалуйста, подготовьте подробную информацию о возникшей ситуации и Ваших действиях, приведших к ней, а также информацию о конфигурации используемого компьютера и периферийного оборудования.

Служба технической поддержки принимает обращения по электронному адресу: support@eremex.ru

Примечание. При возникновении проблем с установкой и/или запуском системы убедитесь, что инструкции, описанные в документе «Администрирование системы», точно выполнены.

Документ доступен по адресу:

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

1.4 Рекомендации по использованию документации

Документация по системе представлена в виде справочника, задача которого состоит в иллюстрации работы системы в объеме, достаточном для эффективного использования. Некоторые фрагменты текста могут не иметь абсолютной полноты описания, или не иметь абсолютного соответствия с какой-либо версией программы. Тем не менее, приведенные описания дают достаточную информацию о работе функционала программы.

Вопросы, о предпочтительной методике использования системы (Почему надо использовать тот или иной функционал? Как организовать проектирование в соответствии со стандартом? и т.д.) рассматриваются только в рамках учебных курсов. Информация об учебных курсах доступна по адресу в интернете:

https://www.eremex.ru/learning-center/

2 Разработка скриптов Delta Design

2.1 Общая информация о скриптах

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

2.1.1 Пример. Главный класс скрипта

Пример 1. Главный класс скрипта, выводящий строку «Hello, World!»

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

2.2 Описание класса ScriptBase

`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 выводит сообщения в журнал Delta Design. Только чтение.

`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.

`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)`

Открывает плату заданного проекта, projectName – имя 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.

`Logger GetLogger(string logFilePath)`

Получает содержание журнала. Аргумент logFilePath - путь к файлу журнала. Если значение logFilePath не задано, возвращается журнал Delta Design.

`Атрибут ScriptClass обозначает главный класс скрипта, где присутствует точка входа – функция Main(). Данный атрибут не наследуемый.`

2.3 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.

2.3.1 Пример скрипта, открывающего схему проекта

Пример 2. Открытие схемы проекта

```
[ScriptClass]
public class SchematicOpenScript : ScriptBase
{
	public async Task Main()
	{
		// Открыть схему проекта
		var sch = await OpenSchematic("ddBox");
		if (sch == null)
		{
			Log.WriteLine(string.Format("Проект '{0}'
			не найден!", "ddBox"));
			return;
		}
		Log.WriteLine("");
	}
}
```

2.3.2 Свойства и методы работы со схемой

`string Name`

название схемы.

`string GetAttribute(string name)`

Получает значение атрибута схемы. Здесь name – имя атрибута.

`void SetAttribute(string name, string value)`

Устанавливает значение атрибута схемы.

name – имя атрибута

value – значение атрибута

2.3.3 Свойства и методы работы с листами

`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

name – название листа схемы.

`async Task ShowPage(string name)`

Показывает заданный лист схемы. Делает заданный лист текущим

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

Создаёт лист схемы.

name – название листа
borderTemplate – название форматки листа.

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

Переименовывает лист схемы.

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

`async Task DeletePage(string name)`

Удаляет лист схемы.

name – название листа.

2.3.4 Свойства и методы работы с нетлистом

`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.

designator - позиционное обозначение компонента.

`NetXO GetNetInfo(string netName)`

Получает объект с информацией о цепи схемы.

netName – имя цепи.

`BusXO GetBusInfo(string busName)`

Получает объект с информацией о шине.

busName – имя шины.

2.3.5 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.

libraryName - Имя библиотеки

componentName - Имя компонента

location - Координаты точки размещения компонента на схеме

sheet - Название листа (необязательный параметр). Если не указано, то берётся текущий лист.

gate - Номер секции (необязательный параметр). Если не указано, то берётся первый.

partName - Название радиодетали (необязательный параметр). Если не указано, то берётся первая радиодеталь компонента.

designator - Позиционное обозначение (необязательный параметр). Если значение не указано, то оно автогенерируется системой.

angle - Угол поворота выраженный в градусах. Поддерживаются только значения кратные 90 градусам. Если значение не указано, то компонент вставляется без поворота.

Функция возвращает позиционное обозначение размещённого компонента.

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

Размещает проводник на схеме

points – список точек проводника

net - Название цепи (необязательный параметр). Если значение не указано, то генерируется новое уникальное название цепи.

sheet - Название листа (необязательный параметр). Если имя листа не указано, то берётся текущий лист.

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.

location - Координаты порта на схеме

symbol - Название УГО порта

sheet - Название листа схемы (необязательный параметр). Если название не указано, то 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.

location – Координаты размещения порта на схеме

symbol - Название УГО порта

sheet - Название листа схемы (необязательный параметр). Если название не указано, то 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.

points – набор координат точек шины на схеме

name - Название шины (необязательный параметр). Если название не указано, то генерируется новое уникальное название шины.

nets - Список/диапазон цепей в шине (необязательный параметр). Если значение не указано, то создаётся шина с произвольным набором цепей.

sheet - Название листа схемы (необязательный параметр). Если значение не указано, то берётся текущий лист схемы.

2.3.6 Функции выбора объектов на схеме

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

Выбирает компонент на схеме.

designator - позиционное обозначение компонента на схеме.

gate - номер секции. Если значение не указано, то берётся первая секция компонента.

2.3.7 Функции прокладки проводника на схеме

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

Находит путь проводника из точки в точку.

startPoint - начальная точка.

endPoint - конечная точка.

Возвращает массив точек, определяющий путь проводника.

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

Находит путь проводника из вывода компонента в другой вывод.

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».

2.4 Функции для 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),
```

которая описана выше в классе ScriptBase.

2.4.1 Пример открытия платы

Пример 3. Открытие платы проекта

```
[ScriptClass]
public class PcbOpenScript : ScriptBase
{
	public async Task Main()
	{

		// Открыть плату
		var pcb = await OpenPcb("ddBox");
		if (pcb == null)
		{
			Log.WriteLine(string.Format("Проект '{0}' 
			не найден!", "ddBox"));
			return;
		}
	}
}
```

2.4.2 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.

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, то берётся значение 0 (без поворота). Угол отсчитывается против часовой стрелки и должен быть кратен 90°

technology - Технология (плотность) монтажа (необязательный параметр). Если не указана, то берётся плотность, заданная для платы.

async Task PlaceComponentsFromFile()

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

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.

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.

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.

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] - назначение правил совместного использования фанаута несколькими выводами (целое число)

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

Args[15] - ? (create/delete).

2.4.3 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 последовательно для заданного множества точек на печатной плате.

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

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. То есть это точки, в которых геометрия трека фиксируется. 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, то 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).

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

2.4.4 Функции выбора объектов на плате

`async Task SelectComponent(string designator)`

Выбирает компонент на плате.

designator – позиционное обозначение компонента.

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

Выбирает контактную площадку компонента.

componentDesignator - позиционное обозначение компонента

padNumber - номер контактной площадки.

2.4.5 Функции отображения слоёв

`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.

2.4.6 Функции удаления объектов

`async Task RemoveComponent(string designator)`

Удаляет компонент с платы, при этом компонент в нетлисте остаётся.

designator – позиционное обозначение компонента.

2.4.7 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.

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.

netName – имя цепи.

`PcbDiffPairXO GetPcbDiffPairInfo(string name)`

Получает информацию о дифференциальной паре.

name – имя дифференциальной пары.

2.5 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.

2.6 Функции экспорта данных

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

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

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).

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).

projectName - имя выгружаемого проекта.

ddcFile – имя и адрес файла .ddc.

async Task ExportStandards(string ddsFile)

Экспортирует стандарты DeltaDesign.

ddsFile - имя файла экспорта стандартов.

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

Экспортирует проект в IDF (3d).

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.

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++).

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. Создаёт текстовый файл с перечислением ошибок.

projectName - имя экспортируемого проекта

boardName - имя платы

drcFile – имя и адрес файла отчета о нарушениях