Pleo 1.1
Porting Guide

Table of Contents

Table of Contents **2**
Overview **3**
Pleo 1.0.x *3*
Pleo 1.1.x *4*
Porting *5*
Name Changes *7*
Other Coding Issues *9*
Document Revision History *10*

Overview

The initial releases of the Pleo software – versions 1.0.x – were designed and built with Pawn version 3.2. Ugobe added two additional features to the Pawn 3.2 subsystem: code blocking and incremental execution. Code blocking is used to allow the execution of arbitrarily large scripts, and incremental execution prevents any script from blocking the execution of the rest of the system.

In the Pleo 1.1.x series of software we have moved to the latest Pawn 3.3 version. This version includes a built-in feature called overlays that replaces our home-grown code blocking. It also includes an additional feature – code packing – that generates smaller compiled code. In addition, the incremental execution feature has been re-implemented using the Pawn monitor hook instead of modifying the core Pawn VM code.

NOTE: This change means that all projects or applications built for Pleo 1.0.x will NOT run on Pleo 1.1.x. and projects or applications built for Pleo 1.1.x will NOT run on Pleo 1.0.x.

Our goal is to be as source code compatible as possible. If you follow this guide, all you should have to do is adjust your Pawn compiler command line settings, rebuild, and you will be ready for Pleo 1.1.

Pleo 1.0.x

The Pleo 1.0.x series of firmwares use Pawn version 3.2 internally. The build tools include the stock pawn compiler – pawncc.exe or pawncc32.exe – and a special post-processor – pleocc.exe - that converts the compiled code in AMX files into blocks.

The Pawn compiler command line is specified in the UPF file, in the options/tools/pawn element, A typical command line looks like the following:

```
      <pawn value="../../bin/pawncc %i -v2 -S128 -C- %I -
o%o"/>
```

where:

- ../../bin/pawncc is the Pawn compiler executable
- %i is the place holder for the input .p Pawn source file
- -v2 turns on verbose output from the compiler, providing more information
- -S128 defines the size of the stack, in cells
- -C- turns off compression of the code (this is incompatible with our Pawn VM run-time).

- %I is a place holder for the include path(s) specified in the options/include element of the UPF
- -o%o specifies the output filename, where %o is the placeholder for the destination path and name

The build tools are instructed to run the pleocc post processor either with an options/block element in the project UPF, or is specified on the command line to the ugobe_project_tool with a -b. The command line for the code blocking tool – pleocc – is typically something like the following:

```
<block value="../../bin/pleocc -b512 -v %i"/>
```

where:

- ../../bin/pleocc is the path to the pleocc executable
- -b512 specifies the size of each block
- -v turns on verbose output mode
- %i is a placeholder for the input AMX file

The pleocc tool will code block the AMX file in place, and set a bit (0x40) in the AMX header that this AMX file has been code blocked. Note this bit-setting is a non-standard extension to the AMX fie format.

Pleo 1.1.x

In the Pleo 1.1.x series of firmware, we have moved to the Pawn 3.3 version of compiler and VM. This gives us the following advantages:

- **Overlays**: overlays are similar to our custom code blocking system, but breaks up the code on function boundaries, not a fixed sized block. This takes better advantage of available memory, since only those functions/overlays will be loaded that need to be loaded. And there is no padding with no-ops as in the code blocking case, which means less wasted memory. See the -V option
- **Code packing**: there are new Pawn instructions that combine both a Pawn operator and its parameters into one opcode, reducing the size of the code generated. On average we have seen a 35% reduction in code size. This of course allows more code to run in the same amount of reserved memory. See the -O option
- **ARM assembly execution core**: there is also a new ARM assembly version of the core opcode execution function in Pawn 3.3. This should allow faster execution of our Pawn code, but thus far no measurements have been taken.

These features taken together allow us to better utilize our limited memory that is reserved for code - currently 8K. There is still some balancing to be done in the

swapping of overlays to and from memory, but our initial starting point is no worse than the current status.

NOTE: Do NOT use the debug level option ‘-d0’ option when compiling scripts for Pleo 1.1. Options '-d1' or '-d2' are fine. We currently depend on the 'run-time checks' that are inserted with the ‘-d1’ or ‘-d2’ in the firmware to execute Pawn code incrementally.

Porting

The key components of porting your code involves the following:

- Removing the code blocking tool – pleocc – from your build.
- Adding the new compiler switches to your Pawn command line to take advantage of the new features.
- Adjusting any source code based on the latest Pleo include files, which have undergone some cleanup.

Removing code blocking

The code blocking tool is told to run either through the options/block element in the UPF file, or the -b option to the ugobe_project_tool. If you have block option in your UPF, you may comment it out as follows:

```
<!--  <block /> -->
```

or remove it completely. Also ensure that there is no '-b' option being passed to the ugobe_project_tool, commonly placed in a build.bat or rebuild.bat batch file.

Adding new compiler options

There are two options that you can add to your scripts to take advantage of the new features. Note that these are optional. If all of your compiled code fits in the reserved code area, then you do not need to code pack or overlay your code. It is usually a good idea to try variations to see what works for you.

- **Overlays**: the new overlay support is enabled with the '-V' command line switch. This switch can take an optional parameter which specifies the maximum size any one overlay can be. In the current Pleo 1.1 implementation, this is 8K, though we recommend you keep the overlays much smaller to allow for better use of the limited code cache. We suggest at most 2K. So, in this case, your command line would include a '-V2048'.
- **Code packing**: code packing is enabled when the optimization is set to it's maximum level, using the '-O2' command line switch. This will enable the use of the new Pawn 3.3 opcodes, which reduce code size in our testing about 35%.

We have cleaned up the Pleo include files, to ensure that your Pawn source is using all the native functions properly. Here are some of the changes, the results you may see, and how to adjust your code for them.

- Removal of all int: tags. This was an extraneous tag, which had no useful meaning. The native type in Pawn is a 'cell', which in our case is a 32-bit integer, so there is no need for this tag. You may see tag mismatch warnings. In these cases, simply remove the int: tags from your code.
- Removal of all void: tags. Again, this tag has little meaning; that is, we cannot define it properly, so it should not be used. You may see tag mismatch warnings. Simply remove all void: tags.
- Multiple tags on some functions: in our system, we often have functions that can take more than one type. In previous include files we did not properly mark these functions (we did not know how). We have since discovered the proper

syntax for these functions. Using the property_set function as an example, this is what is was before:

```
native int: property_set(property_name: property, value);
```

it is now like this:

```
native property_set({property_name,user_property_name}:
property, value);
```

Note the use of multiple tags. This may cause issues if your project has defined a user property (listed in user_properties.inc) of the same name as the system (defined in pleo/properties.inc). This will result in an error, since the compiler cannot know which enumeration value you want, since both types match. You can fix this in two ways:

1. Remove the offending duplicate user property, or

2. Perform a 'cast' of your property to tell the compiler what type you mean. For example:

```
property_set(property_name:property_pose, pose_sitting);
```

NOTE: We have simply pre-pended the enumeration value with the proper enumeration type we wish to set.

Name Changes

Init or Startup script

In Pleo 1.0, there is a specially named script – INIT.AMX – which will be loaded and executed if found at the root of an SD Card upon insertion. After this script finishes, the main application – whether from SD or DF – will be executed.

In Pleo 1.1, since the Pawn version has changed and thus the INIT.AMX script cannot be executed, we have change the special name we look for to STARTUP.AMX. The behavior is identical to Pleo 1.0, but we simply look for a different name.

So, if your project included an init.p in order to do some initial processing in Pleo 1.0, this script will have to renamed to startup.p in Pleo 1.1.

Main Application

- In Pleo 1.0, the firmware would look for a file named 'pleo.urf' to execute. No other URF would be executed (unless explicitly given in an app load command).
- In Pleo 1.1, we can and will execute any other URF file except 'pleo.urf'. We scan the SD Card, looking for all files ending with .URF. The first one that is not named 'pleo.urf' will be executed. Note also that the base name can be greater than 8 characters – up to 32 characters in fact.
- Note also a change in the project tool that parses UPF files: it will now use the name of the 'ugobe_project' element as the name for the output URF file. This should reduce the need to rename the resultant files created.
- These changes allow the potential to create one SD Card that can execute on both Pleo 1.0 and Pleo 1.1 firmware versions.

Other Coding Issues

Here are some additional Pawn programming tips which are not directly related to porting. That is, they apply to both Pawn 3.2 and Pawn 3.3.

3. The warning number 203 - “symbol is never used: identifier” - can be issued on small helper functions, resulting in a very large number of warnings when compiling larger scripts. To eliminate these warnings, mark your small helper functions with the stock keyword. This tells the compiler that these functions may not always be called.

4. Pawn, by default, uses unpacked strings. That is, a string where each character takes up one cell. This is wasteful of memory. To save memory, it is recommended that you compress all strings. You do this the #pragma pack 1 pre-processor directive. Put this at the top of your .p file(s), and the compiler will treat all strings as packed.

5. It is extremely useful to review the Pawn language reference from CompuPhase. The latest versions can be found here: http://code.google.com/p/pawnscript/

Document Revision History

Revision	Date	Comment
0.1		*Initial version*
0.2		*Add Startup Script Items*
0.3		*Add pleo.urf note and rename section to Name Changes*
0.4	*May 07, 2008*	*Some formatting updates*
0.5	*May 30, 2008*	*Formatting changes*