Pawn

embedded scripting language

Time Functions Library

August 2007

Abstract

The "Time Functions Library" adds a set of general purpose functions to the PAWN scripting language. The library provides an interface to standard "time of the day" as well as a millisecond-resolution timer.

The software that is associated with this application note can be obtained from the company homepage, see section "Resources"

Typeset with TEX in the "Computer Modern" and "Palatino" typefaces at a base size of 11 points.

Introduction

The "PAWN" programming language depends on a host application to provide an interface to the operating system and/or to the functionality of the application. This interface takes the form of "native functions", a means by which a PAWN script calls into the application. The PAWN "core" toolkit mandates or defines *no* native functions at all (the tutorial section in the manual uses only a *minimal* set of native functions in its examples). In essence, PAWN is a bare language to which an application-specific library must be added.

That non-withstanding, the availability of general purpose native-function libraries is desirable. The "Time Functions Library" discussed in this document intends to be such a general-purpose module.

This application note assumes that the reader understands the PAWN language. For more information on PAWN, please read the manual "The PAWN booklet — The Language" which is available from the company homepage.

Implementing the library

The "Time Functions Library" consists of the two files AMXTIME.C and TIME.INC. The C file may be "linked in" to a project that also includes the PAWN abstract machine (AMX.C), or it may be compiled into a DLL (Microsoft Windows) or a shared library (Linux). The .INC file contains the definitions for the PAWN compiler of the native functions in AMXTIME.C. In your PAWN programs, you may either include this file explicitly, using the `#include` preprocessor directive, or add it to the "prefix file" for automatic inclusion into any PAWN program that is compiled.

The "Implementer's Guide" for the PAWN toolkit gives details for implementing the extension module described in this application note into a host application. The initialization function, for registering the native functions to an abstract machine, is `amx_TimeInit` and the "clean-up" function is `amx_TimeCleanup`. In the current implementation, calling the clean-up function is not required.

If the host application supports dynamically loadable extension modules, you may alternatively compile the C source file as a DLL or shared library. No explicit initialization or clean-up is then required. Again, see the Implementer's Guide for details.

Usage

Depending on the configuration of the PAWN compiler, you may need to explicitly include the TIME.INC definition file. To do so, insert the following line at the top of each script:

```
#include <time>
```

The angle brackets "`<...>`" make sure that you include the definition file from the system directory, in the case that a file called TIME.INC or TIME.P also exists in the current directory.

From that point on, the native functions from the file I/O support library are available.

The `settimer` function sets up the interval (or the *delay* for a one-shot timer) for the `@timer` callback function. To get a time event, the script must implement the `@timer` callback function and configure the timer with `settimer`.

An event-driven program that prints a period ("`.`") every second is:

Listing: **event-driven program to print a dot each second**

```
#include <time>

main()
    settimer 1000       /* interval is in milliseconds */

@timer()
    print "."
```

For comparison, below is a flow-driven program that does the same thing. It needs two loops: an inner loop to check for overflowing a second and an outer loop to continue printing dots after each second lapse. The program below is designed for purpose of demonstration, instead of timing quality. As it is, it is prone to timer drift. The event-driven alternative above is more accurate.

Listing: **flow-driven program to print a dot each second**

```
#include <time>

main()
    {
    for ( ;; )
        {
        new stamp = tickcount()
        while (tickcount() - stamp < 1000)
            {}
        print "."
        }
    }
```

Public functions

@timer — A timer event occurred

Syntax: `@timer()`

Returns: The return value of this function is currently ignored.

Notes: This function executes after the delay/interval set with `settimer`. Depending on the timing precision of the host, the call may occur later than the delay that was set.

If the timer was set as a "single-shot", it must be explicitly set again for a next execution for the `@timer` function. If the timer is set to be repetitive, `@timer` will continue to be called with the set interval until it is disabled with another call to `settimer`.

See also: `delay`, `settimer`

Native functions

cvttimestamp — Convert a timestamp into a date and time

Syntax:
```
cvttimestamp(seconds1970, &year=0, &month=0, &day=0,
             &hour=0, &minute=0, &second=0)
```

`year`	This will hold the year upon return.
`month`	This will hold the month (1–12) upon return.
`day`	This will hold the day of (1–31) the month upon return.
`hour`	This will hold the hour (0–23) upon return.
`minute`	This will hold the minute (0–59) upon return.
`second`	This will hold the second (0–59) upon return.

Returns: This function always returns 0.

Notes: Some file and system functions return timestamps as the number of seconds since midnight, 1 January 1970, which is the start of the Unix system epoch. This function allows to convert these time stamps into date and time fields.

See also: `gettime`, `getdate`, `settimestamp`

delay — Halts execution a number of milliseconds

Syntax:
```
delay(milliseconds)
```

`milliseconds`
The delay, in milliseconds.

Returns: This function currently always returns zero.

Notes: On some platforms, the `sleep` instruction also delays for a given number of milliseconds. The difference between the `sleep` instruction and the `delay` function is that the `delay` function does not yield events and the `sleep` instruction typically yields. When yielding events is, any pending events are handled. As a result, the `delay` function waits *without* handling any pending events and the `sleep` instruction waits and deals with events.

See also: `tickcount`

---

getdate Return the current (local) date

Syntax: `getdate(&year=0, &month=0, &day=0)`

`year`	This will hold the year upon return.
`month`	This will hold the month (1–12) upon return.
`day`	This will hold the day of (1–31) the month upon return.

Returns: The return value is the number of days since the start of the year. January 1 is day 1 of the year.

See also: `gettime`, `setdate`

---

gettime Return the current (local) time

Syntax: `gettime(&hour=0, &minute=0, &second=0)`

`hour`	This will hold the hour (0–23) upon return.
`minute`	This will hold the minute (0–59) upon return.
`second`	This will hold the second (0–59) upon return.

Returns: The return value is the number of seconds since midnight, 1 January 1970: the start of the Unix system epoch.

See also: `getdate`, `settime`

setdate Set the system date

Syntax: `setdate(year=cellmin, month=cellmin, day=cellmin)`

`year` The year to set; if this parameter is kept at its default value ("`cellmin`") it is ignored.

`month` The month to set; if this parameter is kept at its default value ("`cellmin`") it is ignored.

`day` The month to set; if this parameter is kept at its default value ("`cellmin`") it is ignored.

Returns: This function always returns 0.

The date fields are kept in a valid range. For example, when setting the month to 13, it wraps back to 1.

See also: `getdate`, `settime`, `settimestamp`

settime Set the system time

Syntax:
```
settime(hour=cellmin, minute=cellmin,
        second=cellmin)
```

`hour` The hour to set, in the range 0–23; if this parameter is kept at its default value ("`cellmin`") it is ignored.

`minute` The minute to set, in the range 0–59; if this parameter is kept at its default value ("`cellmin`") it is ignored.

`second` The second to set, in the range 0–59; if this parameter is kept at its default value ("`cellmin`") it is ignored.

Returns: This function always returns 0.

The time fields are kept in a valid range. For example, when setting the hour to 24, it wraps back to 23.

See also: `gettime`, `setdate`, `settimestamp`

settimer Configure the event timer

Syntax: `settimer(milliseconds, bool: singleshot=false)`

`milliseconds`
: The number of milliseconds to wait before calling the `@timer` callback function. Of the timer is repetitive, this is the interval. When this parameter is 0 (zero), the timer is shut off.

`singleshot`
: If `false`, the timer is a repetitive timer; if `true` the timer is shut off after invoking the `@timer` event once.

Returns: This function always returns 0.

Notes: See the chapter "Usage" for an example of this function, and the `@timer` event function.

See also: `@timer`, `tickcount`

settimestamp Sets the date and time with a single value

Syntax: `settimestamp(seconds1970)`

`seconds1970`
: The number of seconds that have elapsed since midnight, 1 January 1970. This particular date, 1 January 1970, is the "Unix system epoch".

Returns: This function always returns 0.

Notes: The function `getdate` returns the number of seconds since 1 January 1970.

See also: `getdate`, `setdate`, `settime`

tickcount Return the current tick count

Syntax: `tickcount(&granularity=0)`

`granularity` Upon return, this value contains the number of ticks that the internal system time will tick per second. This value therefore indicates the accuracy of the return value of this function.

Returns: The number of milliseconds since start-up of the system. For a 32-bit cell, this count overflows after approximately 24 days of continuous operation.

Notes: If the granularity of the system timer is "100" (a typical value for Unix systems), the return value will still be in milliseconds, but the value will change only every 10 milliseconds (100 "ticks" per second is 10 milliseconds per tick).

This function will return the time stamp regardless of whether a timer was set up with `settimer`.

See also: `settimer`

Resources

The PAWN toolkit can be obtained from **www.compuphase.com** in various formats (binaries and source code archives). The manuals for usage of the language and implementation guides are also available on the site in Adobe Acrobat format (`PDF` files).

Index

◇ Names of persons (not products) are in *italics*.
◇ Function names, constants and compiler reserved words are in `typewriter font`.