Como estender o Pure Data através de *externals* em C

Flávio Luiz Schiavoni
fls@ime.usp.br

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

10 de junho de 2014

Atenção!

Esta apresentação, os exemplos utilizados e o tutorial estão disponíveis nos seguintes endereços:

`http://www.ime.usp.br/~fls`
`https://github.com/flschiavoni/pd-external-tutorial`

Objetivo da apresentação

Introduzir ao desenvolvimento de *externals* para Pure Data, considerando:

- Compilação.
- Estrutura do código.
- Exemplos passo a passo.

Objetivo da apresentação

Introduzir ao desenvolvimento de *externals* para Pure Data, considerando:

- Compilação.
- Estrutura do código.
- Exemplos passo a passo.

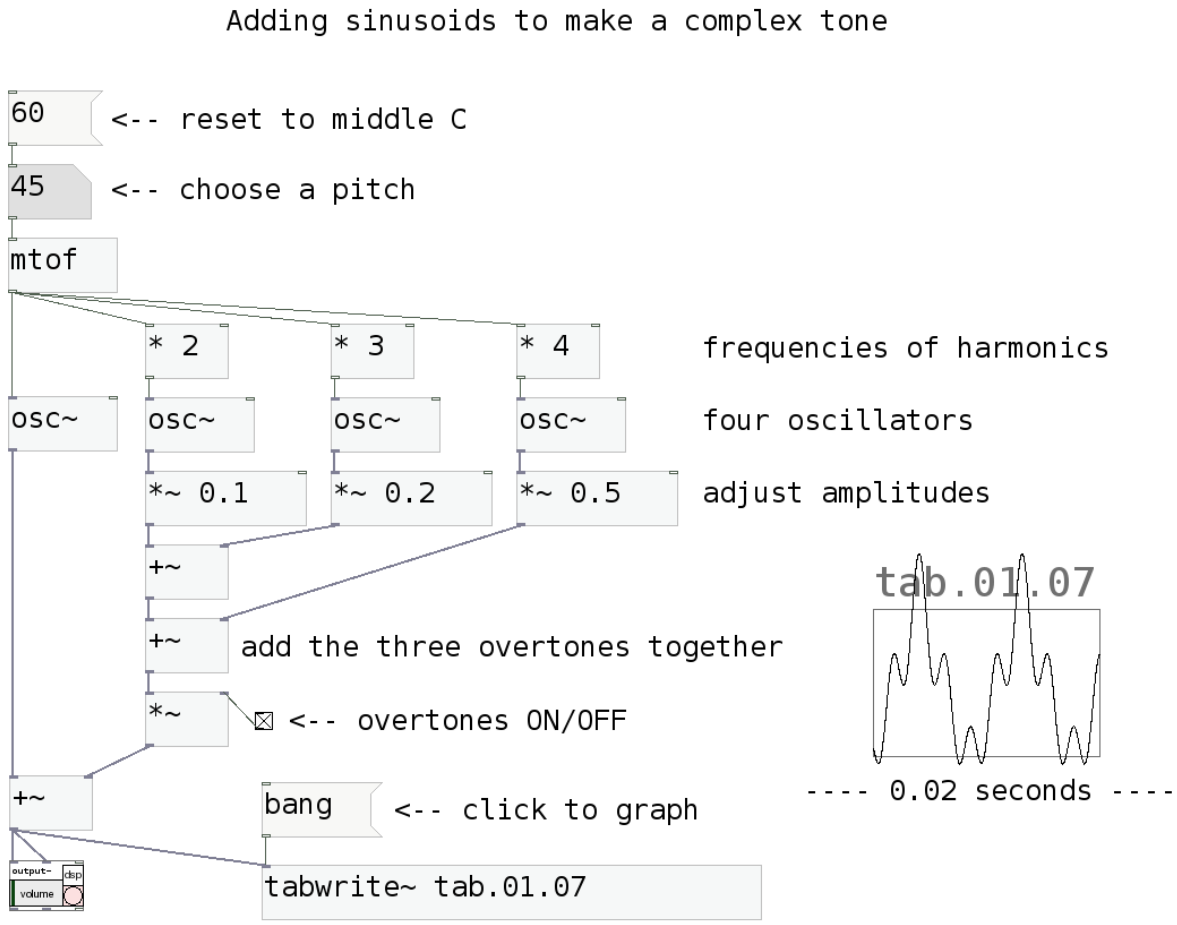
Observação: não são objetivos desta apresentação:

- Ensinar a programar em C.
- Ensinar algoritmos de processamento de sinais.
- Ensinar a usar Pure Data.

Estrutura da apresentação

Estrutura da apresentação

Pure Data


Adding sinusoids to make a complex tone
60
<-- reset to middle C
45
<-- choose a pitch
mtof
* 2
* 3
* 4
frequencies of harmonics
osc~
osc~
osc~
osc~
four oscillators
*~ 0.1
*~ 0.2
*~ 0.5
adjust amplitudes
+~
+~
add the three overtones together
*~
<-- overtones ON/OFF
+~
bang
<-- click to graph
output~
dsp
volume
tabwrite~ tab.01.07
tab.01.07
---- 0.02 seconds ----

Formas de estender o Pure Data

Existem algumas formas de estender as funcionalidades do Pure Data:

- Subpatches.
- Externals.
- Alterações no código-fonte (C e Tcl/Tk).

Formas de estender o Pure Data

Existem algumas formas de estender as funcionalidades do Pure Data:

- Subpatches.
- Externals.
- Alterações no código-fonte (C e Tcl/Tk).

Este seminário trata da extensão do Pure Data através da criação de *externals* em C.

Organização do código-fonte

Algumas informações sobre o código do Pure Data:

- Publicado sob a licença Standard Improved BSD License.
- Organizado de forma orientada a objetos.
  - Classes são tipos.
  - Objetos (gráficos) do Pure Data são instâncias de classes. dados e assinaturas de funções: `m_pd.h`.
- Existe um arquivo de cabeçalho com constantes, tipos, estruturas de dados e assinaturas de funções: `m_pd.h`.

Compilação

```
1 EXTNAME=meu_external
2 cc -DPD -fPIC -Wall -o ${EXTNAME}.o -c ${EXTNAME}.c
3 ld -shared -lc -lm -o ${EXTNAME}.pd_linux ${EXTNAME}.o
4 rm ${EXTNAME}.o
```

O Pure Data procura por objetos com nomes diferentes em cada sistema:

- `meu_external.pd_linux` (GNU/Linux).
- `meu_external.pd_irix5` (Irix 5).
- `meu_external.pd_darwin` (Mac OS X).
- `meu_external.dll` (MS Windows).

Arquivos de ajuda

Um arquivo de ajuda é um patch do Pure Data com:

- Um nome informativo: `meu_external-help.pd`.
- Instruções de uso.
- Exemplos de utilização.

A seguinte função associa um arquivo de ajuda a uma classe de *external*:

```
1 class_sethelpsymbol(meu_external_class,
      gensym("meu_external_class-help"));
```

Utilizando *externals*

Passos para utilizar um *external* no Pure Data:

1. Escreva um arquivo `.c` com as funções, classes e métodos.
2. Compile o código-fonte para criar uma biblioteca compartilhada.
3. Informe ao Pure Data o caminho para o *external* através de uma das opções abaixo:
   - Na criação do objeto, insira o caminho completo (relativo ou absoluto) para o objeto do *external*.
   - Na linha de comando, utilize a opção `-path <caminho>`.
   - Na interface gráfica, acesse a opção `File` → `Path...`.
4. Crie um objeto na interface gráfica do Pd com o nome do arquivo do *external*, sem a extensão.

Utilizando *externals*

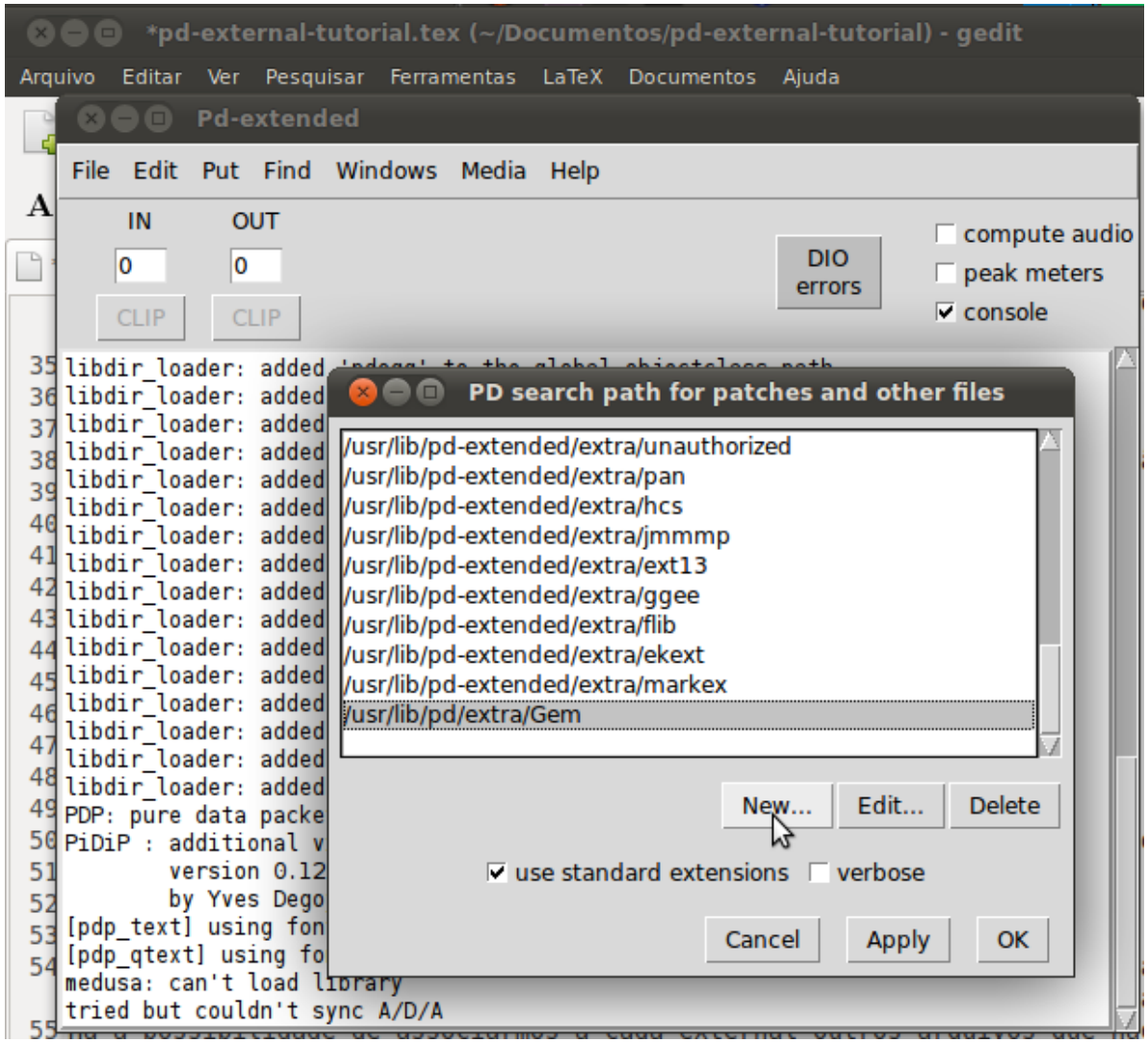


Figura : Adicionando o diretório de um *external* ao caminho de busca do Pure Data.

Estrutura da apresentação

O mínimo para um *external* (1/2)

1. Um tipo e uma estrutura que representam uma classe:

```
1  static t_class *example1_class;
2
3  typedef struct _example1 {
4    t_object x_obj;
5  } t_example1;
```

2. Uma função de configuração da classe:

```
6  void example1_setup(void) {
7    example1_class = class_new(
8      gensym("example1"),           // Nome simbolico
9      (t_newmethod) example1_new, // Construtor
10     0,                            // Destrutor
11     sizeof (t_example1),          // Atributos
12     CLASS_NOINLET,                // Flags da classe
13     0                             // Tipos
14   );
15 }
```

O mínimo para um *external* (2/2)

3. Um método construtor:

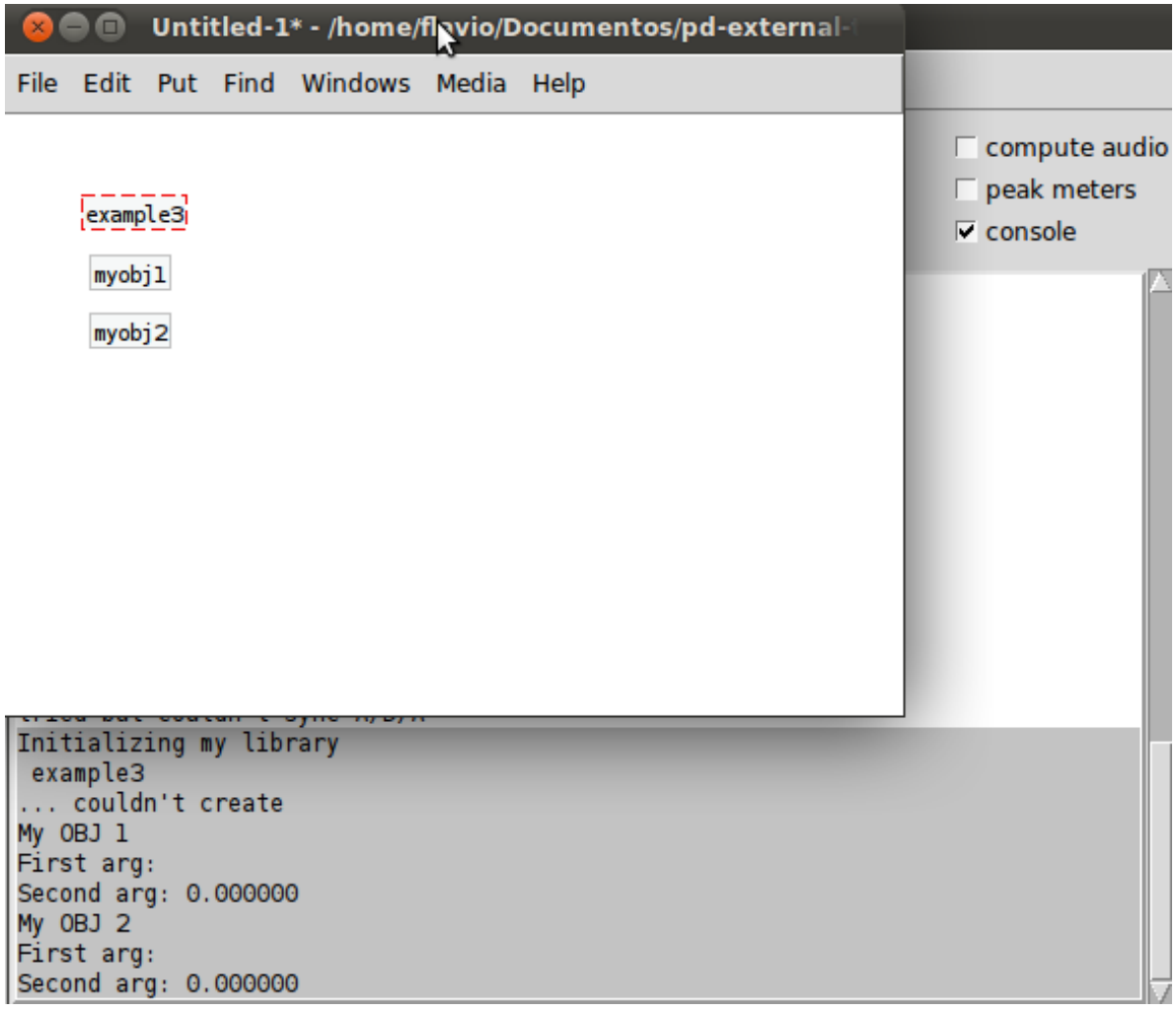
```
16 // Construtor da classe
17 void * example1_new(void) {
18     t_example1 *x = (t_example1 *)
           pd_new(example1_class);
19     return (void *) x;
20 }
```

Bibliotecas (1/2)

Podemos definir diversas classes numa mesma função de inicialização:

```
1  void example2_setup(void) {
2    post("Initializing my library");
3
4    myobj1_class = class_new(
5      gensym("myobj1"),
6      (t_newmethod) myobj1_new, // Constructor
7      0,
8      sizeof (t_myobj1),
9      CLASS_NOINLET,
10     0);
11
12   myobj2_class = class_new(
13     gensym("myobj2"),
14     (t_newmethod) myobj2_new, // Constructor
15     0,
16     sizeof (t_myobj2),
17     CLASS_NOINLET,
18     0);
19 }
```

Bibliotecas (2/2)



Variáveis globais

Variáveis globais são acessadas por todas as instâncias de objetos e funções do *external*:

```
1  int count = 0;
2  void * example4_new(void) {
3      t_example4 *x = (t_example4 *)
           pd_new(example4_class);
4      post("Counter value: %d",count);
5      count++;
6      return (void *) x;
7  }
```

Variáveis de instância

Variáveis declaradas dentro da estrutura de uma classe são como atributos dos objetos:

```
1  static t_class *example_class;
2
3  typedef struct _example {
4      t_object x_obj;
5      t_int counter;
6  } t_example;
7
8  void * example_new(void) {
9      t_example *x = (t_example *)pd_new(example_class);
10     post("Counter value: %d",x->counter);
11     x->counter++;
12     return (void *) x;
13 }
```

Estrutura da apresentação

Tipos definidos no arquivo `m_pd.h`

tipo do pd	descrição
`t_atom`	átomo
`t_float`	valor de ponto flutuante
`t_symbol`	símbolo
`t_gpointer`	ponteiro (para objetos gráficos)
`t_int`	valor inteiro
`t_signal`	estrutura de um sinal
`t_sample`	valor de um sinal de áudio (ponto flutuante)
`t_outlet`	*outlet* de um objeto
`t_inlet`	*inlet* de um objeto
`t_object`	objeto gráfico
`t_class`	uma classe do pd
`t_method`	um método de uma classe
`t_newmethod`	ponteiro para um construtor (uma função `_new`)

Símbolos

Um símbolo é um valor constante de uma *string*:

- Os símbolos são mantidos em uma tabela por razões de performance.
- A função `gensym(char *)` faz a busca/criação de símbolos.

Mensagens

Uma mensagem é composta de um seletor e uma lista de átomos.

Seletores:

seletor	rotina de busca	endereço de busca
`bang`	`gensym("bang")`	`&s_bang`
`float`	`gensym("float")`	`&s_float`
`symbol`	`gensym("symbol")`	`&s_symbol`
`pointer`	`gensym("pointer")`	`&s_pointer`
`list`	`gensym("list")`	`&s_list`
`--` (signal)	`gensym("signal")`	`&s_signal`

Tipos de átomos:

- `A_FLOAT`: um valor numérico (de ponto flutuante).
- `A_SYMBOL`: um valor simbólico (string).
- `A_POINTER`: um ponteiro.

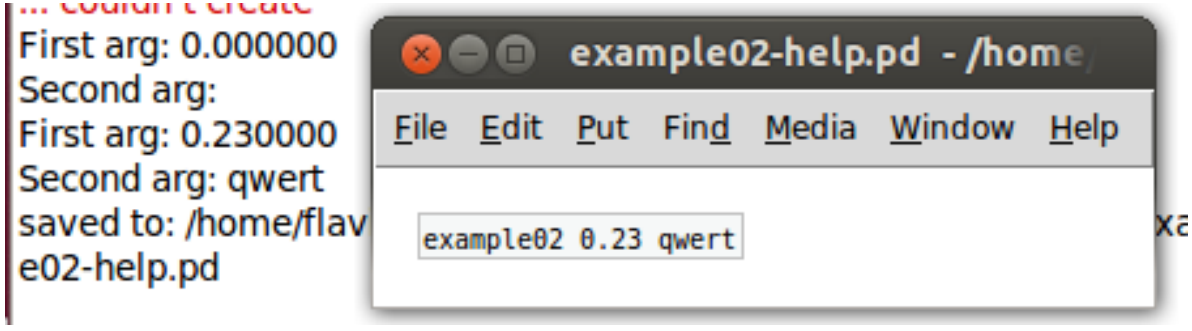
Outros tipos de átomo

Existem diversos tipos de átomo definidos pelo arquivo `m_pd.h` que podem ser utilizados para passagem de parâmetros:

- `A_NULL`
- `A_FLOAT`
- `A_SYMBOL`
- `A_POINTER`
- `A_SEMI`
- `A_COMMA`
- `A_DEFFLOAT`
- `A_DEFSYM`
- `A_DOLLAR`
- `A_DOLLSYM`
- `A_GIMME`
- `A_CANT`

Estrutura da apresentação

Construtor com parâmetros (1/2)



Construtor com parâmetros (2/2)

```
1  // Constructor of the class
2  void *example2_new(t_symbol * arg1, t_floatarg arg2) {
3      t_example2 *x = (t_example2 *)
           pd_new(example2_class);
4      post("First arg: %s", arg1->s_name);
5      post("Second arg: %f", arg2);
6      return (void *) x;
7  }
8
9  void example2_setup(void) {
10   example2_class = class_new(gensym("example2"),
11     (t_newmethod) example2_new, // Constructor
12     0,
13     sizeof (t_example2),
14     CLASS_NOINLET,
15     A_DEFFLOAT, // Constructor parameter 1
16     A_DEFSYMBOL, // Constructor parameter 2
17     0);
18 }
```

Construtor com A_GIMME (1/2)

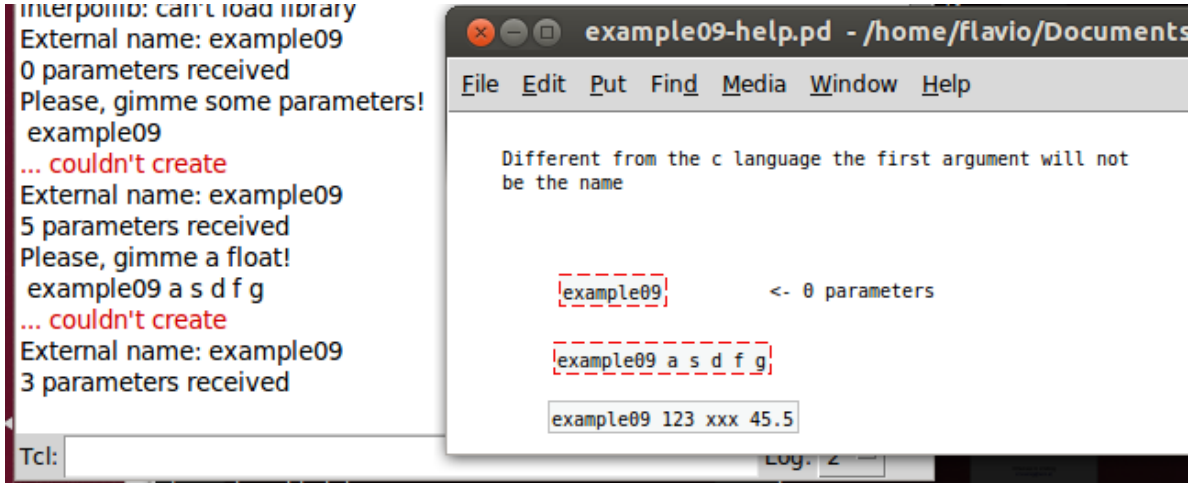


Figura : Diferente da linguagem C, o primeiro parâmetro não é o nome do external.

Só é possível utilizar 5 parâmetros com tipos definidos. Para passar mais parâmetros, é necessário utilizar o A_GIMME.

Construtor com A_GIMME (2/2)

```
1  // Constructor of the class
2  void *example9_new(t_symbol *s,
3                     int argc, t_atom * argv) {
4      t_example9 *x = (t_example9 *)
           pd_new(example9_class);
5      post("%d parameters received",argc);
6      return (void *) x;
7  }
8
9  void example9_setup(void) {
10     example9_class = class_new(gensym("example9"),
11       (t_newmethod) example9_new, // Constructor
12       (t_method) example9_destroy, // Destructor
13       sizeof (t_example9),
14       CLASS_NOINLET,
15       A_GIMME, // Allows various parameters
16       0); // LAST argument is ALWAYS zero
17 }
```

Destrutor

```
1  // Destroy the object
2  void example9_destroy(t_example9 *x) {
3    post("You say good bye and I say hello");
4  }
5
6  void example9_setup(void) {
7     example9_class = class_new(gensym("example9"),
8       (t_newmethod) example9_new, // Constructor
9       (t_method) example9_destroy, // Destructor
10      sizeof (t_example9),
11      CLASS_NOINLET,
12      A_GIMME, // Allows various parameters
13      0); // LAST argument is ALWAYS zero
14 }
```

Para liberar memória, podemos utilizar:

```
1  void freebytes(void *x, size_t nbytes)
```

Estrutura da apresentação

Inlets e outlets

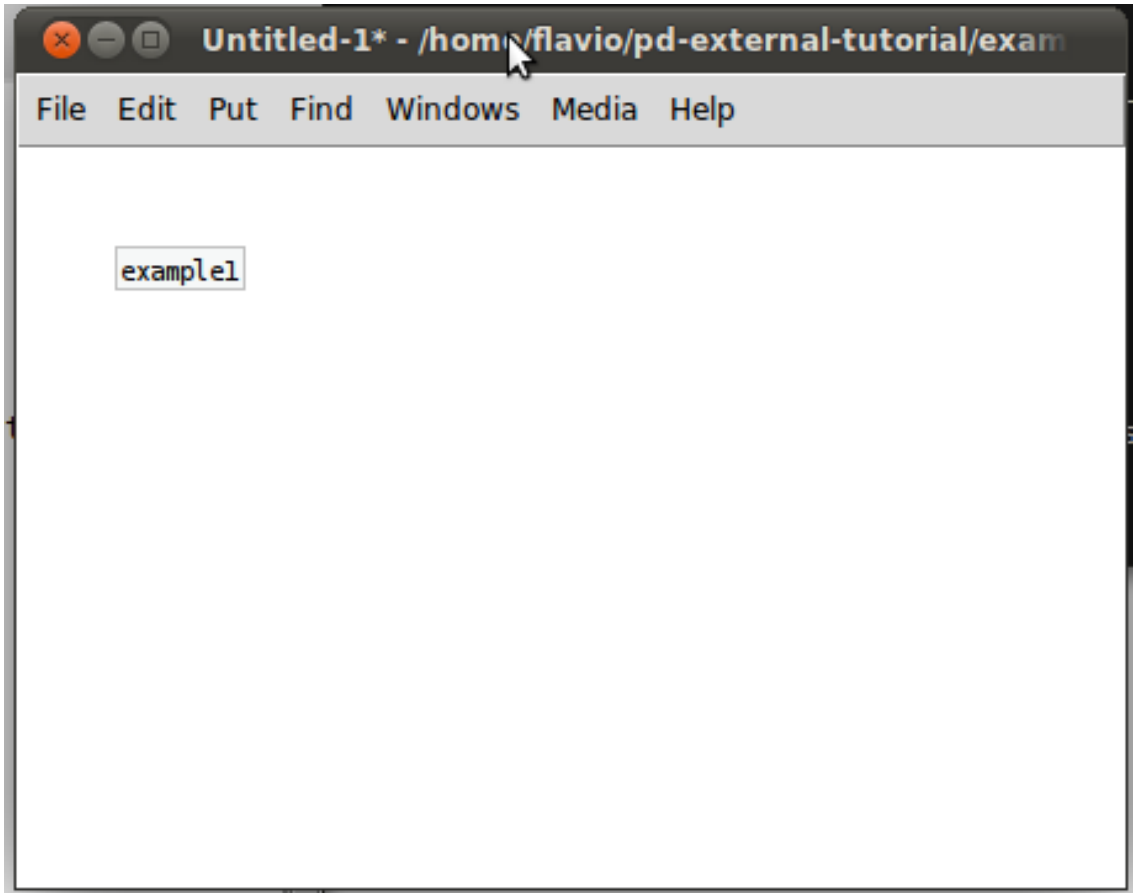
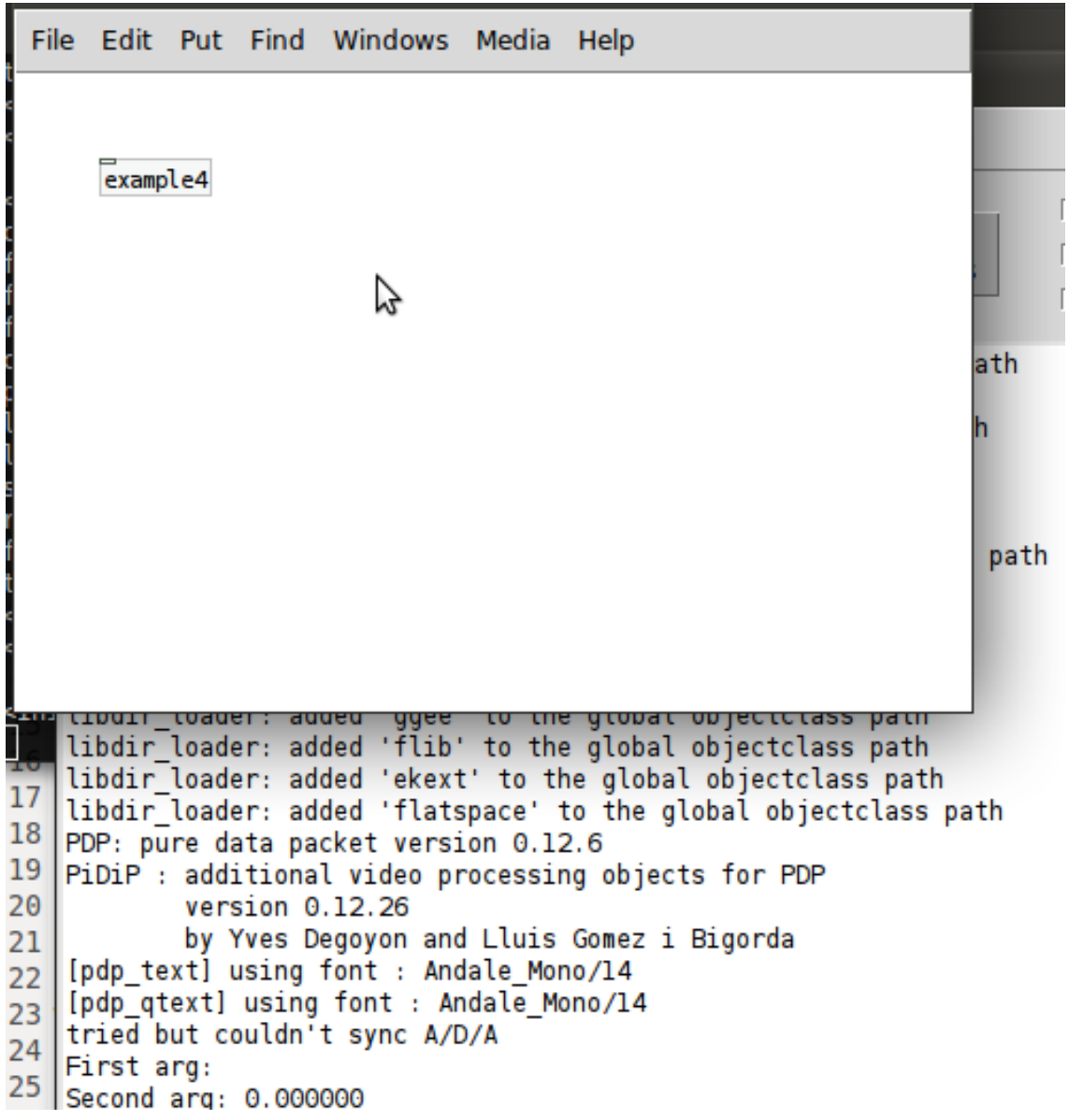


Figura : Para que serve isso!?

Inlets passivos (1/3)



Inlets passivos (2/3)

```
1  static t_class *example4_class;
2
3  typedef struct _example4 {
4    t_object x_obj;
5    t_float my_float;
6  } t_example4;
7
8  // Constructor of the class
9  void *example4_new(t_symbol *arg1, t_floatarg arg2) {
10   t_example4 *x = (t_example4*)pd_new(example4_class);
11   post("First arg: %s", arg1->s_name);
12   post("Second arg: %f", arg2);
13   floatinlet_new(&x->x_obj, &x->my_float);
14   return (void *) x;
15 }
```

Inlets passivos (3/3)

As funções para criar inlets passivos dos tipos mais comuns são:

- `floatinlet_new(t_object *owner, t_float *fp)`
- `symbolinlet_new(t_object *owner, t_symbol **sp)`
- `pointerinlet_new(t_object *owner, t_gpointer *gp)`

Inlets ativos (1/2)

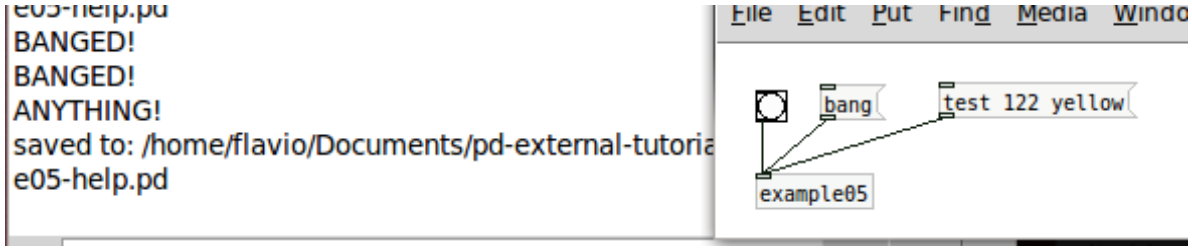
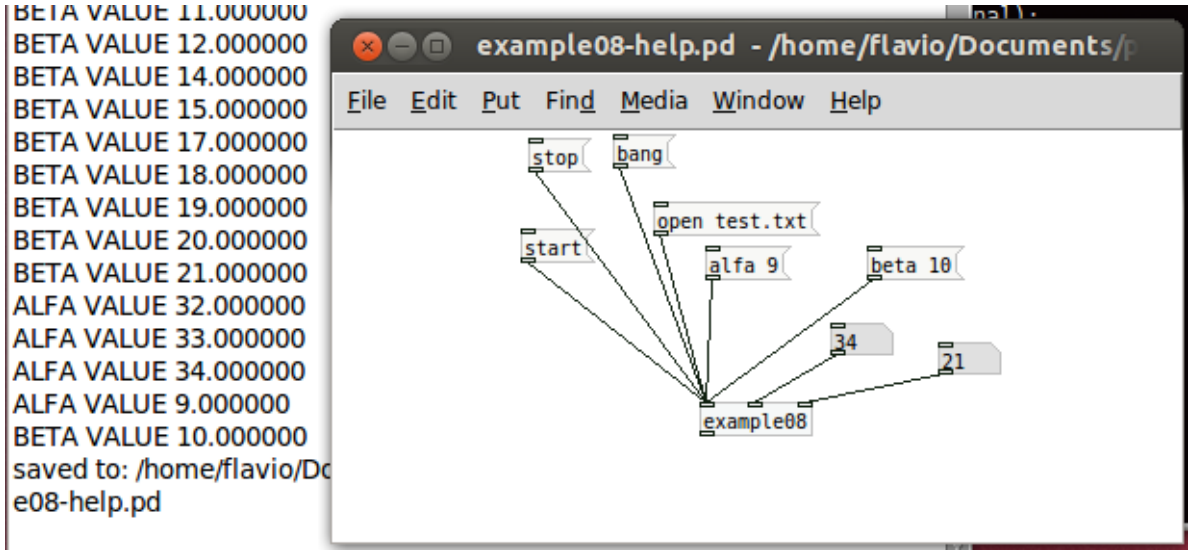


Figura : Inlets ativos.

Inlets ativos (2/2)

```
1  // inlet-methods receive the object as first argument.
2  void example5_bang(t_example5 *x) {
3    post("BANGED!");
4    post("My_float value: %f",x->my_float);
5  }
6
7  void example5_anything(t_example5 *x, t_symbol *s,
       int argc, t_atom *argv){
8    post("ANYTHING!");
9  }
10
11 void example5_setup(void) {
12   example5_class = class_new(gensym("example5"),
13     (t_newmethod) example5_new, // Constructor
14     0,  sizeof (t_example5), CLASS_DEFAULT,
15     0); // LAST argument is ALWAYS zero
16   class_addbang(example5_class, example5_bang);
17   class_addanything(example5_class,
         example5_anything);
18 }
```

Tratamento de mensagens (1/4)



Tratamento de mensagens (2/4)

```
1  void example08_setup(void) {
2    example08_class = class_new(gensym("example08"),
3      (t_newmethod) example08_new, // Constructor
4      (t_method) example08_destroy, // Destructor
5      sizeof (t_example08), CLASS_DEFAULT, 0);
6    // leftmost inlet will trigger methods upon
7    // these messages.
8    class_addmethod(example08_class, (t_method)
         example08_start, gensym("start"), 0);
9    class_addmethod(example08_class, (t_method)
         example08_start, gensym("bang"),  0);
10   class_addmethod(example08_class, (t_method)
         example08_stop, gensym("stop"),  0);
11   class_addmethod(example08_class, (t_method)
         example08_open, gensym("open"),  A_DEFSYMBOL,0);
12   // associate messages with inlets 2 and 3
13   class_addmethod(example08_class, (t_method)
         example08_alfa, gensym("alfa"), A_DEFFLOAT,0);
14   class_addmethod(example08_class, (t_method)
         example08_beta, gensym("beta"), A_DEFFLOAT,0);
15 }
```

Tratamento de mensagens (3/4)

```
16 // Constructor of the class
17 void * example08_new(void) {
18   t_example08 *x = (t_example08 *)
         pd_new(example08_class);
19   // creates inlets for distinct messages
20   inlet_new(&x->x_obj, &x->x_obj.ob_pd,
         gensym("float"), gensym("alfa"));
21   inlet_new(&x->x_obj, &x->x_obj.ob_pd,
         gensym("float"), gensym("beta"));
22   x->my_outlet = outlet_new(&x->x_obj,
         gensym("bang"));
23   return (void *) x;
24 }
```

Tratamento de mensagens (4/4)

```
25 void example08_start(t_example08 *x){
26   post("START / BANG");
27 }
28
29 void example08_stop(t_example08 *x){
30   post("STOP");
31 }
32
33 void example08_open(t_example08 *x, t_symbol *s){
34   post("open %s",s->s_name);
35 }
36
37 void example08_alfa(t_example08 *x, t_floatarg f){
38   post("ALFA VALUE %f",f);
39 }
40
41 void example08_beta(t_example08 *x, t_floatarg f){
42   post("BETA VALUE %f",f);
43 }
```

Outlets (1/4)

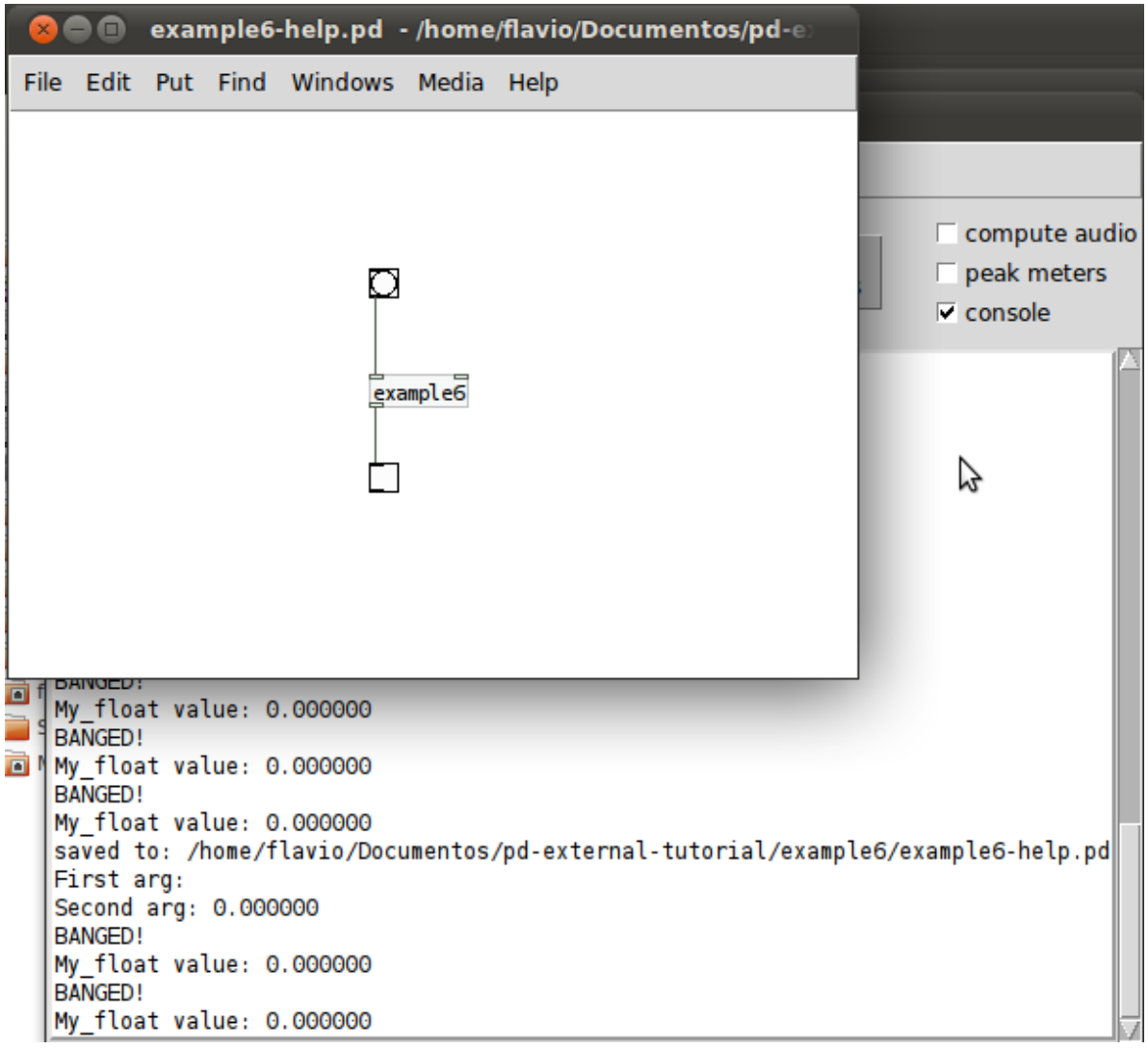


Figura : Um external bem útil que recebe um bang e envia um bang.

Outlets (2/4)

```
1    x->my_outlet = outlet_new(&x->x_obj,
         gensym("bang"));
2    return (void *) x;
3  }
4
5  void example06_setup(void) {
6    example06_class = class_new(gensym("example06"),
7      (t_newmethod) example06_new, // Constructor
8      0, sizeof (t_example06), CLASS_DEFAULT,
9      A_DEFFLOAT,  // 1st constructor parameter
10     A_DEFSYMBOL, // 2nd constructor parameter
11     0); // LAST argument is ALWAYS zero
```

Outlets (3/4)

```
12      (void)argv;
13      post("ANYTHING!");
14  }
15
16  // Constructor of the class
17  void * example06_new(t_symbol * arg1, t_floatarg
        arg2) {
18    t_example06 *x = (t_example06 *)
          pd_new(example06_class);
19    post("First arg: %s", arg1->s_name);
20    post("Second arg: %f", arg2);
21    // passive inlet
22    floatinlet_new(&x->x_obj, &x->my_float);
```

Outlets (4/4)

```
23 typedef struct _example06 {
24   t_object x_obj;
25   t_float my_float;
26   t_outlet *my_outlet; // defines an outlet
27 } t_example06;
28
29 // The BANG method, first inlet
30 void example06_bang(t_example06 *x) {
31   post("BANGED!");
32   post("my_float value: %f",x->my_float);
33   outlet_bang(x->my_outlet); // Bang my outlet
34 }
35
36 void example06_anything(t_example06 *x, t_symbol *s,
       int argc, t_atom *argv){
37    (void)x;
38    (void)s;
```

Estrutura da apresentação

Inlets para DSP

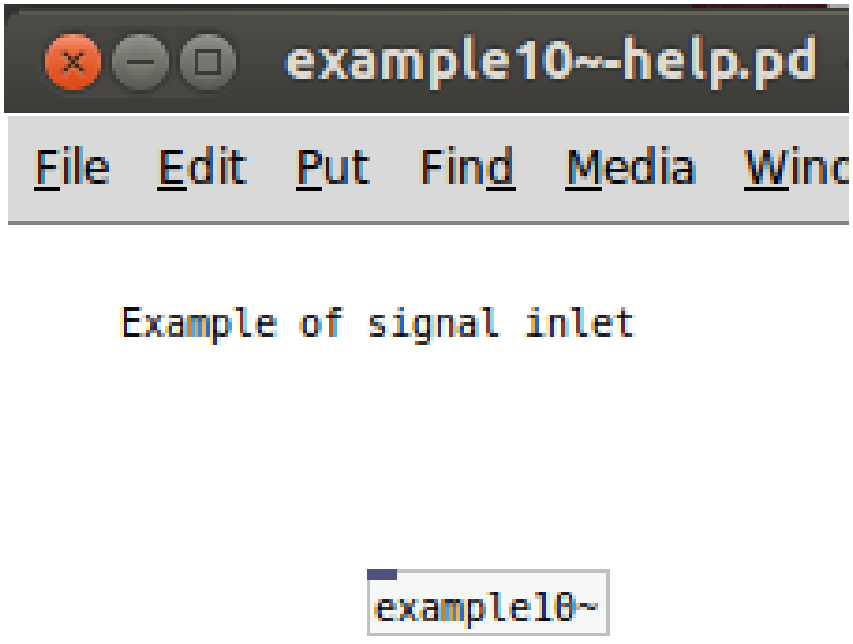


Figura : Primeiro Inlet DSP

Inlets para DSP

Alguns pontos importantes:

- Existem macros para facilitar a definição do primeiro inlet para classes do tipo `CLASS_DEFAULT`.
- Inlets adicionais são instanciados no método `_new()`.

Código para o primeiro inlet DSP (1/2)

Estrutura e métodos _perform() e _dsp()

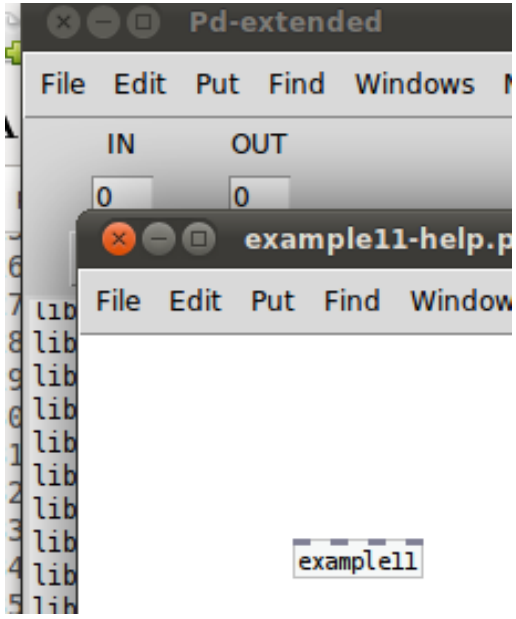
```
1  typedef struct _example10 {
2      t_object x_obj;
3      t_float x_f; /* inlet value when set by message */
4  } t_example10;
5
6  static t_int * example10_perform(t_int *w){
7    t_float      *in = (t_float *) (w[1]);
8    int           n  = (int) (w[2]);
9    t_example10 *x  = (t_example10 *) (w[3]);
10   // do something ...
11   return (w + 4); // next block's address
12 }
13
14 static void example10_dsp(
15           t_example10 *x, t_signal **sp){
16   // adds a method for dsp
17   dsp_add(example10_perform, 3, sp[0]->s_vec,
         sp[0]->s_n, x);
18 }
```

Código para o primeiro inlet DSP (2/2)

Método _setup()

```
1  void example10_tilde_setup(void) {
2    example10_class = class_new(gensym("example10~"),
3      (t_newmethod) example10_new, // Constructor
4      (t_method) example10_destroy, // Destructor
5      sizeof (t_example10),
6      CLASS_DEFAULT,
7      A_GIMME,
8      0);
9    // this declares the leftmost, "main" inlet
10   // as taking signals.
11    CLASS_MAINSIGNALIN(example10_class, t_example10,
          x_f);
12    class_addmethod(example10_class, (t_method)
          example10_dsp, gensym("dsp"), 0);
13 }
```

Inlets DSP adicionais (1/3)



Inlets DSP adicionais (2/3)

Método _new()

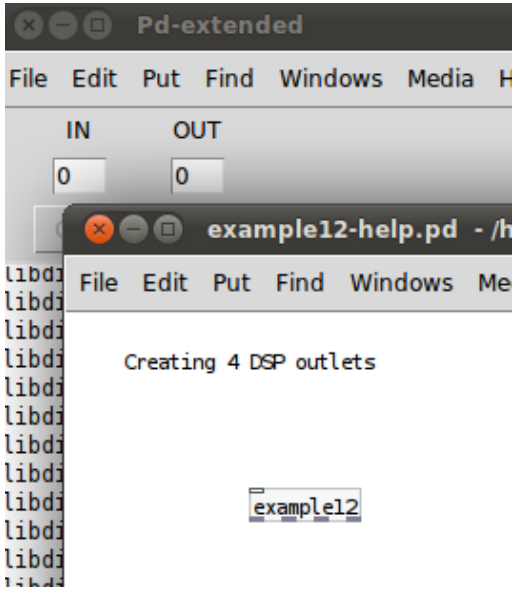
```
1 // Constructor of the class
2 void * example11_new(t_symbol *s, int argc, t_atom *
      argv) {
3     t_example11 *x = (t_example11 *)
          pd_new(example11_class);
4     inlet_new(&x->x_obj, &x->x_obj.ob_pd, &s_signal,
          &s_signal); // second signal inlet
5     inlet_new(&x->x_obj, &x->x_obj.ob_pd, &s_signal,
          &s_signal); // third signal inlet
6     inlet_new(&x->x_obj, &x->x_obj.ob_pd, &s_signal,
          &s_signal); // fourth signal inlet
7     return (void *) x;
8 }
```

Inlets DSP adicionais (3/3)

Métodos _perform() e _dsp()

```
1  static t_int * example11_perform(t_int *w){
2    t_float *in1 = (t_float *)(w[1]);
3    t_float *in2 = (t_float *)(w[2]);
4    t_float *in3 = (t_float *)(w[3]);
5    t_float *in4 = (t_float *)(w[4]);
6    int n = (int)(w[5]);
7    t_example11 *x = (t_example11 *)(w[6]);
8    return (w + 7); // proximo bloco
9  }
10
11 static void example11_dsp(
12         t_example11 *x, t_signal **sp){
13   dsp_add(example11_perform, 6, sp[0]->s_vec,
         sp[1]->s_vec, sp[2]->s_vec, sp[3]->s_vec,
         sp[0]->s_n, x);
14 }
```

Outlets DSP (1/2)



Outlets DSP (2/2)

Método _new()

```
1  typedef struct _example12 {
2      t_object x_obj;
3      t_outlet * x_outlet_dsp_0;
4      t_outlet * x_outlet_dsp_1;
5      t_outlet * x_outlet_dsp_2;
6      t_outlet * x_outlet_dsp_3;
7
8  } t_example12;
9
10 void * example12_new(void){
11     t_example12 *x = (t_example12 *)
           pd_new(example12_class);
12     x->x_outlet_dsp_0 = outlet_new(&x->x_obj,
           &s_signal);
13     x->x_outlet_dsp_1 = outlet_new(&x->x_obj,
           &s_signal);
14     x->x_outlet_dsp_2 = outlet_new(&x->x_obj,
           &s_signal);
15     x->x_outlet_dsp_3 = outlet_new(&x->x_obj,
           &s_signal);
16     return (void *) x;
```

Inlets e outlets criados dinamicamente (1/4)

Cenário: o objeto possui um número de inlets (e outlets) igual a um parâmetro passado no momento de sua criação.

Inlets e outlets criados dinamicamente (2/4)

Estrutura e método _dsp()

```
1  typedef struct _multigain {
2      t_object x_obj;
3      t_int count;
4      t_float gain;
5      t_inlet * x_inlet_gain_float;
6      t_sample ** invec;
7      t_sample ** outvec;
8  } t_multigain;
9  static void multigain_dsp(t_multigain *x, t_signal
       **sp){
10     if(x->count < 1) return;
11     int i = 0;
12     for(; i < x->count; i++){
13         x->invec[i] = getbytes(sys_getblksize() *
               sizeof(t_sample));
14         x->invec[i] = sp[i]->s_vec;
15         x->outvec[i] = getbytes(sys_getblksize() *
               sizeof(t_sample));
16         x->outvec[i] = sp[x->count + i]->s_vec;
17     }
18     dsp_add(multigain_perform, 2, x, sp[0]->s_n);
```

Inlets e outlets criados dinamicamente (3/4)

Método _new()

```
1  void * multigain_new(t_floatarg count_arg){
2      t_multigain *x = (t_multigain *)
           pd_new(multigain_class);
3      x->count = (int)count_arg;
4      short i;
5      for (i = 0; i < x->count; i++) {
6          inlet_new(&x->x_obj, &x->x_obj.ob_pd,
               &s_signal, &s_signal); // signal inlets
7          outlet_new(&x->x_obj, &s_signal);
8      }
9      x->outvec = getbytes(sizeof(t_sample) * x->count);
10     x->invec = getbytes(sizeof(t_sample) * x->count);
11     x->x_inlet_gain_float = floatinlet_new(&x->x_obj,
           &x->gain);
12     return (void *) x;
13 }
```

Inlets e outlets criados dinamicamente (4/4)

Método _perform()

```
1  static t_int * multigain_perform(t_int *w){
2      t_multigain *x = (t_multigain *)(w[1]);
3      int n = (int)(w[2]), i = 0, j = 0;
4      float gain = x->gain;
5      for(; i < x->count ; i++)
6          for(j = 0 ; j < n ; j++)
7              x->outvec[i][j] = x->invec[i][j] * gain;
8      return (w + 3); // proximo bloco
9  }
```

Estrutura da apresentação

Tcl/Tk

Características das linguagens:

- Tool Command Language / Toolkit GUI.
- Licenças tipo BSD.
- A interface gráfica do Pd é escrita em Tcl/Tk
- `http://tcl.tk`

Tcl/Tk

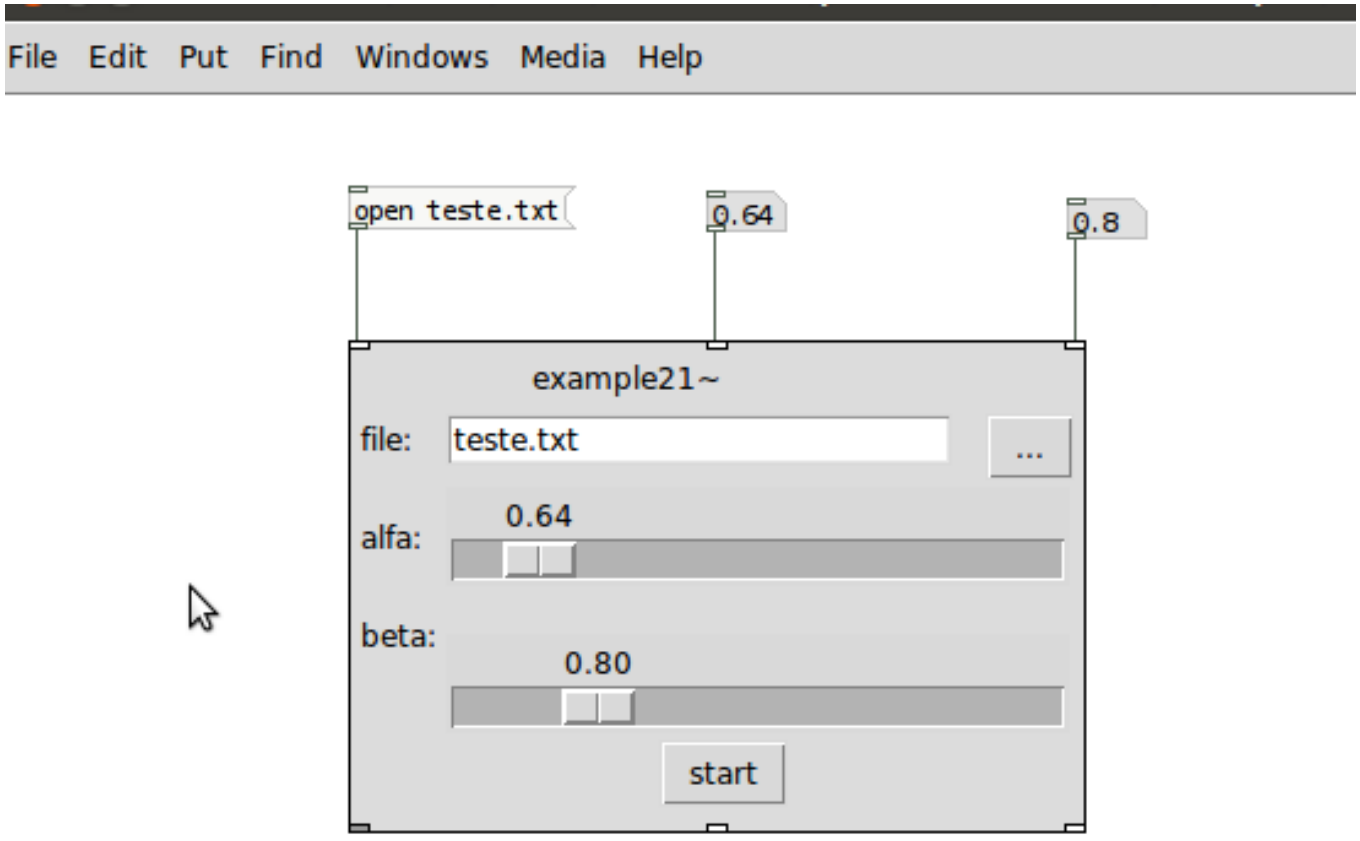


Figura : Exemplo 21

Estrutura da apresentação

Gerenciamento de memória

Arquivo: `m_memory.c`

```
void *getbytes(size_t);
void *getzbytes(size_t);
void *copybytes(void *, size_t);
void freebytes(void *, size_t);
void *resizebytes(void *, size_t, size_t);
```

Átomos

Arquivo: `m_atom.c`

```
t_float atom_getfloat(t_atom *);
t_int atom_getint(t_atom *);
t_symbol *atom_getsymbol(t_atom *);
t_symbol *atom_gensym(t_atom *);
t_float atom_getfloatarg(int, int, t_atom *);
t_int atom_getintarg(int, int, t_atom *);
t_symbol *atom_getsymbolarg(int, int, t_atom *);
void atom_string(t_atom *, char *, unsigned int);
```

Relógios

Arquivo: `m_sched.c`

```
t_clock *clock_new(void *, t_method);
void clock_set(t_clock *, double);
void clock_delay(t_clock *, double);
void clock_unset(t_clock *);
double clock_getlogicaltime(void);
double clock_getsystime(void);
double clock_gettimesince(double);
double clock_getsystimeafter(double);
void clock_free(t_clock *);
```

Inlets

Arquivo: `m_obj.c`

```
t_inlet *inlet_new(t_object *, t_pd *, t_symbol *,
  t_symbol *);
t_inlet *pointerinlet_new(t_object *, t_gpointer *);
t_inlet *floatinlet_new(t_object *owner, t_float *fp);
t_inlet *symbolinlet_new(t_object *, t_symbol **);
t_inlet *signalinlet_new(t_object *, t_float);
void inlet_free(t_inlet *);
```

Outlets

Arquivo: `m_obj.c`

```
t_outlet *outlet_new(t_object *, t_symbol *);
void outlet_bang(t_outlet *);
void outlet_pointer(t_outlet *, t_gpointer *);
void outlet_float(t_outlet *, t_float);
void outlet_symbol(t_outlet *, t_symbol *);
void outlet_list(t_outlet *, t_symbol *, int, t_atom
  *);
void outlet_anything(t_outlet *, t_symbol *, int,
  t_atom *);
t_symbol *outlet_getsymbol(t_outlet *);
void outlet_free(t_outlet *);
```

Classes

Arquivo: `m_class.c`

```
void class_addcreator(t_newmethod, t_symbol *,
  t_atomtype, ...);
void class_addmethod(t_class *, t_method, t_symbol *,
  t_atomtype, ...);
void class_addbang(t_class *, t_method);
void class_addpointer(t_class *, t_method);
void class_doaddfloat(t_class *, t_method);
void class_addsymbol(t_class *, t_method);
void class_addlist(t_class *, t_method);
void class_addanything(t_class *, t_method);
void class_sethelpsymbol(t_class *, t_symbol *);
char *class_getname(t_class *);
char *class_gethelpname(t_class *);
void class_setsavefn(t_class *, t_savefn);
t_savefn class_getsavefn(t_class *);
void class_setpropertiesfn(t_class *, t_propertiesfn);
```

Saída de texto

Arquivo: s_print.c

```
void post(const char *fmt, ...);
void startpost(const char *fmt, ...);
void poststring(const char *s);
void postfloat(t_floatarg f);
void postatom(int argc, t_atom *argv);
void endpost(void);
void error(const char *fmt, ...);
void verbose(int level, const char *fmt, ...);
void bug(const char *fmt, ...);
void pd_error(void *object, const char *fmt, ...);
void sys_logerror(const char *object, const char *s);
void sys_unixerror(const char *object);
void sys_ouch(void);
```

Aritmética de sinais

Arquivo: d_arithmetic.c

```
t_int *plus_perform(t_int *);
t_int *zero_perform(t_int *);
t_int *copy_perform(t_int *);
void dsp_add_plus(t_sample *, t_sample *, t_sample *,
  int);
void dsp_add_copy(t_sample *, t_sample *, int);
void dsp_add_scalarcopy(t_float *, t_sample *, int);
void dsp_add_zero(t_sample *, int);
```

Configurações do DSP

Arquivo: `s_main.c`

```
int sys_getblksize(void);
t_float sys_getsr(void);
```

Configurações de áudio

Arquivo: `s_audio.c`

```
int sys_get_inchannels(void);
int sys_get_outchannels(void);
```

Fast Fourier Transform

Arquivos: d_fft.c, d_fft_fftw.c, d_fft_mayer.c

```
void pd_fft(t_float *, int, int);
void mayer_fht(t_sample *, int);
void mayer_fft(int, t_sample *, t_sample *);
void mayer_ifft(int, t_sample *, t_sample *);
void mayer_realfft(int, t_sample *);
void mayer_realifft(int, t_sample *);
```

Matemática

Arquivo: `d_math.c`

```
t_float mtof(t_float);
t_float ftom(t_float);
t_float rmstodb(t_float);
t_float powtodb(t_float);
t_float dbtorms(t_float);
t_float dbtopow(t_float);
t_float q8_sqrt(t_float);
t_float q8_rsqrt(t_float);
```

Estrutura da apresentação

Referências

- Repositório oficial de externals do Pure Data[1].
- Tutorial do IOHannes[2].
- Código fonte do Pure Data[3].

Agradecemos a comunidade por todos os *externals* desenvolvidos e que serviram para aprender o que apresentamos aqui.

[1] http://pure-data.svn.sourceforge.net/viewvc/pure-data/trunk/externals
[2] http://iem.at/pd/externals-HOWTO/pd-externals-HOWTO.pdf
[3] http://pure-data.git.sourceforge.net/

Dúvidas?

Obrigado!

`http://www.ime.usp.br/~fls`
`https://github.com/flschiavoni/pd-external-tutorial`
`http://compmus.ime.usp.br/`

Um agradecimento especial ao André Bianchi por ter me ajudado a escrever o tutorial e as transparências.