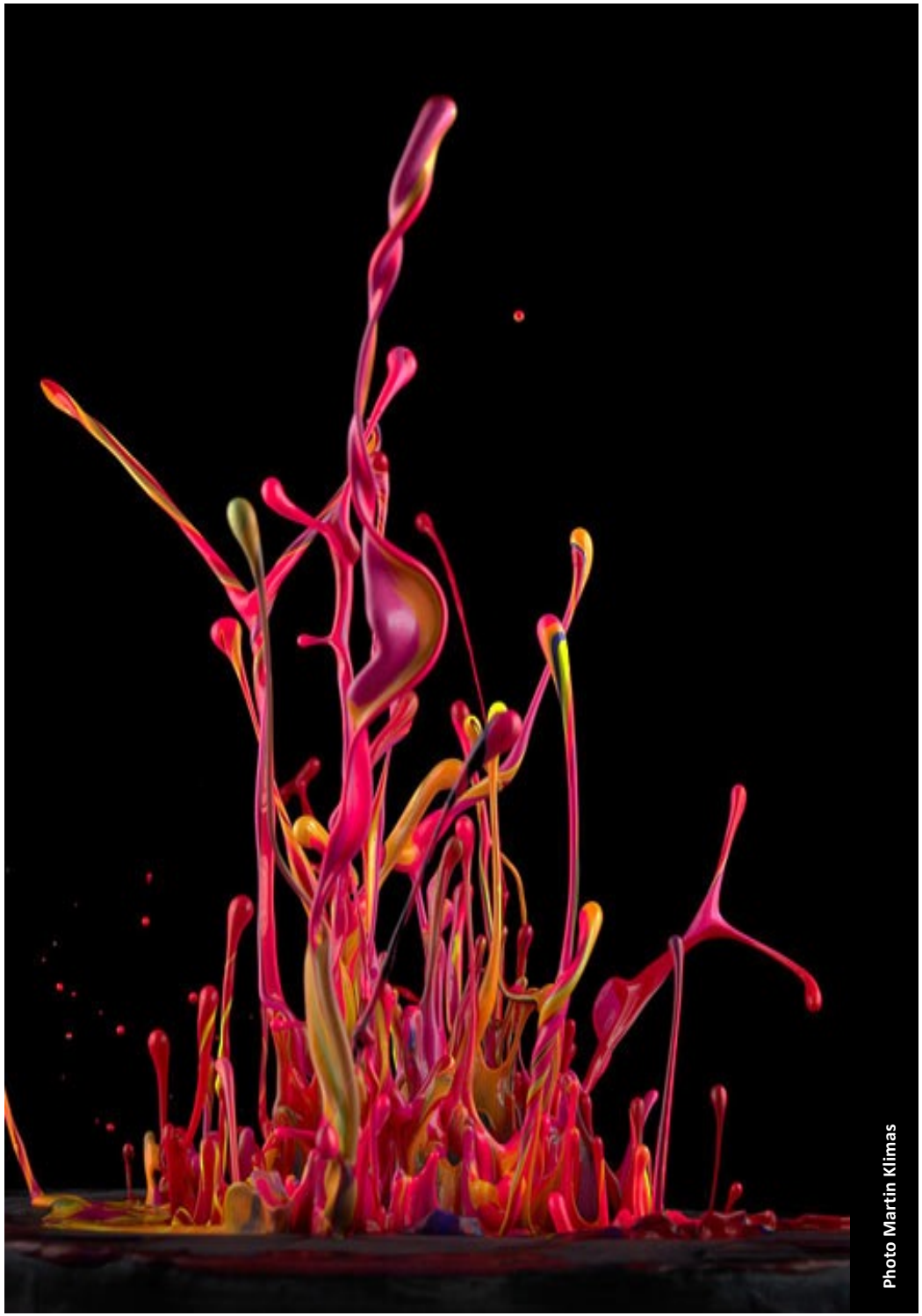


EPFL

Programmation pour Ingénieur

LabVIEW 2

ME 3e semestre

rev. 2025.1

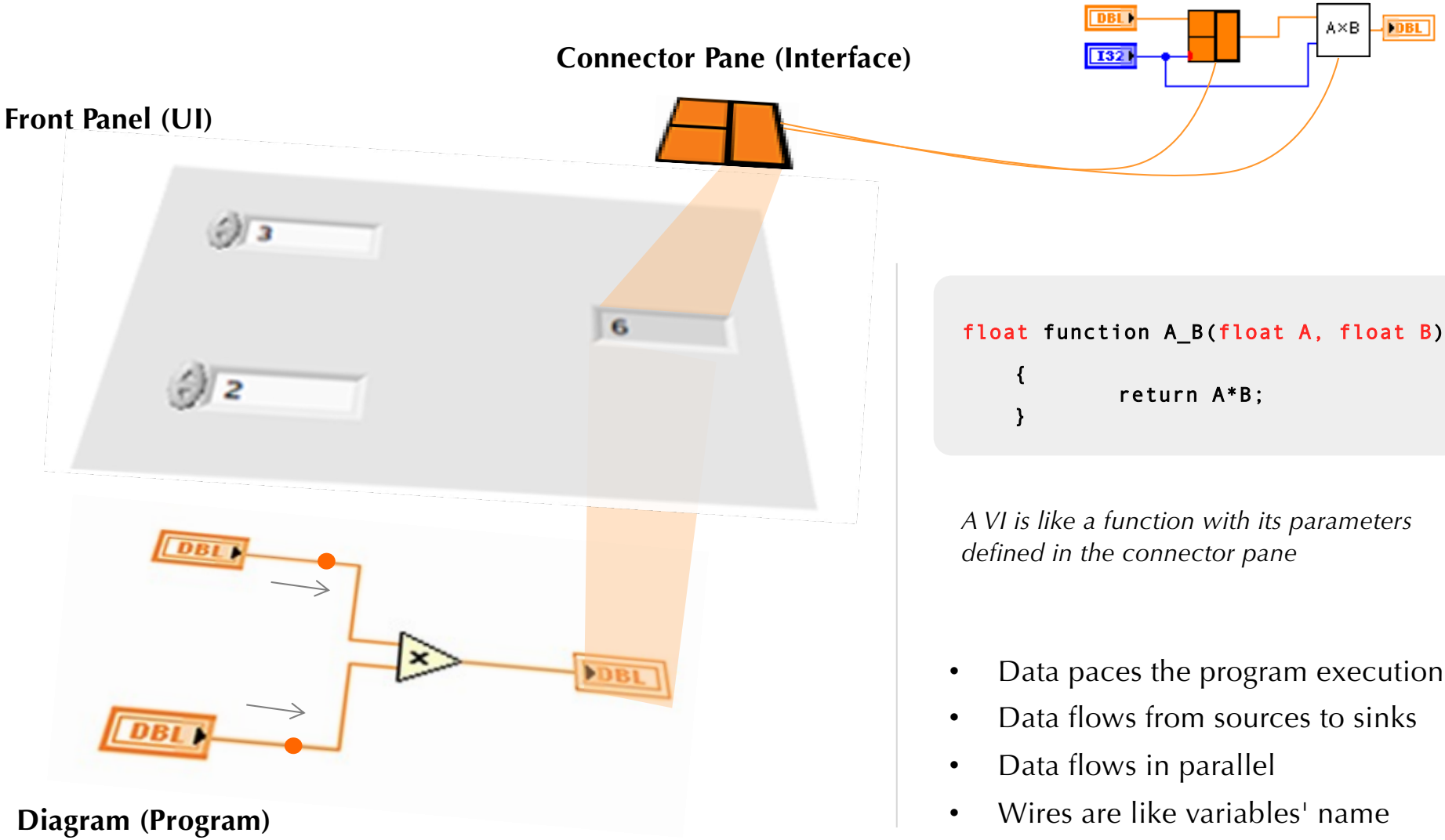
Christophe Salzmann

Laboratoire d'Automatique

Last week

- VI: front panel – diagram – connector pane
- Data flow
- Basic functions/structures
- Polymorphism
- Shift register
- My First VI

Virtual Instrument (VI)



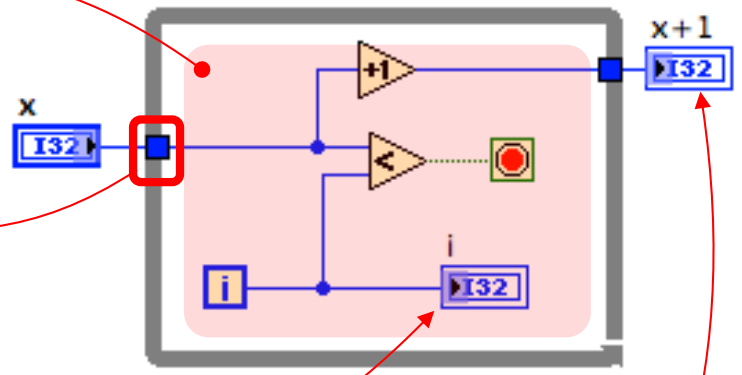

```
float function A_B(float A, float B)
    {
        return A*B;
    }
```

A VI is like a function with its parameters defined in the connector pane

- Data paces the program execution
- Data flows from sources to sinks
- Data flows in parallel
- Wires are like variables' name

G - structures

- Le contenu des structures définissent l'équivalent d'un bloc C/C++ { … }
- Les données (fils) qui entrent dans les structures sont évalués aux limites de la structure
- Un *indicator/control* à l'intérieur d'une structure sera évalué/mis à jour à chaque itération
- Un *indicator/control* à l'extérieur d'une structure sera évalué/mis à jour avant ou après l'exécution de la structure

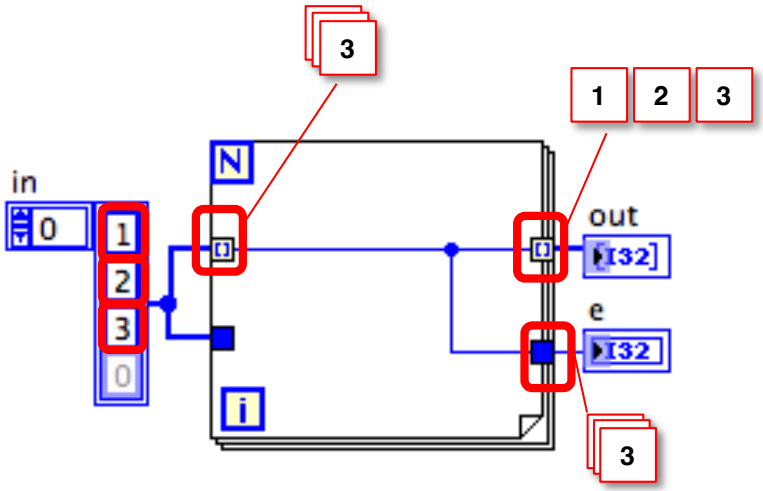


Boucle - for

- N nombre d'itérations definie par la **taille** du tableau d'entrée (in:3)

- auto indexing
- no auto-indexing

- Accède au i^ème élément du tableau d'entrée



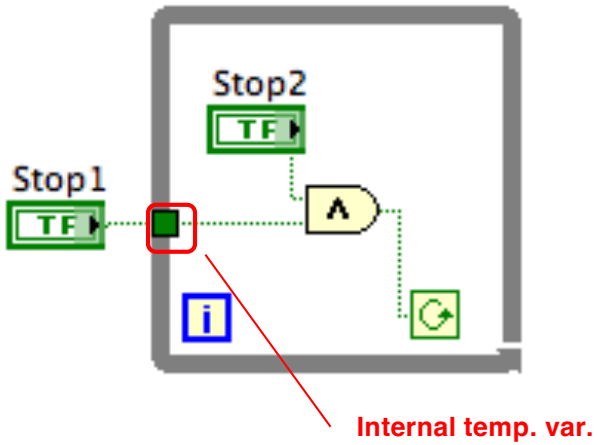
- out[] = {1, 2, 3}
- e = 3
- i = 2, N = 3

```
for(i=0;

i<sizeof(in[]);
        i++)
   {
     out[i] = in[i];
     e = in[i];
   }
```

Boucle - while

Les fils sont évalués à l'entrée de la boucle, l'intérieur de la boucle est similaire à un bloc en c

- Stop1 est évalué une fois avant l'entrée dans la boucle
- Stop2 est évalué à chaque itérations de la boucle

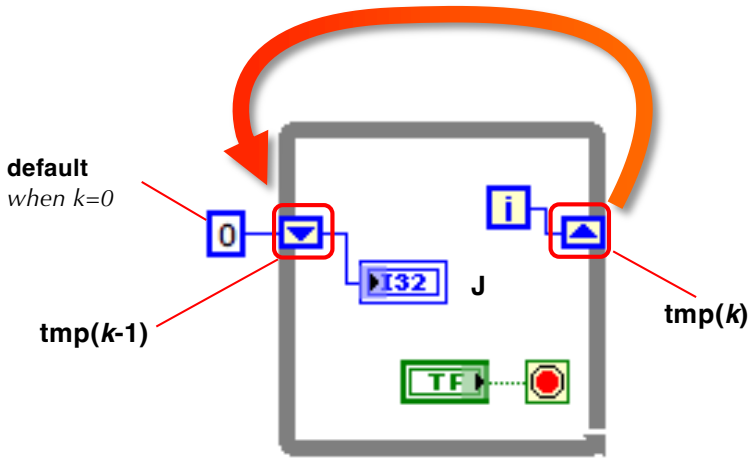


- *La boucle s'arrête après une itérations si Stop1 est False*
- *Si Stop1 est True, la boucle s'arrête quand Stop2 est False*

```
tmp = Stop1;
do {
      ...
   }
  while( tmp & Stop2
);
```

Dataflow & Shift register

Les shift registers sont des variables dépendantes du temps qui mémorisent la (les) valeur(s) précédente(s)



$J = \mathbf{0},0,1,2,3, \ldots$

```
int tmpJ_k_1 = 0; // default
 // tmpJ_k_2 = 0;
int tmpJ, J, Stop;
do {
    J = tmpJ_k_1;
    tmpJ = i;
    tmpJ_k_1 = tmpJ;
  } while (!Stop);
```

Demos

IEEE754 conv

Long SR

SR default value

Uninitialized SR

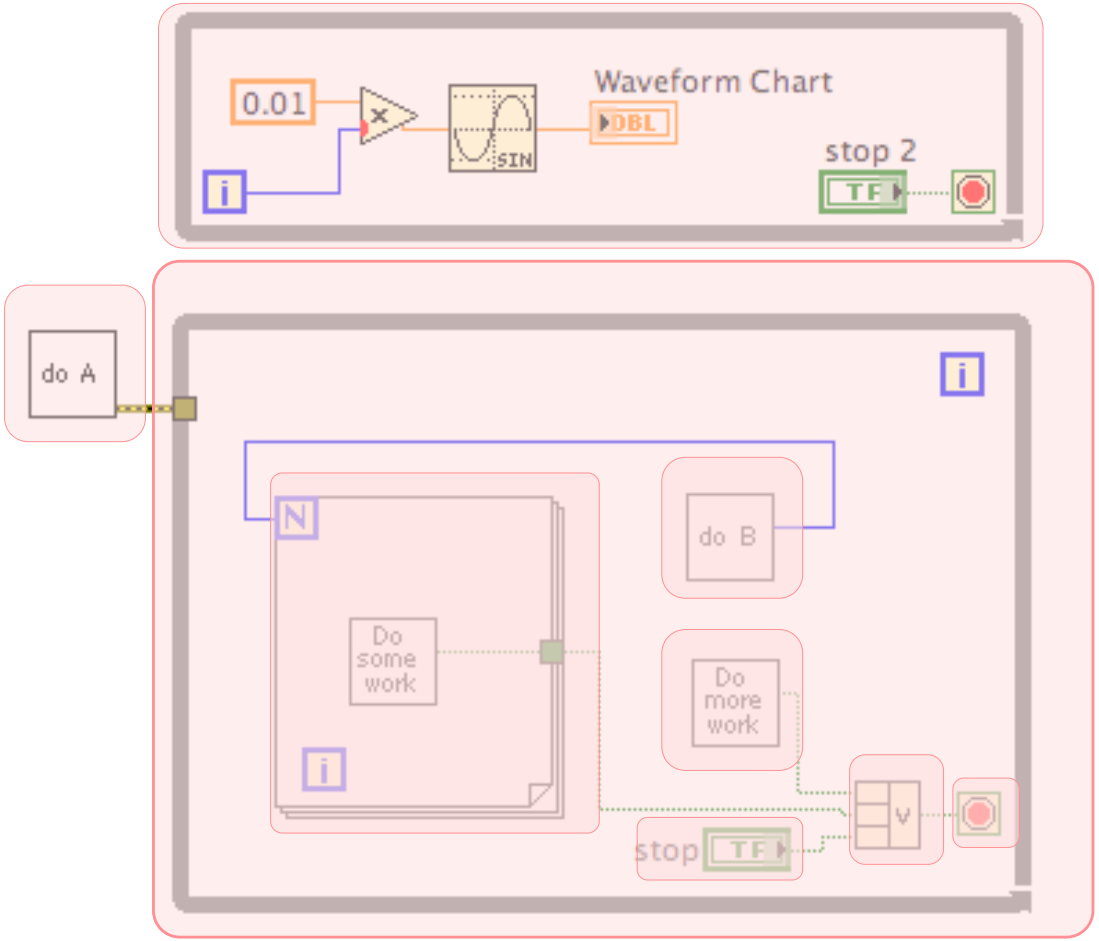
Dataflow dependency

This week

- More data flow
- More shift register
- Files
- More functions (array + string)
- Talking to the OS

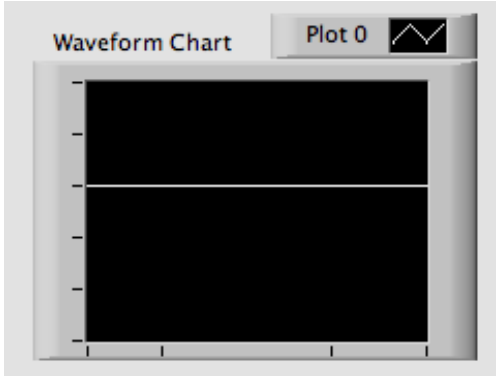
Dataflow

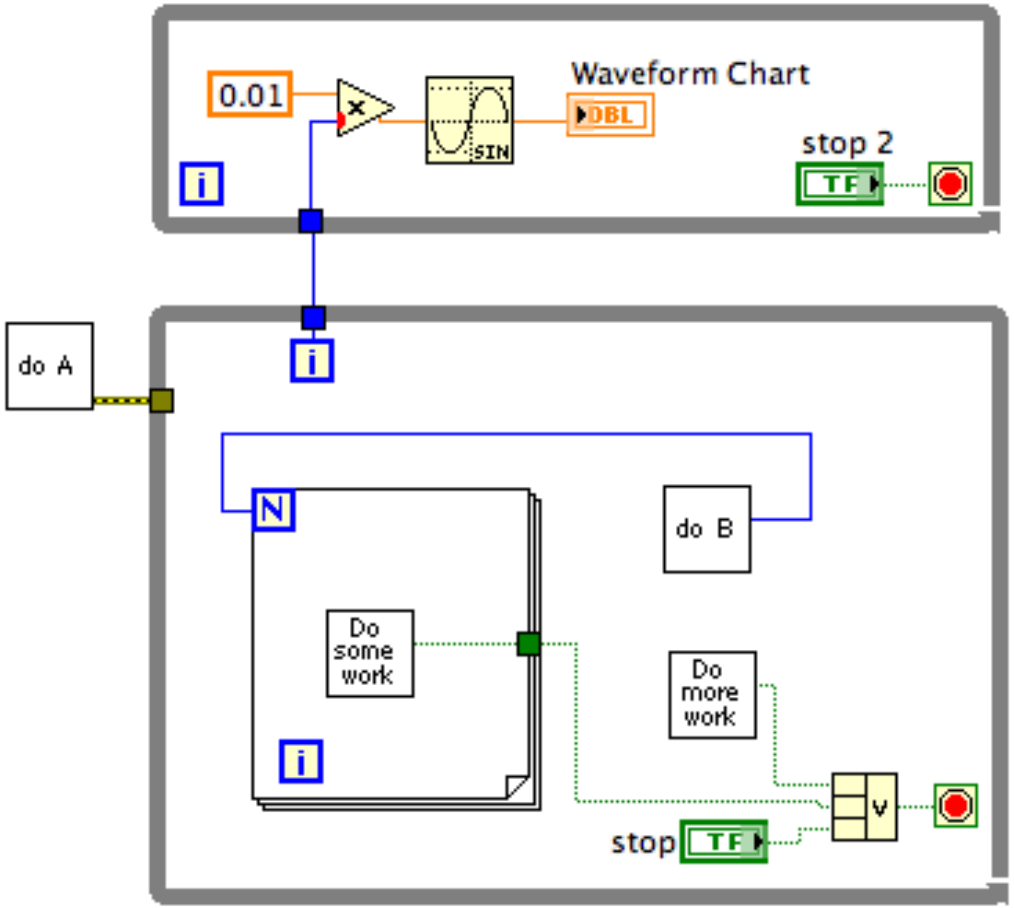
- Pour être exécuté un noeud doit avoir toutes ses entrées définies
- Deux noeuds indépendants sont exécutés en parallèle



Dataflow

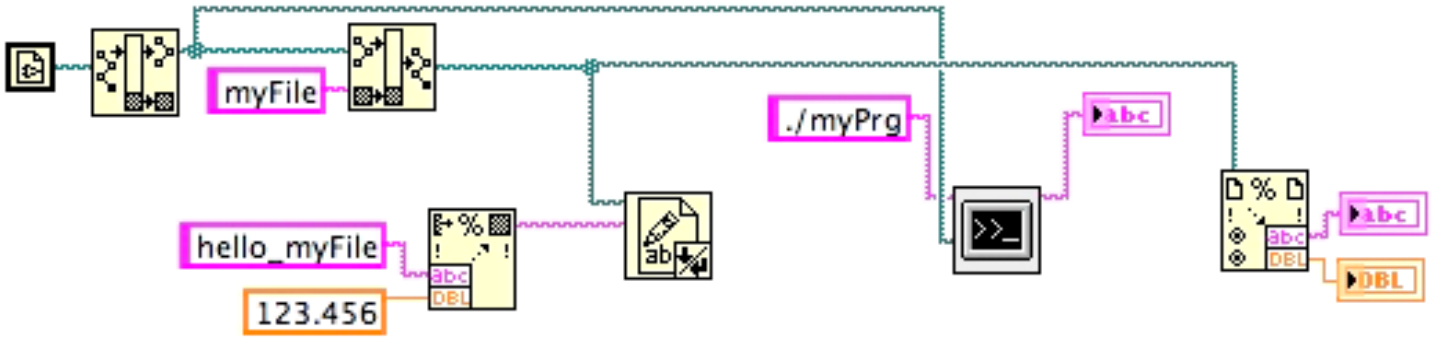
- Pour être exécuté un noeud doit avoir toutes ses entrées définies
- Deux noeuds indépendants sont exécutés en parallèle





Dataflow order

Ordre d'exécution ?

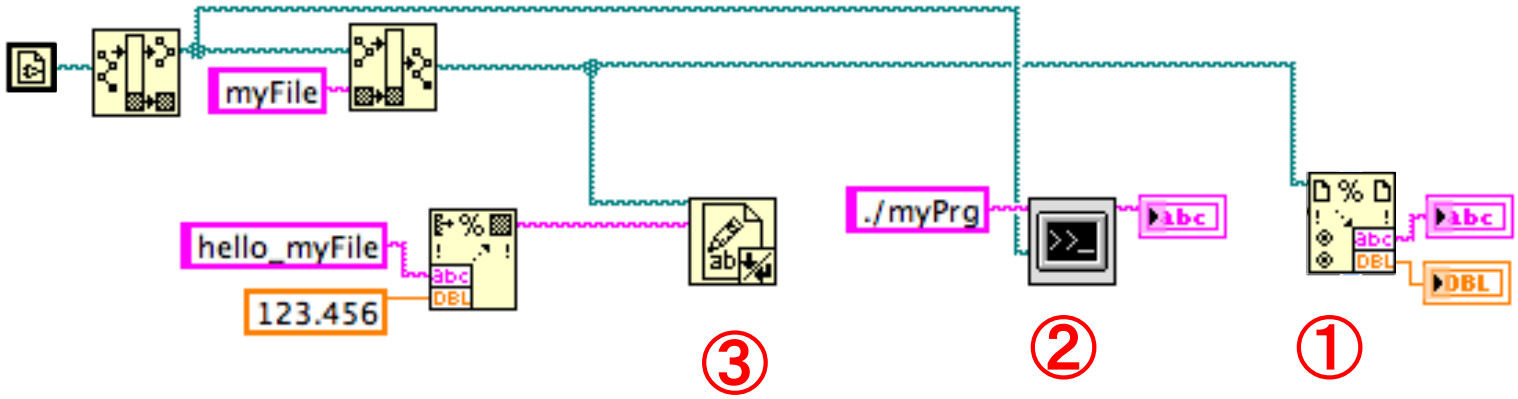


Dataflow order

Ordre d'exécution ?

Exatement l'opposé de ce qui est attendu et voulu!

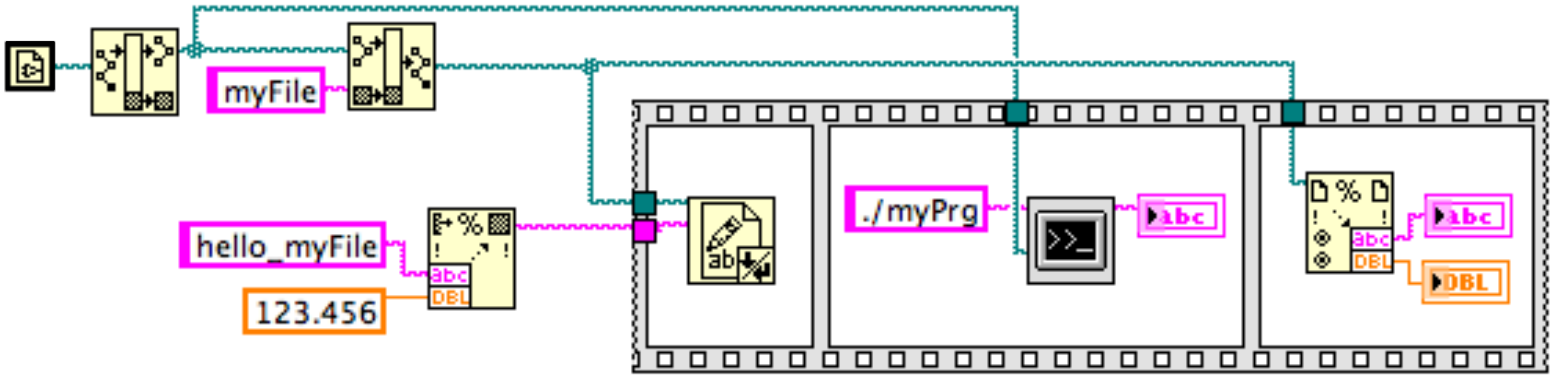
Dans l'exemple ci-dessous l'ordre d'exécution ne peut PAS être determiné en regardant le *diagram*! Le VI doit être exécuté pour déterminer la chronologie des 3 appels paralleles car il n'y a aucune dépendance temporelle entre ces appels .



Comment forcer l'exécution?

Dataflow order

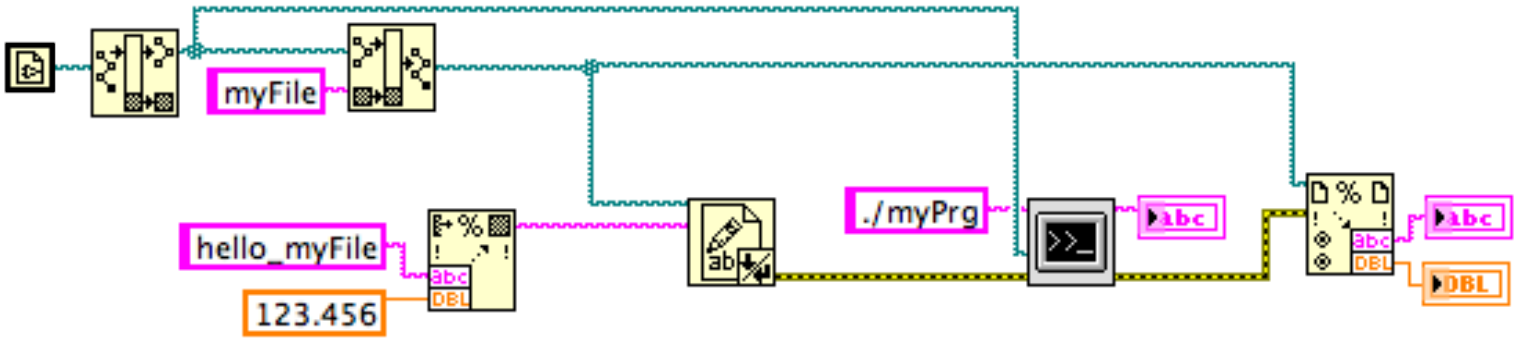
Utiliser une structure *sequence* pour définir l'ordre d'exécution



Risque de perturber le flot optimale de LabVIEW

Dataflow order

Utiliser le fils d'erreur pour définir l'ordre d'exécution

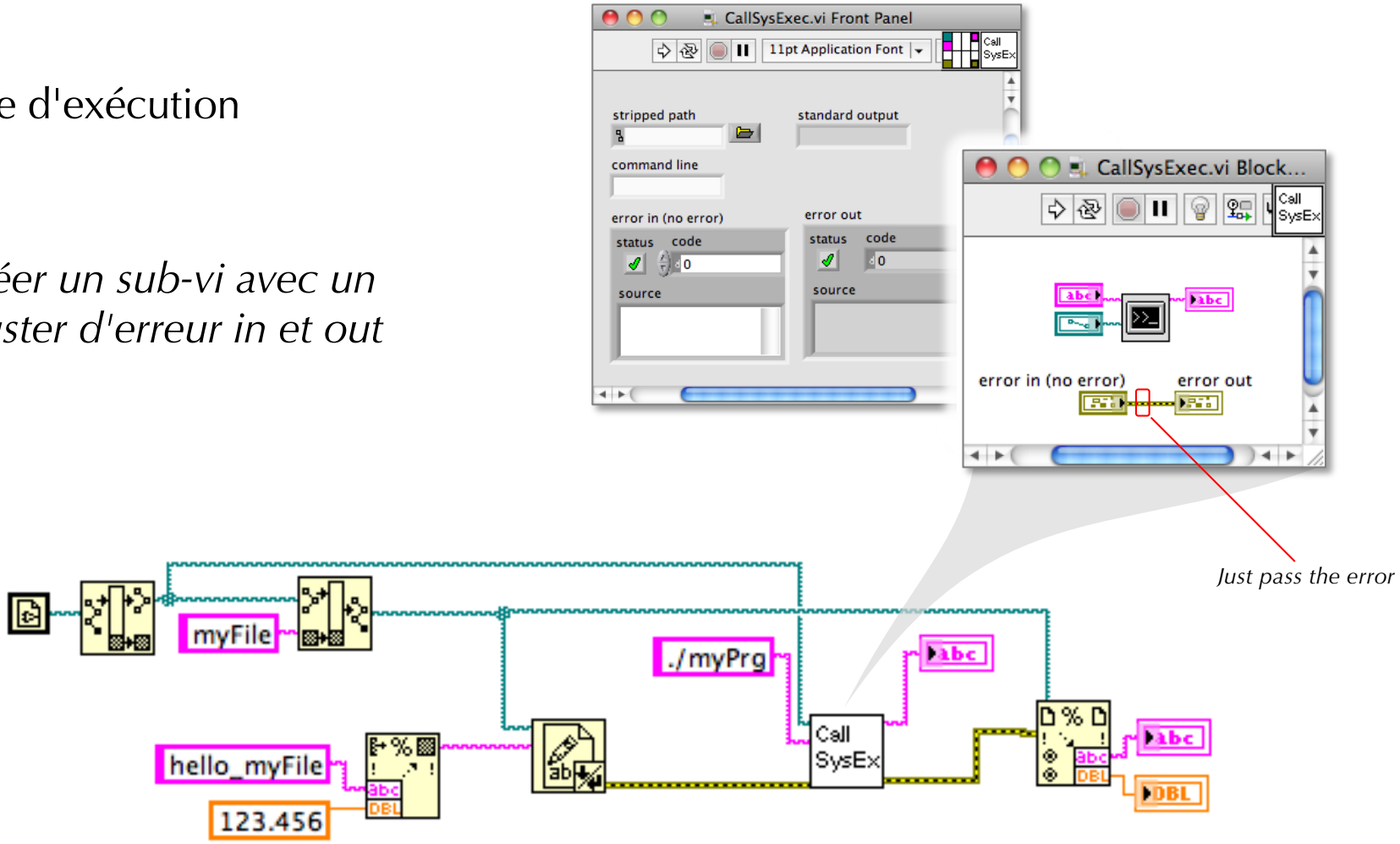


Comment faire si le VI **SysExec** n'avait pas de connecteurs d'erreur ?

Dataflow order

Ordre d'exécution

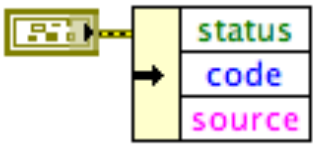
Créer un sub-vi avec un cluster d'erreur in et out



Gestion des Erreurs

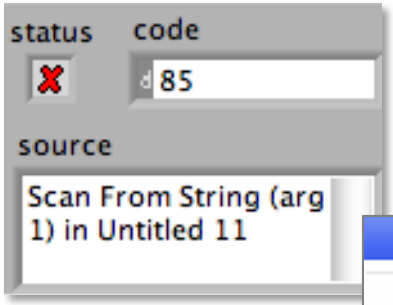
La gestion des erreurs est primordiale en LabVIEW, il en est de même pour les autres langages

LabVIEW utilise un *error cluster* pour la gestion des erreurs.

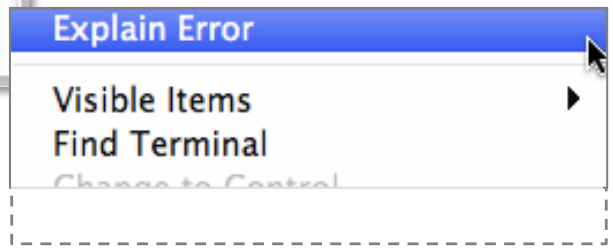


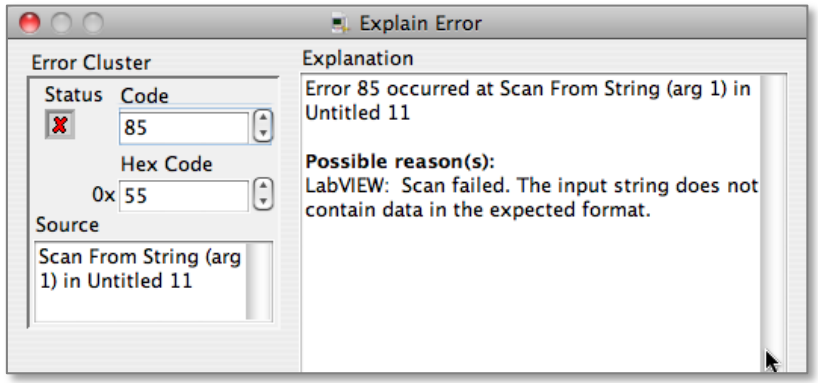
Error cluster

Status:	**true**	error
	false	no error
Code:	**0**	no error
	>0	warning code
	<0	error code
Source:	*additional info*	

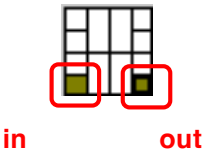


Right-click sur le bord droit d'un *error cluster* pour avoir une information détaillés sur l'erreur courante





Gestion des Erreurs



Par convention les *errors clusters* sont connectés en bas à droit et à gauche du *connector pane*.

Assurez vous que par défaut le *cluster error* d'entrée est défini comme à *no error* .

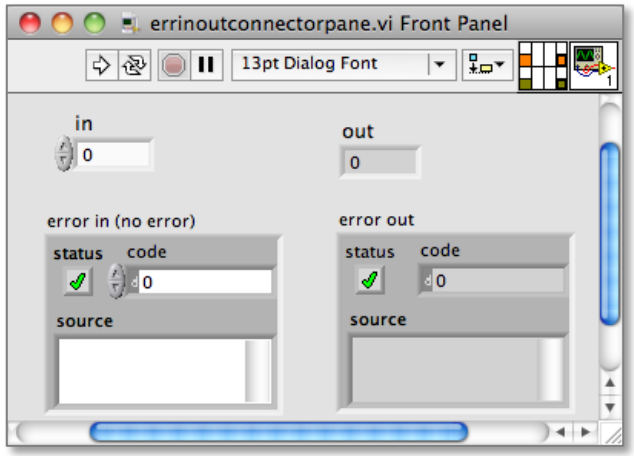
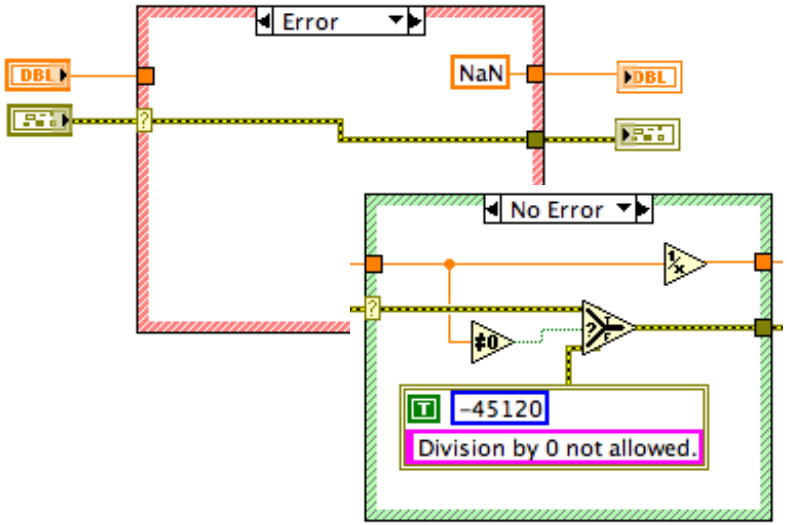


Diagram standard

Dans le *diagram*, s'il y a une erreur elle doit être directement passée au *cluster* de sortie et aucun code ne sera exécuté.

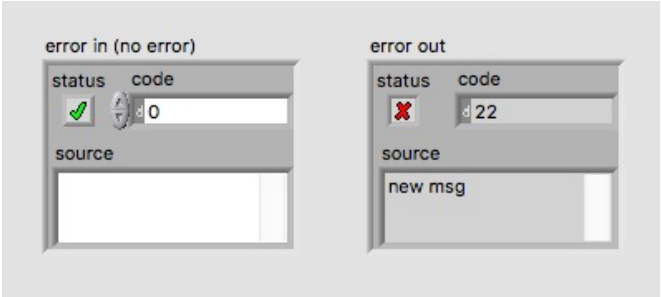
S'il n'y a pas d'erreur d'entrée, à vous de faire la gestion interne adéquate

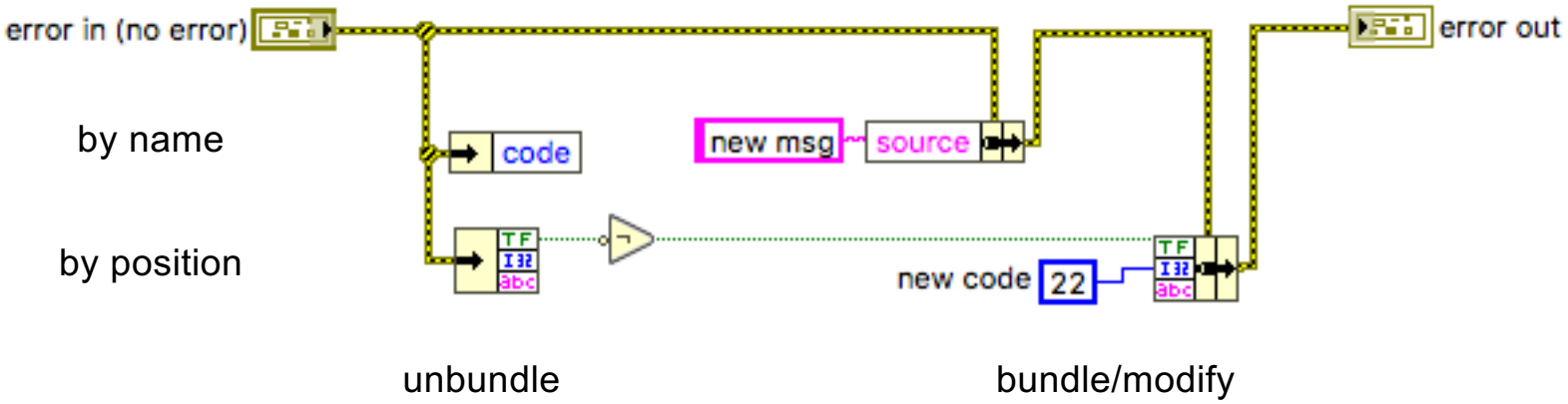


Error cluster

Cluster-> structure avec des éléments de types différents

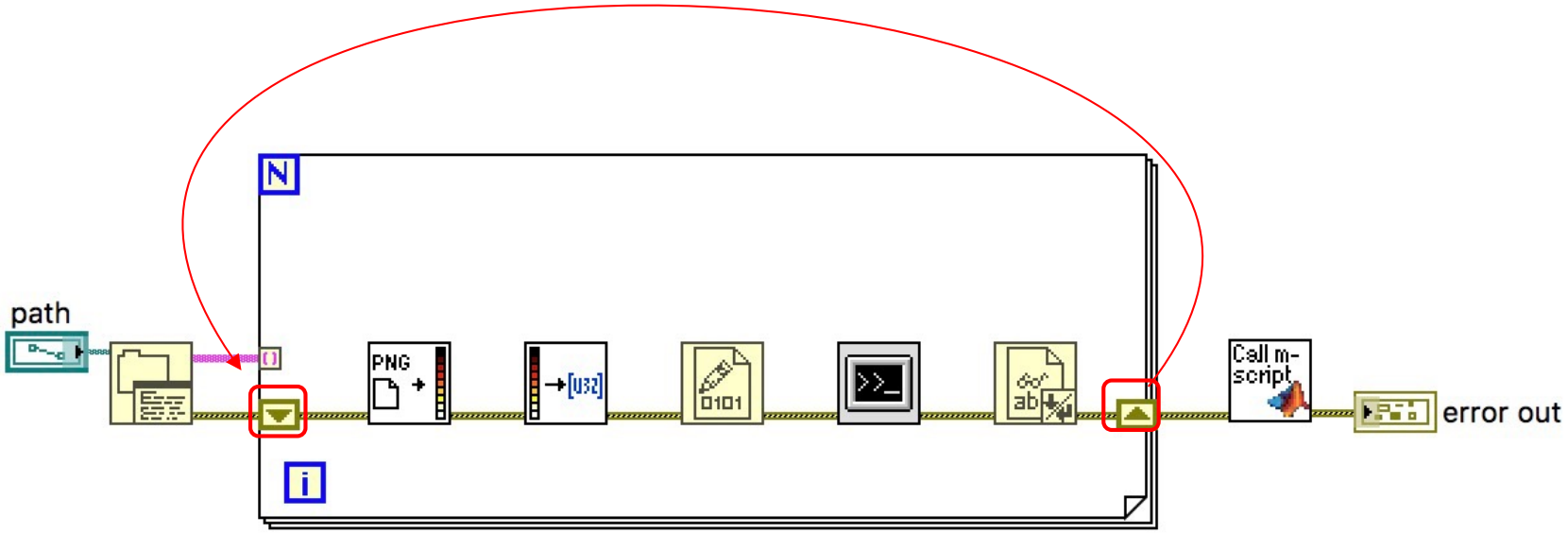
- Equivalent des `struct` en C/C++
- Le nombre d'éléments des clusters est fixe
- La couleur des clusters indique si les éléments sont de taille fixe (brun) ou non (rose)
- Il est possible de mettre un Cluster dans un Cluster
- Les éléments (champs) du Cluster peuvent être accédés par leur nom (recommandé) ou leur position





Gestion des Erreurs

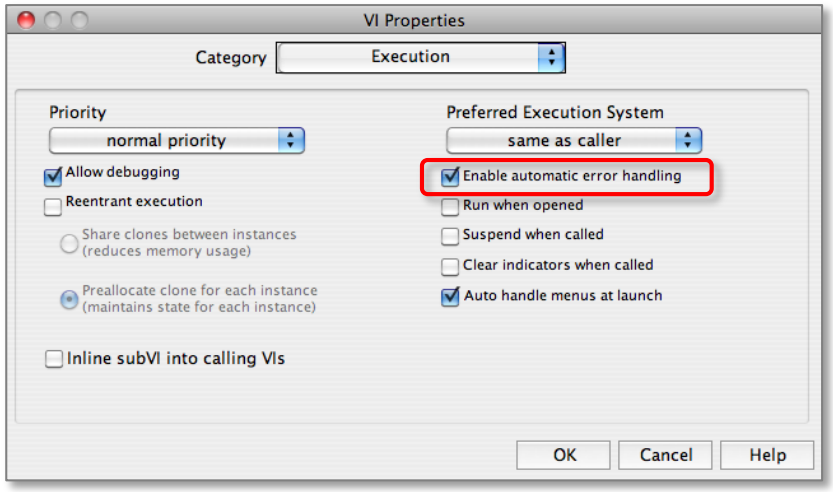
Chainez les Vis à l'aide des *errors* pour contrôler l'ordre d'exécution des VIs.
Dans les boucles (for, while), utilisez un *shift-registers* pour mettre à jour l'état des erreurs à chaque itération.



Gestion des Erreurs

De manière similaire aux exceptions c++, LabVIEW à un moyen pour afficher automatiquement les message d'erreurs si le connecteur d'erreur en sortie n'est pas connecté.

Cette option peut être désactivée via les propriétés du VI (right click sur l'icone du VI)



Fichiers

La gestion des fichier en LabVIEW est très similaire à celle de c/c++, C'est juste plus simple ☺!

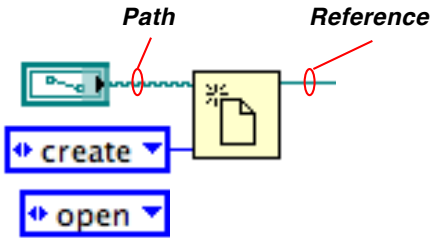
- LabVIEW emploie les appels OS standards
- Les règles sont les mêmes: accès, permission, quota, etc.
- Même type de gestions que le C/C++
- LabVIEW gère aussi bien les fichiers au format *text* or *binary*

File path "HD/myfiles/a.txt"

String to File path "HD/myfiles/b.txt"

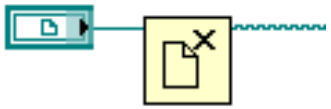
File reference 0x2486172920

Fichiers

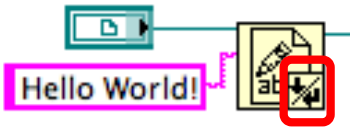


Open/create/replace file

Le chemin (path) doit être défini sinon un dialogue sera affiché.
Plusieurs options peuvent être définie (create/replace/etc.)

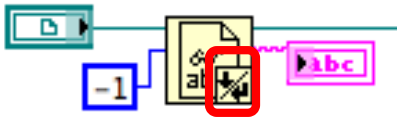


Close file



Write to text file

Option pour adapter le caractère de fin de ligne à la plateforme courante.



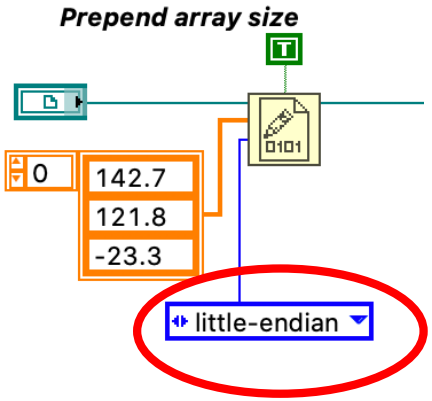
Read from text file

Spécifie le nombre de caractères à lire,
-1 -> tout le fichier

Adapte le caractère de fin de ligne à la plateforme courante.

```
FILE* F;
F=fopen("a.txt","r");


fclose(F);


fputs(F,"Hello World!");


fread(buffer,lenght,F);
```

Fichiers

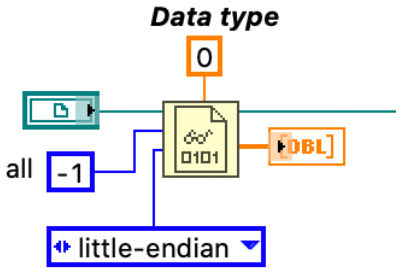


Write to binary file

Permet d'écrire des données binaires, p.ex. un tableau de 3 éléments, la taille (3) sera écrite avant les données du tableau. Pour un tableau multidimensionnel le VI écrira plusieurs tailles

```
0000 0003              3 elements
4061 D666 6666 6666    142.7
405E 7333 3333 3333    121.8
C037 4CCC CCCC CCCD    -23.3
```

Δ endianess



Read from binary file

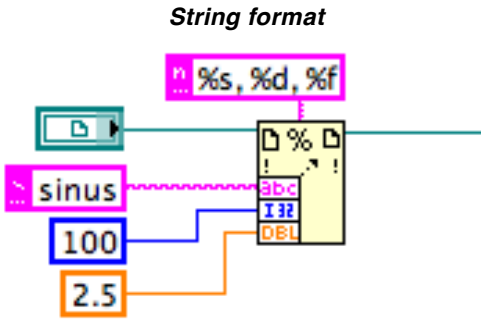
Data format défini comment interpréter les données binaires lues sur le disque.

-'1' pour lire l'entier du fichier.

```
FILE* F;
F=fopen("a.bin","wb");

double A[]=
   {142.7, 121.8, -23.3};

for (int i(0);i<3; i++) {
      fwrite(&A[i], 8,F);
    }
```

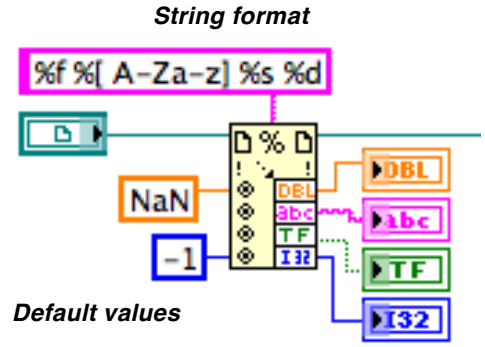
```
F=fopen("b.bin","rb");


double A[maxsize];
fread((char*)A,
      maxsize*sizeof(double));
```

Fichiers



Format into (text) File

Converti les entrées en une **string** conformément au **string format** et écrit cette *string* dans un fichier



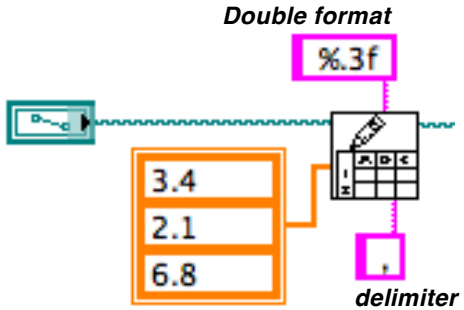
Scan from (text) file

Lit un fichier et interprète les entrées conformément au **string format**

Les valeurs par défaut (***Default values***) seront utilisées en cas d'erreur de conversion.

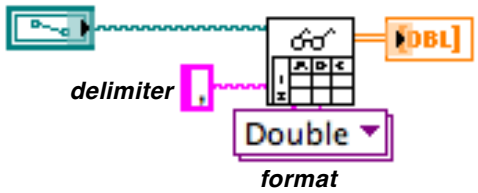
```
f.open("a.txt");


f<<"sinus"<<100<<2.5;


a.txt
sinus, 100, 2.5


double D = NaN;
string S;
boolean B;
int I=-1;

f>>D>>S>>B>>I;
```

Fichiers



Write Spreadsheet to File

Ex. Converti le tableau de double en une **string** conformément au **Double format** et écrit cette *string* dans un fichier

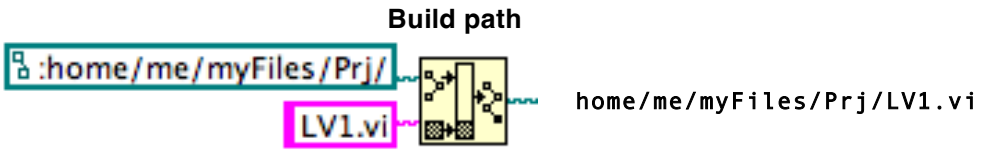


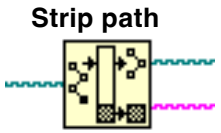
Read from Spreadsheet File

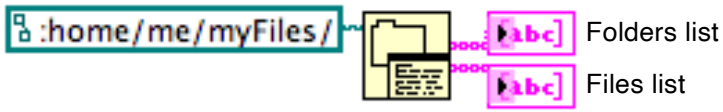
Lit un fichier tableur et interprète les entrées conformément au **format** et **delimiter** choisi

```
double A = {3.4, 2.1, 6.8};

string F = "%.3f";

string P = "a.txt";

S = WriteSpreadSheetToFile
            (P,A,F,',');


double A[];

String D = ',';

string P = "a.txt";

A = ReadFromSpreadSheetFile
            (P,'double',D);
```

Fichiers

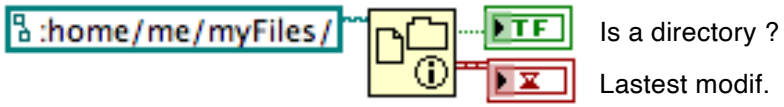






List folders

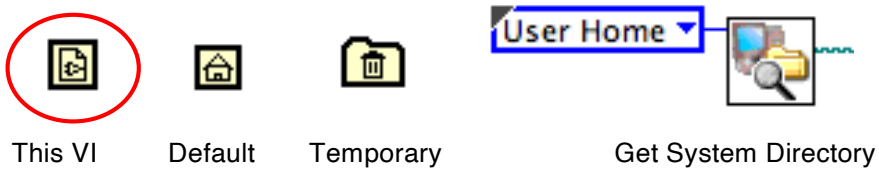
Liste tous le fichiers et dossiers d'un chemin donné



File/Dir info

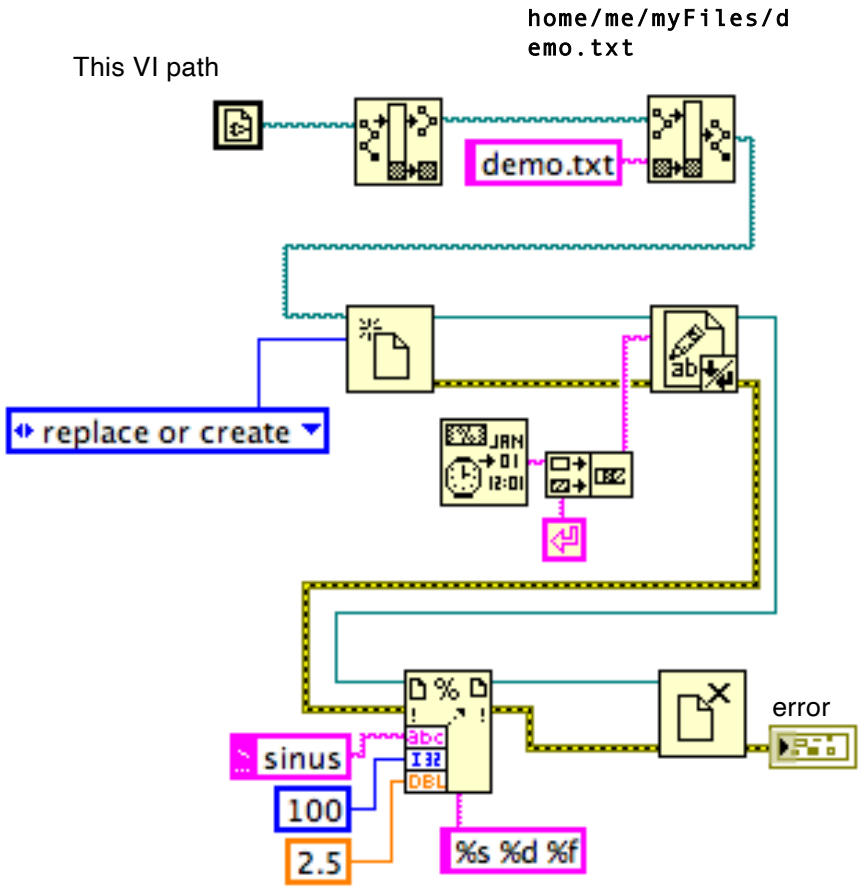
Returne les infos sur pour unfichier/dossier donné

Additional utilities libraries



Path constants

Fichiers – exemple complet



Construis un chemin de fichier basé sur l'emplacement du VI courant

Crée (ou remplace) un nouveau fichier

Lit l'heure courante et l'ajoute au fichier, ajoute '\r' à la suite

Ajoute "sinus 100 2.5" au fichier

Ferme le fichier

Affiche le cluster d'erreur

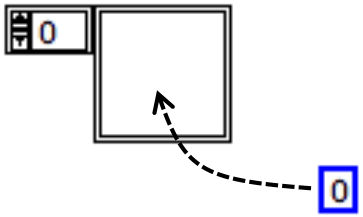
```
string path = LV_GetCrtVIPath();

path += "demo.txt";


ofstream f;
f.open (path.c_str());


time_t t;
time (&t);

f<< ctime(&t)<<"\r";



f << "sinus" << 100 << 2.5;

f.close()


if (f.fail()) ...
```

Demo

Path, write-read files

Tableau *array*

Tableau (Array) -> structure contenant des éléments du même type



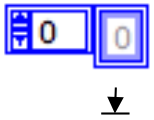


Tableau vide
0 est grisé
1 visible

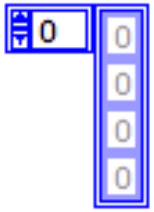
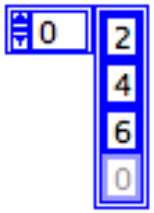
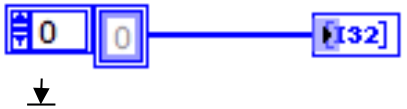


Tableau vide
0s sont grisés
4 visibles

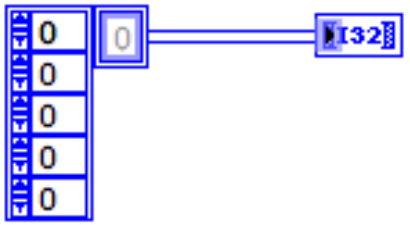


3 éléments
1 vide
4 visibles



1D array

2D array



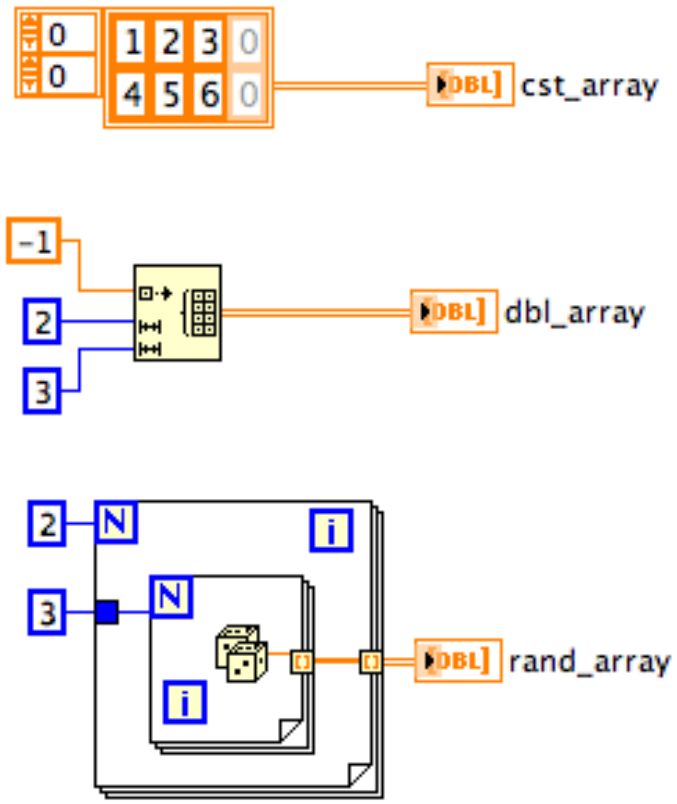
5D array

Demo

Arrays

Array functions

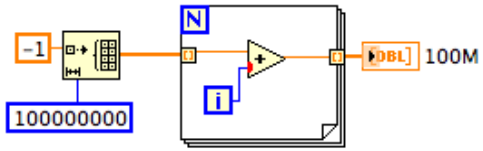
Initialize arrays



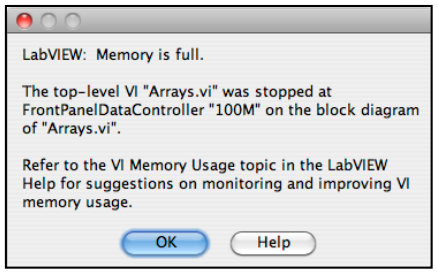

```
const dbl cst_array[2][3] =
      {{1,2,3} , {4,5,6}};


dbl dbl_array [2][3] =
      {{-1,-1,-1},{-1,-1,-1}};


dbl rnd_array[][];
for (r=0; r<2; r++)
   for (c=0;c<3;c++)

rand_array[r][c]=rand();
```

How big an array ?

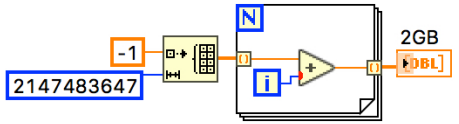
In LabVIEW 2013 32bit



100'000'000 pts x 8 Bytes ~ 800 MBytes



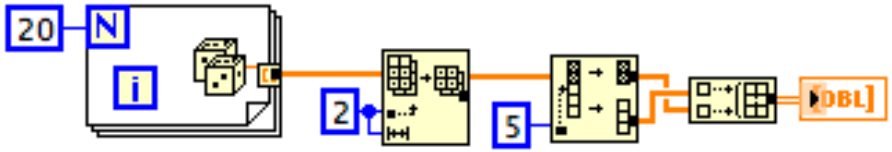
In LabVIEW 2017 64bit



2147483647 = MaxInt = 2^{31}-1 ~ 2 GBytes

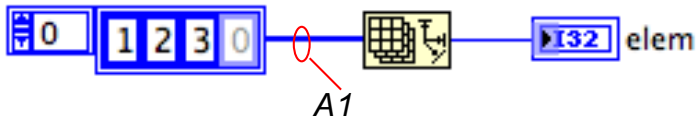
LabVIEW 2017-64bit can handle such size

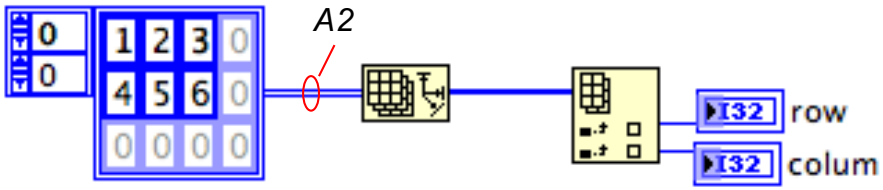
- Utilisez quand c'est possible des primitives *in-place* qui ne travaillent directement sur les tableaux sans les copier
- *Check the Buffer Allocations,* Tools»Profile»Show Buffer Allocations



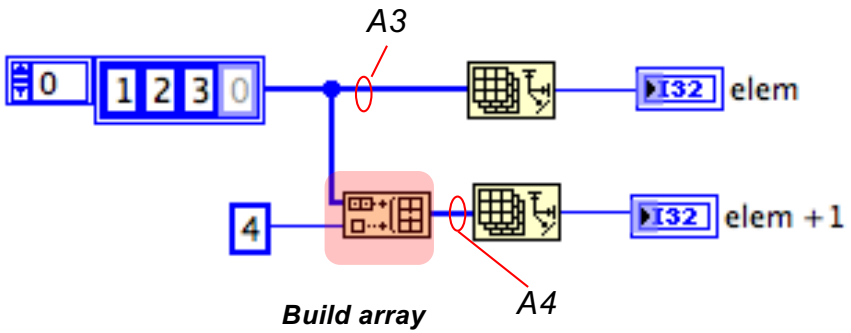
Array functions

Les tableaux LabVIEW connaissent leur(s) taille(s)





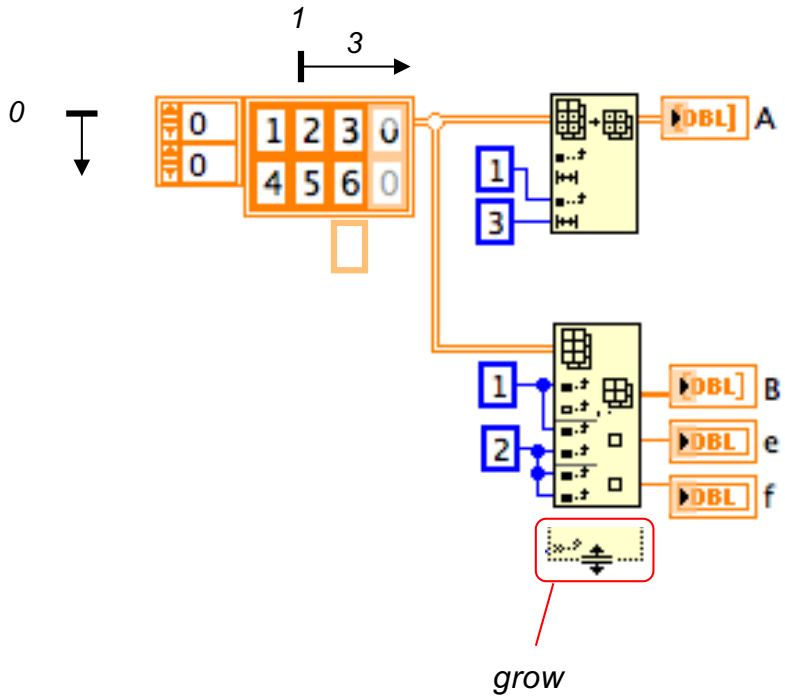
Les tableaux LabVIEW sont dynamiques




```
int A1[3] = {1,2,3};

elem = ArraySize(A1);



int A2[3] = {{1,2,3},
{4,5,6}};

row = Get(ArraySize(A2),0);

Colum = Get(ArraySize(A2),1);



int A3[3] = {1,2,3};

elem = ArraySize(A3);

A4= BuildArray(A3,4);

elem_1 = ArraySize(A4);
```

Array functions

Accès aux éléments du tableau équivalent aux [ ] en c



A[]={{2,3}, {5,6}}

B[]={4,5,6}

e = 6

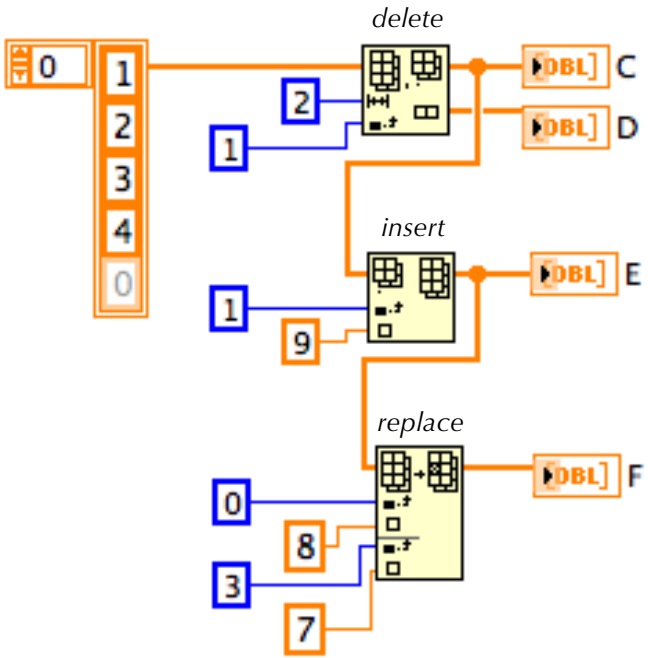
f = 0

Unconnected inputs are set to default (see help window for default values)

```
int Array[6] = {{1,2,3},
                {4,5,6}};
A = GetSubArray(Array,def,all,
                          1,3);


B = GetIndexArray(Array, 1,
all);
e = GetIndexElem(Array, 1, 2);
f = GetIndexElem(Array, def,
2);
```

Array functions

Delete/insert/replace array elements



C[] = {1,4}

D[] = {2,3}

E[] = {1,9,4}

F[] = {8,9,4}

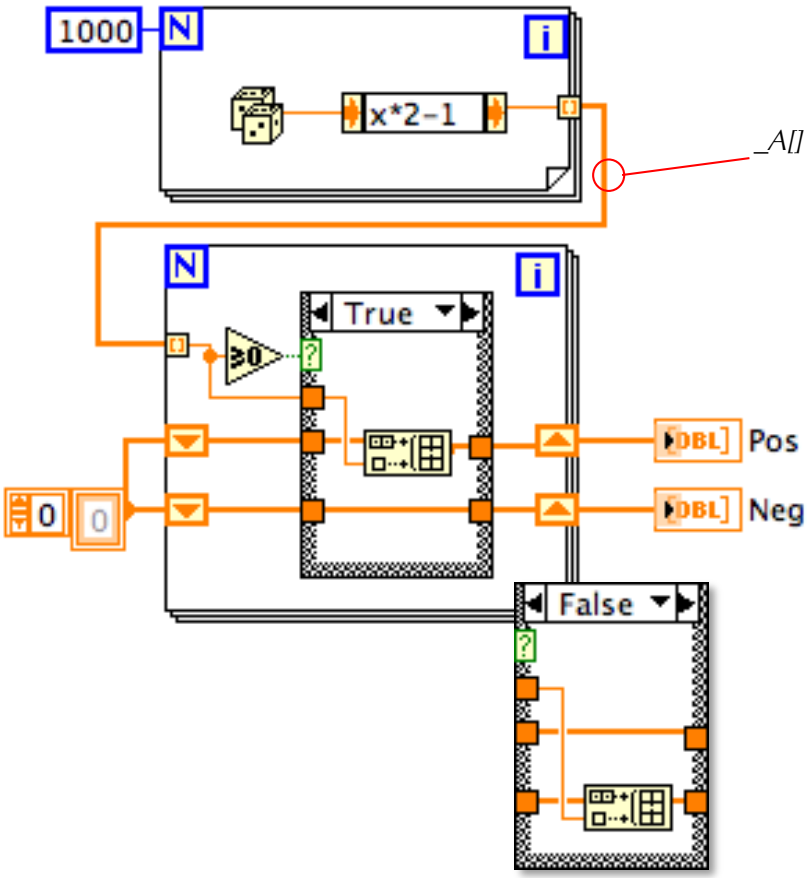
```
int Array[4] = {1,2,3,4};

Delete(Array,2,1,C,D);


Insert(C,1,9.0,E);


Replace(E,0,8.0,F);

Replace(F,3,7.0,F);
```

Demo

- Split array
  - With SR
  - With conditional indexing
  - Sort + threshold

Array functions

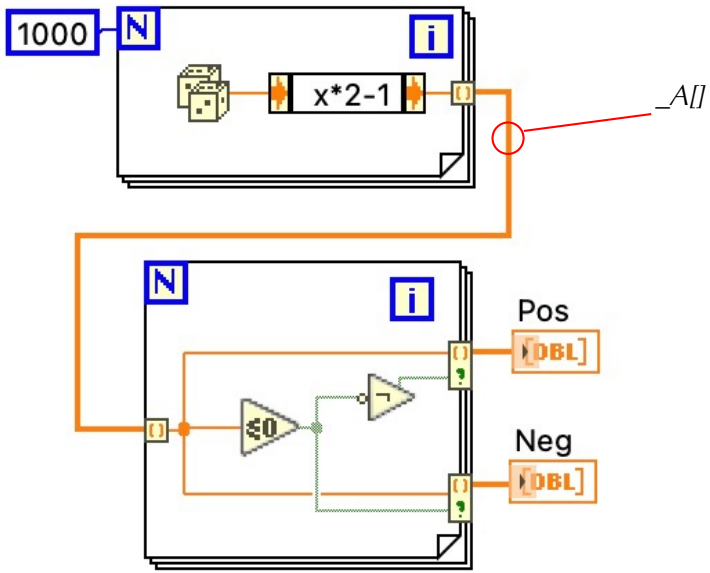
Ex: sort in 2 arrays positive from negative values




```
double  Pos[] = {};
double  Neg[] = {};
double  _A[];

for (i=0; r<1000; i++)
    _A[i]=rand() * 2 - 1;

for (j=0; j<size(_A),j++) {
  if (_A[j]≥ 0)
         AppendArray(Pos, _A[j]);
      else
         AppendArray(Neg, _A[j]);
```

Array functions

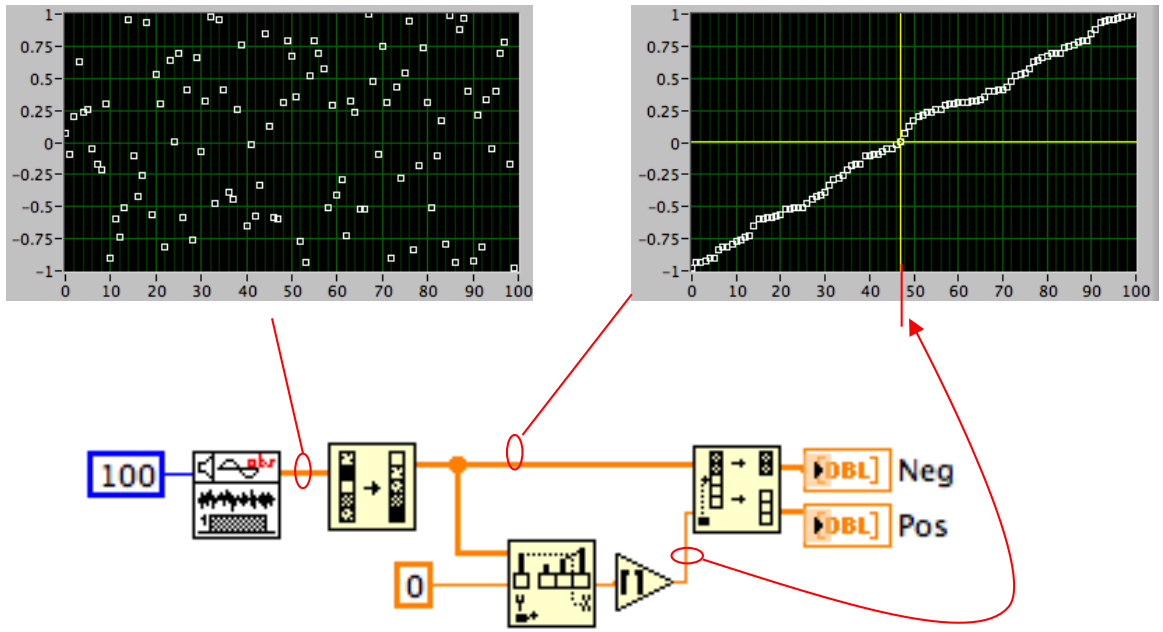
Ex: sort in 2 arrays positive from negative values




```
double  Pos[] = {};
double  Neg[] = {};
double  _A[];

for (i=0; r<1000; i++)
    _A[i]=rand() * 2 - 1;

for (j=0; j<size(_A),j++) {
  if (_A[j]<= 0)
        indexArray(Neg, _A[j]);
      else
        indexArray(Pos, _A[j]);
```

Array functions

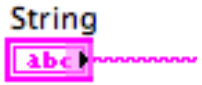
Ex: sort positive from negative values, alternative solution (values indexes differ)




```
double A[100];

double B[100];

double Neg[], Pos[];

Double  T=0;

int i;


GenRand(100, &A);

Sort(A, &B);

for (i=0, i<SizeArray(B), i++){

   if (B[i]<T) && (B[i+1]>=T))

        break;

   i++;

}

Split(B, i, &Neg, &Pos);
```

Strings

- Les *strings* LabVIEW sont comme des string en C/C++.
- Les *strings* peuvent être redimensionnée dynamiquement, LabVIEW se charge de la gestion de la mémoire, elles ont une taille limites de 2^{32} ~4Millard de chars
- Les *strings* LabVIEW connaissent leur taille
- Un caractère est une *string* d'un élément, un caractère peut être représenté comme un unsigned 8bits Integer ou [U8].
- Les *strings* peuvent être employées comme des *streams* de bytes



C

```
typedef struct
LVString {
  long size;
  char[] string;
}
```

or C++

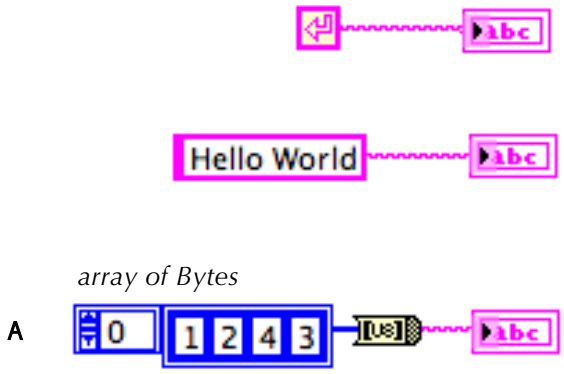
```
String string;
```

Demo

Strings

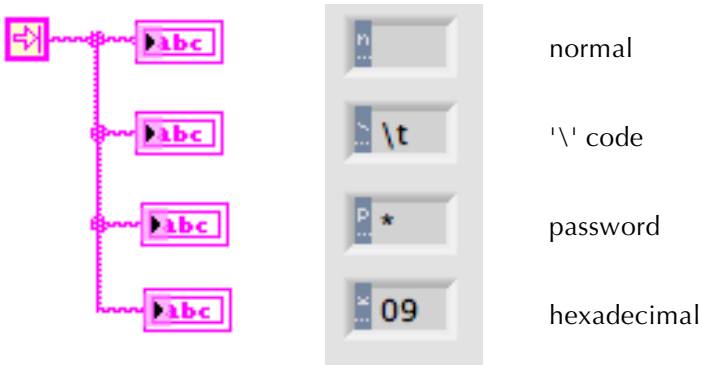
String functions

Initialize Strings



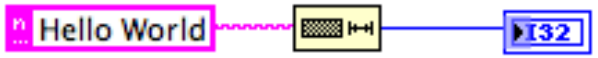

```
String S[] = '\r';

String S[] = 'Hello World';


UInt8 A[] = {1,2,4,3};

String S[] = (String)A;
```

Display formats



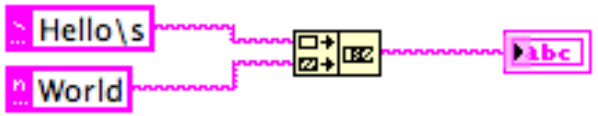
String functions

String Length



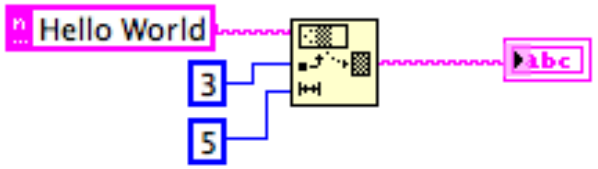

```
Length = 11
```

Concatenate



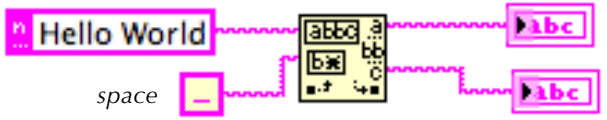

```
S = "Hello World"
```

String Subset



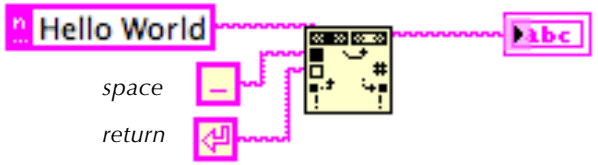

```
S = "lo Wo"
```

Match Pattern




```
S1 = "Hello"
S2 = "World"
```

Search & Replace

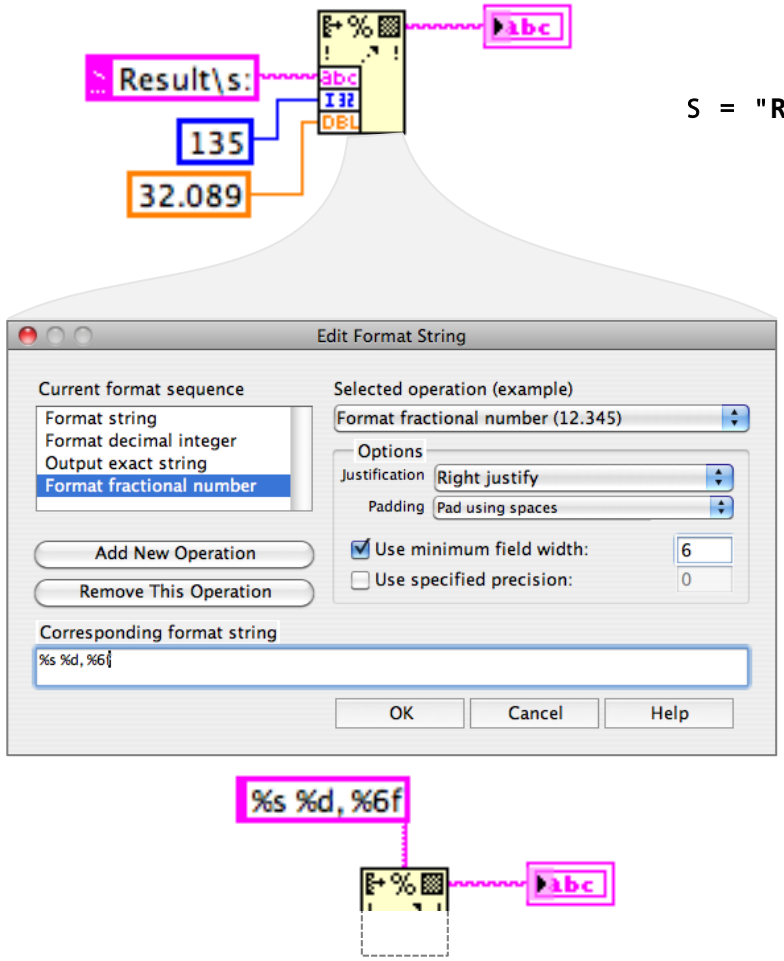



```
S = "Hello
     World"
```

```
Length = sizeof("Hello World");


S = S + "Hello " + "World";
strcat(S,"Hello ");
strcat(S,"World");


 S = StrSubset("Hello World",3,5);


 S[2] = SplitStr("Hello World","\s");


 S = ReplaceStr("Hello World","\s","\r");
```

String functions

Format Into String




```
S = "Result : 135, 32.089000"
```

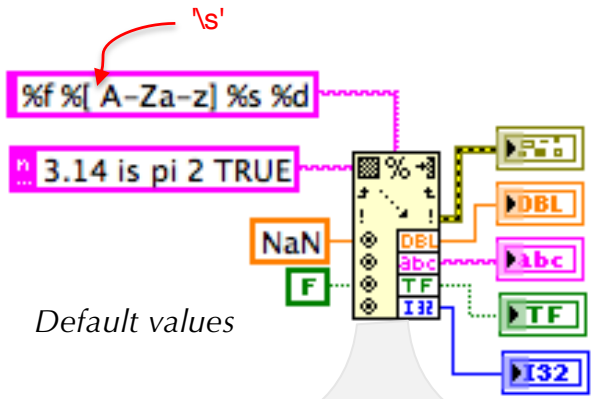
Right – click la fonction **Format into String** *pour éditer précisément le format de la conversion*

String format will be automatically added to the node

```
sprintf(S,"Result :%d, %6f",
                        135,32.089);
```

String functions

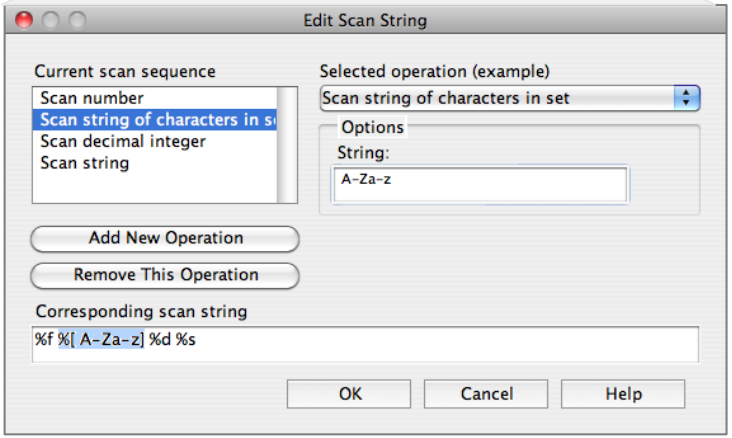
Scan from String



Default values

```
Error = "Scan From String (arg 4) in Strings.vi"
D = 3.14
S = "is pi"
B = FALSE
I = 0
```

Error because the last two elements are inverted



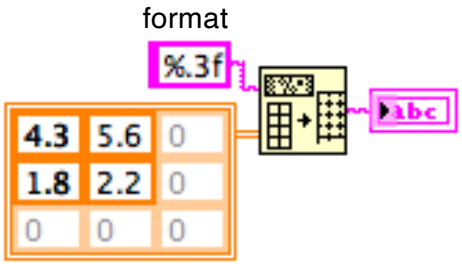
Right – click la fonction **Format into String** *pour éditer précisément le format de la conversion*

```
cin >> D >> S >> B >> I;

If (cin.fail()) ...


sscanf(S,"%f %[ A-Za-z] %s %d",
                    &D, S, &B, &i);
```

Use %s *for a single string (no delimiter like* \s, \r, *etc) otherwise specify the allowed characters in* [], *in the above example* [\s A-Za-z]

String functions

Array to Spreadsheet String

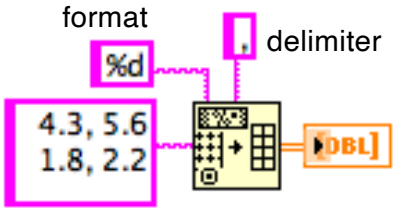



```
S = "4.300 5.600
     1.800 2.200"
```

%.3f -> double with 3 digits after '.'

```
Double A = {{4.3, 5.6} {1.8, 2.2}};
String F = "%.3f";
S = ArrayToSpreadSheetStr(A,F);
```

Spreadsheet String to Array



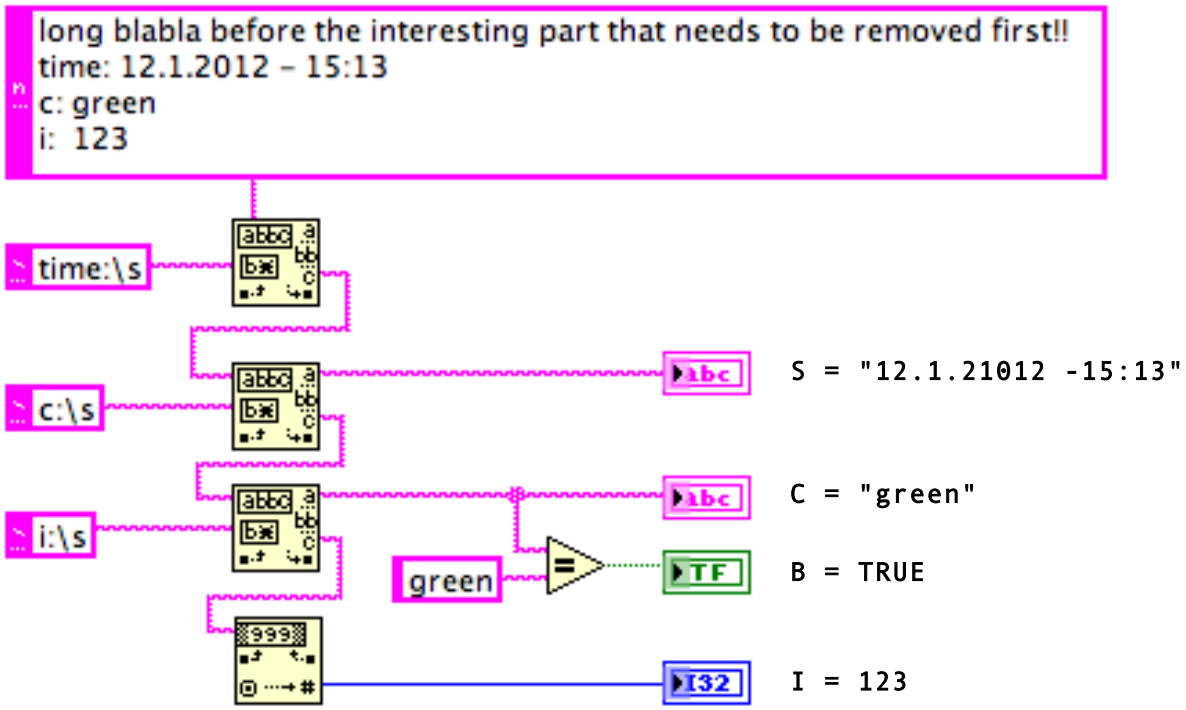

```
A = {{ 4   6 }
     { 2   2 }}
```

%d -> convert into integer (IEEE 754)

```
String S = "4.3, 5.6\r1.8,2.2";
String F = "%d";
String D = ',';
A = SpreadSheetStrToArray(S,F,D);
```

String functions

Example of string parsing

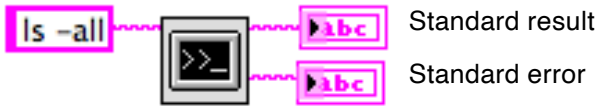


Decimal String to number

```
String LS = "long blabla before the
   interesting part that needs to be
   removed first!!\rtime: 12.1.2012 -
   15:13\rc: green\ri:  123"


MatchPattern(LS, "time:\s",
                 NULL,NULL,After);

MatchPattern(After, "c:\s",
                 S,NULL,After2);

MatchPattern(After2, "i:\s",
                 C,NULL,After3);

B = (C=="green");

atoi(After3,I)
```

Talking to the Unix OS

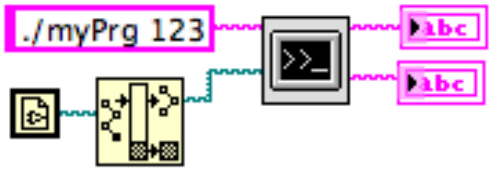
- `System Exec` fourni un accès aux fonctions de l'OS
- L'effet est similaire à l'entrée de commandes dans le terminal/CMD
- Les sorties standard (cout) et d'error (cerr) sont retournées




```
Applications
Developer
Library
Network
System
Users
…
private
sbin
tmp
usr
var
```

Note: on Windows, prepend '**`cmd /c`** to you call, ex: '**`cmd /c dir C:`**'

Talking to the Unix OS




```
argv[1]: 123
no error!
```

The VI directory

Launch a program called "myPrg" with '123' as parameter.

The program and the calling VI are located in the same directory.

```
// -- myPrg.c --

#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* const argv[])
{
    printf("argv[1]: %s\n",argv[1)";
    fprintf(stderr,"no error!\n")              ;

    return 0;
}
```

```
> cd my/vi/path
> ./myPrg 123
argv[1]: 123
no error!
>
```