# Bridging MOOS Communities with `pShare`

Paul Newman, University of Oxford

May 25, 2013

....ten years on

# Contents

# 1 Introduction

MOOS-V10 brings with it a new command line application which allows data to be shared between MOOS communities. Recall that in MOOS parlance a "community" is the set of programms talking to a particular instance of a `MOOSDB` including the `MOOSDB` itself. In some ways `pShare` is just a modern, better written version of `pMOOSBridge` in others it offers much greater flexibility and functionality. Here is a quick summary of `pShare` functionality

- it offers UDP communication between communities
- it can share data over multicast channels
- it allows renaming of variables as they are shared
- it supports wildcard sharing - so you can specify a sinple regular expression for which what should be shared
- it supports dynamic configuration (via MOOS) of sharing/forwarding/renaming
- it supports command line configuration and from a .moos file
- it can be completely configured from the command line

It is worth, straight off the bat, understanding how in usage terms `pShare` is different from `pMOOSBridge`

- `pShare` unlike `pMOOSBridge` only supports UDP (or multicast which is a kind of UDP)
- You need one instance of pShare per community (compare this to `pMOOSBridge` where a single instance could be used to bridge any number of communities)
- Currently `pShare` only supports sharing of up to 64K messages

## 1.1 Why UDP ?

UDP is, of course, a lossy affair - there is no guarrantee that messages will get through and indeed `pShare` is intended for use in just such situations. Use `pShare` when you want, when possible, to get messages between communities and yet you don't mind dropping a few messages. Perhaps this applies when you have a deployed robot out in the wilds and the wireless link simply doesn't suport tcp/ip as well as you might hope. [1]

[1] If you do mind loosing things then you must use tcp/ip (standard MOOS) and if you have a lossy connection you will spend years waiting for data.

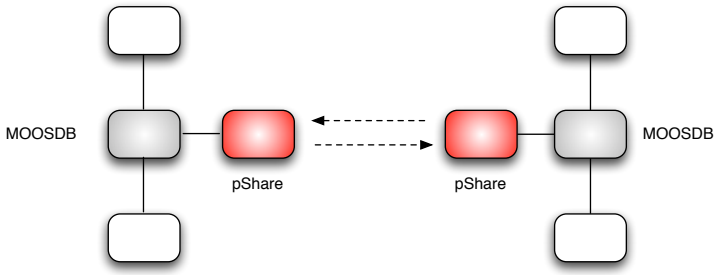


Figure 1: A simple use of `pShare:` two communities are linked by two instances of pShare - one in each community.

# 2 Basic Operation

Figure 1 shows a typical and simple use case of `pShare.` Here two communities are linked by two instances of `pShare.` Each is connected to a `MOOSDB` and each pShare is configured (by a means we will get to in a minute) to subscribe to messages from the `MOOSDB` (issued by clients). These messages are forwarded over a udp link to the other `pShare` instance which inserts them into it's own `MOOSDB.` The important point here is that if process "A" in community "P" has message M shared via `pShareP` and `pShareQ to` process B in community Q then when B receives M it will still have A as its source and P as its source community. So to process B it looks like A is actutaly in its own community (Q).

A more complicated (marginally) example is show in Figure 2 . Here the left hand community is sharing as an output data to the top right and bottom right communities but only receiving data from the bottom right.

## 2.1 Different Ways Sharing

Each instance of `pShare` can be configured such that it can

- forward a named message (like 'X') to any number of specific udp ports on any number of other machines
- can rename a message before forwarding
- receive and forward on to its own `MOOSDB` messages from any number of other `pShares`
- forward messages on predefined or any number of multicast channels
- receive messages on any number of multicast channels

how to do this is best explained with some examples and that will happen in Section 4. Before that it is worth explaining the merit of multicast channels.

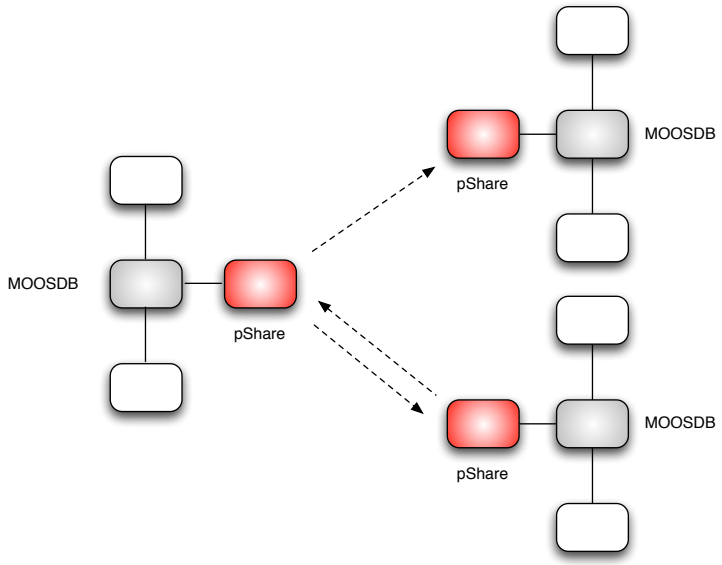


Figure 2: A simple use of `pShare:` three communities are linked by two instances of pShare - one in each community but data sharing is not symmetric.

### 2.2 Sharing via Multicast Channels

Imagine you as an application developer knew that other communities (but you do not know which ones *a-priori*) would be interested in a variable called `X`. Now if you knew exactly who wanted it you could configure a standard UDP shares to mutually agreed ports (presumably one port per community) on which other `pShares` are listening. But you don't. So what to do? Well you could use `pShare's` ability for packets to predefine multicast channels ( these are really simply multicast addresses behind the scenes) you can tell pShare to forward MOOS messages out to a multicast_channel and later on any number of other `pShare` instances can subscribe to this channel and receive them.

## 3 The form of command line configuration strings

### 3.1 Output

The '-o' switch allows you to configure which messages to forward (share), how to rename them and where to send them. At its highest level the the '-o' switch is followed by a comma separated list of

**mappings** "`-o= mapping , mapping, mapping,...`" and each mapping describes how one MOOS variable is routed to any number of destinations. Each mapping contains mulitple...

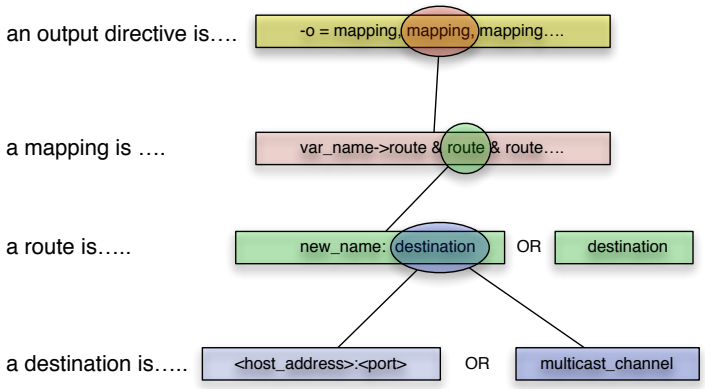


Figure 3: Specifying `pShare` forwarding behaviour from the command line. Examples are given in4.

**routes** and has the form of "`var_name->route & route...`" so a variable name pointing to an ampersand-separated list of "routes". Now, each route describes a...

**destination** for a message and it has the format "`new_name:destination_address:destination_port`" or "`new_name:multicast_channel`". The new_name part can be ommited in which case the variable is not renamed. Figure 3 describes this hiearchy pictorially and some concrete examples are given in Section 4.

### 3.2 Input

The -i switch is much simpler. It tells an instance of `pShare` how to listen to for incoming traffic. The format is always -i=localhost:<port_num> or -i=multicast_<N> where N is a number between 0 and 255. Multiple listens can be specified in a comma separated list.

## 4 Command line Configuration

Imagine we have two communities A and B. Lets also assume that they reside on different machines. Machine A has ip address 192.168.0.10 and machine B has ip address 192.168.0.4.

| | |
|---|---|
| description | **share X from A to B** |
| terminal A command line | `-o='X->192.168.0.4:10000'` |
| terminal B command line | `-i=localhost:10000` |

| | |
|---|---|
| description | **share X from A to B as Y** |
| terminal A command line | `-o='X->Y:192.168.0.4:10000'` |
| terminal B command line | `-i=localhost:10000` |

| | |
|---|---|
| description | **share X from A to B as X and Y** |
| terminal A command line | `-o='X->92.168.0.4:10000 & Y:192.168.0.4:10000'` |
| terminal B command line | `-i=localhost:10000` |

| | |
|---|---|
| description | **share X from A to B as X and Y via two different ports** |
| terminal A command line | `-o='X->92.168.0.4:10000 & Y:192.168.0.4:20000'` |
| terminal B command line | `-i=localhost:10000,localhost:20000` |

| | |
|---|---|
| description | **share X and Y to B** |
| terminal A command line | `-o='X->192.168.0.4:10000 , Y->192.168.0.4:10000'` |
| terminal B command line | `-i=localhost:10000` |

| | |
|---|---|
| description | **share X via multicast** |
| terminal A command line | `-o='X->multicast_7'` |
| terminal B command line | `-i=multicast_7` |

| | |
|---|---|
| description | **share X via multicast and rename** |
| terminal A command line | `-o='X->Y:multicast_7'` |
| terminal B command line | `-i=multicast_7` |

| | |
|---|---|
| description | **share X on several channels** |
| terminal A command line | `-o='X->Y:multicast_7 & Z:multicast_3'` |
| terminal B command line | `-i=multicast_7` |

| | |
|---|---|
| description | **share X via multicast and rename** |
| terminal A command line | `-o='X->Y:multicast_7'` |
| terminal B command line | `-i=multicast_7,multicast_3` |

| | |
|---|---|
| description | **share X as several new variables on the same multicast channel** |
| terminal A command line | `-o='X->Y:multicast_7 & Z:multicast_7'` |
| terminal B command line | `-i=multicast_7` |

**Tip:** don't forget to put single quotes around the routing directives to prevent your shell from interpretting the '>' character.

# 5 Configuring pShare from a .moos file

We have seen some examples on how to configure `pShare` on the command line (because that is insanely useful) but of course it can also be configured by reading a configuration block in a .moos file just like any `MOOSApp` can. The key parameter names are

**Output** which can have the same format as the -o flag on the command line or a more verbose as illustrated below. There can be as many "Output" directives in a configuration block as you need. The verbose form specifies one share per invocation while the compact form specifies as many as you wish. The verbose form of the Output directive is a tuple of token value pairs where the tokens are

**src_name** the name of the varible ot be shared

**dest_name** the name it should have when it arrives at its destination - this is optional, if it is not present then no renaming occurs

**route** a description of the route which could be for udp shares host-name:port:udp or for multicast shares "multicast_X". This is much as it is for the command line configuration.

**Input** which can have the same dense format as the -i flag on the command line as described above or a more verbose, intuitive form illustrated below. In the long hand version you use a single token value pair with a token name of "**route**" as described above. This specifies the fashion in which this instance of pShare should listen - be that on mulitple ports for udp traffic or on a multicast channel for multicast action.

Listing 1: Configuring pShare from a configuration block

```
ProcessConfig=pShare
{
    //a verbose way of sharing X, calling it Y and sending
    //on mulitcast_8
    Output = src_name =X,dest_name=Z,route=multicast_8

    //a verbose way of sharing Y calling it YY and sending
    //it to port 9832 on this machine
    Output = src_name =Y, dest_name = YY, route=192.6.8.3:9832

    //a verbose way of sharing T, sending it without name change
    //to port 9832 on a remote machine
    Output = src_name =T, dest_name = TT, route=192.3.4.5:9832

    //a dense specification which sends X to port 10000 via
    //udp on a remote machine
    //and Y to a different machine while renaming it to 'T'
    Output = X->192.168.0.4:10000
```

```
    output = Y->T:192.168.0.5:10000

    //specify in what places we wish to listen to receive
    //the output of other instances of pShare

    //we can do this one at a time using the route directive
    Input = route=multicast_6

    //or we can specifiy multiple routes at once. Note that
    //we have to use an & character to separate different routes
    //or it looks like a list of mal-formed token value pairs
    Input =route=multicast_21&localhost:9833&multicast_3


    //we can of course also use wildcards - this is where it gets↩
         interesting
    //lets share any varialble in community which is 2 charcters ↩
        long and begins
    //with X
    Output = src_name = X?, route = localhost:9021

    //we could be more specific and say we only want to share ↩
        such variables from
    //a named process. So here we say only share two letter ↩
        variables beginning with Q
    //form a process called procA
    Output = src_name = Q?:procA, route = localhost:9021


}
```

# 6 Wildcard Sharing

It won't have escaped your attention that MOOS-V10 offers support for wildcarding -that is specifying a pattern which represents a whole set of named variables.(So for example '*' means all variables because the regular expressions character '*' matches all sets of characters). `pShare` can utilise this functionality to make sharing many variables trivial. You can also specify to only share variables from a specific process.

So lets start with a command line example. We can share all variables in a community thus:

| description | **share all variables onto channel 7** |
|---|---|
| terminal A command line | `-o='*->multicast_7'` |
| terminal B command line | `-i=multicast_7` |

And we can be a little more precise and only forward variables which begin with the letters "SP"

| | |
|---|---|
| description | **share all variables onto channel 7 which begin with "SP"** |
| terminal A command line | `-o='SP*->multicast_7'` |
| terminal B command line | `-i=multicast_7` |

or which begin with "K" end with "X" followed by any single character

| | |
|---|---|
| description | **starting with X ending with a K plus 1 character** |
| terminal A command line | `-o='X*K?->multicast_7'` |
| terminal B command line | `-i=multicast_7` |

We can also be explict about which processes we want to forward from. So for example say we just wanted to forward messages from teh process called "GPS":

| | |
|---|---|
| description | **share all variables from "GPS" onto channel 7** |
| terminal A command line | `-o='*:GPS->multicast_7'` |
| terminal B command line | `-i=multicast_7` |

And of course the process name also supports wild cards so we we can do

| | |
|---|---|
| description | **var ending in "time" from a proc starting "camera_"** |
| terminal A command line | `-o='*time:camera_*->multicast_7'` |
| terminal B command line | `-i=multicast_7` |

A good question is what does it mean to rename a wildcard share ? Well that simply serves as suffix to the shared variable name

| | |
|---|---|
| description | **share all variables onto channel 7 with renaming** |
| terminal A command line | `-o='*->T:multicast_7'` |
| terminal B command line | `-i=multicast_7` |

which means a variable "X" will be shared as "TX" - the parameter T is acting as suffix. Similarly a variable called "donkey" would endup being shared in this example as "Tdonkey".

Finally of course wildcard shares can be specified in configuration files as shown below.

Listing 2: Configuring pShare from a configuration block

```
ProcessConfig = pShare
{

        //we can of course also use wildcards - this is where it ↩
            gets interesting
        //lets share any varialble in community which is 2 ↩
            charcters long and begins
        //with X
        Output = src_name = X?, route = localhost:9021

        //we could be more specific and say we only want to share↩
             such variables from
        //a named process. So here we say only share two letter ↩
            variables beginning with Q
        //form a process called procA
        Output = src_name = Q?:procA, route = localhost:9021

    //we can be more general and send any variable beginning with↩
         W fomr a process
    //whos name ends in A to multicast channel 7
    Output = src_name = W*:*A, route = multicast_7
}
```

## 6.1 Caret Sharing: A Special Case of Wildcard Sharing

pShare supports a special case of wildcard sharing in that it can forward a variable under a new name where the new name is derived from the part of the original name which matches a '*' wildcard charcter. Granted this sounds complicated. Its easiest to understand this with a few examples. The important point to remember though is the syntax for this special case it is `*<str>->^` or `<str>*->^` where <str> is any string that does not contain a * or ?.

| description | **A_X gets shared as A on multicast 7** |
|---|---|
| terminal A command line | `-o='*_X->^:multicast_7'` |
| terminal B command line | `-i=multicast_7` |

Note in that in the kind of sharing the '^' means the part of the variable name which matches the single '*' wildcard character on the src filter. This wildcard character can only occur at the beginning or teh end of the variable pattern. So we can also have:

| description | **A_X gets shared as X on multicast 7** |
|---|---|
| terminal A command line | `-o='A_*->^:multicast_7'` |
| terminal B command line | `-i=multicast_7` |

# 7 Instigating Dynamic Shares On The Fly

`pShare` can be told to start sharing data dynamically by any MOOS Process simply by publishing a correctly formatted string. The format is simple - its is pretty much the same as a line in a configuration file. You need to write a string `"cmd = <directive>"` to the variable `PSHARE_CMD` where `<directive>` is a output or input directive such as you would write in a configuration file. Here are some examples:

- "cmd = Output , src_name = X?, route = localhost:9021"
- " cmd = Output , src_name =T, dest_name = TT, route=192.3.4.5:9832"

The ability to dynamically instigate shares turns out to be very useful if you don't know what needs to be shared when `pShare` first starts and that only gets figured out by other processes.