

Kontron[®]

JIDA32 Library API

Technical Manual

Rev. 1.8

Kontron[®]
Embedded Modules GmbH
Brunnwiesenstraße 16
94469 Deggendorf/ Germany

PN of Manual:	Jida32.doc
Manual Rev.:	1.8
File:	Jida32.doc

Table of Contents

TABLE OF CONTENTS	2
USER INFORMATION	6
Trademarks	6
General	6
Warranty	7
INTRODUCTION	8
REQUIREMENTS	8
INSTALLATION	8
Windows NT/2000/XP	9
Windows 98/ME	9
Windows 95	10
Windows CE	10
Linux	11
VxWorks	13
ADDITIONAL PROGRAMS	15
LCD Hot Key Support	15
Sample Programs	16
PROGRAMMING OVERVIEW	17
Initializing the DLL	17
Establishing a connection to a board	18
Generic board functions	19
VGA functions	19
Storage areas	19
I2C buses	20
Watchdog	20

CPU Performance	21
Input/Output Port	21
Board Sensor Functions	21
JIDA32 LIBRARY API PROGRAMMER'S REFERENCE	22
JidaDllGetVersion	22
JidaDllInitialize	22
JidaDllUninitialize	22
JidaDllIsAvailable	23
JidaDllInstall	23
JidaBoardCount	24
JidaBoardOpen	24
JidaBoardOpenByName	25
JidaBoardClose	25
JidaBoardGetName	26
JidaBoardGetInfo	26
JidaBoardGetBootCounter	27
JidaBoardGetRunningTimeMeter	28
JidaBoardGetOption	29
JidaBoardSetOption	29
JidaBoardGetBootErrorLog	30
JidaVgaGetContrast	32
JidaVgaSetContrast	32
JidaVgaGetContrastEnable	33
JidaVgaSetContrastEnable	33
JidaVgaGetBacklight	34
JidaVgaSetBacklight	34
JidaVgaGetBacklightEnable	35
JidaVgaSetBacklightEnable	35
JidaVgaEndDarkBoot	35
JidaStorageAreaCount	36

JidaStorageAreaType	36
JidaStorageAreaSize	37
JidaStorageAreaBlockSize	37
JidaStorageAreaRead	38
JidaStorageAreaWrite	38
JidaStorageAreaErase	39
JidaStorageAreaEraseStatus	39
JidaI2CCount	40
JidaI2CIsAvailable	40
JidaI2CType	40
JidaI2CRead	41
JidaI2CWrite	41
JidaI2CWriteReadCombined	42
JidaI2CReadRegister	42
JidaI2CWriteRegister	43
JidaIOCount	44
JidaIOIsAvailable	44
JidaIORead	44
JidaIOWrite	45
JidaIOXorAndXor	45
JidaIOGetDirection	46
JidaIOSetDirection	46
JidaIOGetDirectionCaps	47
JidaIOGetName	47
JidaWDogCount	48
JidaWDogIsAvailable	48
JidaWDogTrigger	48
JidaWDogDisable	49
JidaWDogSetConfig	49
JidaWDogSetConfigStruct	50

JidaWDogGetConfigStruct	50
JWDCONFIG Data Structure	51
JidaPerformanceGetCurrent	52
JidaPerformanceSetCurrent	52
JidaPerformanceGetPolicyCaps	53
JidaPerformanceGetPolicy	53
JidaPerformanceSetPolicy	54
JidaTemperatureCount	55
JidaTemperatureGetCurrent	55
JidaTemperatureGetInfo	56
JidaFanCount	57
JidaFanGetCurrent	57
JidaFanGetInfo	58
JidaVoltageCount	59
JidaVoltageGetCurrent	59
JidaVoltageGetInfo	60
DOCUMENT REVISION HISTORY	61

User Information

Copyright 2004 Kontron[®] Embedded Modules GmbH.

In this document Kontron[®] Embedded Modules GmbH will also be referred to by the short form "Kontron[®]".

The information in this document has been carefully checked and is believed to be accurate and reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Kontron[®] reserves the right to make changes to any portion of this manual to improve reliability, function or design. Kontron[®] does not assume any liability for any product or circuit described herein.

Trademarks

AT and IBM are trademarks of International Business Machines

XT, AT, PS/2 and Personal System/2 are trademarks of International Business Machines Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

Intel is a registered trademark of Intel Corporation.

All other products and trademarks mentioned in this manual are trademarks of their respective owners.

The reproduction, transmission or use of this document or its contents is not permitted without expressed written authority.

Offenders will be liable for damages. All rights created by patent grant or registration of a utility model or design, are reserved.

© Kontron[®] Embedded Modules 2004

General

For the circuits, descriptions and tables indicated no responsibility is assumed as far as patents or other rights of third parties are concerned.

The information in the Technical Descriptions describes the type of the boards and shall not be considered as assured characteristics.

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Warranty

Each board is tested carefully and thoroughly before being shipped. If, however, problems should occur during the operation, please check your user specific settings of all boards included in your system. This is often the source of the fault. If a board is defective, it can be sent to your supplier for repair. Please take care of the following steps:

1. The board returned should have the factory default settings since a test is only possible with these settings.
2. In order to repair your board as fast as possible we require some additional information from you. Please fill out the attached Repair Form and include it with the defective board.
3. If possible the board will be upgraded to the latest version without additional cost.
4. Upon receipt of the board please be aware that your user specific settings were changed during the test.

Within the warranty period the repair is free of charge as long as the warranty conditions are observed. Because of the high test expenditure you will be charged with the test cost if no fault is found. Repair after the warranty period will be charged.

This **Kontron[®]** product is warranted against defects in material and workmanship for the warranty period from the date of shipment. During the warranty period **Kontron[®]** will at its option either repair or replace defective products.

For warranty service or repair the product must be returned to a service facility designated by **Kontron[®]**.

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance or handling by buyer, unauthorized modification or misuse, operation outside of the product's environmental specifications or improper installation or maintenance.

Kontron[®] will not be responsible for any defects or damages to other products not supplied by **Kontron[®]** that are caused by a faulty **Kontron[®]** product.

Introduction

Most **Kontron**[®] PC boards are equipped with unique hardware features that cannot be accessed with standard API. The JIDA interface allows you to access these features in a hardware independent manner under popular 32-bit operating systems.

The library interface works under any flavor of Win32, as well as Linux and VxWorks. The library communicates with a platform dependent driver. At the present time drivers are available for Windows 9x, Windows NT, Windows 2000, Windows XP, Windows CE, Linux, and VxWorks.

The API was renamed from "JIDA Win32 API" to "JIDA32 Library API" as it is now available for a variety of operating systems. There are no changes for using the API under Windows.

The Watchdog functionality has been integrated into the JIDA API. The use of the obsolete JWDOG.DLL is strongly discouraged.

Requirements

- PC with any number of **Kontron**[®] boards that have BIOS support
- Any of these operating system:
 - Windows 9x
 - Windows NT 4.0
 - Windows 2000
 - Windows XP
 - Windows CE 2.x/3.0/4.0/4.1/4.2
 - Linux 2.2/2.4
 - Wind River Tornado 2.0 (VxWorks 5.4) with Pentium or 486 BSP

Installation

The drivers are dynamically installed upon running the sample application `JIDATST.EXE`. You can do that in your own application as well.

Under Windows NT/2000/XP you need Administrator rights to install the drivers, i.e. running `JIDATST.EXE` for the first time.

The JIDA API includes a function `JidaDllInstall` that allows you to perform the necessary steps to set up the required drivers in an operating system independent manner. Please note that it still your responsibility to copy the required files into the Windows directory before calling `JidaDllInstall`. The files are listed below for each operating system along with installation instructions if you do not want to use the JIDA install function.

Please note that the `JIDA.DLL` is binary compatible between Windows 9x and NT/2000/XP. A different version with the same name is supplied for Windows CE.

The `Jida.h` header file is the same on all operating system versions. For Linux and VxWorks you need to include the file `JWinDefs.h` before including `Jida.h` to pull in missing type declarations. You also need to include any OS specific header file before that like `windows.h` or `vxWorks.h`.

Note that for individual operating systems or boards **Kontron**[®] may occasionally release separate packages of the JIDA32 Library files or drivers that will eventually be incorporated into this complete package. So always check for individual updates first.

Windows NT/2000/XP

The following files must be copied to the Windows NT System32 directory:

```
JIDA.DLL
DRIVERS\JIDAN.SYS
```

To set up the drivers manually the following entries must be made to the system registry:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\JidaN]
"Type"=dword:00000001
"Start"=dword:00000002
"ErrorControl"=dword:00000001
"Group"="Extended Base"
```

Please note that the Restart option is not implemented under Windows NT/2000/XP. The Restart option of the watch dog API issues an NMI when the watch dog expires. If desired you can add your own NMI Handler. By default NT will produce a **blue screen**. This behavior is by design.

Windows 98/ME

The following files must be copied to the Windows System32 directory:

```
JIDA.DLL
DRIVERS\JIDAN.SYS
JWDOGV.VXD
JWDOGR.EXE
```

For Win 98/ME copy the JIDAN.SYS to the Windows\system32\drivers directory.
The driver *****MUST***** be in that directory.

If had an older version of the Jida Drivers installed then **DO NOT** load the JIDAV.VXD anymore.
The "device=jidav.vxd" line must be **removed** from system.ini.

To set up the drivers manually the following entries must be made to the system registry:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\JidaN]
"Type"=dword:00000001
"Start"=dword:00000002
"ErrorControl"=dword:00000001
"Group"="Extended Base"
```

To set up the watchdog driver manually the following line must be added to [386Enh] section of the SYSTEM.INI file located in the Windows directory:

```
device=JWDOGV.VXD
```

Note that if you do not intend to use the watchdog functionality then you do not need to load the JWDOG.VXD which sets up an NMI handler.

Note that the JWDOG.VXD is never installed automatically. You must add this line manually.

Windows 95

The following files must be copied to the Windows 9x system directory:

```
JIDA.DLL
JIDAV.VXD
JWDOGV.VXD
JWDOGR.EXE
```

To set up the drivers manually the following line must be added to [386Enh] section of the SYSTEM.INI file located in the Windows directory:

```
device=JIDAV.VXD
device=JWDOGV.VXD
```

Note that if you do not intend to use the watchdog functionality then you do not need to load the JWDOG.VXD which sets up an NMI handler.

Note that the JWDOG.VXD is never installed automatically. You must add this line manually.

Windows CE

The following files must be copied to the Windows directory:

```
JIDA.DLL
JIDAC.DLL
```

To do that add these line to the MODULES section of any *.BIB file:

```
jida.dll      $(_FLATRELEASEDIR)\jida.dll      NK  SH
jidac.dll     $(_FLATRELEASEDIR)\jidac.dll  NK  SH
```

If you want to run the demo also add:

```
jidatst.exe   $(_FLATRELEASEDIR)\jidatst.exe  NK  S
```

Place the three files into any files directory.

To set up the drivers manually add these line in any *.REG file:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\JidaC]
"Dll"="jidac.dll"
"Prefix"="JDA"
"Index"=dword:1
"Order"=dword:9
```

Build your new Windows CE operating system.

Linux

A **separate** package is available includes pre-compiled JIDA modules for the following kernels:

- 2.2.x
- 2.4.x

The Linux JIDA package contains the following files/directories:

JidaDrv	Contains partially linked versions of the JIDA kernel driver for 2.2.x and 2.4.x kernels
JidaLib	Contains a partially linked version of the JIDA interface library
JidaTst	Contains the source code for the general JIDA test application
JWDogTst	Contains the source code for the JIDA watchdog test application
Makefile	Main makefile

Prerequisites

In order to build and install the Linux JIDA driver and interface library you need to have ROOT privileges.

Please make sure, that /usr/src/linux contains or points to the sources (or at least the header files) of the kernel for which you want to build the JIDA package.

Build and Installation

Type '**make all**' to build the JIDA kernel driver matching the kernel version located in /usr/src/linux, the JIDA interface library for your installed libc version, and the corresponding test utilities.

The make utility will automatically chose the appropriate partially linked driver library for the kernel located in /usr/src/linux and build the kernel driver module jida.o.

If the build is successful, the following JIDA files can be found in the respective directories:

Directory	Files(s)	Description
JidaDrv	jida.o	The JIDA kernel driver module for the respective kernel.
JidaLib	Jida.h and JWinDefs.h or jidakiss.h	Header files to be included by the user's application accessing the JIDA interface.
	libjida.a, libjida.so	Static and shared JIDA interface libraries to be linked to the user's application.
JidaTst	jidatst	General JIDA test application.
JWDogTst	jwdogtst	JIDA watchdog test application.

Running '**make install**' will copy the JIDA modules to the appropriate directories:

jida.o	/lib/modules/'KERNEL_VERSION'/misc
libjida.so, libjida.a	/usr/lib
Jida.h, JWinDefs.h, jidakiss.h	/usr/include
jidatst, jwdogtst	/usr/bin

Now you should run the sample application `jidatst`, which will display the following message:

```
JIDA system driver is incompatible or not installed.  
Would you like to install it? (yes or no)
```

If you answer this question with 'yes' or 'y', the device node `/dev/jida` is created and the driver module `jida.o` is loaded.

Afterwards some basic JIDA test calls will be performed which display their results on the screen. If you see these results, the JIDA interface is operational.

In order to be able to access the JIDA interface from your own (non-root) application, you have to make sure, that the JIDA driver module is loaded.

There are two recommended ways to do this:

1. Add an `insmod jida` instruction to one of the Linux initialization scripts.
2. Add the following entry to your `/etc/conf.modules` file:

```
alias char-major-99 jida
```

This way `jida.o` will be loaded dynamically on calling `JidaDllInitialize`. However, this method only works if either `KERNELD` or `KMOD` are active and you are using a `jida.o` compiled for your kernel version.

Please make sure to include the files `Jida.h` and `JWinDefs.h` in your JIDA application.

As an alternate method you can include the `jidakiss.h` **instead** of the previous two header files. This header file uses only simple ANSI C data types and does not rely on any other typedefs made in other header files. It is however untested and solely provided for the convenience of users who have had limited exposure to other operating systems or who find the use of too many abstract typedefs difficult to understand. There is no support for 16 bit characters.

VxWorks

A **separate** package is available that contains the VxWorks version of JIDA.

You must install the following products first:

- Wind River Tornado 2.0 (VxWorks 5.4, UGL 1.2)
- BSP Pentium or 486

VxWork issues

- Because VxWorks uses 8 bit characters there is no support for UNICODE.
- The Restart option of the watch dog API issues an NMI when the watch dog expires. If desired you can add your own NMI Handler.
- You need to include the `JWinDefs.h` header file before including the `Jida.h`.

As an alternate method you can include the `jidakiss.h` **instead** of the previous two header files. This header file uses only simple ANSI C data types and does not rely on any other typedefs made in other header files. It is however untested and solely provided for the convenience of users who have had limited exposure to other operating systems or who find the use of too many abstract typedefs difficult to understand.

Installation

Please make sure that Tornado has been installed properly.

All paths are relative to the Tornado root that is C:\Tornado by default.

Board Support Package BSP

First of all it is recommended that you create a new BSP based on the PENTIUM or 486 BSP. The location of the new BSP will be referred to as `target\config\<BSP>` in this documentation.

Driver Files

Copy these files to your Tornado directory

```
target\lib\objPENTIUMgnuvx\JIDA.a
target\lib\objI80486gnuvx\JIDA.a
target\lib\objPENTIUMgnuvx\JIDA.o
target\lib\objI80486gnuvx\JIDA.o
target\h\jida.h
target\h\win2unix.h
```

Using the Driver

After that you are ready to create a new Tornado project based on the new BSP.

There are a number of ways to include the `jida.a` library to your project. One way of doing that is to add the name `jida.a` to the `EXTRA_MODULES` or `LIB` macro in the projects settings.

As an alternative the JIDA Interface is also provided as a single object file `jida.o`. Please note that can either use `jida.a` or `jida.o` but not both at the same time.

The library also contains two demo functions that correspond to the JIDATST and JWDOGTST in the documentation.

You can call them with `jidaTest()` and `jidaTestWD()` respectively. Please note that need to run `jidaTest()` before running `jidaTestWD()`. The functions will only be pulled in if they are referenced.

Answer the questions with y for yes, n for no, o for ok, and c for cancel. You need to press return after the letter.

The Restart option of the watch dog API issues an NMI when the watch dog expires. If desired you can add your own NMI Handler.



Additional Programs

LCD Hot Key Support

This tool enables hot keys to change the contrast and back light voltage for LCD panels under Windows 9x/NT/2000/XP.

You need these files in addition to the JIDA binaries:

JLcdKeyX.exe
JLcdKeyD.dll

Run JLcdKeyX.exe from your StartUp group or registry Run key.

The hot key assignment is as follows:

Key	Action
Ctrl+Alt+1	Decrease contrast
Ctrl+Alt+2	Increase contrast
Ctrl+Alt+3	Set contrast to 50%
Ctrl+Alt+F1	Decrease backlight
Ctrl+Alt+F2	Increase backlight
Ctrl+Alt+F3	Set backlight to 66%

IMPORTANT NOTE: The JIDA.DLL MUST be located in the Windows or Windows System directory for this to work. It is NOT sufficient that the JIDA.DLL is in the same directory as JLcdKeyX.exe. There is no error message to indicate this faulty configuration. The hot keys simply have no effect.

Sample Programs

The sample programs `JIDATST` and `JWDOGTST` are just that: sample programs. They are not intended to serve any useful purpose. To learn how they work please look at the provided source code. Please note that `JIDATST` dynamically loads the driver while `JWDOGTST` does not.

THESE PROGRAMS ARE INTENDED FOR SOFTWARE DEVELOPERS AND ARE OF NO INTEREST TO END USERS!

YOU SHOULD NEVER SHIP THESE PROGRAMS (AS THEY ARE) TO END USERS!

PLEASE NOTE THAT AFTER RUNNING JWDOGTST THE BOARD MIGHT BE *RESET*** OR PRODUCE A ***BLUE SCREEN*** WITHOUT PRIOR NOTICE. DATA MIGHT BE LOST! THIS BEHAVIOR IS BY DESIGN. SEE THE SOURCE CODE FOR DETAILS!**

Programming Overview

All API functions are exported from the `JIDA.DLL` dynamic link library. UNICODE is supported. `JIDA.DLL` is binary compatible between Windows 9x and NT/2000/XP. A different version with the same name is supplied for Windows CE.

A header file `Jida.h` and import library `JIDA.LIB` for C/C++ are provided in the `INC` and `LIB` directories. The header file is the same on all Windows versions.

The file `JIDATST.CPP` contains an example that demonstrates the JIDA functionality under Microsoft Visual C++.

The file `JWDOGTST.CPP` contains an example that demonstrates the basic watchdog functionality under Microsoft Visual C++.

Initializing the DLL

Before any other API is to be used you must initialize the DLL using `JidaDllInitialize`. Before your application terminates you must call `JidaDllUninitialize` to allow proper resource clean up.

- `JidaDllGetVersion`
- `JidaDllInitialize`
- `JidaDllUninitialize`
- `JidaDllIsAvailable`
- `JidaDllInstall`

Establishing a connection to a board

The JIDA API is based on the board concept. A board is a physical hardware component. At the moment each board must have a BIOS or DOS TSR that either provides a 16 bit real mode or 32 bit protected mode entry point that contains support functions for the underlying hardware.

Each board has a unique seven letter name that corresponds directly with the physical type of board. Examples are "P388", "P488", "LEU1" or "SPRINT5". Boards are also divided into classes. The currently defined classes are "CPU", "VGA", and "IO". Each board has one primary class but it can also have any number of secondary classes.

This allows you to talk to a class of boards that has a particular functionality without knowing the exact name of the board.

The function `JidaBoardCount` can be used to query the number of available boards either in total or for a given class. Note that since one board can belong to any number of classes the total number of boards is not necessarily the sum of all of the number of boards for each class. For example "LEU1" is primarily a CPU class board but it also has an on board VGA, so it has that class as its secondary class.

The maximum possible configuration at the moment would be a CPU board with onboard VGA, a second VGA and an IO card plugged into a slot.

So you would have a total of three physical boards. One for CPU class, two boards for VGA, and one for IO.

Most JIDA API calls take a board handle as a first parameter. You can obtain such a handle thru one of two functions: `JidaBoardOpen` which takes a class name or board index as a parameter and `JidaBoardOpenByName` which takes a unique board name.

You can keep the handle open for as long as you like. The handle must be closed with `JidaBoardClose`.

- `JidaBoardCount`
- `JidaBoardOpen`
- `JidaBoardOpenByName`
- `JidaBoardClose`

Generic board functions

A number of `JidaBoard*` functions allow you to retrieve general board class independent information about the board.

- `JidaBoardGetName`
- `JidaBoardGetInfo`
- `JidaBoardGetBootCounter`
- `JidaBoardGetRunningTimeMeter`
- `JidaBoardGetBootErrorLog`

VGA functions

`JidaVga*` functions are implemented by boards that belong to the VGA class. They primarily control LCD backlight brightness and contrast.

- `JidaVgaGetContrast`
- `JidaVgaSetContrast`
- `JidaVgaGetContrastEnable`
- `JidaVgaSetContrastEnable`
- `JidaVgaGetBacklight`
- `JidaVgaSetBacklight`
- `JidaVgaGetBacklightEnable`
- `JidaVgaSetBacklightEnable`
- `JidaVgaEndDarkBoot`

Storage areas

Each board can have a number of storage areas. A storage area is piece of physical memory that usually provides persistent storage for the user's application. All of the `JidaStorageArea*` functions take a type as a second parameter. This type is a combination of one of the predefined type constants for EEPROM, FLASH, CMOS, or RAM and a zero-based index of the area if there are more areas of a particular type.

- `JidaStorageAreaCount`
- `JidaStorageAreaType`
- `JidaStorageAreaSize`
- `JidaStorageAreaRead`
- `JidaStorageAreaWrite`

I2C buses

JidaI2C* functions provide access to the onboard I2C bus. Note that since I2C addresses may change you should not use these functions to access any **Kontron[®]** onboard devices. You should use these functions only if you have your own devices connected to the onboard bus.

- JidaI2CCount
- JidaI2CIsAvailable
- JidaI2CType
- JidaI2CReadRegister
- JidaI2CWriteRegister
- JidaI2CRead
- JidaI2CWrite
- JidaI2CWriteReadCombined

Watchdog

Some **Kontron[®]** PC boards are equipped with a watchdog component that allows the system to be reset when the running application has stopped responding.

This works by setting up a time interval either in the BIOS setup or thru API functions like JidaWDogSetConfig that are called directly by the application. After that the application must continuously call another API function named JidaWDogTrigger that triggers the watchdog. If it fails to call that function within the set up time period the PC is reset.

The **Kontron[®]** Watchdog Win32 API is obsolete and has been incorporated into the JIDA API.

- JidaWDogCount
- JidaWDogIsAvailable
- JidaWDogTrigger
- JidaWDogGetConfigStruct
- JidaWDogSetConfigStruct
- JidaWDogSetConfig
- JidaWDogDisable

Porting from JWDOG API to JIDA:

All watchdog functions have been moved to JIDA. All functions have been renamed from JWDog* to JidaWDog* and now have a JIDA handle and a type as the first two parameters. See JWDogTst.cpp and Jida.h for details.

Easy steps to convert your application to the new JIDA watchdog API:

- Replace JWDogInitialize with JidaDllInitialize
- Replace JWDogUninitialize with JidaDllUninitialize
- Declare a variable HJIDA hJida;
- Call JidaBoardOpen(JIDA_BOARD_CLASS_CPU,0,0,&hJida) to obtain a JIDA board handle
- Call JidaBoardClose(hJida) to free the JIDA handle at the end
- Replace all function prefixes from JWDog to JidaWDog
- Prefix these two arguments to all watchdog calls: hJida,0,
- Replace the header include from jwdog.h to Jida.h
- Link with jida.lib instead of jwdog.lib

CPU Performance

JidaPerformance* functions provide access to the CPU Performance settings.

- JidaPerformanceGetCurrent
- JidaPerformanceSetCurrent
- JidaPerformanceGetPolicyCaps
- JidaPerformanceGetPolicy
- JidaPerformanceSetPolicy

Input/Output Port

JidaIO* functions provide access to Digital Input/Output Ports. Many boards do not have any user accessible IO ports, so these functions return errors. XScale boards usually do have support for GPIOs.

- JidaIOCount
- JidaIOIsAvailable
- JidaIORead
- JidaIOWrite
- JidaIOXorAndXor
- JidaIOGetDirection
- JidaIOSetDirection
- JidaIOGetDirectionCaps
- JidaIOGetName

Board Sensor Functions

These functions provide access to the board temperature, fan, and voltage sensors.

- JidaTemperatureCount
- JidaTemperatureGetInfo
- JidaTemperatureGetCurrent

- JidaFanCount
- JidaFanGetInfo
- JidaFanGetCurrent

- JidaVoltageCount
- JidaVoltageGetInfo
- JidaVoltageGetCurrent

JIDA32 Library API Programmer's Reference

JidaDllGetVersion

Declaration

```
DWORD WINAPI JidaDllGetVersion(void);  
Declare Function JidaDllGetVersion Lib "JIDA" () As Long
```

Returns:

The major version number of the API is located in the upper 16 bits and the minor version number of the API in the lower 16 bits of the return value.

Description:

This function returns the version number of the JIDA API. This is the only function that can be called before `JidaDllInitialize()`. To check for version compliance you can compare the major number to the constant `JidaDllVersionMajor` as defined in your version of the header file.

JidaDllInitialize

Declaration

```
BOOL WINAPI JidaDllInitialize(void);  
Declare Function JidaDllInitialize Lib "JIDA" () As Integer
```

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This initializes the JIDA API for use by the application. A failure indicates that the driver is not properly installed or outdated. Calls to `JidaDllInitialize` and `JidaDllUninitialize` can be nested but must be balanced.

JidaDllUninitialize

Declaration

```
BOOL WINAPI JidaDllUninitialize(void);  
Declare Function JidaDllUninitialize Lib "JIDA" () As Integer
```

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function must be called before the application terminates after it has successfully called `JidaDllInitialize`. Calls to `JidaDllInitialize` and `JidaDllUninitialize` can be nested but must be balanced.

JidaDllsAvailable

Declaration

```
BOOL WINAPI JidaDllsAvailable(void);  
Declare Function JidaDllsAvailable Lib "JIDA" () As Integer
```

Returns:

TRUE (1) if a previous call to `JidaDllInitialize` was successful and the JIDA functionality is available. FALSE (0) on failure.

Description:

This function allows you to determine if the JIDA functionality is available.

JidaDllInstall

Declaration

```
BOOL WINAPI JidaDllInstall(BOOL install);
```

Parameters:

`install`
TRUE (1) on for install. FALSE (0) for uninstall.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function installs or uninstalls the underlying JIDA driver. You can call this function from your setup program or if `JidaDllInitialize` fails. Note that the system may need to be rebooted for the changes to become effective. Under NT you need administrative rights to use this function successfully. After this function succeeds you can call `JidaDllInitialize` a second time. If this function fails again then a reboot is required to load the drivers (under Windows 9x). If it succeeds then the drivers have been loaded dynamically and JIDA is ready to be used (under Windows NT and Windows CE). The driver will also load every time you start Windows in the future. Please note that under Windows CE the drivers will only be loaded again if the registry is retained on reboots.

Note that under Windows 9x the `JWDOG.VXD` which is required by the `JidaWDog*` functions is never installed automatically by this function. You must add this driver manually. See the Windows 9x installation section.

JidaBoardCount

Declaration

```
DWORD WINAPI JidaBoardCount(LPCTSTR pszClass, DWORD dwFlags);
```

Parameters:

pszClass

The class name of the board. So far the following classes have been defined:

JIDA_BOARD_CLASS_CPU

Basic CPU boards with BIOS

JIDA_BOARD_CLASS_VGA

VGA (LCD) boards with video BIOS

JIDA_BOARD_CLASS_IO

IO cards

This value can be NULL in which case the total number of boards will be returned

dwFlags

Can be any combination of the following flags:

JIDA_BOARD_OPEN_FLAGS_PRIMARYONLY

Count only boards that do have the given class name as a primary class.

Otherwise any boards that fit the given class in any way will be returned.

Returns:

Number of available boards.

Description:

JIDA_BOARD_OPEN_FLAGS_PRIMARYONLY

JidaBoardOpen

Declaration

```
BOOL WINAPI JidaBoardOpen(LPCTSTR pszClass, DWORD dwNum, DWORD dwFlags,
PHJIDA phJida);
```

Parameters:

pszClass

See JidaBoardCount.

dwNum

Zero based number of the board within the given class.

dwFlags

See JidaBoardCount.

phJida

Pointer to a location that will receive the handle to the board.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Opens a board that fits the given class.

JidaBoardOpenByName

Declaration

```
BOOL WINAPI JidaBoardOpenByName(LPCTSTR pszName, PHJIDA phJida);
```

Parameters:

pszName

Name of the board. Currently defined names are:

"D101", "D201", "D401", "D501", "P386", "P388", "P488", "P586", "P588", "P489",
"PGX1", "ROI1", "PISB", "LEU1", "LEV1", "LEU2", "LEU3", "LEU6", "LEUE", "BQC3",
"BQP3", "BQG1", "MOD1", "MOD2", "MOD5", "MOD6", "MOD7", "MOD8", "MOD9",
"XBD1", "XBD2", "XBD3", "TAHOE", "A586", "SIEB", "SIM1", "MODI", "WOB1",
"MULTI4", "SPRINT5", "SPRINT6", "APG1", "MIO2".

phJida

Pointer to a location the will receive the handle to the board.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

JidaBoardClose

Declaration

```
BOOL WINAPI JidaBoardClose(HJIDA hJida);
```

Parameters:

hJida

Board handle.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Closes the connection to a board.

JidaBoardGetName

Declaration

```
BOOL WINAPI JidaBoardGetName(HJIDA hJida, LPTSTR pszName, DWORD dwSize);
```

Parameters:

hJida
Board handle.

pszName
Location that will receive the name of the board.

dwSize
Size of the buffer location. Should be at least JIDA_BOARD_MAX_SIZE_ID_STRING characters.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the unique name of the board.

JidaBoardGetInfo

Declaration

```
BOOL WINAPI JidaBoardGetInfo(HJIDA hJida, PJIDABOARDINFO pBoardInfo);
```

Parameters:

hJida
Board handle.

pBoardInfo
Location of a JIDABOARDINFO structure that will receive the static information of the board. The structure contains the following information:

DWORD dwSize
Must be initialized by the caller with sizeof(JIDABOARDINFO).

DWORD dwFlags
Reserved. Always 0.

TCHAR szPrimaryClass[JIDA_BOARD_MAX_SIZE_ID_STRING]
Primary class name. See JidaBoardCount for possible values.

TCHAR szBoard[JIDA_BOARD_MAX_SIZE_ID_STRING]
Name of the board. See JidaBoardOpenByName for possible values.

TCHAR szBoardSub[JIDA_BOARD_MAX_SIZE_ID_STRING]
Subname of the board or empty. Usually the same as the board name and of no particular use.

TCHAR szManufacturer[JIDA_BOARD_MAX_SIZE_ID_STRING]
Manufacturer name. Usually "JUMP".

SYSTEMTIME stManufacturingDate
Manufacturing date.

SYSTEMTIME stLastRepairDate
Date that the system was last repaired or refurbished. Valid only if later then the manufacturing date.

TCHAR szSerialNumber[JIDA_BOARD_MAX_SIZE_SERIAL_STRING]
10 character unique serial number of the board

WORD wHardwareRevision
Hardware revision number. Major number in upper byte. Minor number in lower byte.

WORD `wFirmwareRevision`
Firmware revision number. Major number in upper byte. Minor number in lower byte.

WORD `wJidaRevision`
BIOS Jida interface revision number. Major number in upper byte. Minor number in lower byte. Only valid if driver has the option of querying the BIOS.

WORD `wFeatureNumber`
Feature Number of the BIOS or the driver. Usually 0 or 1.

TCHAR `szClasses[JIDA_BOARD_MAX_SIZE_CLASSES_STRING]`
Comma separated list of all class names that are applicable for the board. See `JidaBoardCount` for possible values.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves a structure that contains information about the board that does not change.

JidaBoardGetBootCounter

Declaration

```
BOOL WINAPI JidaBoardGetBootCounter(HJIDA hJida, LPDWORD pdwCount);
```

Parameters:

`hJida`
Board handle.

`pdwCount`
Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieve the watchdog boot counter.

JidaBoardGetRunningTimeMeter

Declaration

```
BOOL WINAPI JidaBoardGetRunningTimeMeter(HJIDA hJida, LPDWORD pdwCount);
```

Parameters:

hJida

Board handle.

pdwCount

Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

The time period in hours that the system has been running.

JidaBoardGetOption

Declaration

```
BOOL WINAPI JidaBoardGetOption(HJIDA hJida, DWORD dwOption, LPDWORD  
pdwSetting);
```

Parameters:

hJida
Board handle.
dwOption
pdwSetting
Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

No functionality in the current version of JIDA.

JidaBoardSetOption

Declaration

```
BOOL WINAPI JidaBoardSetOption(HJIDA hJida, DWORD dwOption, DWORD  
dwSetting);
```

Parameters:

hJida
Board handle.
dwOption
dwSetting
New value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

No functionality in the current version of JIDA.

JidaBoardGetBootErrorLog

Declaration

```
BOOL WINAPI JidaBoardGetBootErrorLog(HJIDA hJida, DWORD dwType, PDWORD
pdwLogType, LPBYTE pBytes, PDWORD pdwLen);
```

Parameters:

hJida
Board handle.

dwType
Reserved. Must be 0.

pdwLogType
Pointer to location that will receive the type of the boot error log

0
PhoenixBIOS release 6.0/6.1

1
AMIBIOS core 8.

pBytes
Pointer to location that will receive the error log structure. If this pointer is NULL then no data is returned but the `pdwLogType` and entries `pdwLen` will still be updated. This can be used to determine the size of the buffer needed for a subsequent call to this function.

pdwLen
Pointer to location that will receive the length of the boot error log.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function retrieves the boot or POST error log. The POST error log is a list of serialized entries. An entry consists of a length byte followed by POST error information. The length byte specifies the size of the entry not including the length byte. The error information is system core BIOS specific. The type code returned by the function can be used to identify the structure of the POST error information. The error log is terminated by a null entry, i.e. a length byte of 0.

PhoenixBIOS release 6.0/6.1 POST error log entry structure

Offset	Size	Name	Description
+00h	BYTE	length	length of this POST error log entry not including this field, i.e. 6
+01h	BYTE	errCode	error code (see table below)
+02h	BYTE	errSubCodeValid	0 if the errSubCode field is invalid / otherwise this field contains the number of valid hexadecimal digits (i.e. nibbles) in the errSubCode field
+03h	DWORD	errSubCode	optional numerical error data

Phoenix BIOS release 6.0/6.1 POST error codes

Name	Value	Description
ERR_DISK_FAILED	000h	harddisk error / errSubCode contains the drive number
	001h-00Fh	reserved for other disk errors
ERR_KBD_STUCK	010h	stuck key detected / errSubCode may contain a scan code
ERR_KBD_FAILED	011h	keyboard test failed
ERR_KBD_KCFAIL	012h	keyboard controller test failed
ERR_KBD_LOCKED	013h	keyboard is locked
	014h-01Fh	reserved for other keyboard errors
ERR_VIDEO_SWITCH	020h	CGA/MDA video configuration error
ERR_LOCAL_MEMORY	021h	UMA video memory initialization failure
	022h-02Fh	reserved for other video errors
ERR_SYS_MEM_FAIL	030h	system memory error
ERR_SHAD_MEM_FAIL	031h	shadow memory error
ERR_EXT_MEM_FAIL	032h	extended memory error
ERR_MEM_TYPE_MIX	033h	invalid combination of memory module types
ERR_MEM_ECC_SINGLE	034h	single bit ECC memory error
ERR_MEM_ECC_MULTIPLE	035h	multiple bit ECC memory error
ERR_MEM_DECREASED	036h	size of available memory decreased
ERR_DMI_MEM_FAIL	037h	not enough memory for DMI info structure
	038h-03Fh	reserved for other memory errors
	040h-04Fh	reserved for other errors
ERR_CMOS_BATTERY	050h	CMOS battery down
ERR_CMOS_CHECKSUM	051h	CMOS checksum invalid
ERR_PW_CHECKSUM	052h	CMOS checksum invalid
	053h-05Fh	reserved for other CMOS errors
ERR_TIMER_FAILED	060h	system timer error
	061h-06Fh	reserved for other timer errors
ERR_RTC_FAILED	070h	realtime clock error
ERR_RTC_INV_DATE_TIME	071h	invalid realtime clock date/time
	072h-07Fh	reserved for other realtime clock errors
ERR_CONFIG_FAILED	080h	system configuration error
ERR_CONFIG_MEMORY	081h	memory configuration error
	082h-08Fh	reserved for other configuration errors
ERR_NVRAM	090h	NVRAM error
	091h-09Fh	reserved for other NVRAM errors
ERR_COP	0A0h	coprocessor error
	0A1h-0AFh	reserved for other coprocessor errors
ERR_FLOPPYA_FAILED	0B0h	diskette drive A error
ERR_FLOPPYB_FAILED	0B1h	diskette drive B error
ERR_FLOPPYA_INCORRECT	0B2h	incorrect diskette drive type A
ERR_FLOPPYB_INCORRECT	0B3h	incorrect diskette drive type B
	0B4h-0BFh	reserved for other diskette drive errors
	0C0h-0CFh	reserved for other errors
ERR_CACHE_FAILED	0D0h	cache error
ERR_L2_CACHE_RANGE	0D1h	main memory size exceeds cache range
	0D2h-0DFh	reserved for other cache errors
ERR_IO_ADDRESS	0E0h	IO address error
ERR_IO_COM	0E1h	COM port error
ERR_IO_LPT	0E2h	LPT port error
ERR_IO_CONFLICT	0E3h	I/O resource conflict
	0E4h-0EFh	reserved for other I/O errors
	0F0h-0FFh	reserved for other errors

JidaVgaGetContrast

Declaration

```
BOOL WINAPI JidaVgaGetContrast(HJIDA hJida, LPDWORD pdwSetting);
```

Parameters:

`hJida`
Board handle.

`pdwSetting`
Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the contrast setting for the LCD of any VGA class board. The value ranges from 0 to JIDA_VGA_CONTRAST_MAX (255).

JidaVgaSetContrast

Declaration

```
BOOL WINAPI JidaVgaSetContrast(HJIDA hJida, DWORD dwSetting);
```

Parameters:

`hJida`
Board handle.

`dwSetting`
New value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Sets the contrast setting for the LCD of any VGA class board. The value ranges from 0 to JIDA_VGA_CONTRAST_MAX (255).

JidaVgaGetContrastEnable

Declaration

```
BOOL WINAPI JidaVgaGetContrastEnable(HJIDA hJida, LPDWORD pdwSetting);
```

Parameters:

`hJida`
Board handle.

`pdwSetting`
Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the contrast voltage setting for the LCD of any VGA class board. TRUE (1) indicates on, FALSE (0) indicates off.

JidaVgaSetContrastEnable

Declaration

```
BOOL WINAPI JidaVgaSetContrastEnable(HJIDA hJida, DWORD dwSetting);
```

Parameters:

`hJida`
Board handle.

`dwSetting`
New value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Sets the contrast voltage for the LCD of any VGA class board. TRUE (1) indicates on, FALSE (0) indicates off.

JidaVgaGetBacklight

Declaration

```
BOOL WINAPI JidaVgaGetBacklight(HJIDA hJida, LPDWORD pdwSetting);
```

Parameters:

`hJida`
Board handle.

`pdwSetting`
Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the backlight brightness setting for the LCD of any VGA class board. The value ranges from 0 to JIDA_VGA_BACKLIGHT_MAX (255).

JidaVgaSetBacklight

Declaration

```
BOOL WINAPI JidaVgaSetBacklight(HJIDA hJida, DWORD dwSetting);
```

Parameters:

`hJida`
Board handle.

`dwSetting`
New value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Sets the backlight brightness setting for the LCD of any VGA class board. The value ranges from 0 to JIDA_VGA_BACKLIGHT_MAX (255).

JidaVgaGetBacklightEnable

Declaration

```
BOOL WINAPI JidaVgaGetBacklightEnable(HJIDA hJida, LPDWORD pdwSetting);
```

Parameters:

`hJida`
Board handle.

`pdwSetting`
Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the backlight voltage setting for the LCD of any VGA class board. TRUE (1) indicates on, FALSE (0) indicates off.

JidaVgaSetBacklightEnable

Declaration

```
BOOL WINAPI JidaVgaSetBacklightEnable(HJIDA hJida, DWORD dwSetting);
```

Parameters:

`hJida`
Board handle.

`dwSetting`
New value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Sets the backlight voltage for the LCD of any VGA class board. TRUE (1) indicates on, FALSE (0) indicates off.

JidaVgaEndDarkBoot

Declaration

```
BOOL WINAPI JidaVgaEndDarkBoot(DWORD dwReserved);
```

Parameters:

`dwReserved`
Must be 0.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This ends the dark boot if that option has been enabled in the BIOS. The screen will be no longer be black and the current Windows screen will be visible.

JidaStorageAreaCount

Declaration

```
DWORD WINAPI JidaStorageAreaCount(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida

Board handle.

dwType

Possible values are:

JIDA_STORAGE_AREA_UNKNOWN

JIDA_STORAGE_AREA_EEPROM

JIDA_STORAGE_AREA_FLASH

JIDA_STORAGE_AREA_CMOS

JIDA_STORAGE_AREA_RAM

The only type currently supported is JIDA_STORAGE_AREA_EEPROM and 0.

Use 0 to get the total number of storage areas.

Returns:

Number of available storage areas.

Description:

Retrieves the number of persistent user storage areas of the specified type that are on the given board.

JidaStorageAreaType

Declaration

```
DWORD WINAPI JidaStorageAreaType(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida

Board handle.

dwType

Zero based storage area index.

Returns:

Type of storage area.

Description:

Retrieves the type of storage area.

JidaStorageAreaSize

Declaration

```
DWORD WINAPI JidaStorageAreaSize(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida

Board handle.

dwType

Possible values are a zero based index ORed with one of the following flags:

JIDA_STORAGE_AREA_UNKNOWN

JIDA_STORAGE_AREA_EEPROM

JIDA_STORAGE_AREA_FLASH

JIDA_STORAGE_AREA_CMOS

JIDA_STORAGE_AREA_RAM

The only type currently supported is JIDA_STORAGE_AREA_EEPROM and a index of 0.

Returns:

Size of storage area in bytes.

Description:

Retrives the size of the given storage area.

JidaStorageAreaBlockSize

Declaration

```
DWORD WINAPI JidaStorageAreaBlockSize(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida

Board handle.

dwType

See JidaStorageAreaSize for possible values.

Returns:

Block size of storage area or 0 in not split into blocks.

Description:

No functionality in the current version of JIDA.

JidaStorageAreaRead

Declaration

```
BOOL WINAPI JidaStorageAreaRead(HJIDA hJida, DWORD dwType, DWORD dwOffset, LPBYTE pBytes, DWORD dwLen);
```

Parameters:

hJida
Board handle.

dwType
See JidaStorageAreaSize for possible values.

dwOffset
Zero based offset into the storage area.

pBytes
Pointer to location that will receive the bytes.

dwLen
Number of bytes to read.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Read bytes from a storage area.

JidaStorageAreaWrite

Declaration

```
BOOL WINAPI JidaStorageAreaWrite(HJIDA hJida, DWORD dwType, DWORD dwOffset, LPBYTE pBytes, DWORD dwLen);
```

Parameters:

hJida
Board handle.

dwType
See JidaStorageAreaSize for possible values.

dwOffset
Zero based offset into the storage area.

pBytes
Pointer to location that contains the bytes.

dwLen
Number of bytes to write.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Write bytes to the storage area.

JidaStorageAreaErase

Declaration

```
BOOL WINAPI JidaStorageAreaErase(HJIDA hJida, DWORD dwType, DWORD dwOffset, DWORD dwLen);
```

Parameters:

hJida
Board handle.

dwType
See JidaStorageAreaSize for possible values.

dwOffset
Zero based offset into the storage area.

dwLen
Number of bytes to erase.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

No functionality in the current version of JIDA.

JidaStorageAreaEraseStatus

Declaration

```
BOOL WINAPI JidaStorageAreaEraseStatus(HJIDA hJida, DWORD dwType, DWORD dwOffset, DWORD dwLen, LPDWORD lpStatus);
```

Parameters:

hJida
Board handle.

dwType
See JidaStorageAreaSize for possible values.

dwOffset
Zero based offset into the storage area.

dwLen

lpStatus

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

No functionality in the current version of JIDA.

JidaI2CCount

Declaration

```
DWORD WINAPI JidaI2CCount(HJIDA hJida);
```

Parameters:

hJida
Board handle.

Returns:

Number of available I2C buses.

Description:

Retrieves the number of I2C buses on the board. (In the current implementation the last bus is always the JILI bus if present. This may however change in a future version.)

JidaI2CIsAvailable

Declaration

```
BOOL WINAPI JidaI2CIsAvailable(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida
Board handle.
dwType
Zero-based number of the I2C bus.

Returns:

TRUE (1) if the give type of I2C bus is present. FALSE (0) otherwise.

Description:

Queries whether the I2C bus of the given type is available.

JidaI2CType

Declaration

```
DWORD WINAPI JidaI2CType(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida
Board handle.
dwType
Zero-based number of the I2C bus.

Returns:

JIDA_I2C_TYPE_UNKNOWN	unknown or special purposes
JIDA_I2C_TYPE_PRIMARY	primary I2C bus
JIDA_I2C_TYPE_SMB	system management bus
JIDA_I2C_TYPE_JILI	JILI interface

Description:

Queries the bus type of the given I2C bus.

JidaI2CRead

Declaration

```
BOOL WINAPI JidaI2CRead(HJIDA hJida, DWORD dwType, BYTE bAddr, LPBYTE pBytes, DWORD dwLen);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of the I2C bus.

bAddr
Address of the device on the I2C bus, the full 8 bits as it is written to the bus.
Bit 0 should be always 1 to read from regular I2C devices.

pBytes
Pointer to location that will receive the bytes.

dwLen
Number of bytes to read.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads bytes from a device on the I2C bus.

JidaI2CWrite

Declaration

```
BOOL WINAPI JidaI2CWrite(HJIDA hJida, DWORD dwType, BYTE bAddr, LPBYTE pBytes, DWORD dwLen);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of the I2C bus.

bAddr
Address of the device on the I2C bus, the full 8 bits as it is written to the bus.
Bit 0 should be always 0 for regular I2C devices.

pBytes
Pointer to location that contains the bytes.

dwLen
Number of bytes to write.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Writes bytes to a device on the I2C bus.

WARNING: Improperly using this function with certain buses and devices may cause PERMANENT DAMAGE to your system and may prevent your board from booting. The most likely scenario is to accidentally overwrite the configuration data in the EEPROM that is attached to the SMBus and located on the RAM module. This may make the RAM module permanently inaccessible to the system and will therefore stop the boot process.

JidaI2CWriteReadCombined

Declaration

```
BOOL WINAPI JidaI2CWriteReadCombined(HJIDA hJida, DWORD dwType, BYTE
bAddr, LPBYTE pBytesWrite, DWORD dwLenWrite, LPBYTE pBytesRead, DWORD
dwLenRead);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of the I2C bus.

bAddr
Address of the device on the I2C bus, the full 8 bits as it is written to the bus.
Bit 0 should be always 0 for regular I2C devices. During the read cycle this functions sets Bit 0 automatically.

pBytesWrite
Pointer to location that contains the bytes.

dwLenWrite
Number of bytes to write.

pBytesRead
Pointer to location that will receive the bytes.

dwLenRead
Number of bytes to read.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Writes bytes to a device on the I2C bus, then reads the specified number of bytes in combined mode. The difference between using this function instead of the separate write and read functions is that at the end of the write this function does not include a STOP condition. The second START condition for the read is present.

WARNING: Improperly using this function with certain buses and devices may cause PERMANENT DAMAGE to your system and may prevent your board from booting. The most likely scenario is to accidentally overwrite the configuration data in the EEPROM that is attached to the SMBus and located on the RAM module. This may make the RAM module permanently inaccessible to the system and will therefore stop the boot process.

JidaI2CReadRegister

Declaration

```
BOOL WINAPI JidaI2CReadRegister(HJIDA hJida, DWORD dwType, BYTE
bAddr, WORD wReg, LPBYTE pDataByte);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of the I2C bus.

bAddr

Address of the device on the I2C bus, the full 8 bits as it is written to the bus.
Bit 0 should be always 0 for regular I2C devices. During the read cycle this functions sets Bit 0 automatically.

wReg

Register index of the device.

pDataByte

Pointer to location that will receive the byte.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads a register from a device on the I2C bus.

JidaI2CWriteRegister

Declaration

```
BOOL WINAPI JidaI2CWriteRegister(HJIDA hJida, DWORD dwType, BYTE
bAddr, WORD wReg, BYTE bData);
```

Parameters:

hJida

Board handle.

dwType

Zero-based number of the I2C bus.

bAddr

Address of the device on the I2C bus, the full 8 bits as it is written to the bus.
Bit 0 should be always 0 for regular I2C devices.

wReg

Register index of the device.

bData

Data byte.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Writes a register of a device on the I2C bus.

WARNING: Improperly using this function with certain buses and devices may cause PERMANENT DAMAGE to your system and may prevent your board from booting. The most likely scenario is to accidentally overwrite the configuration data in the EEPROM that is attached to the SMBus and located on the RAM module. This may make the RAM module permanently inaccessible to the system and will therefore stop the boot process.

JidaIOCount

Declaration

```
DWORD WINAPI JidaIOCount(HJIDA hJida);
```

Parameters:

hJida
Board handle.

Returns:

Number of available IO Ports.

Description:

On many boards this returns 0.

JidaIOIsAvailable

Declaration

```
BOOL WINAPI JidaIOIsAvailable(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida
Board handle.
dwType
Zero-based number of IO Ports.

Returns:

TRUE (1) if the give type of IO is present. FALSE (0) otherwise.

Description:

On many boards this returns FALSE.

JidaIORead

Declaration

```
BOOL WINAPI JidaIORead(HJIDA hJida, DWORD dwType, LPDWORD pdwData);
```

Parameters:

hJida
Board handle.
dwType
Zero-based number of IO Ports.
pdwData
Pointer to read value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads the current state of the IO Port. This includes the input and output values.

JidaIOWrite

Declaration

```
BOOL WINAPI JidaIOWrite(HJIDA hJida, DWORD dwType, DWORD dwData);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of IO Ports.

dwData
Value to write to the port.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Writes to the output pins of the IO Port.

JidaIOXorAndXor

Declaration

```
BOOL WINAPI JidaIOXorAndXor(HJIDA hJida, DWORD dwType, DWORD dwXorMask1, DWORD dwAndMask, DWORD dwXorMask2);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of IO Ports.

dwXorMask1

dwAndMask

dwXorMask2

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function reads the performs the function:

$$\text{newPortValue} = (((\text{currentPortValue} \text{ xor } \text{dwXorMask1}) \text{ and } \text{dwAndMask}) \text{ xor } \text{dwXorMask2})$$

If both xor-masks have the same value then this function inserts the bits of the xor-mask into the ports bit locations where the and-mask is 1. The other bits remain unchanged.

JidaIOGetDirection

Declaration

```
BOOL WINAPI JidaIOGetDirection(HJIDA hJida, DWORD dwType, LPDWORD pdwData);
```

Parameters:

hJida

Board handle.

dwType

Zero-based number of IO Ports.

pdwData

Pointer to the location that will receive the current direction of the port pins. A 0 bit indicates an OUTPUT, a 1 bit indicates an INPUT pin in the corresponding bit position.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads the current direction of the IO Port pins.

JidaIOSetDirection

Declaration

```
BOOL WINAPI JidaIOSetDirection(HJIDA hJida, DWORD dwType, DWORD dwData);
```

Parameters:

hJida

Board handle.

dwType

Zero-based number of IO Ports.

dwData

Direction of the port pins. A 0 bit indicates an OUTPUT, a 1 bit indicates an INPUT pin in the corresponding bit position.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Changes the direction of the pins of the IO Port. Fixed inputs and fixed outputs cannot be changed.

JidaIOGetDirectionCaps

Declaration

```
BOOL WINAPI JidaIOGetDirectionCaps(HJIDA hJida, DWORD dwType, LPDWORD  
pdwInputs, LPDWORD pdwOutputs);
```

Parameters:

`hJida`
Board handle.

`dwType`
Zero-based number of IO Ports.

`pdwInputs`
Pointer to the location that will receive the pins that are inputs. A 1 indicates a pin in the corresponding bit position is capable of being an input.

`pdwOutputs`
Pointer to the location that will receive the pins that are outputs. A 1 indicates a pin in the corresponding bit position is capable of being an output.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads the direction capabilities of the IO Port. For pins that are both input and output the direction can be changed with the `JidaIOSetDirection` function.

JidaIOGetName

Declaration

```
BOOL WINAPI JidaIOGetName(HJIDA hJida, DWORD dwType, LPSTR pszName, DWORD  
dwSize);
```

Parameters:

`hJida`
Board handle.

`dwType`
Zero-based number of IO Ports.

`pszName`
Buffer that will be used to return the name of the IO Port.

`dwSize`
Size of buffer

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function is not implemented yet and will always return FALSE.

JidaWDogCount

Declaration

```
DWORD WINAPI JidaWDogCount(HJIDA hJida);
```

Parameters:

hJida
Board handle.

Returns:

Number of available watchdogs on this board.

Description:

Always returns 0 or 1 in the current version of JIDA.

JidaWDogsAvailable

Declaration

```
BOOL WINAPI JidaWDogIsAvailable(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida
Board handle.
dwType
Must be 0.

Returns:

TRUE (1) if the give type of I2C bus is present. FALSE (0) otherwise.

Description:

Queries whether the watchdog of the given type is available. Currently only type 0 is implemented.

JidaWDogTrigger

Declaration

```
BOOL WINAPI JidaWDogTrigger(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida
Board handle.
dwType
Must be 0.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function indicates that the application is still working properly and must be called on a continues basis by the application to ensure that the system will not be restarted. This applies only after that watchdog has been activated.

JidaWDogDisable

Declaration

```
BOOL WINAPI JidaWDogDisable(HJIDA hJida, DWORD dwType);
```

Parameters:

hJida
Board handle.

dwType
Must be 0.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This disables the watchdog. The application is not longer required to call `JidaWDogTrigger`.

This function is not implemented in any of the current watchdog versions. Once a watchdog is turned on it can never be turned off again until the next hard reset. This behavior may change in future versions.

JidaWDogSetConfig

Declaration

```
BOOL WINAPI JidaWDogSetConfig(HJIDA hJida, DWORD dwType, DWORD timeout,
                               DWORD delay, DWORD mode);
```

Parameters:

hJida
Board handle.

dwType
Must be 0.

timeout
This is the watchdog timeout in milliseconds. The application must continuously call `JidaWDogTrigger` within that interval to prevent a reboot. Note that the min/max values and the resolution depends on the underlying hardware. In many cases the resolution is 200 ms.

delay
This is an initial delay in milliseconds that will be added to the first timeout period. This allows the application to have a longer initialization phase without calling `JidaWDogTrigger` and still be protected by the watchdog.

mode
This value can either be:
`JWDMoDeRebootPC (0)`
 This will cause a hard reset without shutting down Windows when the watchdog engages.
`JWDMoDeRestartWindows (1)`
 This will shut down Windows in a proper manner when the watchdog engages.
This behavior is only implemented under Windows 9x. Other OS call the NMI handler which should be implemented by the Embedded System Designer.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function activates the watchdog with the given parameters. After that the application must continuously call `JidaWDogTrigger` within the specified interval. The timeout value can be changed at any time. The transition from `JWDMoDeRebootPC` to `JWDMoDeRestartWindows` can only be made once. After that you cannot revert to `JWDMoDeRebootPC`. This behavior may change in future versions.

JidaWDogSetConfigStruct

Declaration

```
BOOL WINAPI JidaWDogSetConfigStruct(HJIDA hJida, DWORD dwType,
PJWDCONFIG pConfig);
```

Parameters:

`hJida`
Board handle.

`dwType`
Must be 0.

`pConfig`
Pointer to a `PJWDCONFIG` structure.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function activates the watchdog with the given parameters. After that the application must continuously call `JidaWDogTrigger` within the specified interval.

JidaWDogGetConfigStruct

Declaration

```
BOOL WINAPI JidaWDogGetConfigStruct(HJIDA hJida, DWORD dwType,
PJWDCONFIG pConfig);
```

Parameters:

`hJida`
Board handle.

`dwType`
Must be 0.

`pConfig`
Pointer to a `PJWDCONFIG` structure.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

This function retrieves the watchdog parameters that were set by the application. The function fails if the values have never been set. In the current implementation it will never return the values set in the BIOS setup pages.

JWDCONFIG Data Structure

Declaration

```
typedef struct {  
    DWORD dwSize;  
    DWORD dwTimeout;  
    DWORD dwDelay;  
    DWORD dwMode;  
} JWDCONFIG, * PJWDCONFIG;
```

Parameters:

dwSize

This must be initialized to `sizeof(JWDCONFIG)` before calling any of that functions that deal with this structure.

dwTimeout

This is the watchdog timeout in milliseconds. The application must continuously call `JidaWDogTrigger` within that interval to prevent a reboot.

dwDelay

This is an initial delay in milliseconds that will be added to the first timeout period. This allows the application to have a longer initialization phase without calling `JidaWDogTrigger` and still be protected by the watchdog.

dwMode

This value can either be:

`JWDModeRebootPC (0)`

This will cause a hard reset without shutting down Windows when the watchdog engages.

`JWDModeRestartWindows (1)`

This will shut down Windows in a proper manner when the watchdog engages.

Description:

This structure is used with `JidaWDogSetConfigStruct` and `JidaWDogGetConfigStruct`.

JidaPerformanceGetCurrent

Declaration

```
BOOL WINAPI JidaPerformanceGetCurrent(HJIDA hJida, DWORD dwType, LPDWORD  
pdwSetting);
```

Parameters:

hJida
Board handle.

dwType
Must be 0.

pdwSetting
Pointer to location that will receive the value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the current CPU performance percentage. The value ranges from 0 to 100.

JidaPerformanceSetCurrent

Declaration

```
BOOL WINAPI JidaPerformanceSetCurrent(HJIDA hJida, DWORD dwType, DWORD  
dwSetting);
```

Parameters:

hJida
Board handle.

dwType
Must be 0.

dwSetting
New value.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Sets the current CPU performance percentage. The value ranges from 0 to 100.

JidaPerformanceGetPolicyCaps

Declaration

```
BOOL WINAPI JidaPerformanceGetPolicyCaps(HJIDA hJida, DWORD dwType, LPDWORD  
pdwSetting);
```

Parameters:

`hJida`
Board handle.

`dwType`
Must be 0.

`pdwSetting`
Pointer to location that will receive the value.
`JIDA_CPU_PERF_THROTTLING`
system supports/utilizes CPU throttling
`JIDA_CPU_PERF_FREQUENCY`
system supports/utilizes CPU frequency switching

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the current CPU performance policy capabilities. The current values can be set and retrieved with the `JidaPerformanceSetPolicy` and `JidaPerformanceGetPolicy` functions.

JidaPerformanceGetPolicy

Declaration

```
BOOL WINAPI JidaPerformanceGetPolicy(HJIDA hJida, DWORD dwType, LPDWORD  
pdwSetting);
```

Parameters:

`hJida`
Board handle.

`dwType`
Must be 0.

`pdwSetting`
Pointer to location that will receive the value.
`JIDA_CPU_PERF_THROTTLING`
system supports/utilizes CPU throttling
`JIDA_CPU_PERF_FREQUENCY`
system supports/utilizes CPU frequency switching

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Retrieves the current CPU performance policy.

JidaPerformanceSetPolicy

Declaration

```
BOOL WINAPI JidaPerformanceSetPolicy(HJIDA hJida, DWORD dwType, DWORD dwSetting);
```

Parameters:

hJida
Board handle.

dwType
Must be 0.

dwSetting
New value. See above.

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Sets the current CPU performance policy.

JidaTemperatureCount

Declaration

```
DWORD WINAPI JidaTemperatureCount(HJIDA hJida);
```

Parameters:

hJida
Board handle.

Returns:

Number of available sensors.

Description:

This function returns the number of available temperature sensors.

JidaTemperatureGetCurrent

Declaration

```
BOOL WINAPI JidaTemperatureGetCurrent(HJIDA hJida, DWORD dwType,  
LPDWORD pdwSetting, LPDWORD pdwStatus);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of sensor.

pdwSetting
Pointer to location that will receive the current value.

pdwStatus
Pointer to location that will receive the current sensor status.

JIDA_SENSOR_ACTIVE
Sensor is operating

JIDA_SENSOR_ALARM
Sensor reports alarm condition

JIDA_SENSOR_BROKEN
Sensor circuit is broken

JIDA_SENSOR_SHORTCIRCUIT
Sensor has a short circuit

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads the current value of the temperature sensor. The values are measured in units of 1/1000th degrees Celsius.

JidaTemperatureGetInfo

Declaration

```
BOOL WINAPI JidaTemperatureGetInfo(HJIDA hJida, DWORD dwType, PJIDATEMPERATUREINFO pInfo);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of sensor.

pdwSetting
Pointer to location that will receive the sensor information.

pInfo
Pointer to location that will receive the sensor information.

DWORD dwSize
Must be initialized by the caller with `sizeof(JIDATEMPERATUREINFO)`.

DWORD dwType
temperature sensor type
JIDA_TEMP_CPU, JIDA_TEMP_BOX, JIDA_TEMP_ENV,
JIDA_TEMP_BOARD, JIDA_TEMP_BACKPLANE, JIDA_TEMP_CHIPSETS,
JIDA_TEMP_VIDEO, JIDA_TEMP_OTHER

DWORD dwFlags
temperature sensor capabilities flags

DWORD dwAlarm
temperature alarm mode

DWORD dwRes
temperature resolution, i.e. how exact the sensor can measure

DWORD dwMin
minimum temperature the sensor can measure

DWORD dwMax
maximum temperature the sensor can measure

DWORD dwAlarmHi
upper alarm threshold, i.e. the value up to which the temperature must rise to generate an alarm

DWORD dwHystHi
upper alarm hysteresis, i.e. how many degrees the temperature must fall to reset an alarm

DWORD dwAlarmLo
lower alarm threshold, i.e. the value down to which the temperature must fall to generate an alarm

DWORD dwHystLo
lower alarm hysteresis, i.e. how many degrees the temperature must rise to reset an alarm

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Returns the temperature information. The values are measured in units of 1/1000th degrees Celsius.

JidaFanCount

Declaration

```
DWORD WINAPI JidaFanCount(HJIDA hJida);
```

Parameters:

hJida
Board handle.

Returns:

Number of available sensors.

Description:

This function returns the number of available fan sensors.

JidaFanGetCurrent

Declaration

```
BOOL WINAPI JidaFanGetCurrent(HJIDA hJida, DWORD dwType,  
LPDWORD pdwSetting, LPDWORD pdwStatus);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of sensor.

pdwSetting
Pointer to location that will receive the current value.

pdwStatus
Pointer to location that will receive the current sensor status.

JIDA_SENSOR_ACTIVE
Sensor is operating

JIDA_SENSOR_ALARM
Sensor reports alarm condition

JIDA_SENSOR_BROKEN
Sensor circuit is broken

JIDA_SENSOR_SHORTCIRCUIT
Sensor has a short circuit

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads the current value of the fan speed sensor. The values are measured in RPM (revolutions per minute).

JidaFanGetInfo

Declaration

```
BOOL WINAPI JidaFanGetInfo(HJIDA hJida, DWORD dwType, PJIDAFANINFO pInfo);
```

Parameters:

`hJida`
Board handle.

`dwType`
Zero-based number of sensor.

`pdwSetting`
Pointer to location that will receive the sensor information.

`pInfo`
Pointer to location that will receive the sensor information.

`DWORD dwSize`
Must be initialized by the caller with `sizeof(JIDAFANINFO)`.

`DWORD dwType`
sensor type
`JIDA_FAN_CPU`, `JIDA_FAN_BOX`, `JIDA_FAN_ENV`,
`JIDA_FAN_CHIPSET`, `JIDA_FAN_VIDEO`, `JIDA_FAN_OTHER`

`DWORD dwFlags`
sensor capabilities flags

`DWORD dwAlarm`
fan alarm mode

`DWORD dwAlarmHi`
upper alarm threshold, i.e. the value up to which the fan speed must rise to generate an alarm

`DWORD dwHystHi`
upper alarm hysteresis, i.e. how many RPMs the fan speed must fall to reset the alarm

`DWORD dwAlarmLo`
lower alarm threshold, i.e. the value down to which the fan speed must fall to generate an alarm

`DWORD dwHystLo`
lower alarm hysteresis, i.e. how many RPMs the fan speed must rise to reset the alarm

`DWORD dwOutVal`
new fan speed control output value

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Returns the fan speed sensor information. The values are measured in RPM (revolutions per minute).

JidaVoltageCount

Declaration

```
DWORD WINAPI JidaVoltageCount(HJIDA hJida);
```

Parameters:

hJida
Board handle.

Returns:

Number of available sensors.

Description:

This function returns the number of available voltage sensors.

JidaVoltageGetCurrent

Declaration

```
BOOL WINAPI JidaVoltageGetCurrent(HJIDA hJida, DWORD dwType,  
LPDWORD pdwSetting, LPDWORD pdwStatus);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of sensor.

pdwSetting
Pointer to location that will receive the current value.

pdwStatus
Pointer to location that will receive the current sensor status.

JIDA_SENSOR_ACTIVE
Sensor is operating

JIDA_SENSOR_ALARM
Sensor reports alarm condition

JIDA_SENSOR_BROKEN
Sensor circuit is broken

JIDA_SENSOR_SHORTCIRCUIT
Sensor has a short circuit

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Reads the current value of the voltage sensor. The values are measured in units of 1/1000th volts.

JidaVoltageGetInfo

Declaration

```
BOOL WINAPI JidaVoltageGetInfo(HJIDA hJida, DWORD dwType, PJIDAVOLTAGEINFO
pInfo);
```

Parameters:

hJida
Board handle.

dwType
Zero-based number of sensor.

pdwSetting
Pointer to location that will receive the sensor information.

pInfo
Pointer to location that will receive the sensor information.

DWORD dwSize
Must be initialized by the caller with `sizeof(JIDAVOLTAGEINFO)`.

DWORD dwType
voltage type
JIDA_VOLTAGE_CPU, JIDA_VOLTAGE_DC,
JIDA_VOLTAGE_DC_STANDBY, JIDA_VOLTAGE_BAT_CMOS,
JIDA_VOLTAGE_BAT_POWER, JIDA_VOLTAGE_AC,
JIDA_VOLTAGE_OTHER

DWORD dwNom
nominal voltage / 0 if unknown

DWORD dwFlags
voltage monitor capabilities flags

DWORD dwAlarm
voltage monitor alarm mode

DWORD dwRes
voltage monitor resolution, i.e. how exact the voltage can be measured

DWORD dwMin
minimum voltage that can be measured

DWORD dwMax
maximum voltage that can be measured

DWORD dwAlarmHi
upper alarm threshold, i.e. the value up to which the voltage must rise to generate an alarm

DWORD dwHystHi
upper alarm hysteresis, i.e. how much the voltage must decrease to reset an alarm

DWORD dwAlarmLo
lower alarm threshold, i.e. the value down to which the voltage must fall to generate an alarm

DWORD dwHystLo
lower alarm hysteresis, i.e. how much the voltage must rise to reset an alarm

Returns:

TRUE (1) on success. FALSE (0) on failure.

Description:

Returns the voltage information. The values are measured in units of 1/1000th volts.

Document Revision History

Filename	Date	Edited by	Rev	Alteration to preceding revision
JIDA32.DOC	01.11.98	DP		Initial version!
JIDA32.DOC	11.11.98	C.Riesinger	1.0	Reformatted
JIDA32.DOC	07.06.99	DP	1.1	Updated for JIDA Win32 1.1 Moved Watchdog API to JIDA Driver available for Win NT and Win CE
JIDA32.DOC	21.06.99	DP	1.1	CHAR to TCHAR
JIDA32.DOC	04.04.00	DP	1.2	Windows 2000
JIDA32.DOC	07.03.01	DP	1.3	Changed "JIDA Win32 API" to "JIDA32 Library API" Added Linux and VxWorks platform Added new boards Added hot key LCD support
JIDA32.DOC	04.12.01	DP	1.4	Added new boards Added Windows XP Added VxWorks jida.o Added jidakiss.h Updated Linux section
JIDA32.DOC	05.12.02	DP	1.5	Changed to Kontron JidaI2CRead/Write is implemented Multiple I2C buses per board are implemented
JIDA32.DOC	13.06.03	DP	1.6	Added JidaVga(G/S)etBacklightEnable Added JidaVga(G/S)etContrastEnable Added JidaPerformance(G/S)etCurrent Added JidaI2CType Implemented JidaWDogGetConfigStruct Implemented JidaIO* functions Windows 98/ME now use WDM JIDA driver Full support of all currently implemented JIDA32 BIOS features Linux and VxWorks are separate packages
JIDA32.DOC	09.11.03	DP	1.7	Added JidaI2CWriteReadCombined Added JidaIOGetDirection Added JidaIOSetDirection Added JidaIOGetDirectionCaps Added JidaIOGetName (not yet implemented)
JIDA32.DOC	06.04.04	DP	1.8	Added JidaBoardGetBootErrorLog Added JidaPerformance(G/S)etPolicy(Caps) Added JidaTemperature* Added JidaFan* Added JidaVoltage* QNX 6 version available on request