16AIO

16-Bit/12-Bit ADC/DAC, 32 Scanned Analog Inputs
4 Analog Outputs, 16-bit Digital I/O

PMC-16AIO/16LCAIO
PC104P-16AIO/16LCAIO
PMC-12AIO/12LCAIO
PC104P-12AIO/12LCAIO

INtime Device Driver User Manual

Manual Revision: August 26, 2018
Driver Release Version 5.2.79.4.1

General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: www.generalstandards.com
E-mail: sales@generalstandards.com
E-mail: support@generalstandards.com

Preface

Additional copies of this manual or other literature may be obtained from:

General Standards Corporation
8302A Whitesburg Dr.
Huntsville, Alabama 35802
Phone: (256) 880-8787
FAX: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to provide general usage information on the 16AIO API Library and underlying 16AIO INtime for Windows Device Driver. The API Library software provides the interface between "Application Software" and the device driver. The driver software provides the interface between the API Library and the actual 16AIO hardware. The API Library and device driver interfaces are primarily IOCTL based.

1.2. Acronyms

The following is a list of commonly occurring acronyms used throughout this document.

Acronyms	Description
API	Application Programming Interface
DMA	Direct Memory Access
GSC	General Standards Corporation
INTx	This refers to PCI’s pin based interrupts, which include INTA, INTB, INTRC and INTD.
MSI	Message Signaled Interrupt
PCI	Peripheral Component Interconnect
PMC	PCI Mezzanine Card

1.3. Definitions

The following is a list of commonly occurring terms used throughout this document.

Term	Definition
…	This is used as a shortcut for the 16AIO install directory path or any of its subdirectories.
16AIO	This is used as a general reference to any board supported by the driver and API Library.
API Library	This is the library that provides application level access to 16AIO hardware.
Application	This is the user mode process, which runs in user space with user mode privileges.
Device Mailbox	These global mailboxes, “`16aio.`*`x`*”, give access to individual 16AIO devices.
Driver	This is the application that communicates directly with the 16AIO hardware.
Driver Mailbox	This global mailbox, “`16aio`”, accesses the driver for informational purposes only.
Library	This is usually a general reference to the API Library.

1.4. Software Overview

NOTE: This release of the INtime driver was developed using Microsoft Visual Studio Enterprise 2015, update 3, with INtime for Windows version 6.3. The development host OS was Microsoft Windows 10 Enterprise 2016 LTSB, 64-bit.

1.4.1. API Library

The primary means of accessing 16AIO boards is via the 16AIO API Library. This library forms a layer between the application and the driver. With the library, applications are able to open and close access to a device and, while open, perform I/O control operations, read data from the board and write data to the board. The interface between the API Library and the device driver is message based. The details of the messaging protocol are not documented here. Refer to the *16AIO API Library Reference Manual* for additional information.

1.4.2. Device Driver

> **NOTE**: The macro `INTIME_MAX_DEV_LIST_SIZE` controls the maximum number of devices the driver will recognize. The default is `32` devices. This macro is defined in `…\16aio\driver\os_main.h`. If this macro is changed, then the driver must be rebuilt.

The device driver is the host software that provides a means of communicating directly with 16AIO hardware. The 16AIO device driver is implemented as a standard INtime application written in the C programming language.

Upon start up, the driver creates a mailbox and a corresponding support thread for each 16AIO device (Device Mailboxes and Device Threads). The Device Mailboxes are named `16aio.`*`x`*, where the *`x`* is the zero based index of the device. The Device Threads listen for activity on their respective Device Mailbox. Each message received is immediately handed off to a Service Thread taken from a Service Thread pool common to all of the 16AIO devices. The Service Thread examines the message content and passes the content to a driver function specific to the message type. The message type corresponds to specific driver operations, such as open, close, read, write and IOCTL processing. When the specific operation is complete the Service Thread sends a response message back to the originating thread, and then returns to the Service Thread pool. For additional information see section 4.6 on page 15.

Upon start up, the driver also creates a Driver Mailbox named `16aio`, which is serviced by the driver's main thread. The Driver Mailbox supports open, close and read operations. The purpose of this mailbox is to provide basic identification information about the driver and the installed 16AIO boards. For additional information see section 4.6.2 on page 16.

1.4.3. Device Driver Threads and Priorities

All driver threads run at the same priority. The default is shown below, but can be modified from the command line with the `–p#` argument. The "`#`" refers to the desired priority, which is limited to the range of 131 to 250.

Thread	**Default Priority**
Device Thread	131
Service Thread	131
Driver Thread	131

1.4.4. Device Driver VSEG

The device driver is a non-XM application. Thus, the driver's VSEG is set explicitly. For the 16AIO device driver VSEG is set to 140MB. This is the default of 8MB plus 4MB for each device the driver can accommodate. The driver's device list is limited to 32 devices. The value 4MB is the size of a one million sample buffer. If the device list is fully populated and each device will receive I/O requests larger than 4MB, then the driver's VSEG may need to be increased.

1.5. Hardware Overview

The 16AIO is a high-speed analog Input/Output board. The 16AIO offers 16-bits of resolution. The 12AIO offers 12-bits of resolution. The inputs are configurable as either 32 single-ended input channels or as 16 differential input pairs. There are also four analog output channels. The input sampling rate is at an aggregate rate of up to 300,000 samples per second for the 16AIO and it is up to 1,500,000 for the 12AIO. The output sampling rate is up to 300,000 samples per second per channel for the 16AIO and up to 400,000 for the 12AIO. The analog channels can be clocked from either of two independently configurable on-board clocks. Input and output clocking can be either synchronized or independent and can use either on-board or external synchronization signals. A synchronization output is included so that multiple boards can operate in unison. The analog I/O voltage range is software selectable as +/-2.5V, +/-5V or +/-10V. Internal auto calibration networks permit periodic calibration to be performed without removing the board from the system. The board also features two independent 32K deep FIFOs; one for input and one for output. The output FIFO can be configured for single-shot or continuous waveform output. A 16-bit bidirectional digital I/O port is also provided, along with two auxiliary I/O lines. The board also includes DMA and interrupt capabilities.

1.6. Restrictions

The 16AIO device driver does not support shared interrupts. When a 16AIO device is redirected to INtime, each 16AIO device must be assigned exclusive use of its interrupt. If a shared interrupt is assigned, then the operation of both device drivers may be negatively impacted. Additionally, assigning a shared interrupt has the potential to reduce system stability. See section 4.2.3, page 13.

1.7. Reference Material

The following reference material may be of particular benefit in using the 16AIO. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AIO User Manual* from General Standards Corporation.

- The PCI9080 PCI Bus Master Interface Chip data handbook from PLX Technology, Inc.

  PLX Technology Inc.
  870 Maude Avenue
  Sunnyvale, California 94085 USA
  Phone: 1-800-759-3735
  WEB: http://www.plxtech.com

2. Installation

2.1. File Extraction

The 16AIO INtime device driver and API Library are distributed as a compressed archive. The default file name is `16aio.intime.tar.gz`, which is a gzip compressed tar file, though the distributed file name may include the driver version number. The archive includes all source code and associated files for the device driver, the API Library, the utility libraries, the sample applications and the documentation. The archive should be decompressed and the contents placed in a suitable location. When the archive is decompressed it places its content in a subdirectory named “`16aio.intime`”. The location below is the default used throughout this manual.

```
c:\gsc\
```

2.2. Documentation

The documentation is placed in the root destination directory. The following table briefly describes the documentation provided.

File	Description
`16aio_api_rm.pdf`	This is a PDF version of the *16AIO API Library Reference Manual*.
`16aio_intime_um.pdf`	This is a PDF version of this user manual.

2.3. Directory Structure

The following table describes the directory structure observed by the source archive.

Directory	Content
`16aio.intime`	This is the driver’s root directory.
…`\api\`	This directory contains the 16AIO API Library (section 6, page 18).
…`\batch\`	This directory contains the batch file that invokes Visual Studio 2015 to build individual archive targets (section 2.4, page 10).
…`\docsrc\`	This directory contains the code samples from the reference manual (section 7.1, page 19).
…`\driver\`	This directory contains the driver and its sources (section 4, page 13).
…`\include\`	This directory contains the header files for the various libraries.
…`\lib\`	This directory contains all of the libraries build from the driver archive.
…`\samples\`	This directory contains the sample applications (section 9, page 21).
…`\utils\`	This directory contains utility sources used by the sample applications (section 8, page 20).

Under each directory that contains a build target there is an additional directory structure. This is illustrated in the below table, which uses the API Library as an example.

Directory	Content
…`\api\`	This directory contains the target’s source files (i.e. `*.c`).
…`\api\mvs15\`	This directory contains the Visual Studio 2015 project files.
…`\api\mvs15\Debug\`	This directory contains the Debug specific intermediate and output build files. *
…`\api\mvs15\Release\`	This directory contains the Release specific intermediate and output build files. *

* All library build targets (`*.lib` and `*.rsl`) are placed under the …`\lib\` subdirectory.

2.4. Overall Build Batch File

The driver archive includes a batch file designed to produce fresh builds of all INtime content included in the archive. The batch file is designed to invoke the build tools included with Visual Studio 2015 as installed in their default location. If your tool set is other than Visual Studio 2015, then the batch file will not likely work as intended.

If using other than Visual Studio 2015, then the underlying batch file will need to be edited for use with your build tools. (See `c:\gsc\16aio.intime\batch\mvs15\_bmake.bat`.) Porting of the project files may also be required. Before attempting to invoke the overall build batch file, exit any IDE or editor accessing any of the included INtime project or source files. Follow the below instructions to perform an overall build.

1. In a Command Window, change to the 16AIO INtime root directory (…`\16aio.intime\`).
2. Invoke the batch file as shown to initiate a clean of all build targets..

   ```
   .\bmake_mvs15.bat clean
   ```

3. Invoke the batch file as shown to initiate a rebuild of all build targets.

   ```
   .\bmake_mvs15.bat
   ```

4. The build process should take about five minutes. At the conclusion of the build process the status of each target build is posted to the screen.

3. Basic Software Architecture

This section describes the general architecture for the basic components that comprise an INtime based 16AIO application. The overall architecture is illustrated in Figure 1 below.

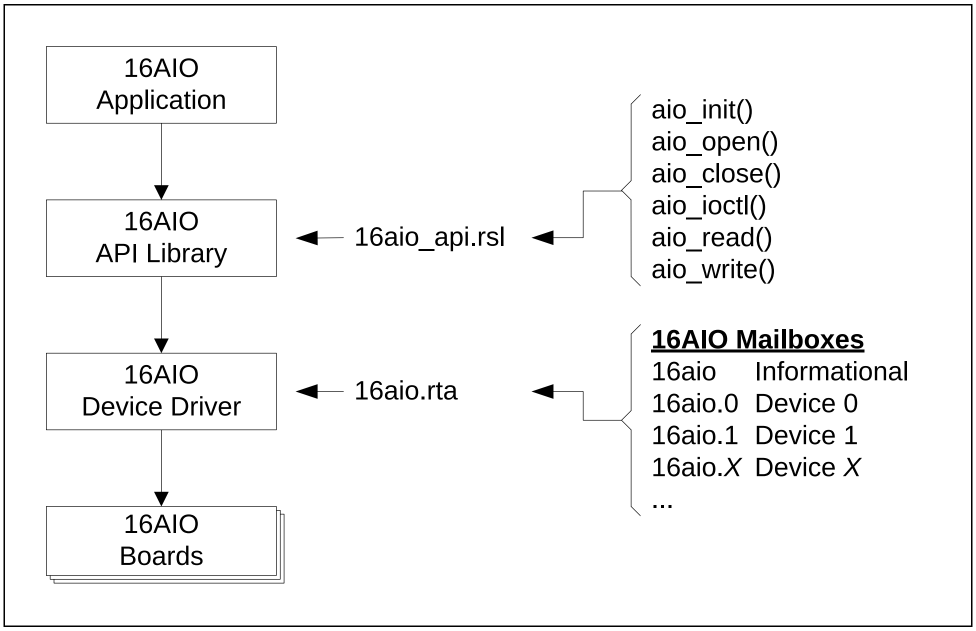

Figure 1 The basic software architecture of INtime based 16AIO applications.

3.1. 16AIO API Library

The API Library is the sole means provided for communicating with the device driver and installed 16AIO hardware. This library implements the application level side of the driver's messaging protocol and is provided as a dynamically linked INtime library (`16aio_api.dsl` with `16aio_api.lib`). The library's interface is defined in the header file `16aio_api.h` and is limited to the services listed in Figure 1. The `aio_ioctl()` supported services are defined in the header file `16aio.h`. Refer to the *16AIO API Library Reference Manual* for additional information.

3.2. 16AIO Device Driver

The device driver is an INtime application, `16aio.rta`, written specifically to provide a software mechanism for interfacing with installed 16AIO hardware. The communications interface to the driver involves an intricate exchange of messages via a set of global mailboxes. These global mailboxes are created when the driver is started, and they are deleted when the driver exits. The Device Mailboxes, named "`16aio.x`", accesses individual 16AIO devices. The trailing "`x`" is the zero based index of the device to access. Via the Driver Mailbox, named "`16aio`", applications interact with the driver for informational purposes only. For additional information see section 4.6.2, page 16.

3.3. 16AIO Boards

At the bottom of Figure 1 are the 16AIO boards. Any number of boards may be utilized. While the end product of an effort may be a software application written for a specific model of 16AIO, the software components provided by General Standards are generally written to function with all supported 16AIO models.

4. Device Driver

4.1. Build

The device driver is built via the Overall Build Batch File (section 10, page 10), but it can also be built individually as follows.

1. Load the Visual Studio solutions file listed below.

   `c:\gsc\16aio.intime\driver\mvs15\16aio.sln`

2. Select the Release configuration. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select the “Release” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped. The end product of build is the application listed below.

   `c:\gsc\16aio.intime\driver\mvs15\Release\16aio.rta`

4.2. Host Preparation

Before the 16AIO INtime driver can be started there are several preparatory steps which must be performed first.

4.2.1. Windows’ Driver Installation

The 16AIO Windows driver must be installed. This step is necessary so that appropriate system resources are assigned to each board. Each board’s resources include memory regions, an I/O region and an interrupt. The assignment of the resources to each 16AIO is performed by the Windows driver. Installation of the Windows driver is described in documentation specific to that driver.

4.2.2. INtime Node Startup

Use of the 16AIO INtime driver on the local host requires that a local INtime Node be started. Refer to the INtime documentation for starting an INtime node.

4.2.3. Device Interrupt Redirection

Use of the 16AIO INtime driver on the local host requires that each 16AIO’s interrupt be redirected from Window to INtime. Follow the below general procedures to manually redirect the 16AIO interrupts.

1. Select the Windows main menu option “*Start* | INtime | INtime Configuration”.

2. From the “INtime Configuration Panel” window that appears, double click on the “INtime Device Manager” icon.

3. From the “INtime Device Manager” window locate and right click on an 16AIO under the “Windows devices” list.

4. From the popup menu that appears, select the menu option titled “Pass to INtime with legacy IRQ”. (For additional information see section 4.2.3.1, page 14.)

5. Save the configuration by selecting the menu option “File | Save the configuration”.

   NOTE: You may be presented with a “System Settings Change” window requesting a reboot.
   If so, perform the reboot after completing the interrupt redirection process.

6. Dismiss the “INtime Device Manager” window by clicking on the close button in the upper right corner of the window.

7. Dismiss the “INtime Configuration Panel” window by clicking on the “Exit” button.

4.2.3.1. Pass to INtime with legacy IRQ

This is the only “Pass to INtime” option that can be used with 16AIO boards. This option makes use of the 16AIO’s pin based interrupts commonly referred to as INTx Interrupts or Legacy Interrupts. These are supported by all PCI based models of the 16AIO, regardless of form factor. These pin based interrupts may be shared among multiple devices. If a device sharing the same interrupt is not passed to INtime, then INtime reports a conflict, thus preventing use of this option with the respective board. In this case either the 16AIO or the other device(s) must be moved to a non-conflicting slot.

NOTE: The 16AIO does not support MSI Interrupts.

4.3. Startup

Follow the below steps to start driver execution.

1. Select the Windows main menu option “*Start* | INtime | RT Application Loader”. From the “RT Application Loader” window navigate to, and select, the driver executable shown below. Add command line argument as needed per the table below. Then click the “Open” button.

   ```
   c:\gsc\16aio.intime\driver\mvs15\Release\16aio.rta
   ```

Argument	Description
`-h`	Display usage/help information.
`-p#`	Adjust the priority of driver threads (131-250) (section 1.4.3, page 8).
`-x`	This initiates driver termination (*exit*).

2. The driver should start and generate a small amount of display output. The driver will continue to execute until told to exit.

   NOTE: During driver startup the display output will include the basic model number of each discovered board. The output will appear as “`16aio: device loaded: 16AIO`”, presuming, for example, the board is a model 16AIO.

   NOTE: At the end of driver startup the last line of the displayed output will indicate the status of the effort. The last line will start with “`16aio: driver loaded:`” if the effort was successful. In this case the driver will remain running and provide access to the boards discovered. The last line will start with “`16aio: driver load failure:`” if the effort was not successful. In this case the driver immediately exits.

4.4. Shutdown

Shutdown the driver using the below options.

4.4.1. Shutdown Using the Driver Command Line

Follow the below steps to tell the driver to exit.

1. Select the Windows main menu option "*Start* | INtime | RT Application Loader". From the "RT Application Loader" window navigate to, and select, the driver executable shown below. Enter the *exit* command line argument from the table below. Then click the "Open" button.

```
c:\gsc\16aio.intime\driver\mvs15\Release\16aio.rta
```

Argument	Description
`-h`	Display usage/help information.
`-x`	This initiates driver termination (*exit*).

2. The driver instance being executed should take only a second or so to complete its task. The driver already running should exit shortly thereafter.

4.4.2. Shutdown Using the *exit* Sample Application

Follow the below steps to tell the driver to exit.

1. Build the *exit* sample application as described in section 9 on page 21.

2. Select the Windows main menu option "*Start* | INtime | RT Application Loader". From the "RT Application Loader" window navigate to the executable listed below. Then click the "Open" button.

```
c:\gsc\16aio.intime\exit\mvs15\Debug\exit.rta
```

3. The application should take a second or so to complete its task. The driver should exit shortly thereafter.

4.5. Version

The driver version number can be obtained in two ways. First, it is reported by the driver both when the driver is loaded and when it is unloaded. This is part of the display output generated by the driver. Second, it is reported in the text obtained by reading from the Driver Mailbox cataloged under the root process. This mailbox is accessed using the API Library call `aio_open()` service using device index `-1`.

4.6. Access

Access to the 16AIO driver and individual 16AIO devices is via mailboxes cataloged into the root process.

> **NOTE:** If a device is in an open state when an application exits, the device will remain in the opened state. If this occurs the driver may have to be reloaded to gain subsequent access to the same device.

4.6.1. Device Mailboxes: 16aio.*x*

The 16AIO driver provides individual mailboxes for access to each 16AIO device. These Device Mailboxes are cataloged into the root process and are named "`16aio.x`". The suffix "`x`" is the zero based index of the device to access. (For example, the first device is accessed via Device Mailboxes “`16aio.0`”.) Using the API Library (section 5, page 17) an application calls `aio_open()` with index *x* to access a Device Mailbox, where “`x`” corresponds directly to the mailbox suffix.

4.6.2. Driver Mailbox: 16aio

The 16AIO driver provides a mailbox named “`16aio`”. This mailbox is maintained so that applications can gain information about the driver and installed devices without having to query individual devices. (Such queries may not be possible as devices may already be open in exclusive access mode.) This Driver Mailbox is accessed using the `aio_open()` with device index `-1`. Once opened, a read request returns the below information.

```
version: 5.2.79.4
32-bit support: yes (native)
boards: 1
models: 16AIO
```

Entry	**Description**
`version`	This gives the driver version number in the form `x.x.x.x`.
`32-bit support`	This reports the driver’s support for 32-bit applications. This will be either “`yes`” or “`no`” for 64-bit driver builds and “`yes (native)`” for 32-bit builds.
`boards`	This identifies the total number of boards the driver detected.
`models`	This gives a comma separated list of the basic model number for each board the driver detected.

4.7. Driver Interface

At the lowest level, direct access to the device driver is through the Device Mailboxes, which involves message based transactions. It is through these mailboxes that communication is established with a device, through which the device is configured and through which data is written to and read from the device. The messaging interface is defined within the driver sources, but it is not described in any documentation. Refer to the *16AIO API Library Reference Manual* for additional information.

The functional interface to the driver is IOCTL based. The IOCTL services are utilized in combination to achieve a higher level of functionality. The IOCTL command codes are defined in the header files `16aio.h`. Their default location is `c:\gsc\16aio.intime\include\`. The IOCTL command codes are documented in the *16AIO API Library Reference Manual*.

5. Main Interface Files

For additional information refer to the *16AIO API Library Reference Manual*.

5.1. Main Header File

Throughout this document references are made to various header files included as part of developing 16AIO INtime applications. For ease of use it is suggested that applications include only the Main Header file shown below rather than individually including those headers identified separately elsewhere in this document. Including this header file pulls in all other pertinent 16AIO header files.

Description	File	Location
Header File	`16aio_main.h`	`…\include\`

5.2. Main Library File

Throughout this document references are made to various library files linked as part of developing 16AIO INtime applications. For ease of use it is suggested that applications link only the Main Library file shown below rather than individually linking those libraries identified separately elsewhere in this document. Linking this library file pulls in all other pertinent 16AIO library files.

Description	File	Location
Static Link Library	`16aio_main.lib`	`…\lib\` *

* Debug and release versions of the libraries are included under corresponding subdirectories.

5.3. Build

The Main Library is built via the Overall Build Batch File (section 2.4, page 10), but it can also be built individually as follows.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\16aio.intime\lib\mvs15\16aio_main.sln
   ```

2. Select the Release application configuration. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select the “Release” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, one success, no failures and one skipped. (It is the Debug version of the library that is skipped.) The final build target is as follows.

   ```
   c:\gsc\16aio.intime\lib\mvs15\Release\16aio_main.lib
   ```

NOTE: If the Debug configuration is selected, then debug versions of the libraries will be located in the corresponding `Debug` subdirectory and the Release build will be skipped.

6. 16AIO API Library

For additional information refer to the *16AIO API Library Reference Manual*.

6.1. Build

The API Library is built via the Overall Build Batch File (section 2.4, page 10), but it can also be built individually as follows.

> **NOTE**: If the API Library is rebuilt, then the Main Library must also be rebuilt (section 5, page 17).

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\16aio.intime\api\mvs15\16aio_api.sln
   ```

2. Select the Release application configuration. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select the “Release” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, two successes, no failures and nothing skipped. The final build targets are as follows.

   ```
   c:\gsc\16aio.intime\lib\mvs15\Release\16aio_api.lib
   c:\gsc\16aio.intime\lib\mvs15\Release\16aio_api.rsl
   ```

> **NOTE**: If the Debug configuration is selected, then debug versions of the libraries will be located in the corresponding `Debug` subdirectory.

6.2. Library Use

The library is used at application compile time, at application link time and at application run time. At compile time include the below listed header file in each source file using a component of the library interface. Also, edit the include search path to locate the header file in the below listed directory. At link time include the below listed static link library with the objects being linked with the application. Also, edit the library search path to locate the static link library in the below listed directory. At run time, the INtime Dynamic Link Library must be included either in the same directory as the using INtime application, or it must be copied to the “RSL search path”.

Description	**File**	**Location**
Header File	`16aio_api.h`	…`\include\`
Static Link Library	`16aio_api.lib`	…`\lib\` *
Dynamic Link Library	`16aio_api.rsl`	Same as `.rta` or in “RSL search path”. †

* Debug and release versions of the libraries are included under corresponding subdirectories.
† Refer to the INtime Configuration Manager for the “RSL search path”.

> **NOTE**: The “RSL search path” is defined via the INtime Configuration application.

7. Document Source Library

For additional information refer to the *16AIO API Library Reference Manual*.

7.1. Build

The Document Source Library is built via the Overall Build Batch File (section 2.4, page 10), but it can also be built individually as follows.

> **NOTE**: If the Document Source Library is rebuilt, then the Main Library must also be rebuilt (section 5, page 17).

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\16aio.intime\docsrc\mvs15\16aio_dsl.sln
   ```

2. Select the Release application configuration. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, two successes, no failures and nothing skipped. The final build targets are as follows.

   ```
   c:\gsc\16aio.intime\lib\mvs15\Release\16aio_dsl.lib
   ```

> **NOTE**: If the Debug configuration is selected, then debug versions of the libraries will be located in the corresponding `Debug` subdirectory.

7.2. Library Use

The library is used at application compile time and at application link time. At compile time include the below listed header file in each source file using a component of the library interface. Also, edit the include search path to locate the header file in the below listed directory. At link time include the below listed static link library with the objects being linked with the application. Also, edit the library search path to locate the static link library in the below listed directory.

Description	**File**	**Location**
Header File	`16aio_dsl.h`	…`\include\`
Static Link Library	`16aio_dsl.lib`	…`\lib\` *

* Debug and release versions of the libraries are included under corresponding subdirectories.

8. Utilities Library

For additional information refer to the *16AIO API Library Reference Manual*.

> **NOTE**: The utility sources include thread creation services. The default stack size designated at thread creation time is 64K bytes. Refer to `…\16aio\utils\os_util_thread.c` for additional information.

8.1. Build

The Utilities Library is built via the Overall Build Batch File (section 2.4, page 10), but it can also be built individually as follows.

> **NOTE**: If the Utilities Library is rebuilt, then the Main Library must also be rebuilt (section 5, page 17).

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\16aio.intime\utils\mvs15\16aio_utils.sln
   ```

2. Select the Release application configuration. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select the “Release” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, two successes, no failures and nothing skipped. The final build targets are as follows.

   ```
   c:\gsc\16aio.intime\lib\mvs15\Release\16aio_utils.lib
   ```

> **NOTE**: If the Debug configuration is selected, then debug versions of the libraries will be located in the corresponding `Debug` subdirectory.

8.2. Library Use

The library is used at application compile time and at application link time. At compile time include the below listed header file in each source file using a component of the library interface. Also, edit the include search path to locate the header file in the below listed directory. At link time include the below listed static link library with the objects being linked with the application. Also, edit the library search path to locate the static link library in the below listed directory.

Description	**File**	**Location**
Header File	`16aio_utils.h`	`…\include\`

Static Link Library	`16aio_utils.lib`
	`gsc_utils.lib`
		`…\lib\` *

* Debug and release versions of the libraries are included under corresponding subdirectories.

9. Sample Applications

The Library archive includes a variety of sample and test applications. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Build Batch File (section 10, page 10), but each may be built individually by loading its respective solution file into Visual Studio. The initial output from each application includes information on its supported command line arguments. The following gives a brief overview of each application.

9.1. aithwaitrate - Analog Input Threshold Wait Rate - …\aithwaitrate\

This application monitors the Input Buffer Threshold flag based on a variety of configurable settings to assist in evaluation the WAIT service. Performance statistics are included in the output.

9.2. aoburst - Analog Output Burst - …\aoburst\

This application outputs a ramp wave on each of the four output channels using output bursting.

9.3. aout - Analog Output - …\aout\

This application outputs a repeating pattern on the four output channels. The pattern is different for each channel, though they are synchronized at the same sample rate.

9.4. auxin - Auxiliray Input - …\auxin\

This application reads the cable’s auxiliary input and reports the values read to the console.

9.5. auxout - Auxiliary Output - …\auxout\

This application writes a pattern to the cable’s auxiliary output line.

9.6. din - Digital Input - …\din\

This application reads the cable’s digital I\O signals and reports the values read to the console.

9.7. dout - Digital Output - …\dout\

This application writes a pattern to the cable’s digital output lines as it is displayed to the console.

9.8. driver_mbox - Driver Mainbox - …\driver_mbox\

This application dumps the content of the Driver Mailbox to the console.

9.9. exit - Driver Exit - …\exit\

This application initiates termination of the driver.

9.10. id - Identify Board - …\id\

This application reports detailed board identification information. This can be used with tech support to help identify as much technical information about the board as possible from software.

9.11. laout - Looping Analog Output - …\laout\

This application outputs different test patterns on the four output channels using the output buffer looping feature.

9.12. regs - Register Access - …\regs\

This application provides menu based interactive access to the board’s registers, and reports other pertinent information to the console.

9.13. rxrate - Receive Rate - …\rxrate\

This application configures the board for its highest ADC sample rate then reads the input as fast as possible. The purpose is to measure the peak sustainable input rate for the host, per the provided command line arguments.

9.14. savedata - Save Acquired Data - …\savedata\

This application configures the board for a modest sample rate, reads a megabyte of data, then saves the data to a hex file.

9.15. signals - Digital Signals - …\signals\

This application configures the board to drive the digital output signals for a user specified period of time. This is done to facilitate setup of test equipment to capture those signals during actual use.

10. Operating Information

This section explains some basic operational procedures for using the 16AIO under INtime. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

10.1. Exception 19, Thread is using all processor time

Some operations carried out by the driver on behalf of the applications are known to the highly CPU intensive and run uninterrupted for what INtime may consider long periods of time. These operations may thus run for so long that they interfere with lower priority threads gaining execution time. This is particularly true of PIO based I/O requests, though it may apply to other operations as well. Consequently, the INtime monitoring tool "SpinDoctor" may throw an exception on the driver when the condition arises. The exception message appears as follows, where the *xxx* sequences refer to a driver process address. As given below, there are ways of preventing, or reducing, occurrences of the exception.

```
Exception 19 at xxxx:xxxx (Thread is using all processor time)
```

10.1.1. Avoid PIO Based I/O Requests

As PIO based I/O requests are known to be highly CPU intensive and lengthy operations, use of DMA may be an option to preventing the exception. While DMA can be very efficient for larger I/O requests, for smaller I/O requests PIO is more efficient. The threshold where DMA outperforms PIO is likely somewhere between requests for 60 to 100 samples. At these small request sizes PIO operations are extremely unlikely to cause the exception. So, unless there is some very uncommon reason for using PIO with larger I/O requests, applications should elect to use DMA.

10.1.2. Increase the Run Time Threshold

The threshold at which SpinDoctor throws an exception is configurable. By adjusting a setting, the driver can possibly be prevented from causing the fault without also preventing the detection of run time excesses by other applications. To access the SpinDoctor configuration settings open the INtime "Node Management" tool, select the "Advanced" tab, then, under the "Section:" list, select the "SPIN" option. Then, adjust the "MAX" option, which is the fault's maximum permissible run time threshold in 10 millisecond intervals.

NOTE: Any change made will take effect after the node is restarted.

10.1.3. Disable Exception Monitoring

By default, SpinDoctor is disabled in runtime environments and enabled in development environments. Thus, to prevent the exception from interfering with development efforts it can be disabled. To access the SpinDoctor configuration settings open the INtime "Node Management" tool, select the "Advanced" tab, then, under the "Section:" list, select the "SPIN" option. Set the option "ACTIVE" to zero to disable SpinDoctor run time monitoring.

NOTE: Any change made will take effect after the node is restarted.

10.1.4. Adjust Application Thread Priorities

Applications are able to adjust the execution priority of their own threads. If an application thread is one of those "low priority" threads receiving insufficient execution time, then the application may be able to address the fault circumstance by increasing its execution priority relative to that of the driver. Refer to the INtime documentation for execution priority guidelines and programming. Also, see section 1.4.3, page 8 for driver priority information.

10.1.5. Adjust Driver Thread Priorities

If the above options are either not suitable or insufficient to prevent the exception, then adjusting the driver’s run time priority may be an option. This is done via a driver command line argument. Refer to section 1.4.3, page 8.

Document History

Revision	**Description**
August 26, 2018	Initial release, version number 5.2.79.4.1. Added `aithwaitrate` sample application.
July 7, 2018	Initial release, version number 5.2.79.4.0.