Adv3Lite Quick Reference

[optional elements]

Things

```
itemName: Thing ‘vocab’ [@location]
    “desc”
    myProp = ‘foo’
    myMeth(bar)
    {
       return bar ?? name;
    }
;
```

vocab =
`‘[article] name; adjs; nouns; pronoun’`

article is optional and can be ‘a’, ‘an’, ‘some’ – mass noun or ‘()’ = qualified name.

pronoun is ‘it’ ‘them’ ‘him’ or ‘her’
For ambiguously plural things use ‘it them’ if the name is singular or ‘them it’ if the name is plural.

Words in the vocab string can be followed by `[n]`, `[adj]` or `[prep]` to mark the part of speech if not the expected default. Weak tokens can be followed by [weak] or placed in parentheses.

Some common properties
`isFixed` - to make nonportable
`isVehicle` – to make it a vehicle
`isLit` – this Thing provides light
`specialDesc` = “A foo sits in the corner. “
`familiar` – the player character knows about this object.

Rooms and Travel Connectors

```
fooPlace: Room ‘roomTitle’ [‘vocab’]
     “This looks a strange room. ”
     north = northRoom
     east = ‘The wall\’s in the way. ‘
     south = “You walk a few paces
        south but turn round when you
        don’t like what you see. “
     west: TravelConnector
     {
        canTravelerPass(traveler)
        {
           return !foo.isIn(traveler);
        }
       explainTravelBarrier(traveler)
        {
          “You can\’t go that way
         carrying the foo. “;
        }
     out asExit(north)
     southeast = fooDoor
     regions = [barRegion, fooRegion]
;

barRegion: Region;
```

Doors

A door connecting fooRoom to barRoom:

```
fooDoor: DSDoor ‘foo door’ @fooRoom
@barRoom
“The door looks reassuringly solid “
;
```

Alternatively:

```
fooDoor: Door ‘foo door’ @fooRoom
  “The door looks reassuringly solid “
   otherSide = barDoor
;
```

Similarly: Passage, PathPassage, StairwayUp, StairwayDown (use `destination` not `otherrSide`), DSPassage, DSPathPassage, & DSStairway

Locks and Keys

```
fooKey: Key ‘foo key’
     actualLockList = [fooDoor,
        fooBox]
;

fooDoor: Door ‘foo door’ @fooRoom
  “It’s a door. ”
   otherSide = barDoor
   lockability = lockableWithKey
   isLocked = true
;

barDoor ‘bar door’ @barRoom
  “It’s another door. ”
  otherSide = fooDoor
  lockability = lockableWithoutKey
;

fooBox: OpenableContainer ‘foobox’
  “desc”
  lockability = lockableWithKey
  isLocked = true
;
```

Something lockable by a separate mechanism:

```
panelDoor: DSDoor ‘secret panel’
   “desc”
   lockability = indirectLockable
   indirectLockableMsg = ‘The panel
    can only be locked and unlocked
    by frobbing the foobar. ‘
;
```

Multiple Containment

A cooker with a pan on top and a pie inside:

```
cooker: Fixture ‘cooker’ @kitchen
   remapOn: SubComponent {}
   remapIn: SubComponent {
              isOpenable = true
              isOpen = nil
            }
;

+ pan: Thing ‘aluminium pan’
   sLoc(On)
;

+ pie: Food ‘apple pie; baked’
   “It looks well baked. “
   sLoc(In)
;
```

MultiLocs

Are Things in several places at once.

```
sky: MultiLoc, Distant 'sky; dark
crescent; moon stars'
   "The sky is dark tonight, with only
a crescent moon showing among the
myriad of stars. "

   notImportantMsg = 'The sky is way
too far above your head. '
   locationList = [outdoors]
   // outdoors could be a Region.
   // You could also list rooms here.
;

moveIntoAdd(loc) // add sky to loc
moveOutOf(loc)// move sky out of loc
moveInto(loc) // move sky to loc and
             out of everything else.
```

Defining New Actions

TActions take one or more direction objects:

```
DefineTAction(Frob)
;

VerbRule(Frob)
  (‘frob’ | ‘foofrob’) multiDobj
                      [or singleDobj]
 : VerbProduction
  action = Frob

 verbPhrase = ‘frob/frobbing (what)’
 missingQ = ‘what do you want to frob’
;
```

TActions take a direct object and an indirect object:

```
DefineTIAction(FrobWuth)
;

VerbRule(FrobWith)
  (‘frob’ | ‘foofrob’) multiDobj
         ‘with’ singleIobj

 : VerbProduction
  action = FrobWith

 verbPhrase = ‘frob/frobbing (what)
    (with what)’
 missingQ = ‘what do you want to frob|
   what do you want to frob it with’
 iobjReply = withSingleNoun
;

/* Example implementation on Thing: */
modify Thing
  dobjFor(Frob)
  {
    preCond = [touchObj

    verify()
    {
      If(!isFrobable)
        Illogical(‘{I} {can\’t} frob
          {that dobj. ‘);
    }

    check()
    {
     if(foo)
       “{I} {can\’t} frob {the dobj}
           while foo. “;
    }

    action
    {
      frobbed = true;
    }

    report()
    {
      “{I} frob{s/?ed}
              <<gActionListStr>>. “;
    }
  }

 isFrobable = nil
 frobbed = nil
;
```

SpecialVerbs

```
SpecialVerb 'ring' 'push' @doorbell;
SpecialVerb 'cross|walk across|go
     over' 'sp#act' [bigbridge,
                        smallbridge];
```

‘sp#act’ triggers SpecialAction which you can use for your own purposes by defining a `dobjFor(SpecialAction)` block on the intended direct object.
For an action that can apply to multiple direct objects in a single command, use ‘sp#acts’.