Pan: *High Level Description Language*

Rafael Ángel García Leiva

Juan José Pardo Navarro

angel.leiva@uam.es

Universidad Autónoma de Madrid

Geneva – January 2005

Pan: *High Level Description Language*

Contents

1. Introduction to pan

1.1 CDB and pan

1.2 How the panc compiler works?

1.3 Characteristics of pan language

1.4 How to organize configuration information

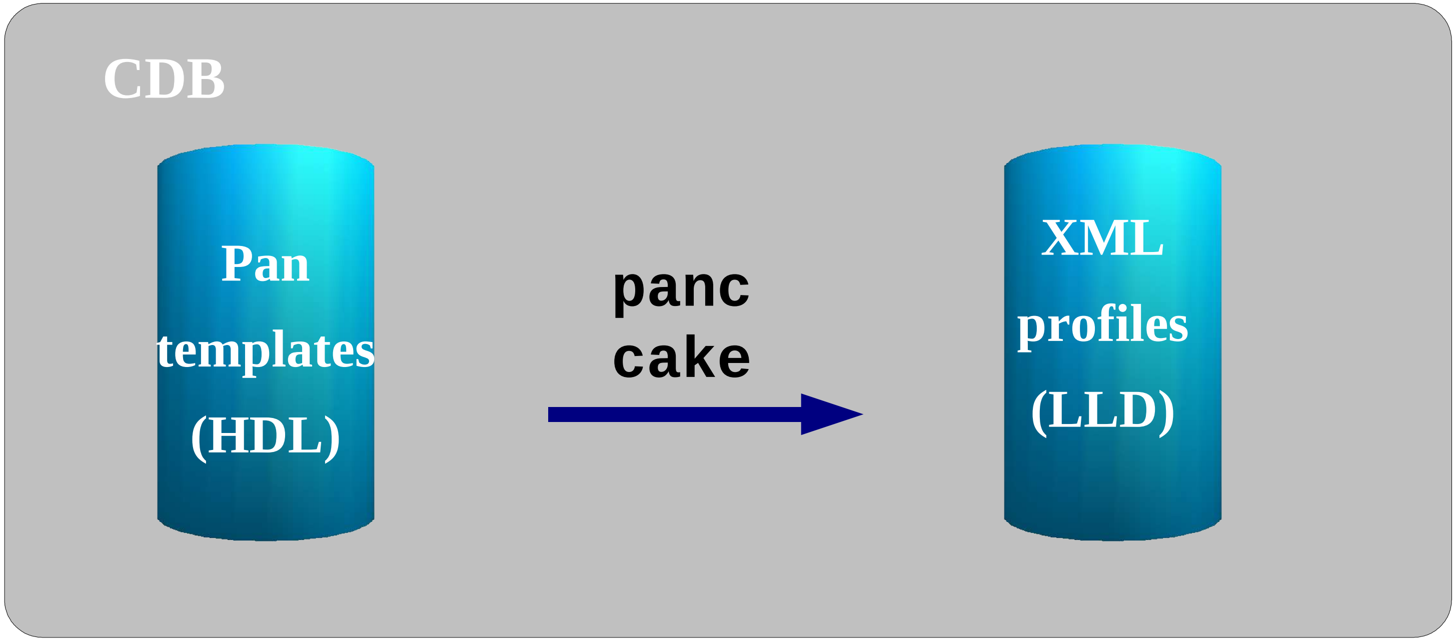
2. Pan syntax

3. Exercices

Pan: *High Level Description Language*

1.1 CDB and Pan

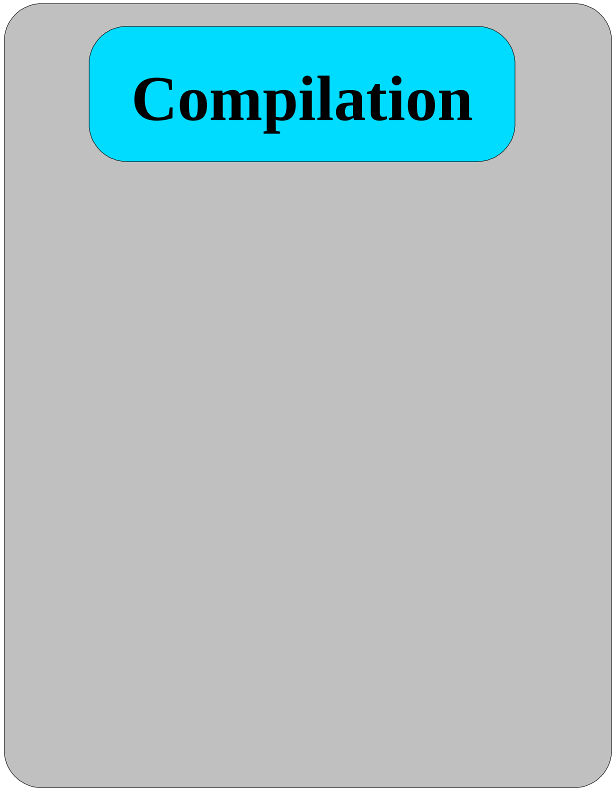
- Profiles:
  - Configuration elements for clients
  - There is a profile on CDB for each client to configure
  - Profiles are writen in pan language



Pan: *High Level Description Language*

1.2 How the panc compiler works

```
panc -x compact profile.tpl
```

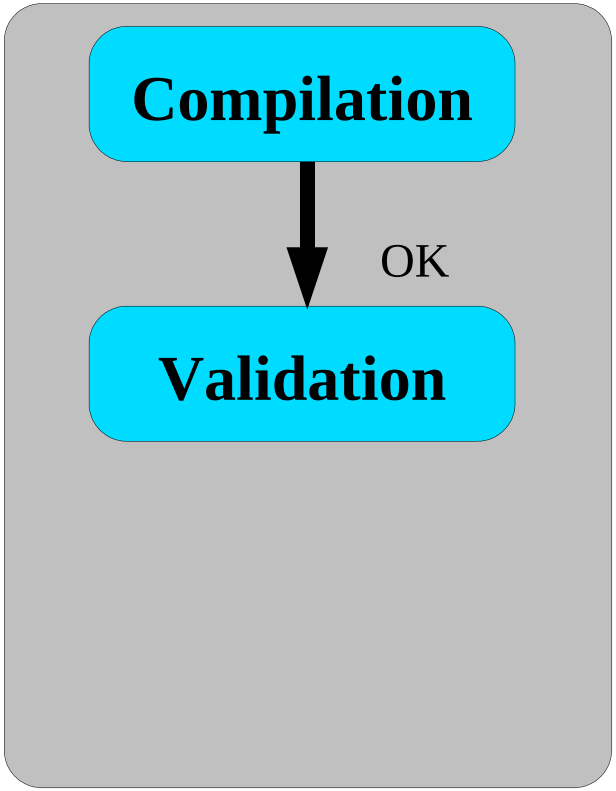


Compilation

Pan: *High Level Description Language*

1.2 How the panc compiler works

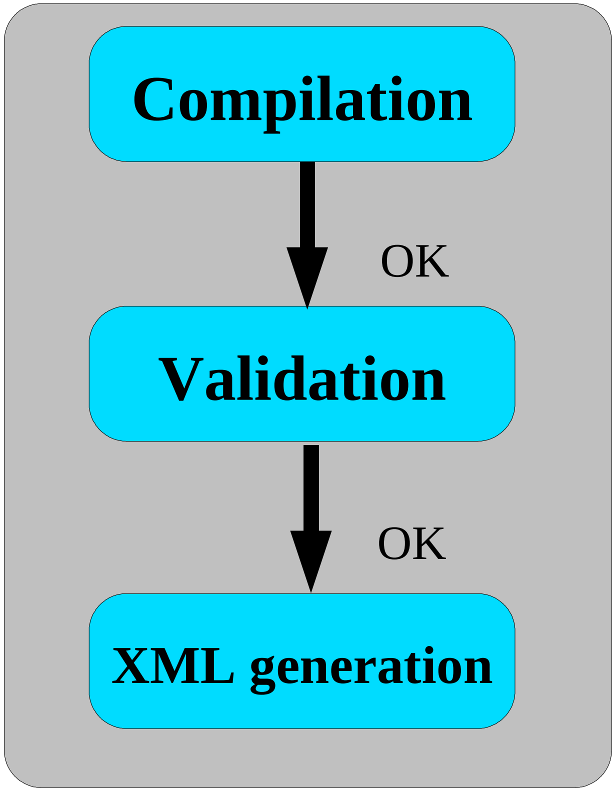
```
panc -x compact profile.tpl
```



Pan: *High Level Description Language*

1.2 How the panc compiler works

```
panc -x compact profile.tpl
```



Pan: *High Level Description Language*

1.3 Pan language characteristics

- Declarative programming language
  - plus data manipulation language
- High level abstraction
- Hierachical organization of information
- Strong type checking

Pan: *High Level Description Language*

1.4 How to organize cofiguration information

- Configuration information is represented by a tree of *elements*:
  - Property (strings, numbers, ...)
  - Resource (list,nlist)
- Every element has unique *path* which identifies its position in tree

Property

/profile/network/interfaces/eth0/ip

Relative path

interfaces/eth0/ip

Resource

/profile/network/interfaces/eth0

/profile/network/interfaces

/profile/network

/profile

Pan: *High Level Description Language*

1.4 How to organize cofiguration information

- Configuration information is represented by a tree of *elements*:
  - Property (strings, numbers, ...)
  - Resource (list,nlist)
- Every element has unique *path* which identifies its position in tree

Example of nlist (interfaces):

```
“/profile/network/interfaces/eth0/ip“ = “150.244.10.200“
“/profile/network/interfaces/eth0/gateway“ = “150.244.10.1“
“/profile/network/interfaces/eth0/netmask“ = “255.255.255.0“

“/profile/network/interfaces/eth1/ip“ = “192.168.0.100“
“/profile/network/interfaces/eth1/gateway“ = “192.168.0.1“
“/profile/network/interfaces/eth1/netmask“ = “255.255.255.0“
```

Pan: *High Level Description Language*

1.4 How to organize cofiguration information

- Configuration information is represented by a tree of *elements*:
  - Property (strings, numbers, ...)
  - Resource (list,nlist)
- Every element has unique *path* which identifies its position in tree

Example of list (RAM):

“/profile/hardware/ram/0/type“ = “DDR“

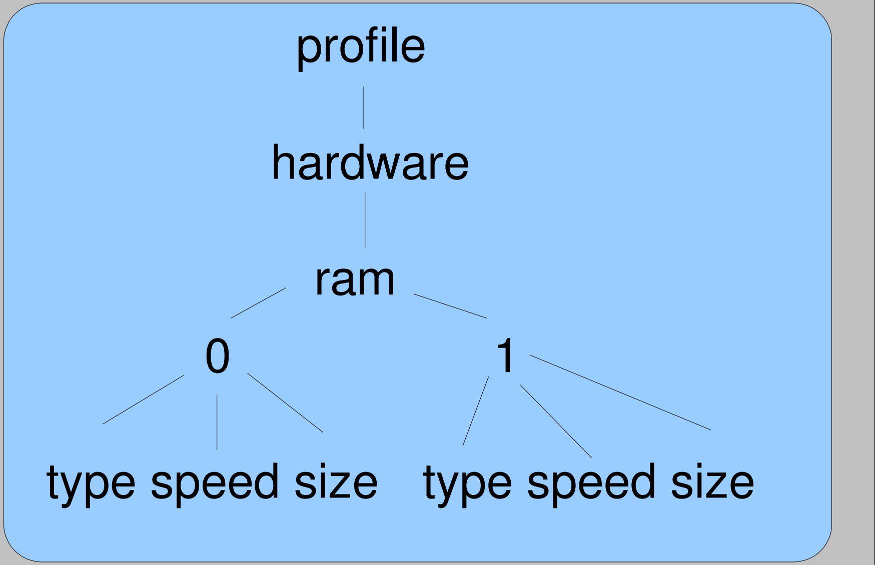
“/profile/hardware/ram/0/speed“ = 133

“/profile/hardware/ram/0/size“ = 256

“/profile/hardware/ram/1/type“ = “DDR“

“/profile/hardware/ram/1/speed“ = 133

“/profile/hardware/ram/1/size“ = 256



Pan: *High Level Description Language*

Contents

1. Introduction to PAN

2. Pan syntax

- **2.1 Templates and Statements**
- **2.2 Pan source files**
- **2.3 Types checking**
- **2.4 Basic type and variables**
- **2.5 Operators**
- **2.6 Complex data types**
- **2.7 Flow Control**
- **2.8 Functions**

3. Exercises

Pan: *High Level Description Language*

2.1 Templates and Statements (1 / 2)

- Main pan statements
  - Include:
    - `include identifier ';'`
  - Assignement:
    - `path '=' dml ';' # where` *`path`* `is a string`
  - delete:
    - `delete path ';'`

Pan: *High Level Description Language*

2.1 Templates and Statements (2 / 2)

- Statements are grouped into templates

Structure template

```
structure template ram;
  "type" = "DDR";
  "speed" = 133;
  "size" = undef;
```

Declaration template

```
declaration template types;
  define type ram_type = {
    "type"  : string
    "speed" : long
    "size"  : long
  };
```

Ordinary template

```
template hardware;
  "/hardware/ram/0" = create ("ram");
  "/hardware/ram/1" = create ("ram");
```

Object template

```
object template lxplus001;
  include types;
  type "/" = root_type;
  include hardware;
  "/hardware/ram/0/size" = 128;
  "/hardware/ram/1/size" = 128;
  "/hardware/total_mem" = 256;
```

Pan: *High Level Description Language*

2.2 Pan source files

- Pan Source Files
  - Filename should match template name
  - Source files end with “.tpl“
- Pan source files can include comments preceded by “#“

```
“/hardware/ram/0“ = create (“ram“) ;  # First RAM module
```

Pan: *High Level Description Language*

2.3 Types checking

- Pan has a strong type checking, it is not allowed to mix different data types

```
    “/x“ = 1 ;       # it is long
    “/x“ = “1“ ;     # Error: String to long
```

- We can add restrictions to any subtree

```
# restriction: swap size must be twice memory size
define type system_type = {
    "swap_size" : long with {
       self == (2* value("/hardware/total_mem"))}
};
```

Pan: *High Level Description Language*

2.4 Basic data types and variables

- Variables
  - There is no need to declare them
- Global Variables
  - They cannot be redifined

object, **self**

```
"swap_size" : long with {
    self == (2* value("/hardware/total_mem"));
```

Pan: *High Level Description Language*

2.4 Basic data types and variables

- Boolean

`true` **or** `false`

- Long

```
245    0       0x20   0xdeadb       0755
```

- Double

```
0.01  3.1416       1e-8  6.034582e23
```

- String

```
'a simple string', 'a bit more \t complex one \n'
```

- Undef
  - Represents non defined elements
  - They must be defined before the create the LLD file

```
“ip“ = undef;
```

Pan: *High Level Description Language*

2.5 Operators

- Arithmetic Operators

Unary: -

Binary: +, -, *, /, %

- String Operator

Binary: +

- Comparison Operators (work on strings and numbers)

Binary: <, <=, >, >=, ==, !=

- Boolean Operators

Unary: !

Binary: &&, |, ||

Pan: *High Level Description Language*

2.6 Complex data types

- Alias

```
define type ulong1 = long with self >= 0;
# it is equivalent to previous one
define type ulong1= long (0..);
define type port = long (0..65535);
```

- Link

```
define type disk_link = long (8..) *
    with match (self, '^(hda|hdb)$');
type “/device“ = disk_link;
# check that device is hda or hdb
“/device“ =  “/hda“;
# check that value is >= 8
“/hda“= 20;
```

Pan: *High Level Description Language*

2.6 Complex data types

- Records

```
define type network {
      “ip“: string;
      “gateway“: string;
};
```

```
# list of record
“/profile/network/interfaces/0/ip“ = “150.244.10.200“
“/profile/network/interfaces/0/gateway“ = “150.244.10.1“
“/profile/network/interfaces/1/ip“ = “192.168.0.100“
“/profile/network/interfaces/1/gateway“ = “192.168.0.1“
```

```
# nlist of records
“/profile/network/interfaces/eth0/ip“ = “150.244.10.200“
“/profile/network/interfaces/eth0/gateway“ = “150.244.10.1“
“/profile/network/interfaces/eth1/ip“ = “192.168.0.100“
“/profile/network/interfaces/eth1/gateway“ = “192.168.0.1“
```

Pan: *High Level Description Language*

2.6 Complex data types

- List

```
x = string []; # shuld go to type declaration
“/dns/0“= “150.244.10.200“;
“/dns/1“= “150.244.10.100“;
x = list (“150.244.10.200“,“150.244.10.100“);
z= x[1]; # set z to “150.244.10.100“
```

- Table (nlist)

```
machines = string {}; # should go to type declaration
“/machines/test1“= “192.168.0.100“;
“/machines/test2“= “192.168.0.101“;
machines = nlist (“test1“,“192.168.0.100“ ,
                  “test2“, “192.168.0.101“);
ip1=machines[“test1“];  ip2=machines[“test2“];
```

Pan: *High Level Description Language*

2.7 Flow Control

- Sequencing
- Branchig

```
if (value  (“/hardware/memory/size“) > 256) {
    swap = 512;
} else { swap = 256 };
```

- Looping

```
              i=0;
              while (i <= 9) {
                    i= i + 1; };
```

Pan: *High Level Description Language*

2.8 Functions

```
define function increment = {
     if (argc==2){
           val = argv[0] + argv[1];
           return (val);
     } else { return 0 };
};

# how to call a function
i=3;
x=increment (i,2);  # x will equal to 5
```

Pan: *High Level Description Language*

2.8 Functions

- **create (name: string,....) : nlist**

```
structure template mount_cdrom
      “device“ = undef;
      “path“ = “/mnt/cdrom“; };
“/system/mounts/0“ = create (“mount_cdrom“, “device“,“hdc“);
# it is equivalent to
“/system/mounts/0“ = create (“mount_cdrom“);
“/system/mounts/0/device“ = “hdc“;
```

- **list (....) : list**

```
“/system/dns“ = list (“150.244.9.200“,“150.244.9.100“);
# it is equivalent to
“/system/dns/0“ = “150.244.9.200“;
“/system/dns/1“ = “150.244.9.100“;
```

Pan: *High Level Description Language*

2.8 Functions

- **value (path:string) : element**

```
 “/x“ = 100;
 “/y“ = 2 * value (“/x“); # /y será 200

“/system/dns“ = list (“150.244.9.200“,“150.244.9.100“);
  z= (value(“/system/dns/0“)
      == value(“/system/dns/1“) ) ; # z is false
 # it is equivalent to
  v= value(“/system/dns“);
  z= (v[0])  == v[1]) ; # z es false
```

- **match (arg: string, regxep : string): boolean**

```
 # device_t is a string either disk or cd
   device_t = string with match (self, ' ^(disk|cd)$');
   device_t = “cdrom“; # Error
```

Pan: *High Level Description Language*

2.8 Functions

- **`first (arg:resource, key:identifier, value:identifier) : boolean`**
- **`next (arg:resource, key:identifier, value:identifier) : boolean`**

```
numlist = list (1,2,4,8);
sum =0;
ok = first (numlist,iter,v);
while (ok) {
    sum = sum + v;
    ok = next (numlist,iter,v);
};
#sum = 15
```

Pan: *High Level Description Language*

For more information:

(see 'quattor web page' -> 'documentation')

- *Pan Language Specification*
  (Version 2.0.1)

- *Pan Tutorial*
  (Version 1.0.2)

Pan: *High Level Description Language*

3. Exercises

1. Introduction to Pan
2. Pan Syntax
3. **Exercices**