ASAM GDI - Platformadapter

API Documentation

rd electronic GmbH
Zweigstelle Dresden
Bernhardstraße 70
01187 Dresden

Tel. +49 351-6563-56-4

Internet: http://www.rd-electronic.de

Document Number:	ASAM-GDI-Platformadapter-UserMan
Issue:	2.0.0.0
Status:	Release
Last Modification:	2017-06-28
Created on:	2006-07-11
Author:	SBa

Important Notice

THE AUTHOR RESERVES THE RIGHT TO CHANGE THE INFORMATION BEING PART OF THIS DOCUMENT WITHOUT ANY NOTICE.

THE INFORMATION CONVEYED IN THIS DOCUMENT HAS BEEN CAREFULLY REVIEWED AND BELIEVED TO BE ACCURATE AND RELIABLE; HOWEVER, NO RESPONSIBILITY IS ASSUMEND FOR INACCURACIES IN THIS DOCUMENT. THE AUTHOR IS NOT RESPONSIBLE FOR ERRORS IN THIS DOCUMENT OR FOR ANY DAMAGE CAUSED AT RANDOM OR FOR CONSEQUENTIAL LOSS IN CONNECTION WITH THE APPLICATION OF THIS DOCUMENT.

Document Change History

Version	Date	Changed by	Modification
1.0.0.0	2006-07-12	BA	• Initial version
1.0.0.1	2006-09-21	BA	• GDI_PA_HANDLE_EXCEPTION interpretation
1.0.0.2	2006-11-01	BA	• See Platformadapter os-Resources Timer and LPTimer
1.0.1.0	2008-07-08	BA	• extended with GDI_PALOG_LEN
1.0.1.1	2010-04-13	BA	• extended with GDI_PA_CONF_DIR
1.0.1.2	2010-06-29	BA	• additional comments in [4.3]
1.0.2.0	2010-09-06	BA	• environment variable GDI_PALOG_NAMESTAMP added
1.0.3.0	2014-03-03	BA	• environment variables added GDI_PALOG_ZIP, GDI_PA_MEM_LEVEL, GDI_PA_MEM_FREE, GDI_PALOG_AGE
1.0.3.1	2014-04-25	BA	• Note for maximal path length • GDI_PALOG_DIR
2.0.0.0	2017-06-28	SBa	• Change to API Documentation • Extraction of configurations in separated document

Contents

1 General

1.1 Identification

This document describes parameterization and behavior of Platformadapter that are not prescribed within the standard document [1].

1.2 Purpose

It is important for the application programmer to know different states of Platformadapter resources, general rules for using, external parameterization and configuration. These parts are not standardized by the ASAM-GDI specification but documented here.

1.3 Referenced Documents

[1] ASAM GDI Specification „ASAM_GDI_4_3_2_Specification.pdf“

2 Platformadapter os – Resources

This chapter lists resources of the Platformadapter are supported via os-functions. Any functions are not described because they don't require any preconditions and haven't special after states.

2.1 General Protection of Resource Objects

All Platformadapter resource objects support multi thread access. If a resource is released by the associated function but currently used by another thread then it isn't destroyed directly. The thread that requires the destruction of a resource object sets a request, deregisters the resource by the Platformadapter and returns immediately. The resource is not available any longer. The last using thread destroys the resource object.

This implementation avoids access violations by early releasing of resources.

2.2 Timer and LPTimer

These resources and theirs functionality are supported by the API functions `os_settimer`, `os_killtimer`, `os_setLPTimer` and `os_killLPTimer`. Both have the same behaviour; another light process calls the call back function `pfnComplete`.

If the Platformadapter generates a new timer thread by a call of `os_settimer` or `os_setLPTimer` and after using it closes this timer, then the timer thread will not be destroyed but reserved for other timers. The thread is suspended for reusing until a new timer is opened.

After a call of `os_killtimer` or `os_killLPTimer` the call back function `pfnComplete` is called no more. Active callbacks are not aborted but the timer resource reusable after returning.

Timers can overlap other timers. That means that more than one timer can exists with the same call back functions and service ID's. These parameters are not validated any longer. It avoids problems by reusing of the same timer with different time outs. The older version of Platformadapter had problems by reusing of the same timer when a call back of the old timer was still active (os_settimer, os_killtimer, os_settimer).

2.3 Semaphore

The functionality of a Semaphore is supported by the functions `os_createSem`, `os_deleteSem`, `os_waitSem` and `os_releaseSem`.

Note that every call of `os_waitSem` decrements the counter of semaphore by one. Also a recall by the same thread decrements this counter. Every call of `os_releaseSem` increments the counter by one. It is not important whether the calling thread have called `os_waitSem` before.

2.4 Debug Channels

The functionality of a Debug Channel is supported by the functions `os_openDebug`, `os_writeDebug` and `os_closeDebug`.

The Platformadapter tries to generate a new log file by a call of `os_openDebug`. Therefore it interprets the environment variable GDI_PALOG_DIR. If the path is valid then a new log file is created there. If GDI_PALOG_DIR is not settled or empty then the Platformadapter returns a valid handle but logging via `os_writeDebug` is deactivated and messages ignored. Also calls of `os_writeDebug` and `os_closeDebug` with this handle will return successfully in this case.

Multi-threaded simultaneous calls of os_writeDebug on the same Debug Channel are synchronized via a Critical Section on Windows and a Mutex on Linux.

By using of empty or NULL- parameters for sink names the PA generates the standard name “PADebugChannel”. This is implemented in following versions:

- PA432 (5.2.7.6) and higher

- PA433 (5.4.2.19) and higher

- PA44 (6.1.5.2) and higher

3 Other os – Functions

This chapter lists functions of the Platformadapter don't use durable resources.

3.1 os_clock

The function returns the processor time used by the calling process. The system function `clock` is called to calculate this information. The return value depends on the operating system.

3.2 os_time

Windows systems don't support time functions have a solution <= 1 µs. That's why the Performance Counter of the system is used for calculation of current time. It is possible that a bit of performance will be lost.

3.3 os_delay

Note that the timeout solution of blocking calls depends on the operating system. On Windows the system call `Sleep` is used. On Linux the Platformadapter calls the system function `select`.

3.4 os_getLPNumber

On Windows the function `GetCurrentThreadId` is called for identification of the calling thread. On Linux function `pthread_self` of the pthread library is used.

Note that identifiers returned identify different running threads. When a thread terminates or was cancelled, its identifier can be reused for threads are created after termination of this thread.

4 Extension Access

Here any rules for using the io – functions are listed. These functions uses internal resource objects represent Extensions, Interfaces and Connections.

4.1 General Protection of Extension Objects

All objects represent any Extension resources support multi thread access. If a resource is closed by a call of `io_conclude` or `io_close` but currently used by another thread then it isn't destroyed directly. The thread that requires the destruction of a resource object sets a request, deregisters the resource by the Platformadapter and returns immediately. The resource is not available any longer. The last using thread destroys the resource object.

This implementation avoids access violations by early releasing of resources.

4.2 Synchronization of Extension Calls

The Platformadapter doesn't synchronize any calls into an Extension. This is a task of the Extension. The Platformadapter only searches for the Extension object identified by a handle and calls the Extension function (`ext_`…) assigned to the `io` – function.

4.3 io_initiate

By a call of this function the Platformadapter checks whether the required Extension is not yet loaded. If the required Extension is ready loaded then only a new handle is generated and returned.

The configuration file ***'rde_pa_conf.cfg'*** contains a routing table for extension routing. This configuration file has to be located in the directory of the Platformadapter. The required entry is identified by the section

[<Gdi-Version>::<extension-name>::<communication-type>]

and contains following routing:

LibraryPath[Linux] = "<routed target path under linux>"

LibraryPath[Windows] = "<routed target path under windows>"

SymbolicExtensionName = "<routed target name of extension library>"

CommTypeName = "<routed target Communication Type>"

If the variable ***LibraryPath*** is empty then the Platformadapter interprets the environment variable ***GDI_PAEXT_DIR*** that perhaps contains the path of extensions. If the required Extension is located on this path then it will be loaded and a new handle returned.

If the variable is not settled or empty then the ***extension-name*** is used without a path for loading. The system searches for the extension library within the working directory and the system library path.

If the variable ***SymbolicExtensionName*** is not settled or empty, then the ***extension-name*** is used directly for loading.

If the variable ***CommTypeName*** is not settled or empty, then the ***communication-type*** is used directly for initialization.