Nemerle за 20 часов?

Тыкаем ЭТО палочкой

«Nemerle – не нужен!»
Общественное мнение

Час 1

Установил nemerle в VS. Открыл 1-ю статью из цикла «Язык Nemerle». Первые 20% статьи – вода и какая-то кустарщина, недостойная даже для студента-шарпера. :) Ок, бегло добрался до HelloWorld

```
using System.Console;

WriteLine("Hello, World!");
```

```
using System.Console;

WriteLine("Hello, World!");
WriteLine("Привет, Мир!");
WriteLine("Мир! Привет!")
```

Паскаль – паскалем. Но студенты-шарпеи, уверен, этого не знают :) Втыкаю примеры в студию... Первое МЛЯ:

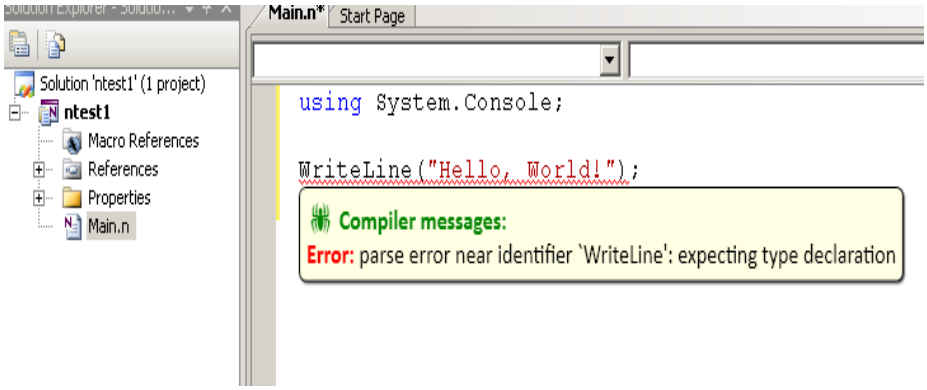

Рис.1
Слабо написать «Привет, мир» без ошибок???

НО КОМПИЛИРУЕТСЯ!!! Ок, едем далее. Чуть не пропустил фразочку: «Последнюю точку с запятой можно смело не указывать». Сходство с паскалем становится более очевидным :)

Продолжил читать статью, на этот раз – до конца. Примеры сопоставления с образцом, функции (напомнили мне ЯП Ruby), рекурсия, макросы (вспомнил LISP)... Самое важное ощущение – автор попытался влепить все возможности языка в одну статью. В итоге, ни об одной возможности яснее не стало. То есть, получился «Nemerle in examples» - достаточно популярный стиль изложения (я лично такое читал и для того же Ruby). Так и пролетел первый час знакомства с этим замечательным ЯП. :)

Час 2 и 3

Второй час изучения Nemerle у меня был посвящен чтению второй и третьей частей цикла статей Влада «Язык Nemerle» в поисках ~~ветхо~~заветного ООП. Не нашел. Конечно, тема калькулятора довольно познавательна с точки зрения авторов «книги Дракона», — как по мне, более всего материал был похож на введение в OCaml, — но дело в том, что сегодня целый день зудели руки броситься в бой, и перекромсать свою полезную библиотечку WXLib в нечто более удобоваримое:

Пример славного монстрокода из библиотечки

C#
`List<Tuple5<A, B, C, D, E>> Zip<A, B, C, D, E>(IList<A> a, IList<B> b, IList<C> c, IList<D> d, IList<E> e)`
Nemerle
???

Не судьба? А вот и нет — Влад настрочил еще статей, одна из которых имеет невзрачное название «Nemerle» — а там, вроде как, нашлись и классы, и интерфейсы, и каменный цветок, и золотой ключик. По крайней мере, мало-мальски минимальное описание классов и дженериков там нашлось. В первом приближении, этой статьей можно пользоваться как справочным пособием. Эх, реально не судьба — почти два часа прошло. Сворачиваюсь, но напоследок проверяю, действительно ли тип «void» (в статьях это словосочетание довольно часто проскакивает) является обыкновенным типом:

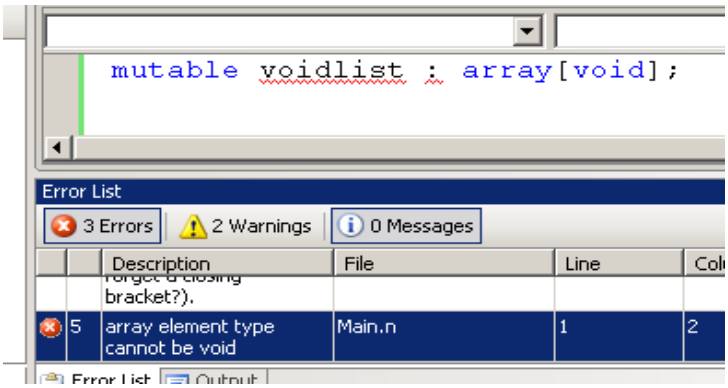

Рис.2
Сказки с наскока не получится.

Один воздушный замок растаял. Ну и хрен с ним. Потом еще будем посмотреть, рухнет ли второй воздушный замок — превращение монстрокода в микрокод за счет обобщения :)

Хватит читать

«-Выходи, чудище, из пещеры! Драться будем!
-Драться, так драться... Но зачем же в жопу кричать?!»
Народный фольклор

Часы 4 — 13

Составив на основе прочитанных четырех статей какое-то суждение о языке, пора приступить к ~~наказанию искусством~~ испытаниям практикой. Задач сегодня — две:

1. Что-то сделать с ключевым словом «void»;
2. Создавать кортежи нужного мне размера, не ограниченного рамками очередного дотнетика.

Отмечу особо — мне сейчас слегонца ~~насрать~~ плевать на то, велосипед ли я делаю, или нет. Плюс — все решения должны быть хотя бы призрачно совместимы с .NET 2.0. Я охотно верю, что в стране эльфов давно пользуются четвертым дотнетом, но в моей суровой реальности до сих пор очень желательна совместимость со вторым. :)

Итак, задача №1: сконвертировать процедурку в функцию. Да не просто а попутно создать соответствующие функциональные типы, чтобы можно было (желательно) одним методом конвертить любые void-методы в функции, возвращающие какой-либо «пустой» тип. Максимум, чего удалось добиться в лучшем из шарпов (где-то час мудохался в четвертом, зато в #Develop), было нечто вроде:

```
// Аналог .NET 3+: struct Void
public sealed class VoidType {
	private static VoidType _instance = null;
	public static VoidType Instance { get { if(_instance == null) _instance = new VoidType();
return _instance;}}
	private VoidType(){}
}
// Аналог .NET 3+: System.Action
public delegate void P();
public delegate void P<T1>(T1 a1);
public delegate void P<T1, T2>(T1 a1, T2 a2);
...

// Аналог .NET 3+: System.Func
public delegate VoidType F();
public delegate VoidType F<T1>(T1 a1);
public delegate VoidType F<T1, T2>(T1 a1, T2 a2);
...
// Наверняка в дотнете есть и писанина, аналогичная этой...
public static class ProcUtil {
	public static F Method2Func(P x){ return (()=>{x(); return VoidType.Instance;}); }
	public static F<T1> Method2Func<T1>(P<T1> x){ return (a1)=>{x(a1); return
VoidType.Instance;}; }
	public static F<T1, T2> Method2Func<T1, T2>(P<T1, T2> x){ return (a1, a2)=>{x(a1, a2);
return VoidType.Instance;}; }
	...
}
```

Прикольно — для **N** формальных параметров метода будем иметь **3N** строк кода в общем случае. Программируем «елочкой» - быстро и эффективно :). Теперь начинаю думать задачу в терминах Nemerle. В функциональные и типовыводящие фишки языка мне что-то не верится — сильно подозреваю что будут те же яйца в плане эффективности кодирования, что и в C#, поэтому даже пробовать не буду, а сразу испробую макросы. Логично? Для постороннего наблюдателя —

наверняка нет. Для меня — да. Глазом видно, что без средств типа шаблонов с переменным кол-вом аргументов а-ля C++ задачка тупо нерешаема.

После ~~40 минут~~ более чем двух часов борьбы с компилятором по различным поводам, чтения кусков «Макросы Nemerle — расширенный курс», вопроса на форум (по состоянию на полдень следующего дня — ~~БЕЗОТВЕТНОГО~~ какой-то совет буквально минут через 20 после первого примечания получил — прим.) в муках ничего не родилось. **EPIC FAIL:**

```
public struct VoidType {} // Кстати, да — что-то и здесь какую-то лажу с Singleton испытал :)

public delegate P() : void;
public delegate P[T1](a1: T1) : void;
public delegate P[T1, T2](a1: T1, a2: T2) : void;

public delegate F(): VoidType;
public delegate F[T1](a1: T1): VoidType;
public delegate F[T1, T2](a1: T1, a2: T2): VoidType;

macro Method2Func(method: expr, s : list[expr]) {
    <[
        ???
    ]>
}
```

Завтра буду читать «...расщиренный курс».

Час 14 — 23

Таки приходится читать цикл «Макросы Nemerle — расширенный курс», и пробую заодно переформулировать вчерашнюю задачу. Раскинув своей извилиной, соображаю, что Method2Func я, может, и построю. А что делать с типами? Кака-с. Значит, либо макро должно генерировать И декларацию типов, И метод их преобразования.

Влад в скайпе молчит (потом ответил прямо на форум — прим.), помощь форума оказалась занимательной (отправили в исходники Nemerle :)), но не особо конструктивной. Начинаю читать с сайта материалы отцов-основателей... Прочел. И вы знаете? Помогло! Макросы, уж не знаю — почему, **НАДО ДЕРЖАТЬ ОТДЕЛЬНО ОТ КОДА**. В соседней сборке, всего лишь. Но, по итогам, после всей этой ебли с плясками, меняю мнение касательно помощи — отсылка (в конкретное место) к исходникам Nemerle оказалась, наверное, самой полезной. И всё равно ничего не выходит. С макросами снова FAIL.

Отдаем Москву французам

«-Ну, что Данило-мастер, не вышла твоя дурман-чаша?
- Не вышла, отвечает.
- А ты не вешай голову-то! Другое попытай. Камень тебе будет, по твоим мыслям.
-Нет, - отвечает, - не могу больше. Измаялся весь, не выходит. Покажи каменный цветок.
- Показать-то, - говорит, - просто, да потом жалеть будешь.»
Хозяйка Медной горы

Час 24

Спросить Влада или кого-то с форума немерлистов — легко! Да вот цель моя — не спросить, да и владов на всех желающих не напасёшься. В общем, ситуация пока даже хуже чем с монадами хаскеля — по монадам хоть литературы завались, а вот с чем кушать немерловый API компилятора — днем с огнем не отыщешь, на крайняк, на официальном сайте и в статьях ничего нет. Начинаю эксперименты, которые помогли бы понять кишки компилятора.

Час 30

Собственно, начал эксперименты. И все ради первой задачи, которая неуловимо скатилась к банальному созданию типа делегата :)

1. Простейший макрос

```
public macro M01(name, expr)
{
  <[ def $name = $expr ]>
}
```

С ним получилось узнать, что рефлектор при попытке перевода сборки в C# падает (а вот в VB.NET или Delphi — нет), и использовать макросборки, например, в C# как минимум, непрактично — другие дотнетовые ЯП видят в сборке класс вида «**_N_operator_508854937_2393Macro**», что, с учетом того, что класс — «**NotInheritable**», достаточно печально.

Час 31 — 40

Перечитал «Defining types from inside macros» + соотв. типы в том, что громко зовется «class library documentation». Какой результат — можно не спрашивать: зачем генерировать документацию на непокрытый описаниями код?..

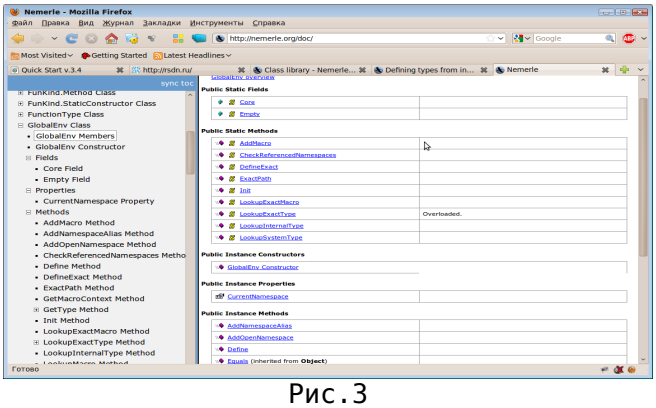

Рис.3

Пытаюсь прикинуть по названиям, что это за штучка – TypeBuilder, и как она может мне помочь. Параллельно, для поколений грядущих, строю в Dia классы, посещаемые при обходе, этой, так сказать, документации:

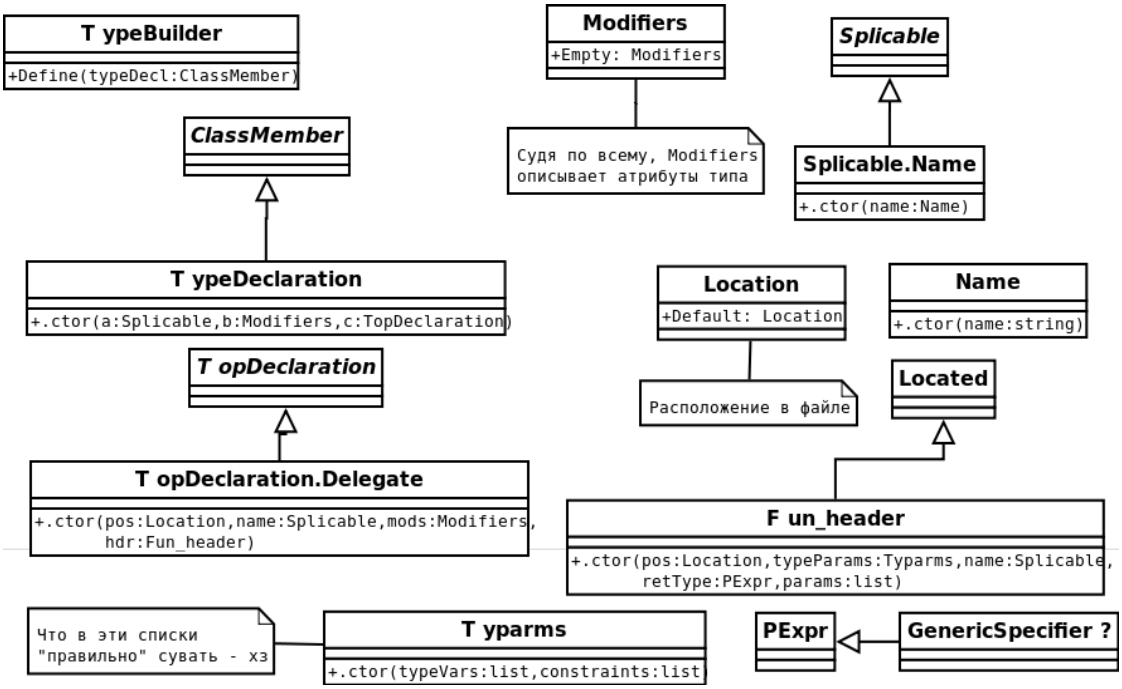

Рис.4
Кишочки. Вид снаружи.

В итоге ~~чтения документации~~ гадания на кофейной гуще, рождается следующий код:

2. Макрос генерации делегата, попытка №2

```
public macro DefDelegate(dname: string, n: int) {
    def context = ImplicitCTX();
    def builder = context.CurrentTypeBuilder;
    def env = context.Env;
    def location = builder.Location;
    mutable args: list[Splicable] = [];

    for(mutable j : int = 0; j<n; j++)
        args = Splicable.Name(Name(string.Format("T{0}", j))) :: args;

    def d = TypeDeclaration(
        Name(dname), // ???
        Modifiers(NemerleAttributes.Public, []),
        TopDeclaration.Delegate(
            location,
            Name(dname),
            Modifiers(NemerleAttributes.Public, []),
            Fun_header( // конструктор выбрал практически наугад
                location,
                Typarms(args, []),
                Name(dname), // шо, опять???
                Nemerle.Compiler.Parsetree.PExpr.Typed(
                    Nemerle.Compiler.Typedtree.TExpr.TypeOf(
                        Nemerle.Compiler.MType.Void()
                    )),
                []
            )
        ));
```

```
    builder.Define(d);
    builder.Compile();

    <[ () ]>
}
```

И сразу же после написания этого кода выясняется, что доки на сайте мало того, что пустые, так еще и **устарели**!!! Честно говоря, после такого открытия желание грызть ~~язык~~ кактус дальше практически пропало. На гуглокоде стало ясно что док нет.
Судя по ObjectBrowser (о, блядь, справка...) Fun_header ВНЕЗАПНО стал PfunHeader... Короче, код стал таким:

2. Макрос генерации делегата, попытка №3

```
public macro DefDelegate(dname: string, n: int) {
    def context = ImplicitCTX();
    def builder = context.CurrentTypeBuilder;
    def env = context.Env;
    def location = builder.Location;
    mutable args: list[Splicable] = [];

    for(mutable j : int = 0; j<n; j++)
        args = Splicable.Name(Name(string.Format("T{0}", j))) :: args;

    def d = ClassMember.TypeDeclaration(
        Splicable.Name(Name(dname)), // ???
        Modifiers(NemerleAttributes.Public, []),
        TopDeclaration.Delegate(
            PFunHeader(
                location,
                Typarms(args, []),
                Splicable.Name(Name(dname)),
                PExpr.Typed( TExpr.TypeOf( FixedType.Void() ) ),
                []
            )
        ));

    builder.Define(d);
    builder.Compile();
    <[ () ]>
}
```

Это хоть компилируется... Но я надежды почти не питаю :) Хотя на голом шарпе создать динамически подобную штучку почти несложно, но как-то все еще хотелось бы, чтоб на стадии компиляции Nemerle мне бы нагенерил всю ту «ёлочку», описанную ещё на 4-м часу изучения этого ЯП... Пробую запихнуть в тестовую консольную прожку... Build... ICE. Не айс. Забиваю до тех лучших времен, когда не надо будет ползать по этим пещерам без фонаря и прочего огня.

Вместо послесловия

> «Большинство тех, кто разобрался в языке, бросают свои проекты и начинают принимать участие в разработке компилятора, просто злой рок какой-то»
> Кочетков

...Теперь я даже знаю — почему :) На самом же деле все обстоит вовсе не так плохо, как я тут расписал — Nemerle **уже** умеет всё, что только можно сделать на C#, и даже больше. Тем более, что на это «**все**» я потратил **всего пару-тройку часов**. ~~Зато остальные 17 протрахался с одним единственным макросом :)~~

```
// Зачем писать уродливый код так...
void Select<A>(FuncX<A, bool> selector, params Tuple2<A, Block>[] actions)
{
    WXUtils.DoLoopOver<Tuple2<A, Block>>(actions, delegate(Tuple2<A, Block> a)
    {
        if(selector(a.Value1))
        {
            a.Value2();
        }
    });
}
...
Select<Type>( x => x==someType,
  FP.Tuple(typeof(Array), ()=>{...},
  FP.Tuple(typeof(StringBuilder), ()=>{...},
  FP.Tuple(typeof(SomeClass), ()=>{...},
  ...
)
```

```
// Если можно так:
match( x ) {
  | x is Array => {...}
  | x is StringBuilder => {...}
  | x is SomeClass => {...}
  ...
}
```

Всё!