***The very quick start guide***

***for ALAN Standard Library v2.0***

1)

Import the five library files (‘lib\_classes.i’, ‘lib\_definitions.i’, ‘lib\_locations.i’, ‘lib\_messages.i’ and ‘lib\_verbs.i’) into your game source (make sure that they are in the same folder as your game source file). Importing happens through placing the following code in your game source file:

IMPORT 'lib\_classes.i'.

IMPORT 'lib\_definitions.i'.

IMPORT 'lib\_locations.i'.

IMPORT 'lib\_messages.i'.

IMPORT 'lib\_verbs.i'.

Alternatively, you can just import the file ‘library.i’ included in the library distribution package, and it automatically includes the five files above. Make sure that ‘library.i’ is in the same folder as the other library files and your own game source file:

IMPORT ‘library.i’.

2)

Define the instance “my\_game” in your own source file, in the following way:

THE my\_game ISA DEFINITION\_BLOCK  
END THE.

The above is the shortest possible formulation of this “meta instance” inside which you can define many things affecting the gameplay on a general level, for example default verb outcomes in your game. Let’s say you’re not happy with the way the library defines the outcome of the *examine* verb for objects and actors (“You see nothing special about [the object or actor].”) Instead, you would like to have “It’s just an ordinary [object or actor].” Here, the *my\_game* instance comes in handy. You can change the default wording for the *examine* verb by editing the *my\_game* instance like this:

THE my\_game ISA DEFINITION\_BLOCK

VERB examine

DOES ONLY “It’s just an ordinary” SAY obj. “.”  
END THE.

END THE.

Now, the default library message for *examine* has been replaced by your own message throughout the game.

The *my\_game* instance can be placed anywhere in your source code (as long as it comes before the start section and naturally is not inside any other instance). It needs to be declared even if you didn’t use it at all, because the game won’t compile correctly without it.

Besides changing the default verb responses, you can edit check messages provided by the library if you wish. You can also add a check of your own to any library verb, edit illegal parameter messages, and add general attributes needed in the game.

THE my\_game ISA DEFINITION\_BLOCK

HAS treasures\_found 0.

HAS time\_left 300.

VERB examine

DOES ONLY

“Nothing special.” -- your own response overriding the

-- library response for ‘examine’

END VERB.

VERB jump

CHECK health OF hero > 5 -- your own check added here,

ELSE “You feel too weak.” -- and not to the library file!

DOES ONLY

“Wheeeee!” -- your own response overriding the

-- library response.

END VERB.

VERB eat

CHECK hero NOT tied\_up

ELSE “You can’t possibly do that while tied up.”

END VERB.

-- Note that there is no DOES ONLY part in the ‘eat’ verb above.

-- The library default outcome for the ‘eat’ verb would be carried

-- out here if the check was passed. In other words, if you’re happy

-- with the default verb response (what happens after DOES), it’s

-- enough to just add a check to the verb in this way.

HAS check\_obj\_not\_obj2\_with "It would be rather absurd to $v something

with itself, don’t you think?".

-- This check message would be shown in all cases throughout the game

-- when the hero would try to use an object with itself (that is, if

-- you don’t like the default that the library provides).

HAS illegal\_parameter\_there: “You can’t $v there.”

END THE my\_game.

All various attributes and editable messages for the *my\_game* instance are listed further down in this manual.

3)

Go on designing your game as usual, implementing locations, actors, objects, and so forth.

-- *end of the very quick start guide.*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_