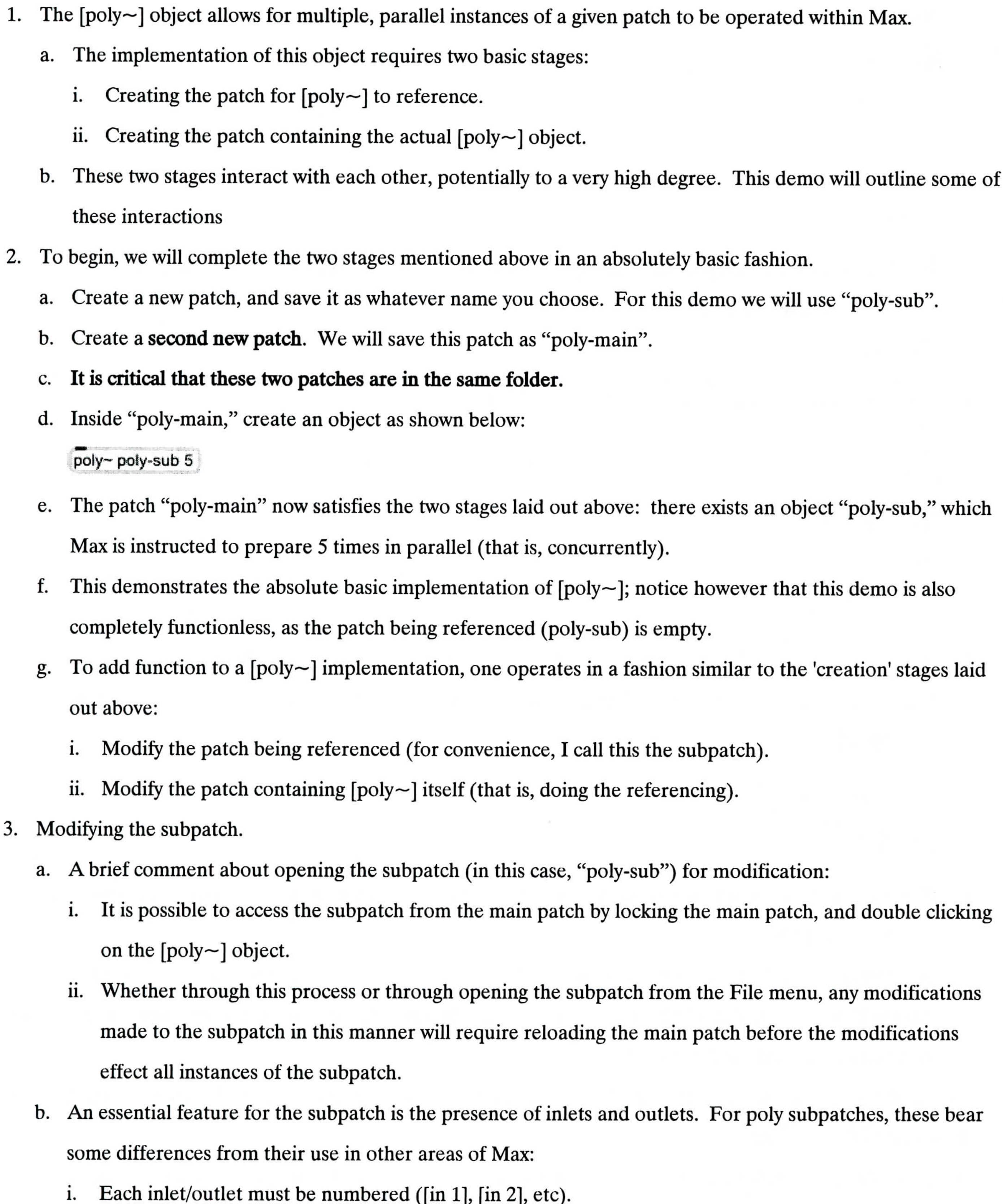
[poly~] object

Electronic Music II

Spring 2014

1. The [poly~] object allows for multiple, parallel instances of a given patch to be operated within Max.
   a. The implementation of this object requires two basic stages:
      i. Creating the patch for [poly~] to reference.
      ii. Creating the patch containing the actual [poly~] object.
   b. These two stages interact with each other, potentially to a very high degree. This demo will outline some of these interactions
2. To begin, we will complete the two stages mentioned above in an absolutely basic fashion.
   a. Create a new patch, and save it as whatever name you choose. For this demo we will use “poly-sub”.
   b. Create a **second new patch**. We will save this patch as “poly-main”.
   c. **It is critical that these two patches are in the same folder.**
   d. Inside “poly-main,” create an object as shown below:

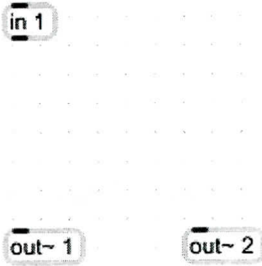


   e. The patch “poly-main” now satisfies the two stages laid out above: there exists an object “poly-sub,” which Max is instructed to prepare 5 times in parallel (that is, concurrently).
   f. This demonstrates the absolute basic implementation of [poly~]; notice however that this demo is also completely functionless, as the patch being referenced (poly-sub) is empty.
   g. To add function to a [poly~] implementation, one operates in a fashion similar to the 'creation' stages laid out above:
      i. Modify the patch being referenced (for convenience, I call this the subpatch).
      ii. Modify the patch containing [poly~] itself (that is, doing the referencing).
3. Modifying the subpatch.
   a. A brief comment about opening the subpatch (in this case, “poly-sub”) for modification:
      i. It is possible to access the subpatch from the main patch by locking the main patch, and double clicking on the [poly~] object.
      ii. Whether through this process or through opening the subpatch from the File menu, any modifications made to the subpatch in this manner will require reloading the main patch before the modifications effect all instances of the subpatch.
   b. An essential feature for the subpatch is the presence of inlets and outlets. For poly subpatches, these bear some differences from their use in other areas of Max:
      i. Each inlet/outlet must be numbered ([in 1], [in 2], etc).

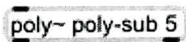
ii. Audio inlets/outlets must be specified by adding a ~. Thus, [in 1] is not an audio inlet; [in~ 1] is.

iii. Notice that the [poly~] object has one inlet by default. Creating [in 1] or [in~ 1] therefore does not cosmetically change the [poly~] object (in the main patch), it only adds function to the subpatch.

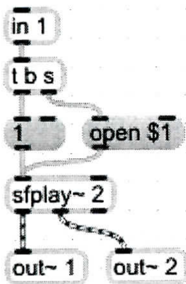
c. With these caveats in mind, I modify "poly-sub" as shown below:



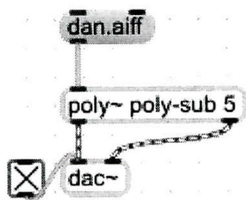
d. If I close and reopen "poly-main," the appearance of my [poly~ poly-sub 5] object has changed to reflect these new inlets and outlets:



e. Now I will further modify the contents of the subpatch:



f. Now, the subpatch will take a symbol from the inlet (a filename), open it in [sfplay~ 2], and play that file immediately. This requires me to change "poly-main" in order to manipulate this feature from there:



g. This back-and-forth relation between the two 'layers' (the main patch and the subpatch) is common when working with [poly~].

h. Notice that the main patch, as it stands now, will only play back dan.aiff once at a time; if I resend the "dan.aiff" message **before** playback has ended, it **restarts** playback rather than **starting a new** instance. The next section will add this functionality.

4. For the [poly~] object to operate multiple instances of the subpatch concurrently, the [thispoly~] object has to be added to the subpatch.

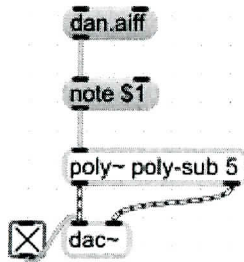
a. [thispoly~] is essentially an indicator switch, which tells [poly~] that the subpatch-instance containing it is

either available for new information or not.

b. To place this in the context of our current patch:

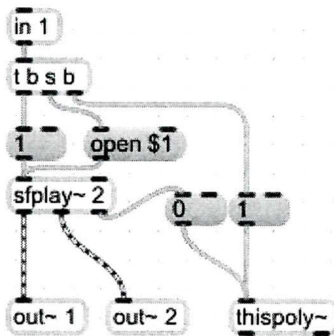
   i. Currently, the message "dan.aiff" is being sent to instance 1 of the subpatch. This is why we only hear a single instance of playback; only one [sfplay~] object is being triggered.

   ii. If the message "dan.aiff" is sent with the word "note" preceding it, [poly~] will automatically find the first (ordinally speaking) available instance of the subpatch to receive the message. This is implemented below (notice there is no change in function):



   iii. [poly~] recognizes availability through a particular setting in an instance's [thispoly~] object. Without this object, all instances are seen as available. Therefore, the first instance available without using [thispoly~] will always be instance 1 (ordinally earliest).

c. I will add the [thispoly~] object to the subpatch as follows:



d. To trace the order of events:

   i. From the inlet, the subpatch-instance receives a symbol (a filename).

   ii. The instance marks itself as unavailable.

   iii. Whichever filename was received is opened by [sfplay~]

   iv. Playback begins.

   v. When playback is complete, the instance marks itself as available.

e. Notice that now, I can overlap instances of "dan.aiff" playing back.

f. Some closing remarks on this function:

   i. As the patch stands, once 5 instances of playback are going on at once, **no further instances can be cued** until the current ones end.

ii. The cessation in patch activity that this creates can be circumvented by sending [poly~] a message, "steal 1". This will turn on voice-stealing, à la MIDI (any new instance cued once all 5 are marked unavailable will **interrupt** the 'oldest' running instance). N.B. This is not on be default, and can be turned off with a message "steal 0".

iii. Alternatively, to avoid the cessation in activity the number of instances can be increased, either by changing the creation argument of [poly~] (for example, instead of [poly~ poly-sub 5], [poly~ poly-sub 10]), or by sending [poly~] a message "voices *n*", where *n* is the desired number of instances.

iv. These messages are illustrated below: