dyn

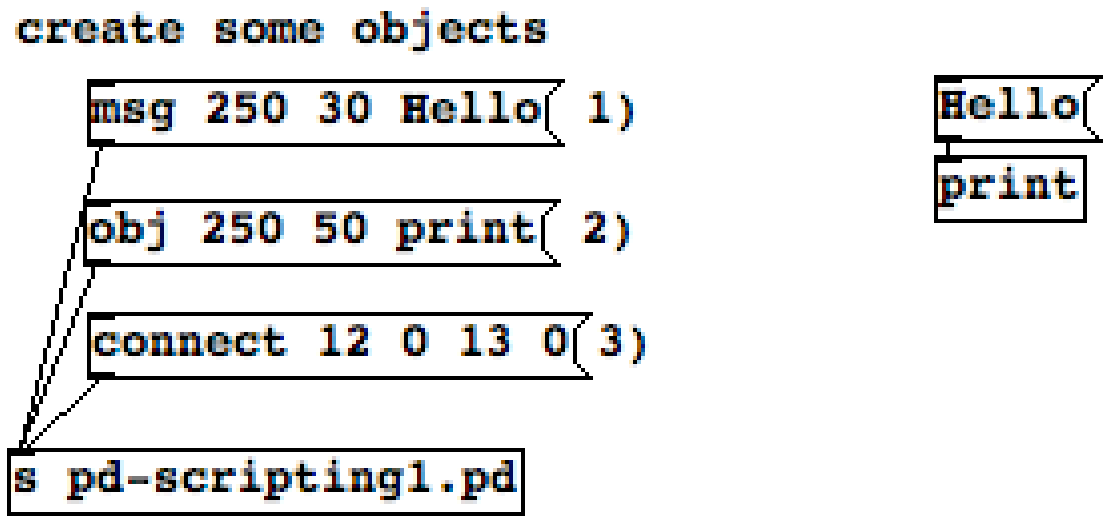
dynamic object management

"next generation PD scripting“

Thomas Grill, Vienna
http://grrrr.org

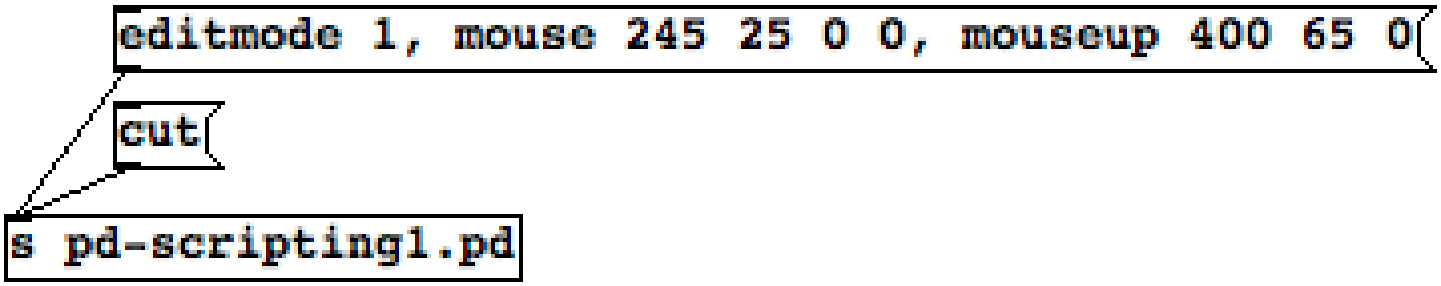
Native PD scripting

Generation etc. of objects by sending messages to the canvas



`connect` message needs index numbers of objects in the patcher, which we need to keep track of.

“Object management” is editor-oriented – in order to delete an object we ought to remember its position.



It is possible to get along with it by creating an empty sub-patcher (zero-based index) and place objects according to their index.

BUT: What if creation fails? We can’t find out!

Scripting in externals (poly~ etc.)

- Use message-based communication – same limitations
- Use PD API functions (m_pd.h, g_canvas.h)
  - *Many of the interesting functions are not public – may be subject to version changes*
  - *Patcher elements (abstractions, sub-patchers, objects, messages) are not treated equally*

The dyn API

- Plain C API, uniform and well-documented
- Thread-safe, asynchronous operation on demand
- Will contain all necessary functions to allow external in-process patcher systems

Object creation / destruction

- All *dyn* objects are referenced by IDs of type `dyn_id`
- Scheduling parameter determines whether operation is synchronous or queued and processed in the background
- Asynchronous operation is callback-based

```
int dyn_NewPatcher(int sched,dyn_id *id,dyn_callback cb,dyn_id
   pid);
int dyn_NewObject(int sched,dyn_id *id,dyn_callback cb,dyn_id
   pid,const t_symbol *obj,int argc,const t_atom *argv);
int dyn_NewMessageStr(int sched,dyn_id *id,dyn_callback
   cb,dyn_id pid,const char *msg);
```

- Objects of any type can be destroyed with a single function

```
int dyn_Free(int sched,dyn_id id);
```

Communication

- Objects can be connected just as in a patcher

```
int dyn_NewConnection(int sched,dyn_id *id,dyn_callback
   cb,dyn_id sid,int outlet,dyn_id did,int inlet);
```

- Messages can be sent directly to an inlet

```
int dyn_Send(int sched,dyn_id id,int inlet,const t_symbol
   *sym,int argc,const t_atom *argv);
```

- Outlets can be eavesdropped

```
int dyn_Listen(int sched,dyn_id *lid,dyn_id id,int
   outlet,dyn_listener cb,void *data);
```

Status information

- All kinds of information can be queried from the system or the various patcher elements.

dyn Python module

- Written in C++ using PyCXX interface
- For use with the py/pyext external
- Simple object-oriented object management
- Above all for ad-hoc solutions, experimenting or testing

```
import dyn                                  # import dyn module
patcher = dyn.Patcher()                     # make new patcher
object1 = patcher.Object("* 2")             # insert * object into patcher
object2 = patcher.Object("print")           # insert print object
object1.Connect(0,object2,0)                # connect the two objects

def send(args):
   object1.Send(0,args)                     # send arguments to inlet of object1
```

Python objects are wrappers to dyn object IDs.

Limitations

- No DSP interface yet
- Many of PDs internal data structures are duplicated – quite inefficient in terms of memory usage

Future

- Could also be implemented for Max/MSP provided that the object management functions be disclosed in the Max API
- May serve as a basis for a future dynamic signal graph generating system (like SCLang)
- Pure Data should adopt a comparable native API and render *dyn* useless