The ALAN Adventure Language

Conversion Guide

from v2 to v3

Preliminary!

Version 3.0beta2

This version of the conversion guide was printed on February 05, 2017

Table of Contents

1 INTRODUCTION

Text adventures or, using a more appropriate term, interactive fiction, is a form of computer game which has many things in common with fiction in book form, role-playing games and puzzle-solving. To create a high quality interactive fiction game, you need to be more of an author than a games programmer.

Alan is a special purpose computer language specifically designed to make it very easy to create such adventure games requiring only limited programming skills.

The author and a very good friend designed the first version of the Alan language in 1985. During many years of incremental improvement, it has now reached its third major version. This means that the language has a sound foundation, based on practical use. Therefore, features have been added as experience have grown, from actual use and understanding of the most prioritised needs.

1.1 Purpose of the Guide

This document tries to give help and advice for those that attempt to port version 2 Alan games to version 3. Although not exactly trivial, it can be rather straightforward and is in not so complicated that it should not be attempted. On the contrary, the author believes that for most games it is simple, but perhaps a bit tedious. The use of some (mis-) features of version 2 can give rise to trouble and these features are listed here together with possible ways to minimize work or work around the problem or even possible well structured rewrites.

A complete package of the Alan Development System v3 should include a crude converter program which can do a lot of the grunt work for you. The source code for that converter is available.

Some expressions in this guide might be difficult to grasp if you are just starting out with Alan v3. Don't hesitate to contact the author, or the Alan mailing list, to get some answers to your questions. Likewise, if you have successfully ported a game, and particularly if you discovered an error in this guide, or even better, found a new tip, we'd like to hear from you!

2 PORTING STRATEGY

Real life has learned this author one lesson: do it bitwise, little by little, one step at a time, even a long journey starts with a small step, …

Start by running your game in a V2 environment. Log the output to a file. While doing this, input any command you can think of. Abuse your game. Ask your evil brother to type some commands. Play it to the end.

Now you have a good starting point:

- A sequence of commands
- A log of the exact output of your game given that input

Copy your game to a new location. Start by running the converter program. It will do the bulk of the straightforward conversions for you. Compile the converted source. Get chocked over the number of errors. Note the number. Pick a section below or an error message in the source, and try to address that by changing the corresponding code. Compile it again. Note the number of errors. Any progress? Good. Repeat.

Once you have a running game, run it in a V3 environment. Log the game output. Input the exact same commands as above. Compare the output. Different? If not, you are done already! Else, you have some debugging to do. Consult the following chapters again. Found anything?

Fix it and run again. Progress! If not, look for appropriate sections in the reference manual, possibly compare it with a v2.x manual. When finally the output of your game is the same as it was in the V2 log, you are done! Well done!

I repeat, don't think you can do it by hacking along. Don't be tempted to change that little spelling mistake, or add that button to the television set. IT WILL RUIN YOUR BASELINE! I.e. you will have nothing to compare it with any longer. Instead, write that idea down on a piece of paper for memory and later action.

Take it easy. Save a new backup copy every time you have made progress, and that's every time. (File versioning software is a blessing.) Eventually you will get there.

By the way, this strategy applies not only to porting Alan games. It applies to all development, this is how Alan v3 was possible; in fact, it applies to the whole life of this author now…

3 THE LITTLE WORDS

There are a couple of new, reserved, words in Alan version 3. These that might cause compiler errors if they where used in old source. This usually occurs because they are used in location names without quotes. The most common problematic words are `THE` and `EVERY`.

The simple and usually correct change is to quote them. E.g.

```
Location l Name Outside the house
```

Needs to be converted to

```
The l Isa location Name Outside 'the' house.
```

Or even

```
The l Isa location Name 'Outside the House'.
```

Do you remember that a location name can be a single quoted identifier?

3.1 Names and Capitalization

Alan V3 has rethought the handling of names, identifiers etc. V2 converted everything to lower case and used that in the game. This caused many problems with actor names and in other cases where you wanted your output to be edited in a special way.

In V3 you can use any case in a name or identifier and that will be preserved. Comparisons are done in a case insensitive way, in both the compiler and the interpreter. This means that you can give the actor a real name with leading capitals and that name will be used in the game output. The player will still be able to input lower case and it will match.

This does away with many occasions where quoted identifiers had to be used together with multiple names to achieve the desired output while keeping the possibility for the player to refer to the items.

When converting a V2 game, especially look out for location and actor names, since these where automatically capitalized. Also, note that the first occurrence of a word will define its capitalization, which includes names of locations. (You might have used capitalization on common words that should not be capitalized, solve this by quoting the complete location name.)

4 OBJECTS, ACTORS AND LOCATIONS

4.1 Declarations

Alan v2 had three built-in entities, or classes. These where built in also into the language. V3 has generalized this into the class structure and the pre-defined classes documented in this manual.

The conversion is tedious but straight-forward (Emacsophiles can probably hack out an elisp-macro):

```
OBJECT o …
    :::
END OBJECT o.
```

Should be translated into

```
The o Isa object …
    :::
End The o.
```

The conversion is analogous for actors and locations.

The converter program that is available, will do this automatically for you.

4.2 Default Attributes

Version 2 allowed attributes to be given to all objects, all actors, all locations or all of these. This was done using the `DEFAULT ATTRIBUTES`, `DEFAULT OBJECT ATTRIBUTES`, `DEFAULT ACTOR ATTRIBUTES` and `DEFAULT LOCATION ATTRIBUTES` respectively.

The new classing mechanism of Alan v3 solves this in a much more flexible way by allowing adding attributes on *any* level in the class hierarchy. You can still add properties, including the special case of attributes, to a class after its definition (to aid in definition of libraries and other general extensions).

```
DEFAULT ATTRIBUTES human. NOT plural.
:
DEFAULT OBJECT ATTRIBUTES moveable.
:
DEFAULT ACTOR ATTRIBUTES real_name.
:
DEFAULT OBJECT ATTRIBUTES size 0.
```

This sequence illustrates the common practice in v2 to add attributes to the various pseudo-classes, and do it in a piece-meal fashion, usually in connection with the definition of some verbs. The above can easily (simple-mindedly) be translated into v3:

```
Add To Every thing Is Human. Not plural. End Add To.
Add To Every object Is moveable. End Add To.
Add To Every actor Has real_name. End Add To.
Add To Every object Has size 0. End Add To.
```

Note that, in V3, you can add all properties, not only attributes, to a class after its definition.

4.3 Articles

Alan v2 allowed the indefinite article to be modified by the author. Indefinite articles were used in listing of containers and default listings of a location for example.

In V3, the article mechanism has been extended to also include definite articles and, to support languages where also the form of the noun change, the **`Indefinite/Definite Form`** has been introduced. This, in conjunction with the new forms of the **`Say`** statement, to indicate definite or indefinite form (**`Say The x. Say An x.`**), allows more control over the presentation of instances. E.g. it is possible to always use the **`Say The`** form in the printout and let the instance or class handle how to print this instance in definite form. An example would be actors with proper names who usually do not want a definite article in front of their name.

You should also look for embedded parameter references in strings (`"You xxx the $o."` etc.). These should be changed to use the new embedded definite and indefinite form printout, `"You xxx $+1."`, or even better, to the form

```
"You xxx" Say The p. "."
```

4.4 Containers

Version 2 and earlier allowed "pure" containers, i.e. entities that where only containers. In early versions, this was the only kind of container and you had to connect that container to the object or actor by statements. In v2.8, you could add the container property to both actors and locations.

Version 3 does not support “pure” containers. One common use was the inventory of the hero. This old style inventory handling will have to be converted to use of the built in container property of the hero. E.g.

```
CONTAINER inventory
   HEADER “You are carrying”
   EMPTY “You are empty-handed.”
END CONTAINER inventory.
…
VERB take
   DOES
      LOCATE o IN inventory.
END VERB take.
```

Will have to be converted to

```
The hero Isa actor
  Container       -- the inventory
     Header “You are carrying”
     Empty “You are empty-handed.”
End The hero.
…
Verb take
   Does
      Locate o In hero.
End Verb take.
```

Pure containers were also sometimes used to collect items that should always be where the hero was. Since pure containers are no longer supported this has to be implemented in another way.

An instance of the predefined class **`entity`** does not have the properties of objects and actors amongst which are the properties of having a location and being described. Therefore, you could declare an instance like

```
The air Isa entity      -- note: inheriting directly from entity
  Container
End The air.
```

This container will be available at all locations but it will never be described so it can be used in the same way as pure containers in version 2.

An alternative is to make the things that should be always available inherit from **`entity`** themselves. This will make all of them available at all times.

4.5 Pronouns

In V2, the language in itself defined a few pronouns. V3 allows an author to take control over the pronoun handling. This should however, make little difference when porting a V2 game.

5 VERBS AND SYNTAX

5.1 Global Verbs

In version 2, global verbs had the semantics of being a generalization of verbs inside objects. I.e. it was assumed that they would be used with parameters.

In version 3, you must add such generalized verbs to a common super-class, e.g. the pre-defined class **`thing`**. Verbs inside classes or instances having no explicit syntax will receive a default syntax similar to the one in version 2:

```
<verb> = <verb> (<class>).
```

The default identifier for the parameter is the name of the class (or class of the instance) in which the verb was declared.

Version 3 assumes that global verbs (verbs outside of any class or instance) are intended to have no parameters. Global verbs without an explicit syntax therefore receive the default syntax of:

```
<verb> = <verb>
```

Global verbs with parameters from pre-3 version source must be moved to an **`Add To`** clause for the appropriate class, usually **`thing`**, but if your verb handles literals you should move it to **`string`** or **`integer`** as appropriate.

You can use the compiler switch **`-information`** to see compiler messages indicating which verbs get which default syntaxes.

5.2 Verbs in Locations

Verbs in locations was, in Alan v2, executed simply when the hero was in them. In v3, this is also true. However, given the fact that location inherits from the same base class, **`entity`**, as objects and actors do, it may be beneficial to study the sections on the Alan v3 class hierarchy and how that affects the execution of verbs.

5.3 Syntax and Syntax Restrictions

In version 2 it was possible to refer to the (single) parameter of a syntax construct using the predefined **`OBJECT`**, which was meant for use with default syntaxes. In v3, this is no longer possible. On the other hand, it is allowed to define a syntax that has “object” as the name of the parameter.

Version 2 allowed restrictions to list multiple classes in the same restriction clause:

```
Syntax v = v (o)
   Where o Isa Actor Or Object …
```

This is no longer possible; each clause must specify a single class:

```
Syntax v = v (o)
   Where o Isa actor Else …
   And o Isa object Else …
```

The contorted example above illustrates the point. Usually what needs to be done is a rephrasing so that the restriction actually refers to the common parent class to the two in the original:

```
Syntax v = v (o)
   Where o Isa Thing …
```

This will allow both actors and objects at that position. If you want to still keep the multiple restrictions, note that successive restrictions should be progressively more restrictive, i.e. the class should be more specialised.

The analysis of the ELSE-part of a restriction clause was incomplete in version 2 and allowed the use of parameters in ways that was not actually safe. This is improved in v3, possibly giving rise to errors in these clauses in games converted from v2.

5.4 Syntax Synonyms

A feature often requested is action or verb synonyms. In v3 this is possible. By declaring multiple but different syntaxes for the same verb, they will in fact work as such:

```
Syntax give = give (o) to (a) Where …
Syntax give = give (a) (o).
```

In v2 this was an error. A usual remedy was often to use separate verbs (with each its own syntax) and list both (or all) in the verb declaration:

```
VERB give, give_to …
```

This was troublesome, and incomplete, since in the above example the parameters are not in the same order. Any such attempt should be replaced by the new feature, and multiple verbs in verb declarations avoided.

6 RULES

The rules for rules has radically changed in Alan v3. They are now triggered when the conditional *becomes* true. They execute without location or actor, so **`Current Location`** and **`Current Actor`** is not defined. Neither is **`Here`** and **`Nearby`**. Output statements have no effect in rule bodies.

The best advice is to study the relevant parts of the manual carefully. Consider the hints and tips that might be applicable to the problem rules was trying to solve. And then experiment to find a way to implement the same thing.

7 EXPRESSIONS

7.1 The LOCATION, OBJECT & ACTOR Variables

Version 2 had the three expressions **`LOCATION`**, **`OBJECT`** and **`ACTOR`**. They referred to: the location where the execution was taking place, the first parameter in the current input, and the currently executing actor respectively.

Those expressions are, obviously, not available anymore. The words are no longer keywords and the identifiers actor, object and location can now be used as any other identifiers. (Remember though that the language predefines them as classes corresponding to their previous semantics.) The need to refer to the current actor (**`ACTOR`** expression) and the current location (**`LOCATION`** expression) remains and has, in V3, been replaced by two new expressions:

- **`Current Actor`**
- **`Current Location`**

The reference to the first parameter in a syntax declaration, using `object,` is not available except for verbs declared in an object without an explicit syntax. (It will receive a default syntax, where the class name is the identifier for the first parameter.)

7.2 Aggregates

Aggregates are functions that calculate values from sets of items. In Alan v2, the **SUM**, **MAX** and **COUNT** aggregates worked on objects only. You could only filter the objects aggregated to be only those at a location.

Alan V3 has generalized all aggregates to allow a list of filters. The following is a V2 aggregate expression:

```
IF COUNT HERE > strength OF bridge THEN …
```

In order to be exactly equivalent in V3 it must be changed so that the aggregation is only performed over objects:

```
If Count Isa object, Here > strength Of Bridge Then …
```

Naturally, if you have made other changes to your code, this might need further analysis.

8 MISCELLANEOUS

8.1 Scripts

Version 3 only allows named script as opposed to the numbered scripts also allowed in v2.x. A simple solution is to put a single character in front of the number. A better solution is to invent descriptive names for the scripts instead.

8.2 Multimedia

Alan v2 had no provisions for multimedia. It has been reported that the SYSTEM statement has been used to some extent. Alan v3 provides the **`Show`** statement to support graphics in a portable way. Note that this might still not be available on all platforms.

8.3 Include directive

In v3 the **`$include`** has been replaced by the **`import`** statement. They work the same, except that the new **`import`** statement can only be placed where declarations (classes, instances etc.) can occur. This ensures a better structure of the contents of files, and aids in developing tools for analysis of Alan source.

8.4 Messages

Most default messages given as a response to the player have changed from V2 to V3. In V3 they all use the definite and indefinite features of course. But the message sections are also defined so that run-time parameters appropriate for attribute testing or printout are available. The most significant change is that many of the identifiers for the messages have changed. You should read the appropriate parts of the Alan Reference Manual for a list of V3 message identifiers.

8.5 New Features

Of course, there are new features not available in v2, but these rarely affect the porting of source from a v2 game, except where new keywords have been introduced in the language. To retain V3 keywords as identifiers you need to quote them. Another option is to replace them completely.