Alan IDE Reference Guide

by Robert DeFord

Version: Final 1.0

Uncopyrighted

1. About This Guide

This guide is written for Alan IDE version 0.1.5 running on the Windows 10 operating system.

This guide covers only the Alan IDE functionality that the author has learned over the course of creating several Alan games with the Alan IDE. As such, it should not be considered to be a complete reference for the Alan IDE.

2. Installing the Alan IDE

The procedure that follows assumes that you are setting up your Alan development system on a Windows 10 computer with the intention of using the Alan IDE to create Alan games.

2.1. Verify your Java Installation

You must have Java installed on your computer in order to use the Alan IDE. Perhaps the easiest way to verify that you have Java is to open the Windows Control Panel and select **View by/small icons.**

- If you see the *Java Control Panel* icon, then you are probably good to go as long as you used the default settings during the Java installation process.
- If you don't see the Java icon, you can get Java from https://www.java.com/en/. Just click on the Free Java Download button and follow the instructions. It's a good idea to stick to the defaults during the installation process.

> According to the Alan web site, a Java runtime environment 1.5 (5.0) or greater is supposed to work. However, for security reasons, it is a good idea update your Java to the latest version. For example, you can double-click the

> Java icon to get the Java control panel, then click on the Update tab to get to the update screen, and then use the **Update Now** button. As a point of reference, the author's Java version is 1.8.0_161 as of this writing.

2.2. Get the right compiler

The Alan IDE uses the *command line* version of the Alan compliler, so you must download the command line Alan SDK even if you've already downloaded and are using the Alan GUI SDK.

1. Download the latest **Alan SDK for Windows (command line)** package from the Alan website: https://www.alanif.se/download-alan-v3/latest-releases.
2. Unzpip the command line SDK into a directory of your choice. For example, you could create a directory on your desktop, name it *Alan*, and then unzip the SDK into it.

2.3. Install the Alan IDE

You will have to install the Alan IDE manually.

1. Go to https://www.alanif.se/download-alan-v3/alanide and download the latest version of the Alan IDE.
2. Unzip it into the directory of your choice. For example, you could use the same directory as the one you put the Alan command line SDK in.
3. Open your chosen directory to see the list of files in it.
4. Create a shortcut to the **AlanIDE.exe** executable file in a convenient place, or simply pin it to the Taskbar.

> Over the course of writing this guide, the author had to change the compatibility mode for the Alan IDE executable to run in Windows 7 compatibiity mode in order to restore certain functionality that *seemed* to be lost after a Windows 10 update. You are advised to do the same:
>
> 1. Right-click on *AlanIDE.exe*, and select **Properties** from the dropdown menu.

2. Select the **Compatibility** tab.
3. Set the *Compatibility mode* to run the program in compatibility mode for **Windows 7**.
4. Click OK.

2.4. Configure the Alan IDE

Before you can use the Alan IDE you must configure it for use with your directory structure.

1. Launch the Alan IDE and wait for it to do an automatic update. Click the **Edit** button on the menu bar to get the drop-down menu.
2. Select **Preferences** from the drop-down menu to open the *Preferences* window.
3. Select **Alan** in the left column of the window to get the *General preferences* settings for Alan IF development screen.
4. Click in the **Selected Compiler** field to select it.
5. Click the **Browse** button for that field, navigate to the directory where you put the Alan command line SDK, and then open that directory.
6. Select **alan.exe** from the open directory to enter its path into the *Selected Compiler* field.
7. Leave the *Path to Standard Library* field blank.
8. (OPTIONAL) Click the **Generate debug information:** check box if you wish to use the Alan compiler outside the IDE workbench to troubleshoot your project. You probably won't need to do this, but it won't hurt to generate the information in case you do.
9. Click the **OK** button to close the Preferences window.

2.5. Choose your Interpreter

You must select which Alan Interpreter that you are going to test your game with while you are using the Alan IDE. You can use any Alan interpreter such as *Arun* or *Gargoyle*, but your best bet is to use the *WinArun* interpreter. It is easy to install, always up to date with the latest Alan release, and it looks a lot better than arun.exe.

- Download the latest *WinArun* installer from the Alan website. (https://www.alanif.se/download-alan-v3/interpreters/interpreters-3-0beta5 as of this writing).

    1. Double-click the installer executable, **winarun3_0beta5.win32.x86.setup.exe,** as of this writing, to install the WinArun interpreter.
    2. Use the Windows operating system to make WinArun the default application for running *.a3c* files so that you can launch an Alan .a3c game by double-clicking on its filename.

You can do this step using the Windows *Control Panel/Default Programs*, or you can right-click on any .a3c file to get the pop-up menu, and then select the *Open With* option.

3. Alan IDE Reference Section

Once you have installed and configured the Alan IDE, you can create an Alan game from start to finish with it. The topics in this section are loosely arranged in a logical order that more or less reflects the series of steps you will take in developing your game.

3.1. Alan IDE overview

When you launch the Alan IDE, it displays a workbench, which consists of a main window with four sub-windows:

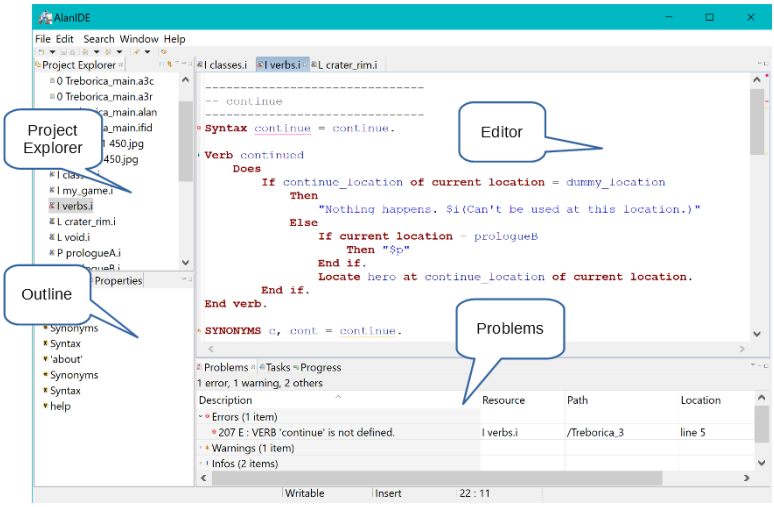

Figure 1. The Alan IDE workbench

- **Project Explorer:** Displays a list of the projects in the Alan IDE's workspace. (This example shows several projects in the IDE's workspace, but there can be any number from zero on up.)
- **Editor:** This is where you write the code for your game. You can open and work on several different files, each under a separate tab. (The default limit is 8 tabs.)
- **Outline:** Displays an outline of the file currently being edited in the Editor. When you double-click one of the items in the outline, the editor will scroll to the code for that item.
- **Problems** This tab lists compiler error messages, if any. When you expand the list, you will see a textual explanation for each problem. When you double-click one of the problems, the editor will open the appropriate file and display the code where the compiler encountered the error.

You can resize the workspace sub-windows. Just use hover the cursor over a border rule until you get the double-arrow, then click and drag the border.

3.2. Starting a new game

Starting a new game is a two-step process. First you create a project to work in, then you create a main file for the new game's source code.

Create a project for the new game

The Alan IDE keeps all the files for a single game in a special container called a *project*. You will need a project for each game that is under development.

1. In the Alan IDE, click on **File** in the menu bar and select **New Wizards/Alan Project** from the pop-up menu to get the *Create New Alan Project* dialog box.
2. Think of a name for your project and type it in the *Project name* field.

By default, the location for your project is the Alan IDE workspace directory. As a general rule, that's the best place for it, so leave the *Use default location* option selected.

3. Click the **OK** button to create the project.

   The Alan IDE will recompile the workspace, and your new project will show up as an empty folder over in the Project Explorer sub-window.

Create a main file for the new game

Once you have an empty project to work in, you then create the first source code file for your new game. This mandatory first file is referred to as the *main file* for the game.

1. Over in the Project explorer, right-click on the name of the new project to get a dropdown menu.
2. If the project is closed, select **Open Project** from the drop-down menu to open that project so you can work in it.
3. Click on **File** in the IDE menu bar, and select **New Wizards/Alan Main File** from the drop-down menu to get the *Create an Alan Source file* dialog box.
4. Put a name for your main file in the File name field. For example, *main.alan*.

By default, the Container field already has a path to the selected project in it, which is what you want, so you can ignore the Browse button.

You can name this primal file with any filename you wish, but you must use the **.alan** extension for it. This extension makes it the main file for you game. In the case where the source code spans more than one file, the compiler looks at the main file first and uses the import statements in that file to compile the complete game.

5. Click the **Finish** button to add the file to the open project.

The Alan IDE will do an automatic save and attempt to compile your new game. Over in the Editor you will see a tab for the main file you just created, and it is currently open for editing. As you can see, the wizard automatically generated the first line of code for you:

```
Start At 1.
```

This line of code defines what is referred to as the **start section** for a game. The start section must always be the **last** chunk of code in the main file for a game. At a minimum, the start section must state the first location that the player sees when the game starts. In this case, this line of code simply tells the compiler to make the location named **l** be the location in the game that the player will start in.

Over in the Project Explorer, you will notice that the *.alan* file you just created is listed as a file in your project. Along with that file, you will notice that the wizard automatically created a *.ifdb* file and added it to your project as well. It contains only the unique IF database identifier code for the game, and you can safely ignore it while developing your game.

At this time, your workbench should now look something like this:

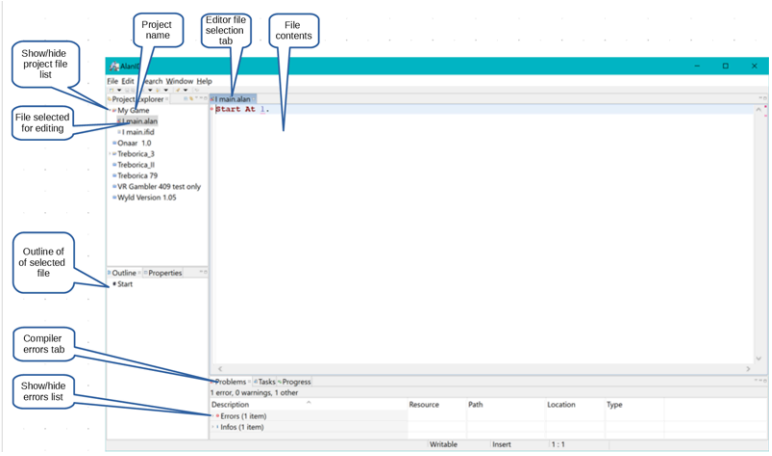

Figure 2. A newly created main file

3.3. Finding and fixing compiler errors

Compiler errors are a common type of error that you will get while developing your games. An example follows that shows an example of the process used to fix a compiler error, but here is a brief overview of that process to give you an idea of what is going on ahead of time:

1. Examine the error message's text to figure out what the compiler is complaining about and what it thinks might resolve the issue.
2. Modify the offending code statement(s) in the source code.
3. *Save* the modified file.

The compiler will recompile the edited code and report on any errors it finds. Hopefully, your modified code will compile without errors.

For example, suppose you just followed the procedure in this guide to create a new .alan main file for a new game and have not yet written any code of your own. Over in the editor, where it is displaying the .alan file, you will see that the compiler is reporting an error down in the Problems sub-window:

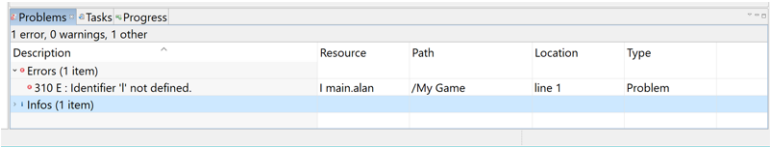

Figure 3. Problems sub-window

This report indicates that the compilier found something it could not compile in the line of code that the *New Main Alan File* wizard automatically created for you. You can fix this particular error with the same technique that you can use later on when you write your own code:

1. Click on the **Errors (1 Item)** line down in the Problems sub-window to select it.
2. Click on the **>** that points to the word Errors to expand the list of errors. (In this case there is only one item on the list.)

As you can see, this error report has several parts:

- **301 E** is the compiler's internal error code.
- **Identifier I not defined.** is a plain text explanation of the error written from the compiler's point of view. In this case, the compiler is telling you that you have referred to an object named **I** that is not defined elsewhere in your source code.
- **main.alan** is the filename of the file where the error occurred. In this case, since you have only one source code file, it may not seem very useful, but it will be later on when your source code spans multiple files.
- **/My Game** is the path to the file. In this example, it is a project folder named *My Game* in the IDE workspace.
- **line 1** is the line number where the error was found in the indicated file.

Line numbering has not been enabled in the IDE editor in the examples in this guide.

Since there is only one line of code in this example's source code, it is pretty easy to see where the offending line of code is. However, if your source code were to span a dozen or so files with thousands of lines of code, that would not be the case.

Fortunately, the Alan IDE provides a simple way to zoom right to the offending line of code; simply double-click the error message in the Problems sub-window.

In response the Editor will open the appropriate file and scroll to display the offending line of code.

To fix the error in this example you would proceed as follows:

1. Create a location object somewhere ahead of the Start section in the main .alan file, and change the L in the offending line to be the name of your new location object. For example, your code could look something like this:

```
The starting_place Isa location
    Name Kansas
    Description "This place looks like the place where the story
        begins."
End The starting_place.

Start At starting_place.
```

2. Select *File/Save* from the Alan IDE main menu.

Any time you Save a file in the Alan IDE, it automatically compiles the file and updates the error report.

If you have written a lot of code without saving and recompiling, you can easily get a confusing array of interdependent errors. Consequently, it is a **very** good idea to save your work after each and every chunk of code you write.

If you have edited multiple files in a project, you should use *File/Save All* from the Alan IDE main menu.

3.4. Adding an existing file to a project

You will no doubt need to add an existing file to your project from time to time. For example, copies of the Standard Library files are usually added to the a project prior to writing any source code, or you might wish to re-use a source code file from another game.

1. Navigate to the file's location on your computer, then right-click on the filename and select **Copy** from the drop-down menu.
2. Over in the Project Explorer, click on the destination project's name to select it.
3. Right-click on the project name to get the drop-down menu, and select **Paste**.

   The file should appear in the list of files for the destination project.

4. In the Project Explorer, double-click the main **.alan** file in the open project to open it in the editor for editing.
5. In the Editor, on a line by itself, somewhere prior the Start section, enclose this statement in single quotes followed by a period: **IMPORT <filename>**, where <filename> is the name of the file you copied into the project. For example:

```
IMPORT 'loc_kansas.i'.
```

When you Save the main file, the compiler will now include the code in that file when it creates the executable file for the game.

3.5. Creating a new file for a project

While it's possible to create a large game with just the main.alan file, trying to work with just one massive file can get cumbersome. It's a good idea to divide your source code into multiple files with meaningful names for each file. Fortunately, the Alan IDE makes the process quick and easy:

1. In the Project Explorer, right-click on the project name to get the drop-down menu, and then select **New/File** to get the New File dialog box.
2. Click in the *File Name* field, then type a name for your new file and end with the **.i** filename extension. This extension designates the file as an *import* file.

Every file in a project that contains **source code** you intend to be part of your game, must end with the .i filename extension. Other files, such as .txt files, can be in the project and used for notes, reference material, and so forth, but they won't be complied into the game.

3. Click the **Finish** button. The file will appear in the Project Explorer as a new file in the project.
4. In the Project Explorer, double-click the main **.alan** file to open it in the editor for editing.
5. On a line by itself, somewhere prior the Start section, enclose this statement in single quotes, **IMPORT <filename>**, and end the line with a period.

For example, with a file named *L sand_cave.i* you would do this:

```
Import 'L sand_cave.i'.
```

From now on, when you Save the main file, the compiler will compile the code in this file along with the other files in the project as it creates the executable file for the game.

There is no limit to the number of files you can have in a project. For example, at a minimum, this author has a file for each location in the game, a file for the custom verbs, a file for each of the game's infrastructure elements, and all the files in the Standard Library.

3.6. Closing an open project

You don't have to close a project between work sessions, and it is entirely possible to have multiple open projects in the Alan IDE workspace. However, it is sometimes a good idea to close an open project. For example, this author likes to keep just the current, active project open between work sessions because the editor could well have multiple tabs open on files from multiple open projects, which means that you can easily edit a file from the wrong project. (It has happened.)

Over in the Project Explorer, right-click a project, and select **Close Project** from the drop-down menu to close the project. In response, the toggle button immediately to the left of the project name will disappear.

3.7. Opening a closed project

When you sit down at the computer to continuing working on a game with the Alan IDE, you will typically have to open an existing project that you closed when you ended a previous work session.

- In the Project Explorer, right-click a project and select **Open Project** from the drop-down menu to open the project.

In response, you will see a toggle button immediately to the left of the project name. That toggle tells you that the project is open.

3.8. Listing the files in a project

Over in the Project Explorer, click on the toggle button immediately to the left of an open project to expand the project and display a list of the files in that project.

3.9. Opening a project file for editing

Over in the Project Explorer, double-click a file in an open project to display the contents of that file in the Editor under its own tab.

3.10. Closing an open file

In the Editor, right-click on the tab for a file to get the dropdown menu and select one of the Close options. Or, you can simply click on the close button on the tab.

3.11. Saving and Compiling your source code

On the IDE menu bar, click on **File**, and select either *Save* or *Save All*. The IDE will save your edits in the open project file (or files) and compile the game to create (or overwrite) an **.a3c** game file, which then appears in the Project Explorer.

Get into the habit of using *File/Save All* to save the changes in **all** the open files in a project instead of *File/Save* to save just the one file. It will save you time and trouble in the long run.

3.12. Testing your game

Just because your game compiles without any compiler errors, does not mean that the game will do what you want for the player, or is bug free. Most authors do incremental testing throughout the development process instead of waiting until the game is complete.

1. Save and compile the project.
2. Over in the Project Explorer, double-click the .a3c game file to launch the game with the interpreter.
3. Play-test the game as if you were the player, taking notes as you go if you see any issues.

4. Close the interpreter, then go back into the Alan IDE and fix the issues by editing your source code files and saving the modified files to compile them.
5. Over in the Project Explorer, double-click the .a3c game file to launch the game with the interpreter.
6. Play-test the game again as if you were the player, taking notes as you go if you find any issues that were not resolved by your code changes.

Repeat these steps over and over again throughout the development of your game.

You can write a special command in your game's source code that will allow you to set up the conditions required to test a specific chunk of code without having to play the game from the start.

3.13. Backing up your project

It is a good idea to back up your project periodically during the development process. For example, at a minimum, this author does a back up at the completion of each day's work, and often after reaching any significant plateau during a work session.

1. Create a backup directory somewhere on your computer system. For example, this author creates a backup directory in an external USB drive.
2. Back in the Alan IDE, in the Project Explorer, click on the *project name* to select it.
3. Right-click on the project name to get the dropdown menu, and select **Export** to get the Export dialog box.
4. Click the **Next** button to get the File System dialog box.

Your author has two computers running the IDE, and they do not behave the same for this procedure. You may have to clik on the General folder in the list, then click the toggle to get a drop-down list, then double_click File System to get the File System screen. The project name should be selected.

5. Put a check mark next to the *name of the project* in the list of open projects to select it.
6. Click the **Browse** button, then navigate to the backup directory you created earlier.
7. Click on the *name* for the backup directory to select it.
8. Click the OK button to put that directory name into the *To Directory* field. (Leave the default options as they are.)
9. Click the **Finish** button.

Now that you've created the backup file, go outside the Alan IDE and use the file explorer to navigate to the backup directory and open it to make sure that a copy of your project is now in the list of files in the backup directory.

Edit the filename in your backup directory to add some sort of unique identifier. For example, add a unique version number to the filename of each backup. These identifiers will keep you from overwriting an existing backup the next time you do a backup of the project. Also, as you accumulate multiple backups for a project, they will help you keep track of them.

3.14. Removing a existing project from the workbench

There are occasions when you wish to remove a project from the Alan IDE's workspace. For example, maybe you are finished with developing a game, or you may have inadvertently messed up the story line so badly that you wish to begin anew, or you've decided to use an entirely different approach to the game's code structures, and so forth.

It is a very good idea to back up the project prior to deleting it from the Alan IDE's workspace.

1. Over in the Project Explorer, right-click on the name of the project you wish to delete and select **Delete** from the drop-down menu to bring up the *Delete Resources* dialog box.

2. **(Optional)** Click on the box next to the *Delete project contents on disk (cannot be undone)* option if you wish to delete the project from your disk (as opposed to simply deleting its name from the Project Explorer list).
3. Click the **OK** button to finish the deletion.

3.15. Copying code from one file to another

When both files were created with the Alan IDE, and located within projects in the workspace, simply open both files in the Editor, then copy-paste the code.

If one of the files was created with an external editor outside the Alan IDE, the process is a little different. In this case, just open that file with the external editor, and copy the code to the clipboard. Then, in the Alan IDE, open the destination file and paste the code into it.

You may wish to create an empty file in an Alan IDE project to serve as the destination file.

The external file must must be a plain text (.txt) file.

3.16. Importing an existing project into the workbench

There are occasions when you wish to work on an existing project you have in storage. For example, you might want to fix bugs that were reported after you have released a game.

1. Over in the Project Explorer, right-click somewhere in the white space outside the list of projects to bring up the popup menu, then select **Import** to bring up the *Import* dialog box.

The author's two systems act differently on this step. You may have to expland the General folder and select *Existing projects* into workspace.

2. Click the **Next** button to advance to the *Import Projects* dialog box.
3. Click on the Browse button next to the Select Root Directory and navigate to where you stored the existing project.

4. Click on the existing project's filename to select it.

You can import more than one project at the same time.

5. Click the **OK** button to it into the *Select Root Directory* field.

   - Down in the **Projects** list, you should see the name of the existing project.

6. **(Optional)** If you have more than one existing project to import, click the Select All button.
7. Leave the default selection *Copy projects into workspace* checked, or check it if it is not already checked.
8. Click the **Finish** button.

The project should appear in the Project Explorer list.

3.17. Changing syntax highlighting colors

The default colors for highlighting the Alan source code in the IDE editor work well for this author, but you may be wish to change them to something you like better.

1. In the IDE menu bar, click on the **Edit** button to bring up the drop-down menu.
2. Select **Preferences** to bring up the Preferences dialog box.
3. Click on **Alan/Syntax Coloring** over in the left hand colum to bring up the *Syntax Coloring* preferences interactive display.
4. Click on the syntax element color you wish to change to bring up the *color selector*.
5. Click on the *desired color*.
6. Click **OK**.

3.18. Showing line numbers

You really do not need line numbers in the IDE Editor, but they are sometimes useful if you are troubleshooting your code with the help of an expert and wish to point out specific lines in your code, or maybe you just want to refer to specific lines in your code comments.

1. In the IDE menu bar, click on the **Edit** button to bring up the dropdown menu.
2. Select **Preferences** to bring up the Preferences dialog box.
3. Select *General/Editors/Text Editors* to bring up the Text Editors dialog box.
4. Click on the Show Line numbers check box to turn on line numbers in the Editor.
5. Click **OK**.

3.19. Changing the default keys

1. In the IDE menu bar, click on the **Edit** button to bring up the dropdown menu.
2. Select **Preferences** to bring up the Preferences dialog box.
3. Select *General/Editors/Keys* to bring up the Text Editors dialog box.
4. Change the **Binding** for each key as desired.
5. Click the **OK** button.