# The dylan.NET Manual v.11.2.2

by Dylan Borg

October 30, 2010

To those who taught me,to my mum, sister and father.

# Contents

# 1 The Compiler

## 1.1 Introduction

This chapter of the manual will speak about the inner workings of the new dylan.NET compiler. For the language syntax look further down this manual. The need for a chapter like this has risen becuase of the new API nature of dylan.NET i.e. now dylan.NET is split in class libraries each doing a step that transforms a basic form of representation into a more complex form. The compiler's work is just that, converting source code written with a basic text editor into an AST, then into MSIL/CIL that is compatible with .NET 3.5 SP1 or higher and Novell Mono 2.6.7 or higher.

Since Mono is the least common denominator between the two frameworks, its libraries shall be used in building the compiler. Mono is also cross-paltform (i.e. works on Windows, Linux and Macintosh OS/X), hence the need to be compatible with it. Go to the Mono website [1] for more info.

## 1.2 The dylan.NET API

The API is split into 4 libraries. These are (a * means that the library is incomplete or not available yet):

**tokenizer.AST.dll** Contains all the AST components such as Tokens, Expression, Statements etc. defined in the dylan.NET language. The other libraries make heavy use of this library.

**tokenizer.Lexer.dll** Contains the Lexer components that can turn a dylan.NET file into statements and tokens.

**tokenizer.Parser.dll*** Contains the Parser components responsible for the optimization of statements. It can recognize the type of statements and tokens.

**tokenizer.CodeGen.dll*** Contains the components that turn the AST into MSIL/CIL code. (Still to be written.)

The version number for all assemblies should match for a given dylan.NET distro. The program **dnc.exe** wraps the 4 libraries and is the main compiler executable. It also is

[1] http://www.mono-project.com contains info about Mono as well as downloads for Windows, Mac and Linux.

an example for the use of the libraries. The libraries and their sources are available from Gitorious. [2]

### 1.2.1 AST

The AST or as I call it, the festival of inheritance contains all dylan.NET language components. The root namespace is dylan.NET.Tokenizer.AST. All classes derive from one of the following classes:

**Token** A standard dylan.NET token comprising an identifier, literal, operator etc. All tokens inherit from this class.

**Expr** A standard dylan.NET expression from which all expressions are derived.

**Stmt** A standard statement from which all other statements are derived. These can be collected in an **StmtSet**.

### 1.2.2 Lexer

The Lexer is what takes all text source files, splits them into lines drom which it makes statements and then splits each line into tokens which it puts inside the corresponding statement. It then store the set of statements into a statement set for handing over to the Parser. The dylan.NET lexer is generally string and character aware i.e. it will not split the token stream when inside a character (e.g. 'c') or when inside a string (e.g. "This is a string"). The spaces in the string used before will not be used to split the string into tokens as the lexer knows that it is a string literal. The lexer also has an ingenious system for recognizing operators that are multi-character such as ++,>=,!=,–, etc. The root namespace is dylan.NET.Tokenizer.Lexer.

### 1.2.3 Parser

The Parser os what takes the **StmtSet** made by the Lexer and transforms it into the specific statements containing specific tokens. etc. The decision is done based on the textual value of the tokens inside the statements. For example a token whose **Value** field says "object" gets converted into an **ObjectTok** which inherits the class **TypeTok** which in turn inherits **Token**. The casting from one type to another is done in a specific fashion and not using the defaukt .NET casting system which is not able to do all the casts needed. During these casts the **new** operator is used a extensively to instantiate the new optimized token, statements etc. and then assignments are used to transfer the information inside the old class into the new class. The root namespace is dylan.NET.Tokenizer.Parser.

[2] http://gitorious.org/dylandotnet/dylandotnet has the latest sources inside the git repo.

### 1.2.4 CodeGen

More on this after the library is actually written. The root namespace shall be dylan.NET.Tokenizer.CodeGen.

# 2 Other Libraries

## 2.1 Introduction

Like other programming languages dylan.NET defines its own specific libraries. The main utility library is **dnu.dll** which contains certain functions helping the dylan.NET programmer. Since the new compiler is written in dylan.NET itself i.e. it is self-hosting it makes use of dnu.dll. This means it has to be built before the compiler if rebuilding the toolset from source. All these libraries below are written in dylan.NET demonstrating that dylan.NET can make great libraries like C# can!

## 2.2 dnu - dylan.NET Utility

This library provides certain constants such as pi,crlf,cr,lf,e etc. One may say, "But .NET already has all that stuff". Yes it is true. But for now dylan.NET cannot create nor access literal fields which means the .NET ones are useless. That is why dnu defines readonly field versions of these constants. In the future this class may get deprecated when compile time constants may be created and used from dylan.NET. It is also useful to know that certain functions in dnu might be original ones and can be needed from C# or VB.NET. The root namespace is dylan.NET.Utils.

## 2.3 sld - SQLite Data

This library provides an easy way to use SQLite database connections. New datatypes storable in SQL databases can be defined and used as file-formats. The main external data formats for dylan.NET are the SQLite database and the XML file. For XML .NET gives us the required abstractions but for SQLite it does not so we need wrapper libraries such as **Mono.Data.Sqlite.dll** and a library to abstract the wrapper such as **sld.dll**.

## 2.4 Others...

Other libraries written in dylan.NET specifically for dylan.NET may rise in the future giving a helping hand to those who wish to invest in dylan.NET. Note that dylan.NET made libraries are generally cross-platform through Novell Mono. If a native library is required, the dylan.NET library is only portable to the OSes for which a nativee library exists. For example, sld is only portable to Windows and Linux that is why XML will be used inside the CodeGen module.

# 3 The Language

## 3.1 Introduction

Finally here it is, the dylan.NET language manual pages.