

# Dylan Einfuehrung

Andreas Bogk und Hannes Mehnert

March 12, 2005

# Geschichte

- ▶ Ralph
- ▶ Apple Dylan, CMU, Harlequin
- ▶ Dylan Interim Reference Manual
- ▶ Dylan Reference Manual

# Apple Dylan

- ▶ Technology Release (basierend auf MCL)
- ▶ Apple Cambridge Labs
- ▶ Implementierung auf 68k, später auch PowerPC
- ▶ 1996 eingestellt aufgrund von Geldmangel

File Edit Text Project Browse Debug Windows Help

Project: tiles (inactive)

Contents of tiles (inactive)

- Dylan
- dylan-framework (inactive)
- Dylan-user
- tiles
- tiles.rsrc

Contents of tiles

- cell view
- tiling-cell
- tiling-view
- tracking
- events
- tiles

Contents of tiling-view

- Copyright (C) 1994, Apple Computer, Inc. All rights reserved.
- tiling view
- <tiling-view> (<cell-view>)
- \$max-rows
- \$max-columns
- new-grid (view :: <tiling-view>, #key) => ()
- draw (view :: <tiling-view>, r :: <region>) => ()
- set-tiling-type (view :: <tiling-view>, tiling-type :: <symbol>) => ()
- compute-coordinates (view :: <tiling-view>) => ()

Source Code of + Warnings of <tiling-view> (<cell-view>)

```
define class <tiling-view> (<cell-view>)
// A tiling-view is a cell-view where the cells are 0-tiles.

slot show-grid?,
  type: <boolean>,
  init-value: #f;

slot show-dual?,
  type: <boolean>,
  init-value: #t;

slot flicker?,
  type: <boolean>,
  init-value: #t;

slot crossing,
  type: <number>,
  init-value: 33/100;

slot leftish,
  type: <integer>,
  init-value: 0;

slot rightish,
  type: <integer>,
  init-value: 0;
end class;
```

tiles

Call Grapher: turtle-forward!«*skeleton-module* (the-turtle :: <turtle>, pen-down :: <boolean>)

Grapher  
turtle-forward!«*skeleton-module* (the-turtle :: <turtle>, pen-down :: <boolean>)

- ↳ turtle-forward!
- ↳ turtle-forward!«*skeleton-module*...
- ↳ + (object1, object2) ↳
  - ↳ + (x :: <small-integer>, y :: <small-integ... ↳
  - ↳ + (x :: <big-integer>, y :: <small-integer... ↳
  - ↳ + (x :: <small-integer>, y :: <big-integer... ↳
  - ↳ + (x :: <big-integer>, y :: <big-integer> ... ↳
  - ↳ + (n :: <integer>, r :: <ratio>) ↳
  - ↳ + (r :: <ratio>, n :: <integer>) ↳
  - ↳ + (p :: <machine-pointer>, o :: <integer>) ↳
  - ↳ + (o :: <integer>, p :: <machine-pointer>) ↳
  - ↳ 11 more (19 total)
- ↳ - (object1, object2) ↳
- ↳ == (object1, object2) => result :: <boolean> ↳
- ↳ LineTo«*mac-toolbox*«*mac-toolbox* (h :: <... ↳
- ↳ MoveTo«*mac-toolbox*«*mac-toolbox* (h :: ... ↳
- ↳ make-point (h :: <integer>, v :: <integer>... ↳
- ↳ point-h ↳
- ↳ point-v ↳
- ↳ position ↳

skeleton-module

## Function Family: + (object1, object2)

Function Family of

+ (object1, object2)

- + (x :: <small-integer>, y :: <small-integer>) => + :: <integer>

No source code available

- + (p :: <machine-pointer>, o :: <integer>)

- + (o :: <integer>, p :: <machine-pointer>)

- + (object1, object2)

- + (r1 :: <ratio>, r2 :: <ratio>)

- + (r :: <ratio>, n :: <integer>)

- + (n :: <integer>, r :: <ratio>)

- + (x :: <big-integer>, y :: <small-integer>) => v :: <integer>

- + (x :: <small-integer>, y :: <big-integer>) => v :: <integer>

- + (x :: <big-integer>, y :: <big-integer>) => v :: <integer>

- + (x :: <double-float>, y :: <double-float>) => result :: <double-float>

- + (x :: <double-float>, y :: <rational>) => result :: <double-float>

- + (x :: <rational>, y :: <double-float>) => result :: <double-float>

- + (p1 :: <point>, p2 :: <point>) => sum :: <point>

- + (a :: <number>, b :: <point>) => result :: <point>

```
define method \+ (a :: <number>, b :: <point>) => result :: <point>
  point(a + b.h, a + b.v);
end method;
```

- + (a :: <point>, b :: <number>) => result :: <point>

```
define method \+ (a :: <point>, b :: <number>) => result :: <point>
  point(a.h + b, a.v + b);
end method;
```

- + (a :: <rect>, b :: <rect>) => result :: <rect>

- + (a :: <rect>, b :: <number>) => result :: <rect>

- + (a :: <number>, b :: <rect>) => result :: <rect>

- + (a :: <rect>, b :: <point>) => result :: <rect>

Info for Selected Class: <turtle> (<object>)

Grapher  
↳ <turtle> (<object>)

↳ <turtle>«skeleton-module

Direct Slots of  
↳ <turtle>«skeleton-module (<object>)

```
position«skeleton-module
  position«skeleton-module (object :: <turtle>) => value :: <point>
    define class <turtle> (<object>)
      slot position :: <point>;
      slot step-size :: <integer>;
      slot rotation :: <integer>;
    end class <turtle>;
  position-setter«skeleton-module
  rotation«skeleton-module
  rotation-setter«skeleton-module
  step-size«skeleton-module
  step-size-setter«skeleton-module
```

Direct Methods of  
↳ <turtle>«skeleton-module (<object>)

```
turtle-forward!«skeleton-module (the-turtle :: <turtle>, pen-down :: <boolean>)
turtle-rotate-left!«skeleton-module (the-turtle :: <turtle>)
turtle-rotate-right!«skeleton-module (the-turtle :: <turtle>)
```

skeleton-module

Heaps

Name	Size
Mac free	571K
Dylan	469K
dylan-framework	420K
free	251K
Mac used	139K
vector	66K
mac-toolbox	58K
pair	28K
string	16K
method	15K
skeleton-library	11K
wrapper	8K

Class	Instances	Total Size
<simple-method>«Dylan»Dylan	32	12,832
<slot-accessor-method>«Dyla...	12	96

Value

```

#<the class <turtle>>
#<the class <turtle>>
#<the method turtle-rotate-right! (turtle) at: #x1EF393FE>
#<the method turtle-rotate-right! (turtle) at: #x1EF393FE>...

```

Commands

```

#<the method turtle-rotate-right! ((turtle) at: #x1EF393FE)
Class: #<the class <simple-method>«Dylan»Dylan>
Argument List: (<turtle>)
Return Types: #()
Rest Type: #<the class <object>>
Size: 436 bytes including 76 bytes of code
0 stw r31,-4(SP)
4 stwu r30,-8(SP)
8 mr r31,loc0 ; or r31,loc0,loc0
12 lwz loc0,18(loc0) ; rotation(code)
16 mr r30,arg2 ; or r30,arg2,arg2
20 bl $+56 ; %a locative to l at: #x1EF3945C
24 mr argV,argZ ; or argV,argZ,argZ
28 li argZ,368 ; addi argZ,r0,368
32 bl $+50 ; %a locative to l at: #x1EF3946C
36 sub: loc0,glink-base,1611 ; addi loc0,glink-base,-1611; floor(code)
40 mr argV,argZ ; or argV,argZ,argZ
44 li argZ,1440 ; addi argZ,r0,1440
48 bl $+312 ; %a locative to l at: #x1EF39578
52 lwz loc0,14(r31) ; rotation-setter(code)
56 mr argZ,r30 ; or argZ,r30,r30
60 mr r31,argV ; or r31,argV,argV
64 bl $+188 ; %a locative to l at: #x1EF39678
68 mr argZ,r31 ; or argZ,r31,r31
72 b $+328 ; %a locative to l at: #x1EF39678

```

Value

```

#<the class <slot-descriptor rotation>
#<the class <slot-descriptor rotation>
#<the slot-descriptor rotation>

```

Commands

```

#<the class <slot-descriptor rotation>
Class: #<the class <slot-descriptor>«Dylan»Dylan>
Type: #<the class <integer>>
Allocation: INSTANCE
Initializer: NONE
Slots: 2
Xinit-desc-bits: 8
Xinit-desc-type-data: #<the class <integer>>
Xinit-desc-init-data: #f
Xinit-desc-key: #f
slot-getter: #<the generic-function rotation (arg-0)
slot-setter: #<the generic-function rotation-setter
slot-class: #<the class <turtle>>

```

Value

```

#f
Class: #<the class <boolean>>
Slots: 0

```

Commands

```

#<the class <integer>>
Class: #<the class <abstract-class>«Dylan»Dylan>
Superclasses: (#<the class <rational>> #<the class <real>> #<the class <complex>> #<the class <number>> #<the class <object>>)
Subclasses: unknown
Slot Descriptors: 0

```

Commands

```

#<the class <turtle>>
Class: #<the class <instance-class>>
Superclasses: (#<the class <object>>)
Subclasses: unknown
Slot Descriptors: 3
#<the slot-descriptor position>
#<the slot-descriptor step-size>
#<the slot-descriptor rotation>

```



#&lt;DYNAMO-MODULE-MODEL skeleton-module #x1CEB...

## Commands

Edit Value

Resample

Id: skeleton-module

Version: 52

Variables: 42

Kind	Int Name	Value
R	• \$edit-menu-id	1002
R	• \$file-menu-id	1001
R	• \$fractal-menu-id	1003
R	• \$library	##<<framework-library> id: 21>
R	• \$skeleton-string-id	1000
R	• \$skeleton-window-title-id	1001
R obj	• <app-behavior>	##<the class <app-behavior>>
R obj	• <my-document>	##<the class <my-document>>
R obj	• <my-view>	##<the class <my-view>>
R obj	• <set-fractal-event>	##<the class <set-fractal-event>>
R obj	• <simple-scripting-behavior>	##<the class <simple-scripting-behavior>>
R obj	• <static-text-message-property>	##<the class <static-text-message-property>>
R obj	• <turtle>	##<the class <turtle>>
R	• behavior-get-property	##<the generic-function behavior-get-property (arg-0, arg-1, arg-2, arg-3)>
R	• build-instructions	##<the generic-function build-instructions (>>
R	• calculate-next-size	##<the generic-function calculate-next-size (arg-0)>
R	• data	##<the generic-function data (arg-0)>
R	• data-setter	##<the generic-function data-setter (arg-0, arg-1)>
R	• direct-turtle	##<the generic-function direct-turtle (arg-0)>
W	• drawing-setup-index	0
R	• drawing-setups	##[#[0, 50, #(<200, 75>), "F-F-F-F", #(#("F" . "F-F+F+FF-F-F+F"))], #[1, 10,
R	• init-skeleton	##<the generic-function init-skeleton (>>
R	• position	##<the generic-function position (arg-0)>
R	• position-setter	##<the generic-function position-setter (arg-0, arg-1)>
R	• rotation	##<the generic-function rotation (arg-0)>
R	• rotation-setter	##<the generic-function rotation-setter (arg-0, arg-1)>
R	• select-drawing-setup	##<the generic-function select-drawing-setup (>>
R	• step-size	##<the generic-function step-size (arg-0)>
R	• step-size-setter	##<the generic-function step-size-setter (arg-0, arg-1)>
R	• the-document	##<the generic-function the-document (arg-0)>
R	• the-document-setter	##<the generic-function the-document-setter (arg-0, arg-1)>
R	• the-fractal	##<the generic-function the-fractal (arg-0)>
R	• the-fractal-setter	##<the generic-function the-fractal-setter (arg-0, arg-1)>
W	• the-turtle	##<<turtle> id: 29>
W	• turtle-axiom	"F-F-F-F"
R	• turtle-forward!	##<the generic-function turtle-forward! (arg-0, arg-1)>
W	• turtle-initial-point	##<point 200,75>
R	• turtle-recursion-depth	3
R	• turtle-rotate-left!	##<the generic-function turtle-rotate-left! (arg-0)>
R	• turtle-rotate-right!	##<the generic-function turtle-rotate-right! (arg-0)>
W	• turtle-rules	##("F" . "F-F+F+FF-F-F+F")
W	• turtle-step-size	2

- ▶ Gwydion Projekt
- ▶ Ziel: Development Environment
- ▶ DARPA finanziert von 1994 bis 1998
- ▶ Dylan Interpreter in C
- ▶ Dylan Compiler nach C in Dylan
- ▶ Seit 1998 Open Source

# Harlequin

- ▶ Mit Lisp bootstrapped (Harlequin LispWorks)
- ▶ Nativer Compiler mit Entwicklungsumgebung fuer Win32
- ▶ Commandline Compiler fuer Linux/x86 (alpha)
- ▶ Seit 2004 Open Source

► Einordnung in Programmiersprachen

## Algol aehnliche Syntax

```
begin
  for (i from 0 below 9)
    format-out("Hello world");
  end for;
end
```

# Dynamisch typisiert

- ▶ Strong vs weak typed
- ▶ Statische vs dynamische Typisierung

# Objektorientiert

- ▶ Alles ist von der Klasse `<object>` abgeleitet
- ▶ Multiple inheritance, aber richtig
- ▶ Superclass linearization

# Klassendefinition

```
define class <square> (<rectangle>)  
  slot x :: <number> = 0, init-keyword: x::  
  slot y :: <number> = 0, init-keyword: y::  
  constant slot width :: <number>,  
    required-init-keyword: width::  
end class;
```

- ▶ Garbage collection

## Keyword arguments

```
define function describe-list
  (my-list :: <list>, #key verbose?) => ()
  format(*standard-output*,
        "{a <list>, size: %d",
        my-list.size);
  if (verbose?)
    format(*standard-output*, ", elements:");
    for (item in my-list)
      format(*standard-output*, " %=", item);
    end for;
  end if;
  format(*standard-output*, "}");
end function;
```

# Higher order functions

- ▶ Anonyme Funktionen (lambda-Ausdruecke)
- ▶ Closures
- ▶ Currying
- ▶ Do, Map, Reduce
- ▶ Function composition

## Anonyme Funktionen und Closures

```
define function make-linear-mapper
  (times :: <integer>, plus :: <integer>)
=> (mapper :: <function>)
  method (x)
    times * x + plus;
  end method;
end function;
```

```
define constant times-two-plus-one =
  make-linear-mapper(2, 1);
```

```
times-two-plus-one(5);
// Returns 11.
```

## Curry, reduce, map

```
let printout = curry(print-object, *standard-output*);  
do(printout, #(1, 2, 3));
```

```
reduce(\+, 0, #(1, 2, 3)) // returns 6
```

```
reduce1(\+, #(1, 2, 3)) //returns 6
```

```
map(\+, #(1, 2, 3), #(4, 5, 6))  
//returns #(5, 7, 9)
```

## Interfacing to C

```
define interface
  #include "ctype.h",
  import: {"isalpha" => is-alphabetic?,
          "isdigit" => is-numeric?},
  map: {"int" => <boolean>};
end interface;

define constant is-alphanumeric? =
  disjoin(is-alphabetic?, is-numeric?);
```

## Generic functions

```
define method double
  (s :: <string>) => result
    concatenate(s, s);
end method;
```

```
define method double
  (x :: <number>) => result
    2 * x;
end method;
```

## Multiple dispatch

```
define generic inspect-vehicle
  (v :: <vehicle>, i :: <inspector>) => ();
define method inspect-vehicle
  (v :: <vehicle>, i :: <inspector>) => ();
  look-for-rust(car);
end;
define method inspect-vehicle
  (car :: <car>, i :: <inspector>) => ();
  next-method(); // perform vehicle inspection
  check-seat-belts(car);
end;
define method inspect-vehicle
  (truck :: <truck>, i :: <inspector>) => ();
  next-method(); // perform vehicle inspection
  check-cargo-attachments(truck);
end;
define method inspect-vehicle
  (car :: <car>, i :: <state-inspector>) => ();
  next-method(); // perform car inspection
  check-insurance(car);
end;
```

## Optionale Typrestriktionen von Bindungen

```
define method foo
  (a :: <number>, b :: <number>)
  let c = a + b;
  let d :: <integer> = a * b;
  c := "foo";
  d := "bar"; // Typfehler!
end
```

```
let collection = #[1, 2, 3];  
for (i in collection)  
  format-out("%=\n", i);  
end for;
```

```
let (initial-state, limit, next-state, finished-state?,
    current-key, current-element) =
  forward-iteration-protocol(collection);
local method repeat (state)
  block (return)
    unless (finished-state?(collection, state, limit))
      let i = current-element(collection, state);
      format-out("%=\n", i);
      repeat(next-state(collection, state));
    end unless;
  end block;
end method;
repeat(initial-state)
```

```
while (1) {
    L_state_2 = L_state;
    if ((L_state_2 < 3)) {
        L_PCTelement = SLOT((heapptr_t)&literal_ROOT,
                            descriptor_t,
                            8 + L_state_2 * sizeof(descriptor_t));
        L_instance = CLS_simple_object_vector_MAKER_FUN(orig_sp, 1, false);
        SLOT(L_instance, descriptor_t, 8 + 0 * sizeof(descriptor_t)) =
            LPCTelement;
        L_temp.heapptr = (heapptr_t)&str_ROOT;
        L_temp.dataword.l = 0;
        /* format-out{<string>} */
        format_out_METH(orig_sp, L_temp,
                       (heapptr_t)&empty_list_ROOT, L_instance);
        L_state = (L_state_2 + 1);
    } else {
        goto block0;
    }
}
block0;
```

# Sealing

```
define sealed domain \+(\<integer>, <integer>);  
  
define sealed class <integer> (<real>)  
  ...  
end class;
```

## Limited types

```
define constant $audio-buffer-size = 2048;
define constant <audio-buffer> =
  limited(<vector>,
    of: <single-float>,
    size: $audio-buffer-size);

define function mix-buffers
  (input1 :: <audio-buffer>, input2 :: <audio-buffer>,
   output :: <audio-buffer>)
=> ()
  for (i from 0 below $audio-buffer-size)
    output[i] = 0.5 * input1[i] + 0.5 * input2[i];
  end for;
end function mix-buffers;
```

## Type unions, singletons

```
define constant <false-or-integer> =  
  type-union(<integer>, singleton(#f));  
  
// Method 1  
define method say (x :: <false-or-integer>)  
  ...  
end method say;  
  
// Method 2  
define method say (x :: <integer>)  
  ...  
end method say;
```

## False-or, one-of

```
define method say (x :: false-or(<integer>)
```

```
  ...
```

```
end method say;
```

```
define method say (x :: one-of("#red", "#green", "#blue"))
```

```
  ...
```

```
end method say;
```

# Syntaktische Konventionen

- ▶ Klassen beginnen und enden mit spitzen Klammern: `<number>`
- ▶ Globale Variablen mit Asterisk: `*machine-state*`
- ▶ Konstanten beginnen mit `$`: `$pi`
- ▶ Praedikatsfunktionen enden mit `?`: `even?`
- ▶ Destruktive Funktionen enden mit `!`: `reverse!`
- ▶ Getters und setters: `element element-setter`

## Non-local exits

```
block (return)
  open-files();
  if (something-wrong)
    return("didn't work");
  end if;
  compute-with-files()
cleanup
  close-files();
end block
```

# Exceptions

```
block ()
  open-files();
  compute-with-files()
exception (<error>)
  "didn't work";
cleanup
  close-files();
end block
```

# Makros

```
define macro with-open-file
  { with-open-file (?stream:variable = ?locator:expression
                    #rest ?keys:expression)
    ?body:body
  end }
=> { begin
    let ?stream = #f;
    block ()
      ?stream := open-file-stream(?locator, ?keys);
      ?body
    cleanup
      if (?stream & stream-open?(?stream))
        close(?stream)
      end;
    end
  end }
end macro with-open-file;
```

## Library and Module

```
define library hello-world
  use dylan, import: all;
  use io, import: { format-out };
  export hello-world;
end library;
```

```
define module foobar
  use dylan;
  use format-out;
  export say-hello;
end module;
```

# Warum Dylan das Leben einfacher macht

- ▶ Keine buffer overflows und buffer underruns
- ▶ Keine double-frees
- ▶ Anwendungsbeispiel: sicherer Ort fuer kryptographische Schluessel

# Existierende Bibliotheken

- ▶ DUIM (Dylan User Interface Manager)
- ▶ Corba (2.0 mit ein paar 2.2 Erweiterungen)
- ▶ ODBC
- ▶ Network
- ▶ Regular Expressions
- ▶ Midi
- ▶ Dood
- ▶ File-system
- ▶ XML-Parser
- ▶ C-Interfaces zu png, pdf, postgresql, sdl, opengl
- ▶ ...