Porting Pawn implementations or scripts from earlier releases

Pawn version 4.0 is not fully backwards compatible with earlier releases. The incompatibilities exist both in the "abstract machine" (the interface to the host program) and in the scripting language itself. This document gives an overview of the required modifications.

The scripting language

- The **`char`** keyword is gone. To declare an array to hold 10 characters, use
  ```
  new array{10}
  ```
  instead of
  ```
  new array[10 char]
  ```
- Literal arrays (array initialization) must now be done with "**`[...]`**" instead of "**`{...}`**". The "**`{...}`**" syntax now means to create packed literal arrays. For example:
  ```
  new primes[5] = [1, 2, 3, 5, 7]        // an unpacked array
  ```
  or
  ```
  new primes{5} = {1, 2, 3, 5, 7}        // a packed array
  ```
- Packed literal strings now use the **`"..."`** syntax; to create an unpacked literal string, use the **`''...''`** syntax. That is: packed strings are between double quotes and unpacked strings are between pairs of single quotes. The first pair of single quotes may also be back-quotes (for example: ``` ``monkey'' ```).
- The **`enum`** keyword is gone. The enumeration functionality is mostly combined with **`const`**. The exceptions are the special "incrementers".
- Arrays based on enumerations are no longer special. This part has been replaced with symbolic subscripts. See the manual on using symbolic subscripts.
- Native functions that return arrays, must be declared with the array dimensions and sizes between the function name and the opening parenthesis of the parameter list. So, for example:
  ```
  native ShuffleCards [52] (Method)
  ```

Changed behaviour of the compiler

The Pawn compiler can now generate code for cell sizes of 16-bit, 32-bit and 64-bit. The cell size is selected with a command line option: **`-C`**. In previous releases, the option **`-C`** specified "compact encoding". In addition, there is now an additional level of optimization.[1] If you launch the Pawn compiler from an IDE or build scripts, you may need to adjust these for the changed meaning of command line options.

The abstract machine (and native function libraries)

The abstract machine has changed significantly from the releases 3.3 and earlier. Notably, the instruction set has been reorganized and reduced, and the interface to accessing memory regions ("variables") in the abstract machine is simplified. These changes make the abstract machine incompatible with the earlier releases —in most cases, you will have to modify your host application to build with the new version of Pawn.

New instruction set and file format

The new abstract machine only runs P-code that is compiled with the recent Pawn compiler. The instruction set has been re-organized and separated into three categories (core instructions, supplemental instructions and packed instructions).

In the file format, the "packed encoding" is no longer supported, as it complicated the abstract machine for reasons that are no longer important (minimizing the size on disk that a compiled script takes). Also, with the introduction of packed opcodes, packed encoding became less effective.

1 The optimization level is now linked to the instruction set. Optimization level 1 generates core instructions only, level two core and supplemental instructions, and level 3 generates the full instruction set (including packed instructions).

Changed macro names

Several macros that are used for conditional compilation were renamed. The macro **JIT** is now called **AMX_JIT** and the macro **ASM32** is now called **AMX_ASM**. If your project uses the JIT or the assembler core of the abstract machine, you may need to modify the (CMake) makefile or your project settings. Note that you also need to add another C file in the project (see "Extra project files" below).

Extra project files

To make the standard C implementation more readable, various "options" have been split off into separate files. By default, the abstract machine uses the ANSI-C core for the P-code interpreter. If you wish to use the core optimized for the GNU GCC or the Intel C/C++ compilers, you need to add the file **amxexec_gcc.c** to your project. Additionally, the main file, **amx.c**, must be compiled to accept this alternate P-code interpreter; to do so, add the macro **AMX_ALTCORE** to the command line options of the compiler.

The macros **AMX_JIT** and **AMX_ASM** imply **AMX_ALTCORE**.

No separate "abstract" and "physical" pointers

In release 3.3 and before, functions like **amx_Allot** and **amx_PushString** returned *two* pointers: one with the *virtual* address relative to the abstract machine and one with the true address that the host application can use to access the allocated memory block.

In the current release, only the "host" pointer (the *physical* pointer) is returned. The abstract machine derives the abstract virtual pointer from this address, if needed. If the host application calls a public function and needs to pass in the address of allocated memory, the new function **amx_PushAddress** translates the pointer to an abstract address and pushes that onto the stack of the abstract machine.

Reference arguments of native functions

In release 3.3 and before, native functions received "reference" parameters as a virtual pointer that was relative to the abstract machine. The native function had to translate that parameter to a physical pointer by calling **amx_GetAddr**. In the current release, reference parameters to native functions are already translated to memory addresses that the native function can use directly ---although a typecast to cast the parameter from type **cell** to a pointer is typically desirable.

Function **amx_GetAddr** no longer exists; it is replaced by the *macro* **amx_Address**. The expansion of the macro depends on the cell size and the CPU architecture, in a typical case, it is merely a typecast. In practice, it means that snippets similar to:

```
cell *cellptr;
amx_GetAddr(amx, params[1], &cellptr);
```

must be replaced by:

```
cell *cellptr = amx_Address(amx, params[1]);
```