Dylan Reference Manual

Draft, September 29, 1995

Background and Goals

Dylan is a general-purpose high-level programming language, designed for use both in application and systems programming. Dylan includes garbage collection, type-safety, error recovery, a module system, and programmer control over runtime extensibility of programs.

Dylan is designed to allow efficient, static compilation of features normally associated with dynamic languages.

Dylan was created out of the belief that programs have become too complex for traditional static programming languages. A new generation of software—software that can be built quickly and enhanced over time—requires higher-level programming tools. The core of these tools is a simple and expressive language, one which protects the programmer from low-level implementation details, but still produces efficient executables.

Dylan was designed from the ground up with a thoroughly integrated object model, syntax, and control structures. It is not source code compatible with any existing languages, and can therefore be more internally self-consistent. At the same time, Dylan's syntax and object-model allow a high-level of integration with libraries written in other languages such as C and C++.

Dylan avoids providing multiple ways of doing the same thing. Quite the opposite, the language often uses a single construct to achieve several ends. For example, Dylan's type declarations improve the efficiency and readability of programs, they ensure type safety, and they provide the basis of polymorphic dispatch, the basic mechanism of object-oriented flow of control.

And while simplicity of language is very important, it should not and need not come at the price of expressiveness. Multi-method dispatch is an example of a Dylan feature that makes the language more powerful and simultaneously makes Dylan programs easier to understand.

Dylan demonstrates that a programming language can be highly expressive, can encourage the use of appropriate abstraction, can make programming more productive, and can make the programming process enjoyable, all without sacrificing the ability to compile into code that is very close to the machine, and therefore very efficient.