

## Introduction

These are the AVIX release notes. AVIX release notes are published in an incremental fashion meaning this document contains the most recent and all preceding release notes. For each published version, the release notes are present in a different section of this document. The version a section is applicable to is noted clearly in the section header and in the names of the chapters contained in each section.

*Before starting to use AVIX make sure to thoroughly read these release notes since issues may be documented here that help you get up and running with AVIX as fast as possible.*

## Summary 5.0.0

AVIX version 5.0.0 comes with the following new functionality:

### Asynchronous support added to pipes:

Pipes are an important communication mechanism to transfer user specified data between threads and between threads and Interrupt Service Routines. The pipe mechanism is enhanced with asynchronous support. Besides the already present blocking read and write operations, two additional functions are offered allowing non-blocking or asynchronous read and write operations. When using these asynchronous functions, the calling thread returns immediately, regardless whether the read/write request is ready or not. As a parameter to asynchronous read/write calls, an event group and one or more event flags are passed. Once the required amount of data is processed, the specified event flag(s) is/are set. This allows the thread to continue with other functionality while the request is handled in the background. Once the thread actually needs to know whether the request has finished, it just starts waiting for the specified event flags to become set. Combined with the already present event flag support for timers and message queue's, this allows for a much cleaner application structure where a thread uses a single point to potentially wait for many different requests to have finished. Besides the most important two functions to asynchronously read and write, the pipe mechanism is enhanced with a number of other functions allowing to cancel a pending operation and to flush a pipe. Especially when using pipes between threads and Interrupt Service Routines, this allows for much more flexible management of the various operations.

In addition to the extended API, the pipe mechanism implementation has undergone a complete overhaul resulting in a decrease in the number of thread context switches involved with pipe operations, leading to a significant performance increase. Finally, a number of pipe functions are further optimized, especially `avixPipe_ReadFromISR` and `avixPipe_WriteFromISR` allowing higher interrupt rates when using pipes.

### Simplification of event group mechanism API:

The AVIX event group mechanism works with event groups and thread event groups. Until now, most event group functions came in two flavors, one receiving an event group id as a parameter and one receiving a thread id as a parameter. Reason for this is that kernel object id's (thread id's, event group id's are type safe and passing a thread id to a function requiring an event group id is not possible. The event group API is simplified by removing all thread id based functions. Still a thread id can be used with the event group functions to manipulate the thread implicit event group. This is accomplished by allowing a thread id to be converted to an event id and passing the result of this conversion to the event group function. This results in a much simpler API while retaining the full functionality and type safety on the API. Even better, the event group functionality is further enhanced with additional functionality. The consequence of the API change is that existing code may need to undergo a minor modification, details of which are found in this document in chapter 'Version 5.0.0 Event Group API Changes background' on page 6.

## New Features 5.0.0

- **Functions added for asynchronous pipe support and generic pipe support**

AVIX pipe services are extended with many new functions:

- `avixPipe_WriteAsync`: Write data to a pipe (non-blocking).
- `avixPipe_ReadAsync`: Read data from a pipe (non-blocking).
- `avixPipe_GetStatusAsyncReq`: Obtain the current status of an asynchronous read or write request.
- `avixPipe_AbortAsyncReq`: Abort a pending asynchronous read or write request.
- `avixPipe_FlushAndAbort`: Abort all pending requests (both synchronous and asynchronous) and discard all current content of the pipe buffer.
- `avixPipe_SetHandlerTracePort`: Assign an I/O port to a pipe which will be set high when the pipe handler is active and low when it is not. A pipe handler is an AVIX internal function establishing the interface between an ISR and a thread.

- **Function added to get the current flags of an event group (`avixEventGroup_GetEventFlags`)**

Using non-blocking function `avixEventGroup_GetEventFlags`, the current value of the flags contained in an event group can be requested.

- **Function added to pulse a thread trace port (`avixThread_PulseTracePort`)**  
For testing and performance tuning purposes, AVIX offers Thread Activation Tracing. This mechanism allows I/O ports to be assigned to threads where for the active thread the I/O port is pulled high and for inactive threads the I/O port is pulled low. For timing measurements within a thread, use can be made of the new function `avixThread_PulseTracePort`. This function causes a short pulse on the I/O port of the active thread. By placing a call to this function at specific points in the thread code, detailed real time timing measurements can be performed.
- **Select WFI or WFE instruction to enter low power mode (AVIX for Cortex-M3 only)**  
Cortex-M3 offers two instructions to enter a low power mode, WFE or WFI. The AVIX idle thread is responsible for executing the desired instruction. In earlier versions, only the WFI instruction was used. Starting with version 5.0.0, the desired instruction can be selected using configuration parameter `avix_WAKEUP_WITH_EVENT`. More details can be found in the applicable port guide and in the AVIX configuration file `avixSystemSettings.h`.

## Changes 5.0.0

- **Thread event group specific functions replaced by regular event group functions**  
For a simplification of the API, all thread event group specific functions are removed and should be replaced by their regular event group counterparts. Note this does not influence the functionality of AVIX. Thread still contain an implicit event group which can be used just as flexible as in previous versions. Below the functions that are removed are specified, together with the function that should be used to replace it.
  - `avixEventGroupThread_Change`, use `avixEventGroup_Change` instead.
  - `avixEventGroupThread_ChangeFromISR`, use `avixEventGroup_ChangeFromISR` instead.
  - `avixEventGroupThread_Wait`, use `avixEventGroup_Wait` instead.
  - `avixMsgQThread_ConnectEventGroupThread`, use `avixMsgQThread_ConnectEventGroup` instead.
  - `avixTimer_ConnectEventGroupThread`, use `avixTimer_ConnectEventGroup` instead.
  - `avixExch_ConnectEventGroupThread`, use `avixExch_ConnectEventGroup` instead.
- **Extension to interface of `avixPipe_WriteFromISR` and `avixPipe_ReadFromISR` to indicate flush.**  
A pipe buffer can be flushed using function `avixPipe_FlushAndAbort`. When during this operation an ISR tries to access the pipe with either `avixPipe_WriteFromISR` or `avixPipe_ReadFromISR` the return value of these functions is negative to indicate the fact the pipe buffer is being flushed.
- **Library format changed from COFF to ELF (AVIX for PIC24-dsPIC only).**  
For AVIX for PIC24-dsPIC, the library format is changed to the ELF format. For all other ports this was already the case so starting with this version, for all ports the AVIX libraries are in the ELF format. Make sure to change the format of your project to the ELF format in the MPLAB project properties.

## Known issues 5.0.0

- none

## Fixed issues 5.0.0

- none

## Documentation changes 5.0.0

- **Documentation updated to reflect new functionality**  
All documentation is updated to reflect new functionality and API changes.

## How to migrate from version 4.6.1 to version 5.0.0

### Background

When installing AVIX, a directory structure is created as described in the applicable Porting Guide.

The Application Programming Interface of version 5.0.0 is not identical to that of version 4.6.1. The EventGroupThread functions are removed and must be replaced by the regular EventGroup functions. This update must be made manually, details are found in chapter 'Version 5.0.0 Event Group API Changes background' on page 6.

Configuration settings are contained in AVIXSystemSettings.h.

The system settings file is installed with default values. It is essential that the settings you are using with version 4.6.1 are migrated to the settings file that comes with version 5.0.0 taking into account the changes mentioned before. This has to be done manually.

The AVIX library name contains its version number and therefore development environment project files must be changed to refer the version 5.0.0 library.

### ***Migration procedure***

In accordance with the above, the following migration procedure is advised:

1. Install the 5.0.0 distribution to a new location in your file system
2. Copy the configuration parameters of the 4.6.1 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or a similar utility.
3. Adapt application code to the simplified Event Group API according the guidelines in chapter 'Version 5.0.0 Event Group API Changes background' on page 6.
4. Replace all version 4.6.1 files in your development structure with the new version 5.0.0 files, maintaining the directory structure of version 4.6.1. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
5. Remove the version 4.6.1 library from your development environment project file.
6. Change the project settings to include the new version 5.0.0 library.
7. Change the project settings to refer the AVIX interface directory.

Before executing these steps you might want to make a backup of the version 4.6.1 files.

## Build and development environment issues 5.0.0

### AVIX for Cortex-M3 (IAR EWARM 6.x)

- none

### AVIX for Cortex-M3 (KEIL MDK 4.x)

- none

### AVIX for PIC32MX (MPLAB8x-C32/XC32)

- **Support added for new XC32 compiler**  
Starting with version 5.0.0, besides the C32 compiler AVIX for PIC32 supports the new XC32 compiler.

### AVIX for PIC24-dsPIC (MPLAB8x-C30/XC16)

- **Support added for new XC16 compiler**  
Starting with version 5.0.0, besides the C30 compiler AVIX for PIC24-dsPIC supports the new XC16 compiler.

### AVIX for PIC32MX (MPLABX-C32/XC32)

- **Support added for MPLABX development environment**  
Starting with version 5.0.0, AVIX for PIC32 supports the new Microchip MPLABX development environment. All demo applications are based on MPLAB8x but can easily be imported in MPLABX by using the MPLABX provided functionality.

### AVIX for PIC24-dsPIC (MPLABX-C30/XC16)

- **Support added for MPLABX development environment**  
Starting with version 5.0.0, AVIX for PIC24-dsPIC supports the new Microchip MPLABX development environment. All demo applications are based on MPLAB8x but can easily be imported in MPLABX by using the MPLABX provided functionality.

---

### IAR EWARM 6.x RTOS Viewer plug-in

- New RTOS viewer compatible with AVIX for CORTEX-M3 version 5.0.0

### KEIL MDK 4.x RTOS Viewer plug-in

- New RTOS viewer compatible with AVIX for CORTEX-M3 version 5.0.0

### MPLAB8x RTOS Viewer plug-in

- New RTOS viewer compatible with AVIX for PIC32 and AVIX for PIC24-dsPIC version 5.0.0

### MPLABX RTOS Viewer plug-in

- none  
Starting with version 5.0.0, AVIX for PIC32 and AVIX for PIC24-dsPIC support the new Microchip MPLABX development environment. No RTOS viewer is available for this environment yet.

## Version 5.0.0 Event Group API Changes background

In order to change your application code to the new Event Group API a number of small changes are required. These changes have to do with the function name and the event group parameter used with these functions.

Event group functions in earlier AVIX versions all came in two flavors. One was intended to be used with regular event groups and those functions received the id of an event group. The other was intended to be used with thread event groups and those functions received the id of a thread.

An example of the 'old' API is shown below:

In case you would like to set flag 0 in an event group this looked like:

```
tavixEventId eventId;
avixEventGroup_Change(eventId, AVIX_EVENT_GROUP_SET, AVIX_EF(0));
```

When setting flag 0 in a thread event group on the other hand, this looked like:

```
tavixThreadId threadId;
avixEventGroupThread_Change(threadId, AVIX_EVENT_GROUP_SET, AVIX_EF(0));
```

Because of the modified Event Group API, the second function is no longer supported and the regular event group function must be used instead. It is however not possible to pass a thread id to the event group function. To accomplish this, a thread id can be converted to an event group id using id attribute `asEventId`. The result of this conversion can be passed to the regular event group function. As a result, the modified functionality to set flag 0 in a thread event group looks like:

```
tavixThreadId threadId;
avixEventGroup_Change(threadId.asEventId, AVIX_EVENT_GROUP_SET, AVIX_EF(0));
```


This modification is required for all 'ThreadEventGroup' functions used in your application. For the name of the function, 'Thread' must be removed and to the threadId passed to that function, attribute `.asEventId` must be added.

The easiest way to accomplish this is to perform a global search on your application source code, looking for 'ThreadEventGroup'. This will find the use of all applicable functions making it easy to execute the required modifications.

To the above scenario, there is one minor exception, which is function `avixEventGroupThread_Wait`. In earlier versions, this function did not receive the id of the thread since this was implicitly the calling thread. After changing the name of this function to `avixEventGroup_Wait`, it is required to pass the id of the thread this function operates on. So for this function only, a parameter must be added, which is the first parameter to this function. This parameter is the id of the calling thread, converted to an event group id. This is obtained using the following construct:

```
avixThread_GetIdCurrent().asEventId
```

So for all occurrences of `avixEventGroupThread_Wait` in your application, the function name must be changed to `avixEventGroup_Wait` and `avixThread_GetIdCurrent().asEventId` must be added as a first parameter to this function.

 *Note that because of this API change, it is also possible to pass the id of another thread (not the thread making the function call) to function `avixEventGroup_Wait`. Although such code will successfully compile it will not work and result in a runtime error. A thread is only allowed to wait for its own event group and waiting for thread event groups of other threads is not possible. This is not a restriction of the new API but has always been the case.*

**This page is intentionally left blank**

## Summary 4.6.1

These are the release notes of AVIX version 4.6.1. The reason for release of this version is added support for the 60/70MHz Microchip EP MCU's in AVIX for PIC24-dsPIC. With this extension using a single AVIX for PIC24-dsPIC license allows AVIX based applications to be developed for all Microchip 16 bit parts that fulfill the AVIX resource requirements.

## New Features 4.6.1

- **Support for PIC24 and dsPIC EP MCU's (AVIX for PIC24-dsPIC only)**  
Starting with version 4.6.1, AVIX can be used with the new Microchip EP MCU's. AVIX is distributed as a library. The older PIC24 and dsPIC MCU's are not binary compatible with the new EP MCU's. For this reason, starting with version 4.6.1, AVIX for PIC24-dsPIC is distributed with two libraries a selection from which must be made based on the type of MCU used. Details are found in the 'AVIX for PIC24-dsPIC Port Guide' document.

## Changes 4.6.1

- **Number of Trace Port Definitions extended**  
Because of the larger number of GPIO ports in the MCU's that are supported by AVIX, the number of Trace Port definitions is extended to 15. The GPIO ports used for tracing are identified by AVIX\_TRACE\_PORT\_A up to and including AVIX\_TRACE\_PORT\_O. These symbols are used as a parameter for `avixThread_SetTracePort` and `avixThread_SetTracePortAndResume` to identify the GPIO port used for Thread Activation Tracing. AVIX\_TRACE\_PORT\_A identifies the lowest numbered GPIO port and AVIX\_TRACE\_PORT\_O identifies the highest numbered GPIO port.

*Note that the number of GPIO ports differs per MCU and using trace port symbols for which the used MCU does not have a GPIO port has no effect.*

## Known issues 4.6.1

- none

## Fixed issues 4.6.1

- **PR140: Trace port issue with PIC32 MCU's having fewer GPIO ports (AVIX for PIC32MX only)**  
Not all PIC32 MCU's have the same number of GPIO ports. For PIC32 MCU's with fewer GPIO ports (e.g. PIC32MX440..) building an AVIX for PIC32MX based application fails because an attempt is made to link with GPIO port symbols that are not present. Starting with version 4.6.1, when building an AVIX for PIC32MX based application the number of ports present on the MCU in use is automatically and correctly determined and building for the mentioned parts will no longer fail.
- **PR141: Locking a semaphore may succeed while count is zero**  
Under certain exotic conditions, locking a semaphore having a count of zero may succeed. Especially when unlocking semaphores from ISR's or from threads that are made ready from an ISR this issue may occur. Starting with AVIX version 4.6.1, semaphores are deal with according the documentation.

***This issue is considered critical. Therefore it is strongly suggested to start using release 4.6.1 as soon as possible.***

- **PR142: Building for dsPIC30F parts with C30 ver. 3.3 or later fails (AVIX for PIC24-dsPIC only)**  
Based on the system timer configured in AVIXSystemSettings.h, AVIX determines which SFR's must be initialized. Because of a change in the dsPIC30F header files in C30 version 3.3 and later, building an application for a dsPIC30F part fails when using this compiler.



## Documentation changes 4.6.1

- **AVIX for PIC24-dsPIC port guide**

This document is updated with information regarding the newly supported EP MCU's.

## How to migrate from version 4.6.0 to version 4.6.1

### Background

When installing AVIX, a directory structure is created as described in the applicable Porting Guide.

The Application Programming Interface of version 4.6.1 is identical to that of version 4.6.0 so version 4.6.1 can be used as a direct replacement for version 4.6.0 without making any change to the application code.

Configuration settings are contained in AVIXSystemSettings.h.

The system settings file is installed with default values. It is essential that the settings you are using with version 4.6.0 are migrated to the settings file that comes with version 4.6.1 taking into account the changes mentioned before. This has to be done manually.

The AVIX library name contains its version number and therefore development environment project files must be changed to refer the version 4.6.1 library.

### ***Migration procedure***

In accordance with the above, the following migration procedure is advised:

8. Install the 4.6.1 distribution to a new location in your file system
9. Copy the configuration parameters of the 4.6.0 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or a similar utility.
10. Replace all version 4.6.0 files in your development structure with the new version 4.6.1 files, maintaining the directory structure of version 4.6.1. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
11. Remove the version 4.6.0 library from your development environment project file.
12. Change the project settings to include the new version 4.6.1 library.
13. Change the project settings to refer the AVIX interface directory.

Before executing these steps you might want to make a backup of the version 4.6.0 files.

## **Build and development environment issues 4.6.1**

### **AVIX for Cortex-M3 (IAR EWARM 6.x)**

- none

### **AVIX for Cortex-M3 (KEIL MDK 4.x)**

- none

### **AVIX for PIC32MX (MPLAB8x-C32)**

None

### **AVIX for PIC24-dsPIC (MPLAB-C30)**

- none

---

### **IAR EWARM 6.x RTOS Viewer plug-in**

none

### **KEIL MDK 4.x RTOS Viewer plug-in**

none

### **MPLAB8x RTOS Viewer plug-in**

none

**This page is intentionally left blank**

## Summary 4.6.0

These are the release notes of AVIX version 4.6. The major reasons for release of this new version are the following:

### Support added for the IAR EWARM 6.x development environment:

Besides support for the KEIL MDK development environment, AVIX for Cortex-M3 is supported by the IAR EWARM 6.x development environment. AVIX for Cortex-M3 is distributed in the form of two binary libraries, one compatible with IAR EWARM 6.x and one compatible with KEIL MDK. Both libraries are distributed subject to the same license conditions as before so with a single AVIX for Cortex-M3 license it is allowed to create applications for all supported Cortex-M3 MCU's with either of the mentioned development environments.

### Support added for version 2.xx of the Microchip C32 compiler:

Microchip introduced a new version of the PIC32MX C32 compiler incorporating many new features. Libraries build with a previous version of this compiler are not compatible with this new version. To retain compatibility with the AVIX installed base but also allow using the new version 2.xx of the C32 compiler, starting with version 4.6, AVIX for PIC32 is distributed with two libraries. For more details see chapter 'AVIX for PIC32MX (MPLAB8x-C32)'

**Important notice:** Version 4.6 required a small number of API changes to be made. As a consequence it may be required to change your existing application code accordingly. These changes are caused by removing a number of dependencies on GNU extensions that were used in AVIX version 4.5 and earlier and that were not supported by all supported development environments. Starting with version 4.6, AVIX is fully C99 compatible and does no longer depend on GNU extensions. The applicable changes are summarized in section 'Changes 4.6.0'. The changes are explained in more detail in a separate section of these release notes (section 'Version 4.6.0 API Changes background').

## New Features 4.6.0

- none

## Changes 4.6.0

- **API Change: Macros AVIX\_TYPESAFE\_EQ and AVIX\_TYPESAFE\_NEQ**  
Constants used on the AVIX API are no longer based on the proprietary AVIX type safety mechanism but are changed to standard C defined constants. As a consequence, when using macros `AVIX_TYPESAFE_EQ` or `AVIX_TYPESAFE_NEQ` in application code these macros must be replaced by the standard C equality operator (`==`) or inequality operator (`!=`) respectively.

Note this only applies to using these macros with type safe constants. Comparing kernel object id's for being equal or not equal to each other still requires use of these macros. More details are found in section 'Version 4.6.0 API Changes'.

- **API Change: Functions `avixMsg_PutKernelObjectId<FromISR>`, `avixMsg_GetKernelObjectId`**  
These functions are used to put or get a kernel object id to or from a message. For passing a kernel object id to these functions use was made of transparent\_unions which is a GNU extension to the C language. Starting with version 4.6.0, transparent unions are no longer used requiring a different syntax to be used when calling these functions. More details are found in section 'Version 4.6.0 API Changes'.
- **API Change: Function `avixKernelObjectId_IsValid` removed from AVIX public API.**  
Starting with version 4.5.0, use of this function has been deprecated in favor of using macro `AVIX_OBJECT_ID_VALID`. Maintaining this function in version 4.6.0 would require an API change because this function depends on GNU extensions. For this reason it is decided to remove this function from the AVIX API starting with version 4.6.0.

**When using function `avixKernelObjectId_IsValid` in your application replace all occurrences of `avixKernelObjectId_IsValid` with `AVIX_OBJECT_ID_VALID`.**

## Known issues 4.6.0

- none

## Fixed issues 4.6.0

- **PR137: Thread waiting for pipe may inadvertently be made running**  
An issue with the pipe mechanism may lead to a thread inadvertently made running. This can be a thread that wants to write and is waiting for empty space in the pipe to become available or a thread waiting for data to become available. This issue is fixed in version 4.6.0 and pipe behavior is compliant with the documentation.

## Documentation changes 4.6.0

- All documentation is updated according the mentioned changes.

## How to migrate from version 4.5.0 to version 4.6.0

### Background

When installing AVIX, a directory structure is created as described in the applicable Porting Guide.

The Application Programming Interface of version 4.6.0 is slightly changed compared with the API of version 4.5.0 (See section Changes 4.6.0). For this reason version 4.6.0 cannot be used as a direct replacement for version 4.5.0. The applicable changes must be made according the description. When the changed functions are not used in your application, version 4.6.0 still can be used as a direct replacement for version 4.5.0

Configuration settings are contained in the system settings file AVIXSystemSettings.h.

The system settings file is installed with default values. It is essential that the settings you are using with version 4.5.0 are migrated to the settings file that comes with version 4.6.0. Migrating the 4.5.0 settings to the 4.6.0 settings file is a manual process. It is strongly advised to use a file comparison tool like BeyondCompare.

The AVIX library name contains its version number and therefore development environment project files must be changed to refer the version 4.6.0 library.

### Migration procedure

In accordance with the above, the following migration procedure is advised:

1. Install the 4.6.0 distribution to a new location in your file system
2. Copy the configuration parameters of the 4.5.0 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or a similar utility.
3. Replace all version 4.5.0 files in your development structure with the new version 4.6.0 files, maintaining the directory structure of version 4.6.0. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 4.5.0 library from your development environment project file.
5. Change the project settings to include the new version 4.6.0 library.
6. Change the project settings to refer the AVIX interface directory.

Before executing these steps you might want to make a backup of the version 4.5.0 files.

## Build and development environment issues 4.6.0

### AVIX for Cortex-M3 (IAR EWARM 6.x)

- [Support for the IAR EWARM development environment for Cortex-M3](#)  
Starting with version 4.6, AVIX for Cortex-M3 supports the IAR EWARM development environment.

### AVIX for Cortex-M3 (KEIL MDK 4.x)

- none

### AVIX for PIC32MX (MPLAB8x-C32)

- [Support for new version 2.xx of Microchip C32 compiler for PIC32MX](#)  
Microchip released a major new version of their C32 compiler for PIC32. Because of the major changes incorporated in this new version, the version 4.5.0 AVIX library is no longer compatible with this new compiler version. To retain compatibility with the AVIX installed base but also allow using the new C32 compiler, starting with version 4.6, AVIX for PIC32 is distributed with two libraries:
  - When using C32 version 1.x, use the AVIX library where the body of the library file name ends with (C32\_1.x).
  - When using C32 version 2.x, use the AVIX library where the body of the library file ends with (C32\_2.x)

Although using the 'wrong' library may build correctly and the application may even work, correct working is not guaranteed and application settings should be changed to the compatible AVIX library.

### AVIX for PIC24-dsPIC (MPLAB-C30)

- none

---

### IAR EWARM 6.x RTOS Viewer plug-in

- [AVIX RTOS Viewer for IAR EWARM development environment](#)  
With support for the IAR EWARM development environment, AVIX is provided with an RTOS viewer compatible with this development environment.

### KEIL MDK 4.x RTOS Viewer plug-in

- [New AVIX RTOS viewer plug-in for KEIL MDK development environment](#)  
Version 2.3 of the RTOS viewer plug-in that came with AVIX version 4.5.0 is not compatible with AVIX version 4.6.0. Advised is to install the new version 3.0 of the KEIL MDK RTOS Viewer Plug-in.

### MPLAB8x RTOS Viewer plug-in

- [New AVIX RTOS viewer plug-in for Microchip MPLAB8x development environment](#)  
Version 2.3 of the RTOS viewer plug-in that came with AVIX version 4.5.0 is not compatible with AVIX version 4.6.0. Advised is to install the new version 3.0 of the MPLAB™ RTOS Viewer Plug-in. This plug-in is compatible both with AVIX for PIC24-dsPIC and AVIX for PIC32MX. Selection of the correct AVIX version is done automatically by the plug-in and no manual settings are required.

## Version 4.6.0 API Changes background

- **Type safe comparison macros:**

AVIX offers two macros to make type safe comparisons. These macros allow programming errors to be detected compile time instead of runtime, leading to a more efficient development process. With AVIX version 4.5.0 and earlier, these macros were used for kernel object id's (thread id, mutex id and so on) and for constants passed to AVIX functions.

Starting with version 4.6.0, these macros are no longer used for constants but only for kernel object id's. For the latter use of these macro's is obligatory and no changes to application code are required.

For constants, when migrating to AVIX version 4.6.0 application changes may be required as described below:

Type safe comparison with AVIX version 4.5.0 and earlier:

```
tavixTimerType, Tt;

Tt = AVIX_TIMER_CYCLIC;

if (AVIX_TYPESAFE_NEQ(Tt, AVIX_TIMER_CYCLIC))
{
    Tt = AVIX_TIMER_CYCLIC;
}
else
{
    Tt = AVIX_TIMER_SINGLE_SHOT;
}
```

Constant comparison with AVIX version 4.6.0:

```
tavixTimerType, Tt;

Tt = AVIX_TIMER_CYCLIC;

if (Tt != AVIX_TIMER_CYCLIC)
{
    Tt = AVIX_TIMER_CYCLIC;
}
else
{
    Tt = AVIX_TIMER_SINGLE_SHOT;
}
```

So for constant comparison only, all occurrences of `AVIX_TYPESAFE_EQ` must be replaced by a standard 'C' == operator. All occurrences of `AVIX_TYPESAFE_NEQ` must be replaced by a standard 'C' != operator.

When compiling an existing application with AVIX version 4.6.0 the lines that need to be changed result in a compilation error.



*Make sure not to replace use of these macros in places where they are used to compare kernel object id's.*

- **Message functions to pass kernel object id's or kernel object id pointers.**

For passing a kernel object id in a message, AVIX offers functions `avixMsg_PutKernelObjectId` and `avixMsg_PutKernelObjectIdFromISR` to put a kernel object id in a message and `avixMsg_GetKernelObjectId` to get a kernel object id from a message.

AVIX version 4.5.0 and earlier used a `transparent_union` for the parameter type of these functions. Doing so, any type of kernel object id could be passed as a function parameter while other types would result in a compilation error. Usage of these functions was straightforward as illustrated below:

```
tavixMsg      msg;
tavixThreadId tid;
```

```
tavixMutexId mid;
...;
avixMsg_PutKernelObjectId(msg, tid);
...;
avixMsg_PutKernelObjectId(msg, mid);
...;
avixMsg_GetKernelObjectId(msg, &tid);
```

Transparent unions are a gnu specific language extension and AVIX version 4.6.0 no longer depends on these extensions. As a consequence for the mentioned functions, use must be made of designated initializers being a C99 feature of the 'C' language.

With AVIX version 4.6.0, the above code sample has to be converted to the code sample shown below:

```
tavixMsg      msg;
tavixThreadId tid;
tavixMutexId mid;
...;
avixMsg_PutKernelObjectId(msg, (tavixKernelObjectId){.thread=tid});
...;
avixMsg_PutKernelObjectId(msg, (tavixKernelObjectId){.mutex=mid});
...;
avixMsg_GetKernelObjectId(msg, (tavixKernelObjectId){.thread=&tid});
```

The difference with AVIX version 4.5.0 and earlier is that in the new situation besides the actual parameter value (tid, mid), also the type of the function parameter and the field in this type the actual parameter is used with have to be specified. When passing a thread id, the field name in the designated initialize construct is .thread. When passing a mutex id, the field name is .mutex.

For the other field names, refer the AVIX User and Reference guide. For more details on the use of C99 designated initializers see <http://gcc.gnu.org/onlinedocs/gcc/Designated-Inits.html>

Although this is a link to GNU documentation, the described feature is ISO C99 compliant and fully supported by all compilers AVIX works with.



**This page is intentionally left blank**

## Summary 4.5.0

The major new feature with version 4.5.0 is support for Exchange kernel objects. Exchange kernel objects allow advanced control over the application structure. Using AVIX Exchange objects, an application is constructed of highly autonomous software components. Because of using Exchange Objects as the major interface mechanism, these components are easy to test and reuse. Furthermore, extending an application with new functionality becomes much easier compared to an application not based on Exchange objects.

With version 4.5, a new document is added to the AVIX document collection, focusing on the new Exchange Mechanism. This document, AVIX\_ExchangeMechanism.pdf, describes the construction of an application consisting of four software components. The design steps to create this application are dealt with in much detail. Furthermore, this application is present as a fully operational demo in many downloads.

## New Features 4.5.0

- **Macro added to define kernel object id variables guaranteed to be initially 'invalid'**  
Added is macro AVIX\_OBJECT\_ID\_DEFINE to define kernel object id variables which are initialized to be invalid. Defining kernel object id's without use of this macro is possible but in this case their value is undefined which may be wrongly interpreted as being a valid kernel object id. Use of macro AVIX\_OBJECT\_ID\_DEFINE is strongly advised to be used for every kernel object id variable.
- **Macro added to test kernel object id variables for being valid**  
Added is macro AVIX\_OBJECT\_ID\_VALID to test a kernel object id for being valid. This macro replaces function avixKernelObjectId\_IsValid. Use of this function is deprecated and it will be removed in a future version.
- **CR132: Pipe read and write access possible from Deferred Interrupt Handlers (DIH's)**  
Pipes read and write operations are now allowed from Deferred Interrupt Handlers (DIH's). These operations are non-blocking and the result value of the operations should be inspected to see how many blocks actually have been processed. More details can be found in the User Guide.
- **CR133: Message queue handling extended with function to flush message queue**  
Using function avixMsgQThread\_Flush removes all messages currently present in a threads message queue and returns the memory to the message pool. This function allows full control over messages, especially in cooperation with exchange objects.
- **CR135: Timer handling extended with function to set period of a counting timer**  
Using function avixTimer\_SetPeriod the period of a (counting) timer can be changed. This function is especially useful to establish a seamless transition to a different period of an active cyclic timer.

## Changes 4.5.0

- **CR134: Allocating a message with a zero sized configured message pool asserts**  
Functions avixMsg\_Allocate and avixMsg\_AllocateFromISR assert when a zero sized message pool is configured through configuration parameter avix\_MSG\_POOL\_NR\_MESSAGES. Previous versions returned a NULL message id in this situation. Note that configuring a non-zero sized message pool does not guarantee a valid message id to be returned. Function avixMsg\_Allocate returns an invalid message id when used with a time-out and the time-out expires. Function avixMsg\_AllocateFromISR returns an invalid message id when the message pool is empty (all messages are allocated).
- **Pipe functions parameter type changed**  
For a number of pipe functions the parameter types have changed to better reflect their meaning. Especially pointers to buffers are changed from 'unsigned char\*' to 'void\*' for the read and 'const void\*' for the write functions respectively. This prevents buffer addresses passed as parameters from having to be cast to 'unsigned char\*'. This change has no impact on existing code.

## Known issues 4.5.0

- none

## Fixed issues 4.5.0

### **PR136: Fixed issue in scatter-load description files (AVIX for Cortex-M3 only)**

For linking, AVIX uses a scatter description file which also contains the definition of the initialization stack. The end of the stack was specified one word too large, both in the documentation (AVIX for Cortex-M3 Port Guide.pdf) as in the sample programs distributed with AVIX. This resulted in erroneous behavior of the debugger and is corrected. Check scatter load files in existing projects for this issue.

## Documentation changes 4.5.0

- **Problem solving chapter added to User Guide**

Added is a chapter containing a list of the most common issues that may exist when working with AVIX and their resolution.

## How to migrate from version 4.1.0 to version 4.5.0

### ***Background***

When installing AVIX, a directory structure is created as described in the applicable Porting Guide.

The Application Programming Interface of version 4.5.0 is identical to that of version 4.1.0 so version 4.5.0 can be used as a direct replacement for version 4.1.0 without making any change to the application code.

Configuration settings are contained in AVIXSystemSettings.h.

The system settings file is installed with default values. It is essential that the settings you are using with version 4.1.0 are migrated to the settings file that comes with version 4.5.0 taking into account the changes mentioned before. This has to be done manually.

The AVIX library name contains its version number and therefore development environment project files must be changed to refer the version 4.5.0 library.

### ***Migration procedure***

In accordance with the above, the following migration procedure is advised:

14. Install the 4.5.0 distribution to a new location in your file system
15. Copy the configuration parameters of the 4.1.0 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or a similar utility.
16. Replace all version 4.1.0 files in your development structure with the new version 4.5.0 files, maintaining the directory structure of version 4.5.0. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
17. Remove the version 4.1.0 library from your development environment project file.
18. Change the project settings to include the new version 4.5.0 library.
19. Change the project settings to refer the AVIX interface directory.

Before executing these steps you might want to make a backup of the version 4.1.0 files.

## Build and development environment issues 4.5.0

### ***AVIX for Cortex-M3 (KEIL MDK)***

- none

### ***AVIX for PIC32MX (MPLAB-C32)***

- none

### ***AVIX for PIC24-dsPIC (MPLAB-C30)***

- none

### ***MPLAB™ RTOS Viewer plug-in***

- Version 2.1 of the RTOS viewer plug-in that came with AVIX version 4.1.0 is not compatible with AVIX version 4.5.0. Advised is to install the new version 2.3 of the MPLAB™ RTOS Viewer Plug-in.

### ***KEIL MDK RTOS Viewer plug-in***

- Version 2.1 of the RTOS viewer plug-in that came with AVIX version 4.1.0 is not compatible with AVIX version 4.5.0. Advised is to install the new version 2.3 of the KEIL MDK RTOS Viewer Plug-in.

This page is intentionally left blank

## Summary 4.1.0

The major new feature with version 4.1.0 is support for extended memory. This is applicable to AVIX for PIC24-dsPIC and AVIX for Cortex-M3.

**AVIX for PIC24-dsPIC:** Some members of the PIC24F family contain Extended Data Space (EDS) memory. This allows for as much as 96KB of RAM to be available. Starting with version 4.1.0, AVIX allows memory pools to be created in this EDS RAM so all available RAM can be accessed using AVIX functions.

**AVIX for Cortex-M3:** Some Cortex-M3 controllers contain multiple banks of RAM. Earlier versions of AVIX only allowed the main bank to be used. Starting with version 4.1.0, allow (part of) a second bank of RAM to be used for AVIX memory pools.

**AVIX for PIC32MX:** Since PIC32MX controllers always contain a single area of RAM, this extension is not available for these controllers. AVIX has always allowed all RAM to be used and continues to do so with version 4.1.0.

For access to the additional RAM, extra functionality is implemented. For compatibility between the different ports, these functions are available for all ports, so also for AVIX for PIC32MX.

Besides this extension, new functions are offered for all ports for timer and error management.

## New Features 4.1.0

- **AVIX Error mechanism accessible for application code**  
A new function (`avixError_Throw`) is added to allow application code to use the same central error handling mechanism used by AVIX. Passed to this function is a numeric application specific error code. This code must be based on a predefined value, `AVIX_USER_ERROR_BASE`.  
To allow easy error checking in application code, two macro's are added, `AVIX_ASSERT` and `AVIX_ASSERT_ALWAYS`.
- **Timer mechanism extended with resume functionality.**  
New functions (`avixTimer_Resume` & `avixTimer_ResumeFromISR`) are added allowing a previously stopped timer to continue counting with the tick count it had when being stopped.
- **Timer mechanism extended with function to query the remaining ticks.**  
A new function (`avixTimer_GetRemainingTicks`) is added to allow a timer to be queried for the remaining number of ticks before it expires. This function works both for running and stopped timers.'
- **Extended memory support (AVIX for PIC24-dsPIC and AVIX for Cortex-M3 only)**  
For Microchip PIC24F controllers offering EDS RAM and Cortex-M3 controllers offering secondary RAM banks, AVIX memory pools can be created in the extended RAM space. For this a new function is added, `avixMemPool_CreateExt`. Please read the applicable port guide for details.
- **System timer override functionality added (AVIX for Cortex-M3 only)**  
By default AVIX for Cortex-M3 uses the Cortex-M3 SysTick timer for its central timing. This timer is stopped when the MCU enters the SLEEPDEEP power mode. For Cortex-M3 based MCU's offering low power counters, a mechanism is added to use a different counter for the central timing making it possible to continue timing functionality when in SLEEPDEEP mode. Details can be found in the AVIX for Cortex-M3 port guide.

## Changes 4.1.0

- **Change in macro name to test memory block id for being valid**  
AVIX offers a macro to test a memory block id for being valid. The name of this macro is changed from `AVIX_MEM_BLOCK_PTR_VALID` in `AVIX_MEM_BLOCK_ID_VALID`. The old name led to confusion regarding its purpose. The change better reflects the purpose of this macro. Please change existing code accordingly.

## Known issues 4.1.0

None

## Fixed issues 4.1.0

- **PR0131: Compilation fails when using timer predefined constants**  
When using one of the predefined constants `AVIX_TIMER_MAX_NR_TICKS` or `AVIX_TIMEOUT_WAIT_FOREVER`, compilation stops with an error reporting macro `BITS_PER_ELEMENT` is not found. This issue is caused by macro `BITS_PER_ELEMENT` not being available in one of the AVIX interface files. The issue has been solved by moving this macro from an AVIX internal header file to `AVIXGeneric.h`.  
  
*Macro `_BITS_PER_ELEMENT` is considered an AVIX internal macro and is not meant to be used in application code.*

## Documentation changes 4.1.0

- **Documentation updated with new functionality.**  
Documentation is updated with all new functionality and changes mentioned before.

## How to migrate from version 4.0.0 to version 4.1.0

### Background

When installing AVIX, a directory structure is created as described in the applicable Porting Guide.

The Application Programming Interface of version 4.1.0 is identical to that of version 4.0.0 so version 4.1.0 can be used as a direct replacement for version 4.0.0 without making any change to the application code.

Configuration settings are contained in `AVIXSystemSettings.h`.

The system settings file is installed with default values. It is essential that the settings you are using with version 4.0.0 are migrated to the settings file that comes with version 4.1.0 taking into account the changes mentioned before. This has to be done manually.

The AVIX library name contains its version number and therefore MPLAB projects must be changed to refer the version 4.1.0 library.

### Migration procedure

In accordance with the above, the following migration procedure is advised:

1. Install the 4.1.0 distribution to a new location in your file system
2. Copy the configuration parameters of the 4.0.0 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or a similar utility.
3. Replace all version 4.0.0 files in your development structure with the new version 4.1.0 files, maintaining the directory structure of version 4.1.0. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 4.0.0 library from your MPLAB project.
5. Change the project settings to include the new version 4.1.0 library.
6. Change the project settings to refer the AVIX interface directory which is a level deeper in this release.

Before executing these steps you might want to make a backup of the version 4.0.0 files.

## Build and development environment issues 4.1.0

### AVIX for Cortex-M3

- none

### AVIX for PIC32MX

- none

### AVIX for PIC24-dsPIC

- **MPLAB version 8.56**  
The simulator present with this version of MPLAB does have an issue leading to a wrong simulation of a bit test instruction combined with auto-increment addressing. Since this instruction is used in AVIX, the simulator produces incorrect results. AVIX-RT is working with Microchip to fix this issue.

### MPLAB™ RTOS Viewer plug-in

- Version 2.0 of the RTOS viewer plug-in that came with AVIX version 4.0.0 is not compatible with AVIX version 4.1.0. Advised is to install the new version 2.1 of the MPLAB™ RTOS Viewer Plug-in.

Version 2.1 of the RTOS viewer plug-in does contain the following changes:

- For memory pools, a field is added indicating whether the memory pools is created in extended memory. This is only applicable for controllers belonging to the PIC24F family having EDS memory.
- For timers that are not running the tick field used to show [n.a]. Version 2.1 shows the content of the tick field as this contains the remaining number of ticks for a stopped timer.

### KEIL MDK RTOS Viewer plug-in

- Version 2.0 of the RTOS viewer plug-in that came with AVIX version 4.0.0 is not compatible with AVIX version 4.1.0. Advised is to install the new version 2.1 of the KEIL MDK RTOS Viewer Plug-in.

Version 2.1 of the RTOS viewer plug-in does contain the following changes:

- For memory pools, a field is added indicating whether the memory pools is created in extended memory. This is only applicable for controllers having multiple banks of RAM.
  - For timers that are not running the tick field used to show [n.a]. Version 2.1 shows the content of the tick field as this contains the remaining number of ticks for a stopped timer.
- The RTOS viewer plug-in for the KEIL MDK development environment is compatible with KEIL MDK version 4.10 only. KEIL published a newer version of its MDK (4.12) but because of changes to the internal interface the AVIX RTOS Viewer Plug-In does not work with this version. Support for KEIL MDK version 4.12 will be present in a forthcoming version of the Plug-In.



This page is intentionally left blank

## Summary 4.0.0

Version 4.0.0 is the successor to version 3.6.0. Version 3.6.0 has not been distributed to existing customers. The main reason for introduction of version 3.6.0 had to do with changes to the setup utility as required by one of our distributors.

**When you are not using version 3.6.0:** Your most current version is 3.5.2. Please read the release notes of version 4.0.0 and the release notes of version 3.6.0.

**When you are using version 3.6.0:** Please read the release notes of version 4.0.0.

The reason for publishing version 4.0.0 is that starting with this version, AVIX is available in a port for a large number of ARM Cortex-M3 based microcontrollers, besides the existing ports for Microchip PIC32MX and PIC24-dsPIC controllers. This implied a large number of changes, not all of them visible to the user.

Important is that although AVIX is evolving to an RTOS deployable on a broader microcontroller portfolio, still the specific features of the individual microcontroller families are used and this usage is even refined further. As a result performance of the Microchip ports have again improved.

To better identify the different AVIX ports available, starting with version 4.0.0, product naming has changed. The different ports are named as follows:

**AVIX for PIC24-dsPIC** Formerly named AVIX-16. This port targets all controllers belonging to the Microchip PIC24F, PIC24H, dsPIC30F and dsPIC33F families fulfilling the AVIX memory requirement. Not supported are the PIC24F controllers implementing Extended Data Space because of changes to the internal architecture of these controllers. PIC24F controllers with Extended Data Space will be supported by a forthcoming AVIX release.

**AVIX for PIC32MX** Formerly named AVIX-32. This port targets all controllers belonging to the Microchip PIC32MX family of microcontrollers fulfilling the AVIX memory requirement.

**AVIX for Cortex-M3** New product. This port targets all controller families based on the ARM Cortex-M3 architecture and supported by the CMSIS software standard. Currently the following controller families are supported:

Atmel	AT91SAM3U
Atmel	AT91SAM3S
Energy Micro	EFM32
NXP	LPC17x
ST Micro	STM32F
Texas Instruments	LM3S
Toshiba	TMPM330

## New Features 4.0.0

- **CR131: Added predefined symbols for AVIX version and target architecture**  
AVIX is extended with a number of predefined symbols available to the application. Using these symbols in the application it is easier to construct code that (after recompilation) works without a change on multiple hardware platforms and/or with different versions of AVIX. These symbols are documented in section 7.5 of the AVIX User and Reference guide.

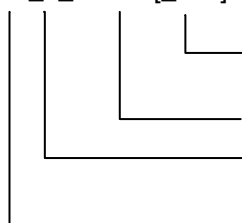
## Improvements 4.0.0

- **Increased performance**  
Because of internal changes, performance of a number of AVIX functions has increased substantially.

## Changes 4.0.0

- Name of configuration parameter file changed**  
 The parameter settings file name used to be AVIX16SystemSettings.h for AVIX for PIC24-dsPIC and AVIX32SystemSettings.h for AVIX for PIC32MX. From this release onward, the difference in naming is removed and for each port the configuration parameter file is named AVIXSystemSettings.h.
- AVIX for PIC32MX only, Interrupt and timer configuration changed**  
 AVIX uses a configurable interrupt and timer. For the selected interrupt and timer the corresponding IF, IE and IP register had to be specified in AVIXSystemSettings.h. Specifying these registers is no longer required. It is now sufficient to only specify the interrupt (CS0 or CS1) and the hardware timer (1..5). The register settings are no longer present in AVIXSystemSettings.h.
- Changed directory structure**  
 AVIX is installed in a number of directories (Cnfg, Doc, Interface, etc.) These directories used to be created in the directory specified in the setup program. To isolate AVIX from other (application) files present here, a directory level is added. In the directory specified in the setup program, now only a single directory is created named \_AVIX. It is in this directory the other subdirectories are created. Details can be found in the "AVIX Port Guide of the port you are using.
- Configuration parameter name avix\_TMR\_CLK\_SRC\_EXT changed**  
 The name of configuration parameter avix\_TMR\_CLK\_SRC\_EXT has changed to avix\_TIMER\_CLK\_SECONDARY. The semantics of this parameter have remained the same. Reason for this change is that \_EXT does no longer reflect the purpose of this parameter since not all Cortex-M3 controllers support an external clock as a secondary input for the main system timer.
- Changed library names**  
 Because of the changed naming of the different ports, the naming convention of the AVIX library has changed in order to make this name better identify the type of library. The new naming convention is as follows:

AVIX\_port\_b\_cccccc[\_cntr].a



Example:

**AVIX\_PIC24-dsPIC\_B\_040000.a**: Basic library for AVIX for PIC24 and dsPIC

## Known issues 4.0.0

None

## Fixed issues 4.0.0

- PR120: Resume does not exit critical section**  
 When resuming a suspended thread and the priority of the resumed thread is lower than the thread executing the resume operation, AVIX leaves its internal critical section active resulting in thread scheduling coming to a halt.

***This issue occurs in release 3.6.0 only and is considered critical. Therefore it is strongly suggested to use release 4.0.0 to replace release 3.6.0.***

- **PR126: AVIX for PIC24-dsPIC only, Linking AVIX for PIC24-dsPIC based application throws memory model error.**

AVIX for PIC24-dsPIC is built using the small memory model to get the smallest and fastest code. All AVIX code is contained in the installed library and consists of a number of object files. Because of the small memory model, all AVIX files may not be further apart from each other than 32KB when linked together. Under certain circumstances, this is however not the case. As a result, certain functions are unable to call each other as reported by a link error. To solve this, all AVIX code is placed in a named section (`._avix_code_`). This allows all AVIX code to be grouped within a single 32KB area in the generated image. For details on this topic and instructions how to add the required information to the linker file, see the 'AVIX for PIC24-dsPIC Port Guide'.

## Documentation changes 4.0.0

- **Separate Port Guide document**  
The User and Reference Guide used to contain an appendix with AVIX for PIC24-dsPIC and AVIX for PIC32MX specific information. This appendix is moved to a separate document, one for each of the three different ports that exist.
- **PR098: AVIX for PIC32MX only: Shadow Register interrupt macro only usable with interrupt priority 7**  
Decided is not to change this. Reason is that an implementation explicitly expecting the ISR to operate at priority 7 is substantial faster than a generic mechanism allowing all possible priorities to use the shadow registers. The restriction is added to the documentation and thereby considered fixed.
- **PR125: Event group polling documented incomplete**  
Event groups allow to be polled. Documentation states polling is accomplished by specifying value `AVIX_EF_NONE` for parameter `desiredEventFlags`. This is incomplete. Polling an event group is accomplished by specifying value `AVIX_EF_NONE` for parameter `desiredEventFlags` **and** value `AVIX_EVENT_GROUP_ALL` for parameter `combine`.

## How to migrate from version 3.6.0 to version 4.0.0

### Background

When installing AVIX, a directory structure is created as described in the applicable Porting Guide. The directory structure of version 4.0.0 has changed as described in section Changes 4.0.0

The Application Programming Interface of version 4.0.0 is identical to that of version 3.6.0 so version 4.0.0 can be used as a direct replacement for version 3.6.0 without making any change to the application code.

Configuration settings are contained in `AVIXSystemSettings.h`.

The system settings file is installed with default values. It is essential that the settings you are using with version 3.6.0 are migrated to the settings file that comes with version 4.0.0 taking into account the changes mentioned before. This has to be done manually.

The AVIX library name contains its version number and therefore MPLAB projects must be changed to refer the version 4.0.0 library.

### Migration procedure

In accordance with the above, the following migration procedure is advised:

1. Install the 4.0.0 distribution to a new location in your file system
2. Copy the configuration parameters of the 3.6.0 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or a similar utility.

3. Replace all version 3.6.0 files in your development structure with the new version 4.0.0 files, maintaining the new directory structure of version 4.0.0. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 3.6.0 library from your MPLAB project.
5. Change the project settings to include the new version 4.0.0 library.
6. Change the project settings to refer the AVIX interface directory which is a level deeper in this release.

Before executing these steps you might want to make a backup of the version 3.6.0 files.

## Build and development environment issues 4.0.0

### AVIX for Cortex-M3

- none

### AVIX for PIC32MX

- none

### AVIX for PIC24-dsPIC

- **MPLAB version 8.56**  
The simulator present with this version of MPLAB does have an issue leading to a wrong simulation of a bit test instruction combined with auto-increment addressing. Since this instruction is used in AVIX, the simulator produces incorrect results. AVIX-RT is working with Microchip to fix this issue.

### MPLAB™ RTOS Viewer plug-in

- Version 1.6 of the RTOS viewer plug-in that came with AVIX version 3.6.0 is not compatible with AVIX version 4.0.0. Advised is to install the new version 2.0 of the MPLAB™ RTOS Viewer Plug-in.

Changes are made to the Plug-In to make a better distinction between application data and other data. For this reason, an About... section is added. Details can be found in the user manual installed with the plug-in.

### KEIL MDK RTOS Viewer plug-in

- The RTOS viewer plug-in for the KEIL MDK development environment is compatible with KEIL MDK version 4.10 only. KEIL published a newer version (4.12) but because of changes to the internal interface the AVIX RTOS Viewer Plug-In does not work with this version. Support for KEIL MDK version 4.12 will be present in a forthcoming version of the Plug-In.

This page is intentionally left blank

## Summary 3.6.0

This section contains release notes for AVIX release 3.6.0. Release 3.6.0 is the successor to AVIX Release 3.5.2. Release 3.6.0 is a maintenance release implying no new features are offered.

## New Features 3.6.0

- None

## Improvements 3.6.0

- **CR117: Added port definition for tracing purposes**  
For tracing purposes a port definition (`AVIX_TRACE_PORT_H`) is added which can be used with functions `avixThread_SetTracePort` and `avixThread_SetTracePortAndResume`. Note this definition can only be used for parts having a port H.

## Changes 3.6.0

- **CR116: Tracing functionality interface changed from macro to functions**  
Starting with version 3.6.0, tracing functions `avixThread_SetTracePort` and `avixThread_SetTracePortAndResume` are implemented as regular C functions. In versions prior to version 3.6.0, this functionality was implemented in the form of macros.  
*This change has no influence on application code.*
- **CR118: Configuration file type changed from assembly to C**  
Starting with version 3.6.0, the type of the configuration file compiled in the user project is changed from assembly to C. For versions prior to version 3.6.0, the configuration file was an assembly file named `AVIXConfig.S`. Starting with version 3.6.0, the configuration file is a C source file named `AVIXConfig.c`. This file is installed as part of the AVIX setup utility. When upgrading an existing project to version 3.6.0 care must be taken to change the project definition to include the new C file.
- **CR119: Added header file for definitions shared between configuration and library**  
The configuration file `AVIXConfig.c` belongs to AVIX and is compiled in the user project. This file contains certain constant values also used in the AVIX library. A file is added to the AVIX setup containing these constant definitions. This file is named `AVIXSharedDefs.h`. `AVIXSharedDevs.h` is used from AVIX interface files and does not need to be included by user code.  
*This change has no influence on application code.*

## Known issues 3.6.0

- **PR098: AVIX-32 Shadow Register interrupt macro only usable with interrupt priority 7**  
This issue is described in more detail in the section containing the release notes of version 3.5. This issue will be fixed in a forthcoming release of AVIX-32.

## Fixed issues 3.6.0

- **PR088: Potentially unused memory block**  
Under certain very exotic conditions, it is possible a thread is waiting for a memory block to become available while the pool the thread is waiting on contains a free block. This issue is fixed such that a thread will only block on a memory pool in case no memory block is available which is according to the documentation.
- **PR115: Function `avixDIH_Queue` uses near call (AVIX-16 only)**  
Function `avixDIH_Queue` defined in `AVIXGeneric.h` is based on inline assembly. The function call in this implementation is changed from a near call to a far call. This issue prevented the function to operate correctly in very large user applications where the call destination was out of reach.

- **PR114: DSP context saving (AVIX-16 only)**  
File AVIXGeneric.h contains macro based code to conditionally save the controller DSP addressing mode registers when using DSP functionality on a dsPIC part. This code contained an invalid condition preventing the DSP context save from working correctly. This issue is fixed from version 3.6.0 onward.

## Documentation changes 3.6.0

- None

## How to migrate from version 3.5.2 to version 3.6.0

### ***Background***

When installing AVIX, a directory structure is created as described in the AVIX User Manual. The directory structure of version 3.6.0 is the same as that of version 3.5.2. The content of the directories created by version 3.6.0 are the same as that of version 3.5.2 with exception of issues described above.

The Application Programming Interface of version 3.6.0 is identical to that of version 3.5.2 so version 3.6.0 can be used as a drop-in replacement for version 3.5.2 without making any change to the application code.

Configuration settings are contained in AVIXxxSystemSettings.h (xx is either 16 or 32 depending on the distribution you are using).

The system settings file is installed with default values. It is essential that the settings you are using with version 3.5.2 are migrated to the settings file that comes with version 3.6.0. This has to be done manually.

The AVIX library name contains its version number and therefore MPLAB projects must be changed to refer the version 3.6.0 library.

### ***Migration procedure***

In accordance with the above, the following migration procedure is advised:

1. Install the 3.6.0 distribution to a new location in your file system
2. Copy the configuration settings of the 3.5.2 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or a similar utility.
3. Replace all version 3.5.2 files in your development structure with the new version 3.6.0 files. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 3.5.2 library from your MPLAB project.
5. Change the project settings to include the new version 3.6.0 library.
6. Remove file AVIXConfig.S from your MPLAB project.
7. Add the new configuration file AVIXConfig.c to your MPLAB project.

Version 1.5 of the RTOS viewer plug-in is not compatible with AVIX version 3.6.0. Version 3.6.0 comes with a new version 1.6 of the RTOS viewer plug-in. This new plug-in can be installed over the existing version of the plug-in.

Before executing these steps you might want to make a backup of the version 3.5.2 files.



## Build and development environment issues 3.6.0

### AVIX-16

- none

### AVIX-32

- none

### ***RTOS Viewer plug-in***

- Version 1.5 of the RTOS viewer plug-in is not compatible with AVIX version 3.6.0. Version 3.6.0 comes with a new version 1.6 of the RTOS viewer plug-in. This new plug-in can be installed over the existing version of the plug-in.

This page is intentionally left blank

## Summary 3.5.2

This section contains release notes for AVIX release 3.5.2. Release 3.5.2 is the successor to AVIX Release 3.5. Release 3.5.2 is a maintenance release implying no new features are offered. Release 3.5.2 contains fixes for a number of issues the details of which you find below.

***Both issues occur in release 3.5 only and are considered critical. Therefore it is strongly suggested to stop using AVIX version 3.5 and replace it with version 3.5.2.***

## New Features 3.5.2

- None

## Improvements 3.5.2

- None

## Changes 3.5.2

- None

## Known issues 3.5.2

- **PR098: AVIX-32 Shadow Register interrupt macro only usable with interrupt priority 7**  
This issue is described in more detail in the section containing the release notes of version 3.5. This issue will be fixed in a forthcoming release of AVIX-32.

## Fixed issues 3.5.2

- **PR107: Resumed ready thread not scheduled**  
Under certain very exotic conditions it might occur that a suspended thread that is being resumed using function `avixThread_ResumeFromISR`, while having the highest priority among all ready threads, will not start running. After the scheduler has been activated because of other application activity this thread will again participate in the scheduling.

***This issue occurs in release 3.5 only and is considered critical. Therefore it is strongly suggested to use release 3.5.2 to replace earlier releases of AVIX.***

- **PR108: Ready threads not scheduled**  
Under certain very exotic conditions it might occur that threads, although ready, will not be scheduled for a long time. This issue only occurs when using functions `avixThread_ResumeFromISR` or `avixSemaphore_UnlockFromISR`.

Only when another thread at the same priority has run again, such a thread will again participate in the scheduling.

***This issue occurs in release 3.5 only and is considered critical. Therefore it is strongly suggested to use release 3.5.2 to replace earlier releases of AVIX.***

## Documentation changes 3.5.2

- None

## How to migrate from version 3.5 to version 3.5.2

### **Background**

When installing AVIX, a directory structure is created as described in the AVIX User Manual. The directory structure of version 3.5.2 is the same as that of version 3.5. The content of the created directories with version 3.5.2 are the same as that of version 3.5.

The Application Programming Interface of version 3.5.2 is identical to that of version 3.5 so version 3.5.2 can be used as a drop-in replacement for version 3.5 without making any change to the application code.

Configuration settings are contained in AVIXxxSystemSettings.h (xx is either 16 or 32 depending on the distribution you are using).

The configuration file is installed with default settings. It is essential that the settings you are using with version 3.5 are migrated to the settings file that comes with version 3.5.2. This has to be done manually.

The AVIX library name contains its version number and therefore MPLAB projects must be changed to refer the version 3.5.2 library.

### **Migration procedure**

In accordance with the above, the following migration procedure is advised:

1. Install the 3.5.2 distribution to a new location in your file system
2. Copy the configuration settings of the 3.5 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or the like.
3. Replace all version 3.5 files in your development structure with the new version 3.5.2 files. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 3.5 library from your MPLAB project.
5. Change the project settings to include the new version 3.5.2 library.

Version 1.5 of the RTOS Viewer plug-in is compatible both with version 3.5.2 and version 3.5.

Before executing these steps you might want to make a backup of the version 3.5 files.

## Build and development environment issues 3.5.2

### **AVIX-16**

- none

### **AVIX-32**

- none

### **RTOS Viewer plug-in**

- The AVIX RTOS viewer plug-in version 1.5 is compatible both with version 3.5 and version 3.5.2.

This page is intentionally left blank

## Summary 3.5

This section contains release notes for AVIX release 3.5. This release is the successor to commercial release 3.1. In between two maintenance releases have been made available either on customer request or to fix customer reported issues. These maintenance releases are 3.2.0 and 3.2.0a.

*The release notes of maintenance releases 3.2.0 and 3.2.0a are copied to the release notes of this section so by reading the release notes of version 3.5, coming from the previous commercial release 3.1, all changes and additions can be found in a single place.*

### Reasons for publication of release 3.5 are:

- Introduction of power management functionality (earlier made available in maintenance release 3.2.0). This functionality allows easy deployment of the controllers IDLE and SLEEP modes in order to reduce energy consumption.
- Addition of a number of functions that may be called directly from an Interrupt Service Routine.
- Internal modifications as preparation to port AVIX to other hardware platforms. These modifications have lead to a significant performance increase.

## New Features 3.5

- **CR085: Power Management Functionality**

**(partly earlier made available in maintenance release 3.2.0)**

From this release onward, AVIX is extended with functionality to exploit the microcontroller's energy saving capabilities. For this purpose three functions and a number of symbols are added to the AVIX API.

- `avixPower_SetMode`
- `avixPower_SetModeFromISR` (new in this release)
- `avixPower_GetMode`
- `avixPower_GetModeFromISR` (new in this release)
- `avixPower_SetCallback`

The prototypes of these functions and the related definitions are present in interface file `avixPower.h` which is installed in the AVIX Interface subdirectory. This interface file is automatically included when including global definition file `AVIX.h`.

To support power management the AVIX configuration file (`AVIXxxSystemSettings.h`) contains an additional parameter `avix_TIMER_CLK_SOURCE` to allow an external oscillator being used as the source for the AVIX system timer.

Finally, a symbol is added `AVIX_SYS_CLOCK_ACTUAL_PERIOD` defining the actual system timer period in microseconds whose value may differ slightly from the configured system timer period because of oscillator resolution. This definition is present in header file `avixTimer.h`. Note the value of this symbol is also show in the system section of version 1.5 of the AVIX RTOS viewer plug-in.

For details on all power management related functions and symbols, please refer the AVIX User Guide section 6.2.6. Also read the controller specific section in the user guide for additional information.

- **CR061: Added functions to obtain the size of a memory block through its id**

Added are two memory pool/block related functions:

- `avixMemPool_GetSizeBlock`
- `avixMemPool_GetSizeBlockFromISR`

These functions allow the size of a memory block to be obtained through a memory block id. These functions are especially useful when using memory blocks to exchange information between threads or threads and ISR's. One thread allocates a memory block and sends its id to another thread. Since the receiving thread does not know from which pool the block is allocated, it does not know the effective size of the memory block. By using these new functions the receiving thread has a means to obtain the size of the memory block before writing data to it.

- **CR100: Added timer functions to be used by Interrupt Service Routines**

Added are two timer related functions:

- avixTimer\_StartFromISR
- avixTimer\_StopFromISR

These functions allow a timer to be started or stopped directly by an Interrupt Service Routine. This prevents the use of an explicit Deferred Interrupt Handler leading to smaller and more efficient application code.

## Improvements 3.5

- **Performance increase**

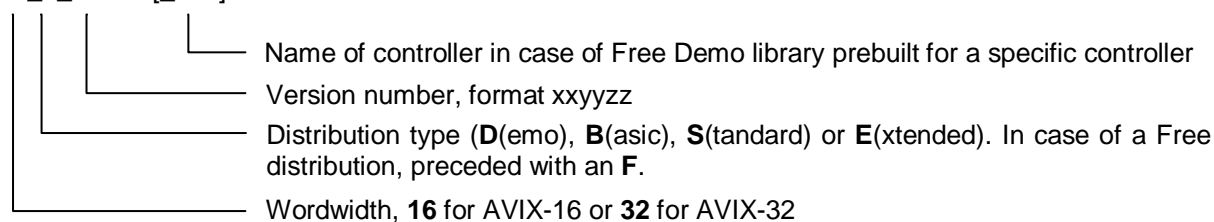
In preparation for porting AVIX to other hardware platforms, its internal scheduling structure is changed leading to a performance increase. Tested against the ThreadMetric test suite, AVIX is the fastest RTOS for PIC24, dsPIC and PIC32MX. Details can be found on the AVIX-RT website ([www.avix-rt.com](http://www.avix-rt.com)).

## Changes 3.5

- **Changed library names**

Starting with version 3.5.0, the naming convention of the AVIX library has changed in order to make this name better identify the type of library. The new naming convention is as follows:

AVIX-aa\_b\_cccccc[\_cntr].a



Examples:

**AVIX-16\_B\_030500.a**: Basic library for AVIX for PIC24 and dsPIC

**AVIX-32\_FD\_030500\_32MX360F512L.a**: Free demo for AVIX for PIC32MX, prebuilt for the PIC32MX360F512L

- **Thread Activation Tracing controlled by configuration setting**

Up to version 3.5, when using Thread Activation Tracing, the application had to be build with a constant `avix_TRACE`. This constant is no longer used and replaced by a configuration parameter.

Depending on the port this configuration parameter is present in `AVIX16SystemSettings.h` for AVIX-16 or in `AVIX32SystemSettings.h` for AVIX-32.

The name of this new parameter is `avix_TRACING`. This new configuration parameter allows for the tracing code not to be executed, leading to an increase in performance. The parameter can have one of three possible values:

0: Tracing code is executed but no trace ports are asserted. This mode is compatible with earlier versions compiled without the now obsolete `avix_TRACE` constant.

1: Tracing code is executed and trace ports are asserted. This mode is compatible with earlier versions compiled with the now obsolete `avix_TRACE` constant.

2: Tracing code is not executed at all leading to a significant performance increase in the scheduling.

More details can be found in the User Manual.

## Known issues 3.5

- **PR098: AVIX-32 Shadow Register interrupt macro only usable with interrupt priority 7**  
With the release of C32 version 1.10B, use of the PIC32MX shadow registers is no longer limited to interrupt priority 7. Macro `avixDeclareISRShadow` for AVIX-32 however does assume it is used with a priority 7 interrupt handler. Make sure when using one of the new PIC32MX parts (PIC32MX5/6/7) and using shadow register based interrupt handlers to use these with priority 7 when using the AVIX interrupt stack mechanism based on `avixDeclareISRShadow`.

When using shadow register based interrupt handlers at priorities different than 7 use the compiler built in Interrupt Service Routine declaration mechanism. This issue is not considered critical since:

- RAM load will hardly increase because it concerns the shadow register based interrupt handlers which have low RAM usage anyway.
- The parts it concerns are equipped with a much larger amount of RAM

This issue will be fixed in a forthcoming release of AVIX-32.

## Fixed issues 3.5

- **PR072: Thread Activation Tracing of idle thread**  
When using thread activation tracing, under certain conditions, the idle thread could be traced as being active together with another thread. In no way does this issue influence functionality.
- **PR089: AVIX-32 internal stack alignment issue (earlier made available in maintenance release 3.2.0)**  
Because of a Microchip documentation issue, AVIX-32 is hurt by an internal stack alignment issue. Under rare conditions, especially when using the C32 version 1.05 floating point libraries with double values this might lead to stack corruption. AVIX-32 is changed such that the stack is aligned correctly under all circumstances.

***This issue is considered critical and therefore it is strongly suggested to use release 3.5 to replace earlier commercial releases of AVIX-32.***

- **PR086: User guide inadvertently states `avixDIH_Queue` is allowed to be called from a thread. (earlier made available in maintenance release 3.2.0)**  
Function `avixDIH_Queue` is used to allow an Interrupt Service Routine to communicate with a thread and will be called from Interrupt Service Routines for this purpose.

The user documentation inadvertently states this function is also allowed to be called from a thread. This is however wrong and might lead to corruption of the DIH queue content resulting in system failure. Although this only happens under very rare conditions, `avixDIH_Queue` should not be called from a thread. Only call this function from Interrupt Service Routines.

Since the sole purpose of `avixDIH_Queue` is for Interrupt Service Routine-thread integration this issue is considered a documentation issue so the AVIX User Guide is updated according the above description.

- **PR092: AVIX-32 variables can become misaligned when using non-integer global variables. (earlier made available in release 3.2.0a)**  
When using non-integer sized global variables in your application, depending on the link order of the different application source modules, certain AVIX-32 internal integer sized variables can become misaligned. (global variables can be regular global variables or static variables both at file or function level).

As a result these variables are not allocated addresses being a multiple of four bytes which is a requirement of the underlying 32bit MIPS architecture. This results in the application to crash.

AVIX-32 is fixed such that under all circumstances the AVIX-32 internal integer sized variables are guaranteed to be allocated on a four byte boundary.



***This issue is considered critical and therefore it is strongly suggested to use release 3.5 to replace earlier commercial releases of AVIX-32.***

- **PR096: AVIX-32 User code executed while a context switch is about to occur**  
Under certain very specific conditions a context switch activated by an AVIX-32 function might be delayed for a few cycles causing the context switch to take place while user code is already executing. When the application manipulates global variables immediately following an AVIX-32 function call this might lead to incorrect values of these global variables.

## Documentation changes 3.5

- Documentation is changed according new features mentioned in these Release Notes.

## How to migrate from version 3.1.0 to version 3.5

### ***Background***

When installing AVIX, a directory structure is created as described in the AVIX User Manual. The directory structure of version 3.5.0 is the same as that of version 3.1.0. The content of the created directories with version 3.5.0 are the same as that of version 3.1.0 with the addition of two header files:

- AVIXPower.h: Definition file for new Power Management functions.
- AVIXPortDef.h: Definition file for preparation of porting AVIX to other hardware platforms.

Both files do not need to be included explicitly since this is taken care of when using AVIX.h.

The Application Programming Interface of version 3.5.0 is identical to that of version 3.1.0 so version 3.5.0 can be used as a drop-in replacement for version 3.1.0 without making any change to the application code.

*Note that version 3.5.0 does contain additional functions which are described above.*

Configuration settings are contained in AVIXxxSystemSettings.h (xx is either 16 or 32 depending on the distribution you are using).

The configuration file is installed with default settings. It is essential that the settings you are using with version 3.1.0 are migrated to the settings file that comes with version 3.5.0. This has to be done manually.

Note that starting with release 3.5.0 the configuration header file contains two additional parameters; make sure to enter these in the configuration file you are using with your application.

- `avix_TIMER_CLK_SOURCE_EXT`: This setting allows the external 32,768Hz crystal to be used for clocking the AVIX system timer in case certain Power Management functionality is used. More details can be found in the User Guide.
- `avix_TRACING`: Besides enabling and disabling Thread Activation Tracing, through this parameter it is also possible to entirely prevent tracing code from being executed leading to faster scheduling.

The AVIX library name contains its version number and therefore MPLAB projects must be changed to refer the version 3.5.0 library.

## ***Migration procedure***

In accordance with the above, the following migration procedure is advised:

1. Install the 3.5.0 distribution to a new location in your file system
2. Copy the configuration settings of the 3.1.0 version you are using to the newly installed configuration file. Preferably this is done with a file comparison tool like BeyondCompare or the like.
3. Replace all version 3.1.0 files in your development structure with the new version 3.5.0 files. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 3.1.0 library from your MPLAB project.
5. Change the project settings to include the new version 3.5.0 library. Note that starting with version 3.5.0, the naming convention of the library has changed according to the above description.
6. Install the new version 1.5 of the RTOS Viewer plug-in.

Before executing these steps you might want to make a backup of the version 3.1.0 files.

## **Build and development environment issues 3.5.0**

### **AVIX-16**

- none

### **AVIX-32**

- none

## ***RTOS Viewer plug-in***

- The AVIX RTOS viewer plug-in is provided in a new version 1.5.

This version is compatible with AVIX version 3.5.0 only. The previous version of the RTOS viewer plug-in (1.3) is not compatible with AVIX version 3.5.0. Make sure when using AVIX version 3.5.0 to also install the new RTOS viewer plug-in.

The functionality of version 1.5 of the RTOS viewer plug-in is identical to that of version 1.3 with the addition of a system parameter showing the actual system timer period in microseconds. This period can differ from the configured period because of the resolution of the clock when using the external 32,768Hz clock for the system timer.

This page is intentionally left blank

## Summary 3.2.0a

This section contains release notes for AVIX version 3.2.0a which is the successor to version 3.2.0.

Version 3.2.0a is a maintenance release containing fixes for issues detected with the previous version. Version 3.2.0a is applicable to AVIX-32 only. For AVIX-16 the most recent version is 3.2.0.

## New Features 3.2.0a

- None.

## Improvements 3.2.0a

- None.

## Changes 3.2.0a

- None.

## Known issues 3.2.0a

- None.

## Fixed issues 3.2.0a

- **PR092: AVIX-32 variables can become misaligned when using non-integer global variables.**  
When using non-integer sized global variables in your application, depending on the link order of the different application source modules, certain AVIX-32 internal integer sized variables can become misaligned. (global variables can be regular global variables or static variables both at file or function level).

As a result these variables are not allocated addresses being a multiple of four bytes which is a requirement of the underlying 32bit MIPS architecture. This results in the application to crash.

AVIX-32 is fixed such that under all circumstances the AVIX-32 internal integer sized variables are guaranteed to be allocated on a four byte boundary.

***This issue is considered critical and therefore it is strongly suggested to use this maintenance release as a replacement for previous versions.***

## Documentation changes 3.2.0a

- None.

## How to migrate from version 3.2.0 to version 3.2.0a

- See version 3.2.0.  
***Make sure to apply all files contained in this release and do not manually mix files of this version with other versions since this will lead to failure.***

## Build and development environment issues 3.2.0a

- See version 3.2.0.

This page is intentionally left blank

## Summary 3.2.0

This section contains release notes for AVIX version 3.2.0 which is the successor to version 3.1.0. This is a minor upgrade essentially for adding power management capabilities to AVIX.

Version 3.2.0 is a maintenance release on specific customer request.

## New Features 3.2.0

- **CR085: Power Management Functionality**

From this version onward, AVIX is extended with functionality to exploit the microcontroller's energy saving capabilities. For this purpose three functions and a number of symbols are added to the AVIX API.

- `avixPower_SetMode`
- `avixPower_GetMode`
- `avixPower_SetCallback`

The prototype of these functions and the related definitions is present in interface file `avixPower.h` which is installed in the AVIX Interface subdirectory. This interface file is included from interface file `AVIX.h`.

To support power management the AVIX configuration file contains an additional parameter `avix_TIMER_CLK_SOURCE` to allow an external oscillator being used as the source for the AVIX system timer.

Finally, a symbol is added `AVIX_SYS_CLOCK_ACTUAL_PERIOD` defining the actual system timer period in microseconds whose value may differ slightly from the configured system timer period because of oscillator resolution. This definition is present in header file `avixTimer.h`. Note the value of this symbol is also show in the system section of version 1.4 of the AVIX RTOS viewer plugin.

For details on all power management related functions and symbols, please refer the AVIX User Guide section 6.2.6.

## Improvements 3.2.0

- None

## Changes 3.2.0

- None

## Known issues 3.2.0

- None

## Fixed issues 3.2.0

- **PR086: User guide inadvertently states `avixDIH_Queue` is allowed to be called from a thread.** Function `avixDIH_Queue` is used to allow an Interrupt Service Routine to communicate with a thread and will be called from Interrupt Service Routines for this purpose.

The user documentation inadvertently states this function is also allowed to be called from a thread. This is however wrong and might lead to corruption of the DIH queue content resulting in system failure. Although this only happens under very rare conditions, `avixDIH_Queue` should not be called from a thread. Only call this function from Interrupt Service Routines.

Since the sole purpose of `avixDIH_Queue` is for Interrupt Service Routine-thread integration this issue is considered a documentation issue so the AVIX User Guide is updated according the above

description.

- **PR089: AVIX-32 internal stack alignment issue**

Because of a Microchip documentation issue, AVIX-32 is hurt by an internal stack alignment issue. Under rare conditions, especially when using the C32 version 1.05 floating point libraries with double values this might lead to stack corruption. AVIX-32 is changed such that the stack is aligned correctly under all circumstances.

***This issue is considered critical and therefore it is strongly suggested to use release 3.5 to replace earlier commercial releases of AVIX-32.***

This fix does not influence functionality or performance.

## Documentation changes 3.2.0

- **User Guide adapted according the above issues**

The AVIX User Guide is adapted according the above described issues. The new Power Management functionality is described in section 6.2.6 of the User Guide. Note there is also a controller specific power management section for each of the controller families supported by AVIX.

## How to migrate from version 3.1.0 to version 3.2.0

### ***Background***

When installing AVIX, a directory structure is created as described in the AVIX User Manual. The directory structure and content of version 3.2.0 is the same as that of version 3.1.0 with the addition of file avixPower.h in the Interface directory for the definitions of the AVIX power management functions and symbols.

The Application Programming Interface of version 3.2.0 is identical to that of version 3.1.0 so version 3.2.0 can be used as a drop-in replacement for version 3.1.0 without making any change to the application code. *Note that version 3.2.0 does contain additional functions and definitions which are described above.*

Configuration settings are contained in AVIXxxSystemSettings.h (xx is either 16 or 32 depending on the distribution you are using). The configuration file is installed with default settings. It is essential that the settings used with version 3.1.0 are migrated to the settings file that comes with version 3.2.0. This has to be done manually. Advised is to use file comparison software like BeyondCompare. Note that starting with version 3.2.0, the configuration file contains an additional parameter avix\_TIMER\_CLK\_SOURCE, details of which you find above and in the AVIX User Guide.

The AVIX library name contains its version number and therefore MPLAB projects must be changed to refer the version 3.2.0 library.

### ***Migration procedure***

In accordance with the above, the following migration procedure is advised:

1. Install the 3.2.0 distribution to a new location in your file system
2. Copy the configuration settings of the 3.1.0 version you are using to the newly installed configuration file.
3. Replace all version 3.1.0 files in your development structure with the new version 3.2.0 files. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 3.1.0 library.
5. Change the project settings to include the new version 3.2.0 library.
6. Install the new version 1.4 of the RTOS Viewer plugin.

Before executing these steps you might want to make a backup of the version 3.1.0 files.

## Build and development environment issues 3.2.0

Issues reported with version 3.1.0 are still applicable with the following additions:

### AVIX-16

- none

### AVIX-32

- none

### *RTOS Viewer plug-in*

- **Actual system timer period added to AVIX RTOS Viewer plug-in:**  
With version 3.2.0 of AVIX comes a new version 1.4 of the AVIX RTOS Viewer plug-in. This new version of the plug-in contains an additional entry in the system section showing the actual system timer period in microseconds.

Although the previous version 1.3 of the AVIX RTOS Viewer plug-in is compatible with AVIX version 3.2.0, advised is to migrate to the new version 1.4 of the plug-in.



This page is intentionally left blank

## Summary 3.1.0

This section contains release notes for AVIX version 3.1.0 which is the successor to version 2.5.0. This is a major upgrade for the following reason:

AVIX is a so called segmented architecture RTOS. The main advantage of the segmented architecture over the unified architecture most other RTOSes implement is the zero latency feature and the ease of use of interrupt service routines. The main advantage of the unified architecture is that it allows more functions to be used from within interrupt service routines. With this new version, AVIX becomes a hybrid RTOS where the advantages of both architectures are combined in a single product. From version 3.1.0 onward, AVIX allows additional functions to be used directly from within interrupt service routines besides the interrupt service routine specific functions that were already available, being the pipe and memory pool functions.

This not only allows for an easier programming model where these functions can be called directly from within an interrupt service routine instead of indirectly by using a DIH but also results in an increase in performance, especially for the interrupt handling. More details can be found in the 'New Features 3.1.0' section of this document.

## New Features 3.1.0

- **CR077: Added functions to be called directly from within Interrupt Service Routines**  
The following functions are added to the AVIX API to be called directly from within an Interrupt Service Routine:

- o `avixSemaphore_UnlockFromISR`
- o `avixThread_ResumeFromISR`
- o `avixMsg_AllocateFromISR`
- o `avixMsg_SendFromISR`
- o `avixMsg_Put<type>FromISR`
- o `avixEventGroup_ChangeFromISR`
- o `avixEventGroupThread_ChangeFromISR`

A detailed description of these functions can be found in the 'AVIX User Guide & Reference Manual'.

- **CR070: Added function to retrieve message sender thread id**  
Added is function `avixMsg_GetSender` returning the id of the thread the message is received from. A detailed description of this function can be found in the 'AVIX User Guide & Reference Manual'.
- **CR081: Added AVIX.h**  
Added is an additional header file AVIX.h which includes all individual AVIX interface files. From now on when using AVIX, only file AVIX.h needs to be included.
- **CR082: Disable round robin scheduling**  
From this version onward AVIX allows round robin scheduling to be disabled by specifying a round robin period of zero (0). This implies that threads at the same priority are no longer automatically preempted after consuming their time slice. When disabling round robin scheduling and still using threads at the same priority, threads must explicitly give up the processor by calling `avixThread_Relinquish` in order for the next thread at that priority to become ready to run. Using a round-robin time slice value of zero (0) has a positive effect on performance. Advised is to only disable round robin scheduling when no threads with the same priority are used.

## Improvements 3.1.0

- **CR064: Performance improvement PIC32 interrupt handling**  
Internal optimization leads to increased performance for PIC32 interrupt handling.
- **CR065: Reduction in thread stack usage for AVIX-32**  
An internal optimization led to an even further reduction of thread stack usage for AVIX-32. Advised is to check your applications stack usage with the AVIX RTOS viewer in order to determine whether your application stacks can be reduced in size leading to more RAM being available to your application.

- **CR071 / CR078: Pipe mechanism queues superfluous DIH's**
- Internally the pipe mechanism makes use of DIH's. Pipe code has undergone an optimization to minimize the number of DIH's used and thereby increase performance.

## Changes 3.1.0

- **Message compatibility mode removed.**  
Starting with AVIX version 2.2, the AVIX message mechanism is based on a standard memory pool. In earlier versions, messages were allocated from the global free store. Version 2.2 and later offered both mechanisms for reason of backward compatibility. Starting with version 3.1.0, the legacy method is removed and the message mechanism will only use the memory pool mechanism.

## Known issues 3.1.0

- **User guide inadvertently states avixDIH\_Queue is allowed to be called from a thread.**  
Function `avixDIH_Queue` is used to allow an Interrupt Service Routine to communicate with a thread and will be called from Interrupt Service Routines for this purpose.

The user documentation inadvertently states this function is also allowed to be called from a thread. This is however wrong and might lead to corruption of the DIH queue content resulting in system failure. Although this only happens under very rare conditions, it is strongly advised not to call `avixDIH_Queue` from a thread and only call this function from Interrupt Service Routines.

Since the sole purpose of `avixDIH_Queue` is for Interrupt Service Routine / thread integration this issue is considered a documentation issue which will be corrected in the next release.

## Fixed issues 3.1.0

- **PR063: Compiling for AVIX-32 with -G0 causes problem**  
When compiling for AVIX-32 with compiler option `-G0`, macro's `avixDeclareISR` and `avixDeclareISRShadow` generate erroneous code. Although this issue is fixed in this new version 3.1, AVIX-RT strongly suggests not to use this compiler option since this generates less efficient code because no use is made of the PIC32 gp register.
- **PR074: Null Message initialization in case of time-out**  
When allocating a message using a time out, it is possible a NULL message id is returned in case no messages are available in the message pool. Inside the allocation function (`avixMsg_Allocate`) the message header is initialized before the message id is returned to the caller. A check is added inside this function to prevent message header initialization in case a NULL message id is returned.
- **PR080: Unexpected behavior in case of deadlock**  
In case of a deadlock using mutexes, the deadlock detecting thread enters an endless loop, effectively causing the application to come to a halt since this thread remains running. Although the deadlock is caused by an issue in the application code, this is undesired behavior. A fix is introduced such that threads involved in a deadlock remain blocked and other threads remain running. Although AVIX is capable of detecting that a mutex operation caused a deadlock, decided is not to issue an error. Reason is that mutex related deadlocks are only one of the deadlock scenarios an application can cause. For deadlocks related to other kernel objects like semaphores or message queues, AVIX is not capable of detecting a deadlock. So the scenario for mutexes is consistent with that of other kernel objects.

## Documentation changes 3.1.0

- **Major overhaul of AVIX user documentation**  
The AVIX user documentation is entirely rewritten leading to a more comprehensible structure. From this version onward, the user documentation contains a clean separation between the user guide and the reference section. Earlier versions contained the user guide information not only in the section named as such but also in other sections.

**From the perspective of usability the following is important:** The Reference section of the manual contains a table with an enumeration of all AVIX supplied functions. These function names are hyperlinked to the detailed description and all detailed descriptions contain hyperlinks back to the

mentioned table. This makes it very easy when using the documentation in electronic form to access all relevant information.

## How to migrate from version 2.5.0 to version 3.1.0

### **Background**

When installing AVIX, a directory structure is created as described in the AVIX User Manual. The directory structure of version 3.1.0 is the same as that of version 2.5.0. The content of the created directories with version 3.1.0 are the same as that of version 2.5.0 with the addition of file AVIX.h in the interface subdirectory, which file can be used as the single header file to include in the application for using AVIX.

The Application Programming Interface of version 3.1.0 is identical to that of version 2.5.0 so version 3.1.0 can be used as a drop-in replacement for version 2.5.0 without making any change to the application code. *Note that version 3.1.0 does contain additional functions which are described above.*

Configuration settings are contained in AVIXxxSystemSettings.h (xx is either 16 or 32 depending on the distribution you are using). The configuration file is installed with default settings. It is essential that the settings used with version 2.5.0 are migrated to the settings file that comes with version 3.1.0. This has to be done manually. Consider using a round-robin time slice value of zero when no threads with the same priority are used by your application.

The AVIX library name contains its version number and therefore projects must be changed to refer the version 3.1.0 library.

### **Migration procedure**

In accordance with the above, the following migration procedure is advised:

1. Install the 3.1.0 distribution to a new location in your file system
2. Copy the configuration settings of the 2.5.0 version you are using to the newly installed configuration file.
3. Replace all version 2.5.0 files in your development structure with the new version 3.1.0 files. Make sure not to mix files of both versions since this might lead to unpredictable results. All files are marked in the header with the version number they belong to.
4. Remove the version 2.5.0 library.
5. Change the project settings to include the new version 3.1.0 library.
6. Install the new version 1.3 of the RTOS Viewer plugin.

Before executing these steps you might want to make a backup of the version 2.5.0 files.

## Build and development environment issues 3.1.0

Issues reported with version 2.5.0 are still applicable with the following additions:

### **AVIX-16**

- none

### **AVIX-32**

- none

***RTOS Viewer plug-in***

- The AVIX RTOS viewer plug-in is provided in a new version 1.3. This version is compatible with AVIX version 3.1.0 only. The previous version of the RTOS viewer plug-in (1.2) is not compatible with AVIX version 3.1.0. Make sure when using AVIX version 3.1.0 to also install the new RTOS viewer plug-in. The functionality of version 1.3 of the RTOS viewer plug-in is identical to that of version 1.2.

This page is intentionally left blank

## Summary 2.5.0

This section contains release notes for AVIX version 2.5.0 which is the successor to version 2.2.1. The major reasons for publishing this new version are:

- Enhanced Pipe implementation
- Making AVIX-16 compatible with multiple PSV pages and DSP functionality offered by the dsPIC30 and dsPIC33 parts

### Important notice:

It is AVIX-RT's policy to keep the AVIX API stable in order for new versions of AVIX to be used with existing applications without requiring any changes to these applications. Reasons can however exit this is hard or even impossible to maintain. Version 2.5.0 introduces a small number of changes to the API which are incompatible with existing application code. Make sure to read the 'Changes 2.5.0' section for details of those changes. For every change its rationale is documented.

All changes are syntactical. In now way are the semantics of AVIX changed and after applying the mentioned changes to the application's source code the behavior of the application will be the same as before.

## New Features 2.5.0

- **Utility macros added for atomic SFR bitfield manipulation**  
When using multiple threads and/or interrupts it is essential manipulations of shared resources is done in an atomic fashion. This is especially true for the Special Function Registers (SFR's). These registers are manipulated bitwise which in general implies a Read-Modify-Write sequence which is vulnerable to corruption when using interrupts or multiple threads. Since it would be far too expensive to place every SFR bitfield manipulation in a critical section guarded by a mutex, a macro based mechanism is developed that guarantees atomic SFR bitfield manipulation without the need of using a mutex. The macros are contained in file AVIXAtomicSFR.h a separate directory tree Utilities/Interface installed in the main install directory. The mechanism is documented in a separate document (AvixAtomicSFRManipulation.pdf), which is installed in the Doc directory. The mechanism is portable between AVIX-16 and AVIX-32.
- **Function added to remove an event group from a timer**  
An event group can be connected to a timer for changing one or more flags when the timer expires. Added is function `avixTimer_DisconnectEventGroup` to disconnect the event group from the timer so it is no longer influenced by the timer expiring. Details can be found in the AVIX user guide.
- **Function added to remove an event group from a message queue**  
An event group can be connected to a message queue to have one or more flags set as long as there are messages present in the queue. Added is function `avixMsgQThread_DisconnectEventGroup` to disconnect the event group from the message queue so it is no longer influenced by messages being placed in the queue. Details can be found in the AVIX user guide.
- **System setting parameter added to control PSV functionality for AVIX-16**  
Added to AVIX16SystemSettings.h is a flag (`avix_MULTI_THREAD_PSV`) to allow multiple PSV pages to be used. Details can be found in the AVIX-16 specific appendix of the User Guide.
- **System setting parameter added to control TBLPAG register usage AVIX-16**  
Added to AVIX16SystemSettings.h is a flag (`avix_MULTI_THREAD_TBLPAG`) to allow each thread to use the TBLPAG register where each thread has its own local copy. Details can be found in the AVIX-16 specific appendix of the User Guide.
- **System setting parameter added to control DSP functionality for AVIX-16**  
Added to AVIX16SystemSettings.h is a flag (`avix_DSP_ENABLED`) to allow a single thread to use DSP functionality without influencing other parts of the application. Details can be found in the AVIX-16 specific appendix of the User Guide.



- **Macro's added to convert kernel object id to and from a void pointer**  
AVIX offers a number of type safe type definitions. These are used for kernel object id's and a number of enumerated types. For passing variables of those types to functions expecting a void\* argument, two macro's are added. These are AVIX\_TYPESAFE\_TO\_VOID and AVIX\_TYPESAFE\_FROM\_VOID. A description of these macros can be found in the user manual, chapter 9.5, "Types and type safety". These macros are intended to be used with arguments passed to avixDIH\_Queue and avixThread\_Create.

## Improvements 2.5.0

- **PR040: Pipe Mechanism extended to support block mode and multiple readers/writer.**  
In earlier versions, AVIX allowed a pipe to be used from a single reader and a single writer thread. This is extended such that a pipe can now be used by multiple reader and multiple writer threads. Note when using pipes to communicate with ISR's, a pipe still may only be used from a single ISR priority level. So when multiple ISR's use a pipe, these ISR's must have the same priority.

Also from version 2.5.0 onwards, pipes are block oriented instead of byte oriented. This is an important improvement allowing your application code to become less complex since the application does no longer need to deal with partial reads or writes. This improvement led to a minor change in the interface of avixPipe\_Create which is documented under 'Changes 2.5.0'.

- **Reduction in thread stack usage for AVIX-32**  
An internal improvement led to an even further reduction of thread stack usage for AVIX-32. Advised is to check your applications stack usage with the AVIX RTOS viewer in order to determine whether the stacks can be made smaller leading to more RAM being available to your application.
- **Minor improvements and performance enhancements**  
A number of minor internal improvements led to a performance increase for a number of functions. This especially resulted in better pipe and interrupt performance making AVIX win five out of the seven tests the open source 'ThreadMetric' test suite is comprised of.

## Changes 2.5.0

- **Pipe mechanism block oriented instead of byte oriented**  
Starting with this version, pipes have become block oriented. As a consequence, avixPipe\_Create receives an additional argument. Instead of specifying the size of the pipe in a single argument representing the number of bytes, from now on the size is specified using two arguments, the first specifying the number of blocks and the second the size in bytes of a single block. A detailed description of this improvement and required changes is provided in a separate section in these Release Notes; '**Improvements to the pipe mechanism and related actions**'. Make sure to read this section since it contains important information when using pipes.
- **AVIX-16 Interrupt declaration macro's requiring an additional parameter**  
Making AVIX-16 compatible with applications using multiple PSV pages (see Improvements 2.5.0) requires ISR's to optionally use the compiler managed PSV page. This requires the AVIX-16 ISR macro's to be used with C30 compatible flags no\_auto\_psv or auto\_psv. To use one of these flags when declaring an ISR, macro's avixDeclareISR and avixDeclareISRShadow receive an additional argument.

**Note this change only involves AVIX-16. When using version 2.5.0, make sure to check for interrupt declarations based on the mentioned macro's and pass an additional parameter based on PSV usage needs. Using value no\_auto\_psv keeps the ISR compatible with earlier AVIX versions.**

- **Boolean type definition removed from AVIX header files.**  
AVIX used to define a bool type with related values FALSE and TRUE. For reasons of compatibility with Microchip software stacks (FAT, Graphical display, Ethernet etc.) the bool type and the FALSE and TRUE constants are removed from the API. This resulted in changing the API of the following functions:
  - avixThread\_Create: Type of last parameter changed from bool to tavixThreadSuspended with possible values AVIX\_THREAD\_READY (i.s.o. FALSE) or AVIX\_THREAD\_SUSPENDED (i.s.o. TRUE).

- `avixMutex_Create`: Type of last parameter changed from bool to `tavixMutexLocked` with possible values `AVIX_MUTEX_UNLOCKED` (i.s.o. FALSE) or `AVIX_MUTEX_LOCKED` (i.s.o. TRUE).
  - `avixSemaphore_Create`: Type of last parameter changed from bool to `tavixSemaphoreLocked` with possible values `AVIX_SEMAPHORE_UNLOCKED` (i.s.o. FALSE) or `AVIX_SEMAPHORE_LOCKED` (i.s.o. TRUE).
  - `avixTimer_Start`: Return value changed from bool to int with possible values 0 (i.s.o. FALSE) or `!= 0` (i.s.o. TRUE). This complies with the C language values for boolean expressions.
  - `avixKernelObjectId_IsValid`: Return value changed from bool to int with possible values 0 (i.s.o. FALSE) or `!= 0` (i.s.o. TRUE). This complies with the C language values for boolean expressions.
  - `avixThread_TimeOutOccured`: Return value changed from bool to int with possible values 0 (i.s.o. FALSE) or `!= 0` (i.s.o. TRUE). This complies with the C language values for boolean expressions.
- **System settings CPU speed in MIPS changed to device speed in Herz**  
The name of system settings parameter `avix_CPU_SPEED_MIPS` is changed to `avix_DEVICE_CLOCKHz`. This is a cosmetic change and is done to let the name better reflect the purpose of the setting. It is likely this parameter is not used in application code in which case this change will not have effect on your application. Might it be used make sure to also change the name in your application code.

## Known issues 2.5.0

- None

## Fixed issues 2.5.0

- **PR029: AVIX-16 is incompatible with dsPIC30/33 DSP functionality**  
AVIX-16 used to be incompatible with DSP functionality using modulo or bit-reverse addressing. This issue has been fixed. For more details also see the New Features 2.5.0 section in this document and the AVIX User Guide.
- **PR038: Number of possible memory pools one lower than configured**  
For AVIX-32, the number of memory pools that could be used was one lower than the configured number of memory pools. This is fixed so from version 2.5.0, configuration parameter `avix_MEM_POOL_COUNT` in file `AVIX32SystemSettings.h` reflects the actual number of memory pools that can be used by the application.
- **PR044: Struct type `tavixKernelObjectIdp` does not contain pointer for semaphore**  
Function `avixMsg_GetKernelObjectId` receives an argument of type `tavixKernelObjectIdp`. This is a transparent union type to allow a pointer to any kernel id type to be passed to this function. The field representing a semaphore id was not declared as a pointer so reading the id of a semaphore from a message would not work. This issue is fixed by declaring the semaphore field as a pointer.

## Documentation changes 2.5.0

- Added is a table containing an overview of all AVIX functions as a quick reference
- Added is a table containing an overview of all AVIX macros as a quick reference
- The AVIX User Guide has been updated with a PSV section accompanying the fix described above
- The AVIX User Guide has been updated with a TBLPAG section accompanying the fix described above
- The AVIX User Guide has been updated with a DSP section accompanying the fix described above.
- Minor changes are made to different sections to make the user manual consistent with changes and extensions to AVIX documented in these release notes.

## Build and development environment issues 2.5.0

Issues reported with version 2.2.1 are still applicable with the following additions:

### AVIX-16

- **Stack corruption issue with MPLAB™ version 8.14 simulator**  
The simulator for the 24 and 3x parts contains a Microchip confirmed issue. When using the simulator without tracing enabled, under certain conditions a stack corruption can occur. With tracing enabled, this stack corruption does not occur. AVIX-RT, being a Microchip Recognized Third Party, has been involved in testing the fix provided by Microchip for this issue which is published in MPLAB™ version 8.15a.
- **For certain 24F parts MPLAB™ version 8.15a simulator corrupts the content of register W0**  
When an interrupt occurs the MPLAB™ version 8.15a corrupts the content of W0. This happens for at least the 'GB' parts (PIC24FJxxxGByyy) and the 'GA1' parts (PIC24FJxxxGA1yy). The 'GA0' parts are found to be simulated correctly (PIC24FJxxxGA0yy)

### AVIX-32

- **MPLAB™ version 8.14 and version 8.15a simulator not working correctly**  
Because of Microchip confirmed issues with the PIC32 simulator of MPLAB™ version 8.14 and version 8.15a, AVIX-32 can not be run with the simulator but only on real hardware (Explorer 16). AVIX-RT is working closely with Microchip to assist in solving this simulator bug.

### RTOS Viewer plug-in

The AVIX RTOS viewer is provided in a new version 1.2. This version is compatible with AVIX version 2.5.0 only. The previous version of the RTOS viewer (1.1) is not compatible with AVIX version 2.5.0. Make sure when using AVIX version 2.5.0 to also install the new RTOS viewer plug-in. Furthermore some minor issues are fixed and the following extensions are made to the RTOS viewer plug-in:

- **Pipe display block oriented in accordance with the changed pipe interface**
- **For pipes column added with maximum pipe usage since the pipe is in use**
- **A thread blocked on a pipe is shown with the number of blocks it either wants to read or write**
- **Memory pool display column added with maximum pool usage since the pool is in use**

## Improvements to the pipe mechanism and related actions

This section provides a detailed description of the improvement to a block oriented approach for pipes, the reason why this is done, the consequences and suggested changes to your application.

### Why is the change made

The pipe mechanism prior to AVIX version 2.5.0 is byte oriented. When reading from or writing to a pipe the number of bytes to transfer is specified. Normally when the function returns the specified number of bytes is transferred. Two exceptions exist. When using a thread based read or write function with a time-out and the time out occurs, not necessarily all bytes are transferred. This is the case if the time-out expires before the request is completely dealt with. On reading this can happen when the requested number of bytes is not yet present in the pipe when the time-out occurs. On writing this can happen when not sufficient empty space is present in the pipe when the time-out occurs.

Also when using read or write functions from an ISR not all specified bytes are transferred, in this case because ISR call's are not blocking so when reading the ISR can only read as many bytes as present in the pipe and when writing the ISR can only write as many bytes as there is empty space in the pipe.

In all cases the application is aware of the number of bytes actually transferred by using the return value of the read and write functions. So under all circumstances the application can recover from a partial read or write and under no circumstance data will get lost.

Recovering from a partial read or write can however be quite involved in case the basic size of the data the pipe is used with is more than a single byte. Besides byte streams, pipes can be used for any other fixed size multi byte data type like kernel id's, pointers or user defined structs. In these cases a partial read or write can also imply not all bytes of a basic element are transferred.

Let's illustrate this with a sample. In this sample code the time-out mechanisms is not shown but for the sake of the sample assume it to be present. Suppose the pipe is used to transfer elements of type myStruct where myStruct is a struct containing more than one byte. Writing a single element to a pipe can look like:

```
myStruct myVar;
int      bytesWritten;

bytesWritten = avixPipe_Write(pipe, (unsigned char*)&myVar, sizeof(myVar));
if (bytesWritten == sizeof(myVar)) {
    /* The write operation succeeded and we are done */
}
```

In case the write has not entirely succeeded, a retry must continue at the byte position where the incomplete operation left. So the else clause for the above sample will look something like:

```
bytesWritten =
    avixPipe_Write(pipe, ((unsigned char*)&myVar)+bytesWritten, sizeof(myVar));
```

Added to the address of the struct is the number of bytes that has already been written in the previous attempt.

Although the above does not yet look too complex, using pipes from ISR's is more involved. An ISR is a function that always runs to completion. Suppose an ISR reads from a pipe and not all data of a basic element is returned. In that case the ISR must 'remember' what is already read in order to continue with the subsequent read where the first read ended. An ISR can however not use local variables and thus this 'bookkeeping' must be maintained in global variables. However, how does the ISR know when to continue? It takes an interrupt to activate the function and this just occurred. How can we generate another interrupt? This potentially leads to a complex software structure.

For this reason, from version 2.5.0 onwards, the pipe mechanism is block oriented. The pipe is aware of the size of the individual elements it contains and a read or write is only done when one or more blocks can entirely be transferred. When for instance specifying a block size of 10 bytes, a read or write will always process an exact multiple of 10 bytes and never something like 11, 12 or 13 bytes. This makes using pipes much easier.

## ***What are the consequences***

From a functional point of view, the new approach is fully backward compatible with the existing approach. It is required to add the additional argument to every call to avixPipe\_Create but in its easiest form, for each of those calls, a block size of 1 can be specified. This will result in the pipe still being byte oriented and your application will function as before.

So as a minimum change:

```
avixPipe_Create(NULL, 100, NULL, NULL);
```

to

```
avixPipe_Create(NULL, 100, 1, NULL, NULL);
```

And your application will work as before but the pipe will still be byte oriented, e.g. the size of a block is one byte.

It might however be the case pipes are used with structs or pointers as the basic type. In these cases it is advised to change the avixPipe\_Create function to specify the size of these elements as the pipe block size. Making this change does have the advantage described in the previous section.

Doing so might involve making changes to the use of the following functions:

```
avixPipe_Read:  
avixPipe_Write:  
avixPipe_ReadFromISR:  
avixPipe_WriteFromISR:  
user specified callback:
```

In versions prior to 2.5.0 these functions use byte oriented arguments and return values. The signature of those functions is not changed but the semantics are. Instead of bytes, blocks are used. So instead of using `avixPipe_Read` where one of the arguments specifies the number of bytes to read, from version 2.5.0 onwards the number of blocks must be specified. Once more, if the block size of the pipe is 1, no change is needed. If however the block size is not 1, the change must be reflected in the use of the mentioned functions.

This is again illustrated with an example. First, suppose a pipe is used with type `myStruct` but the block size is one (1) byte. Reading a single struct is written like:

```
myStruct    myVar;  
int         nrBytesRead;  
  
nrBytesRead = avixPipe_Read(pipe, (unsigned char*)&myVar, sizeof(myVar));
```

When changing the pipe to block mode this code fragment will look like:

```
myStruct    myVar;  
int         nrBlocksRead;  
  
nrBlocksRead = avixPipe_Read(pipe, (unsigned char*)&myVar, 1);
```

The advantage of the second sample is that it is guaranteed the struct is always entirely read. The read function will either return 0 or 1 (assuming a time-out is used) while in the first sample, the struct may be partially read.

For this reason it is advised to have a look at all pipe functions and change the code according the block size used when specifying the pipe.

This page is intentionally left blank

## Summary 2.2.1

This section contains release notes for AVIX version 2.2.1 which is the successor to version 2.2.0. Version 2.2.1 is a minor maintenance release. The main reason for this upgrade is for AVIX to be compatible with the MPLAB™ RTOS Viewer plug-in which is shipped as a free product by AVIX-RT. In the process of generating this release a number of minor issues have been solved.

## New Features 2.2.1

- None, Version 2.2.1 is a maintenance release

## Improvements 2.2.1

- None, Version 2.2.1 is a maintenance release

## Changes 2.2.1

- **PR024: Prevent use of C30 version 3.02**  
Because of a Microchip confirmed issues with C30 version 3.02, AVIX is not compatible with this version of the compiler. A macro is added that checks for use of this version of the compiler generating an error if this version of the compiler is used.
- **PR034: Message body always multiple of basic controller word size**  
The configured size of a message body was the actual size used. If the configured size was not a multiple of the platform basic word size (2 bytes for AVIX-16 and 4 bytes for AVIX-32), the remaining bytes could not be used. A change is made such that the configured message body size is rounded to the first higher or equal multiple of the basic word size and the resulting value can be used.
- **PR037: AVIX-32 Software interrupt fully specified**  
AVIX-32 needs one of the controllers interrupts for internal use. This can be either CS0 or CS1. In version 2.2.0, the selected interrupt was configured through its number. For reason of compatibility from version 2.2.1 on, the complete name must be used in AVIX32SystemSettings.h. So the software interrupt configured in this file is either 'CS0' or 'CS1'.

## Known issues 2.2.1

- **PR029: dsPIC30 and dsPIC33 offer DSP instructions.** When using those instructions with modulo addressing or bit-reversed addressing, AVIX-16 will crash. This will be fixed in a forthcoming version. As a workaround, with version 2.01 you should not use modulo or bit reversed addressing modes.

Regarding usage of DSP instructions: AVIX-16 does not allow more than one thread to use DSP instructions. This is by design and based on the fact that dsPIC30 and dsPIC33 do not allow the DSP state to be fully restored. Since this limitation is based on a hardware restriction there is no solution for this. In a forthcoming version the documentation will be enhanced accordingly.

## Fixed issues 2.2.1

- **PR032: No port definition for port E for use with Thread Activation Tracing**  
The port definitions to be used with Thread Activation Tracing were lacking a definition for port E. This port definition is added to the applicable header.
- **PR033: Problem using \_Get on non-existent memory pool**  
AVIX allows to get access to a not-yet existing kernel object by using a human readable name. The thread executing the \_Get blocks until the desired object is actually created by another thread. A bug in the memory pool mechanism prevented this from working in case when executing the \_Get the memory pool did not yet exist. When executing \_Get for an already existing memory pool functionality was as specified. This issues is fixed such that \_Get works regardless whether at the moment of \_Get the memory pool exists or not.

- **PR035: Not configuring system stack size as multiple of basic word size crashes**  
When not configuring the size of the system stack as a multiple of the basic word size of the platform (2 bytes for AVIX-16 and 4 bytes for AVIX-32) lead to a system crash. A change is made such that the configured size of the system stack is rounded to the first higher or equal multiple of the basic word size and the resulting value is used as the size for the system stack.

## Documentation changes 2.2.1

- Minor, see other sections.

## Build and development environment issues 2.2.1

Issues reported with version 2.2.0 are still applicable with the following additions:

### AVIX-16

- **Bug in C30 version 3.10 generates wrong interrupt code**  
Because of a Microchip confirmed bug in C30 version 3.10, wrong code is generated for interrupt handlers under varying conditions. Advised is to only declare interrupt handlers based on the AVIX supplied macro's since these are not harmed by this bug.

### AVIX-32

#### **MPLAB™ 8.10 simulator not working**

Because of Microchip confirmed bugs in the PIC32 simulator of MPLAB™ version 8.10, AVIX-32 can not be run with the simulator but only on real hardware (Explorer 16). AVIX-RT is working closely with Microchip to assist in solving this simulator bug.




This page is intentionally left blank

## Summary 2.2.0


This section contains release notes for AVIX version 2.2.0 which is the successor to version 2.0.1. Starting with version 2.2.0, besides being available for PIC24 and dsPIC, an entirely new port is added for PIC32 controllers. Product naming is enhanced to identify the port. To differentiate between the AVIX ports, the product named AVIX in earlier versions will be called AVIX-16 from version 2.2.0 onwards. AVIX-16 targets PIC24 and dsPIC controllers. The new port targeting PIC32 controllers is named AVIX-32.

## New Features 2.2.0

- **Time out functionality added:**  
Starting with this version, AVIX offers time out functionality. On potentially blocking functions, a thread can choose to activate a time out which causes the function to return in case the desired resource does not become available within the specified time. In order to use a function with a time out, two function calls must be added to the applicable function. Before the applicable function, the time out must be armed with a call to `avixThread_ArmTimeOut` and behind the function a call to `avixThread_TimeOutOccured` must be added to check for a time out condition. Functions that can be used with a time out are clearly identified in the user manual by this icon .
- **New function added to check for resource availability from a DIH:**  
A new function is added to check whether a resource claimed by a DIH is present or not. The name of this function is `avixKernelObjectId_IsValid`. This function must be used in a DIH when the DIH makes a call to a potential blocking function. For more details see section "Improvements 2.2.0", topic "Deterministic usage of potential blocking calls from DIH's".
- **Message pool mechanism added:**  
In versions preceding version 2.2.0, messages were allocated from the AVIX free store. When allocating a message while the free store was exhausted, an error was generated. Allocating a message was not a potentially blocking call. Starting with version 2.2.0, messages are allocated from a message pool, the size of which is configurable. Allocating a message has become a potentially blocking call, meaning that in case the message pool is exhausted, the calling thread is blocked until either a message becomes available again or an optional time out expires. For backward compatibility, the pre-version 2.2.0 style of message allocation can still be used, details of which can be found in the AVIX user manual. Advised is from now on only to use the message pool based mechanism since AVIX-RT pertains the right to remove this compatibility mode in a forthcoming version.
- **Number of memory pools configurable:**  
Starting with version 2.2.0, the maximum number of memory pools is configurable. This feature is only available for the Extended distribution. The Basic and Standard distribution have a preconfigured maximum number of memory pools.
- **New functions added to put and get a short value to or from a message**  
Starting with version 2.2.0, AVIX is also available for the PIC32 controller. This is a 32 bit controller. In order to allow explicit 16 bit values to be used with messages, functions `avixMsg_PutShort` and `avixMsg_GetShort` are added. By using these functions, manipulating 16 bit values is independent of whether AVIX is used with 16 bit or 32 bit controllers.
- **Interrupt Service Routine definition macro added using shadow registers:**  
Added is macro `avixDeclareISRShadow`. Using this macro an Interrupt Service Routine can be defined that uses the controllers shadow register set leading to a lower interrupt latency. For AVIX-16 this macro can be used with any desirable interrupt priority as long as the Interrupt Service Routines defined with this macro have the same priority. For AVIX-32 this macro may only be used with priority 7 Interrupt Service Routines.

## Improvements 2.2.0

- **Deterministic usage of potentially blocking calls from DIH's:**  
DIH's are extension to Interrupt Service Routines which just like ISR's are not allowed to block on a resource not being available. A DIH, when started, always runs to completion. In versions preceding version 2.2.0 a DIH was only allowed to make a potential blocking function call when the programmer could guarantee the desired resource to be available. Would the resource not be available this would lead to unspecified behavior. Starting with version 2.2.0, a large number of potential blocking functions

are allowed to be used from a DIH where, in case the desired resource is not available, the software can check for this situation by using new function `avixKernelObjectId_IsValid`. Functions that can be called from a DIH where success or failure can be checked are clearly identified in the user manual by this icon: 

- **Timer handling performance increase:**

The internal structure of timer management is improved leading to a better performance of timer handling. This is especially important since the addition of time out functionality will potentially increase the usage of timers.

## Changes 2.2.0

- **Pipe read and write operations return int value:**

Starting with version 2.2.0, `avixPipe_Read` and `avixPipe_Write` return an integer value specifying the number of bytes actually read or written. This extension exists since `avixPipe_Read` and `avixPipe_Write` can now be used with a timeout, potentially leading to those functions returning before the operations are entirely finished. This extension is fully backward compatible with previous versions since the timeout mechanism is added in version 2.2.0. When not using those functions with a timeout, the return value does not need to be checked since the functions only return when the required number of bytes is processed. Depending on the build settings this change might lead to a compiler warning. This warning is harmless and can be ignored. To remove the warning the functions must be cast to type void.

- **AVIXSystemSettings.h renamed to reflect port:**

With the introduction of AVIX-32 for PIC32, the file with configuration settings is renamed to identify the port the file belongs to. `AVIXSystemSettings.h`, as was available with AVIX version 2.01 and earlier is named `AVIX16SystemSettings.h` from version 2.2.0 onwards. `AVIX16SystemSettings.h` contains configuration settings belonging to AVIX-16. `AVIX32SystemSettings.h` is distributed with AVIX-32 and contains the configuration settings for this port.

- **Controller type configuration no longer needed:**

Starting with version 2.2.0, it is no longer needed to configure the controller family in `AVIXSystemSettings.h` and the related entry is removed from this file. When building from MPLAB™, the controller type (and thus family) is defined through the dialog activated by selecting menu entry Configure / Select Device... When building from the command line, for AVIX-16 using C30 the controller type (and thus family) is defined by compiler argument `-mcpu`. For AVIX-32 using C32 the controller type (and thus family) is defined by compiler argument `-mprocessor`.

- **Macro's AVIX\_TYPESAFE\_EQ and AVIX\_TYPESAFE\_NEQ added:**

AVIX offers a number of type safe types allowing certain programming errors to be detected compile time instead of runtime. In order for the user application to manipulate variables of these types, two macro's are added for comparison functionality. A separate chapter is added to the AVIX user manual describing the use of these macro's in combination with type safe types.

## Known Issues 2.2.0

**PR029:** dsPIC30 and dsPIC33 offer DSP instructions. When using those instructions with modulo addressing or bit-reversed addressing, AVIX-16 will crash. This will be fixed in a forthcoming version. As a workaround, with version 2.01 you should not use modulo or bit reversed addressing modes.

Regarding usage of DSP instructions: AVIX-16 does not allow more than one thread to use DSP instructions. This is by design and based on the fact that dsPIC30 and dsPIC33 do not allow the DSP state to be fully restored. Since this limitation is based on a hardware restriction there is no solution for this. In a forthcoming version the documentation will be enhanced accordingly.

## Fixed Issues 2.2.0

- **PR022: Macro AVIX\_EF\_RANGE contains error:**

`AVIX_EF_RANGE` is a macro to generate a bit field of consecutive logical true values to be used with event group functions. The macro contained an error resulting in a harmless compile time warning. Second the macro produced an invalid result when the first bit number argument to the macro was larger than the second bit number argument. Both issues are fixed in version 2.2.0.

## Documentation changes 2.2.0

AVIX-16 and AVIX-32 are ports targeting different controller families. Most of the content of the AVIX user manual is applicable to both ports. Differences exist in the buildtools, buildtool settings, some configuration parameter values and stack usage. To clearly document these differences a change to the user manual has been made. Chapter 6, 'Deployment' and chapter 7, 'Configuration' are moved to an Appendix A, describing these issues for AVIX-16. Added is appendix B, describing deployment and configuration issues for AVIX-32.

## Build and development environment issues 2.2.0

### Common

- **Passing structs by value:**

AVIX type safety is accomplished by a number of AVIX types being declared as a C structure. These small structures are passed by value to applicable functions. C30/C32 allows two different ways to pass values of small structs as function parameters or function result values. AVIX is built in a way these types of structures are passed in the most efficient way. In order for application code to be compatible with AVIX, the application sources must be compiled with the same mechanism. This is accomplished by setting compiler flag **no-pcc-struct-return** (**-fno-pcc-struct-return**). When building from MPLAB™, this flag can be set in the project options dialog through tab 'MPLAB™ C30' for AVIX-16 or tab 'MPLAB™ C32' for AVIX-32. When building from the command line, this flag can be passed like a regular compiler option.

*Failing to specify this flag when compiling application source files will result in a non working program without any warning being given.*

### AVIX-16

- **Compiler version:**

AVIX-16 is not compatible with C30 version 3.02 because of two Microchip confirmed bugs. These bugs are the following:

- Transparent unions (C30 keyword `transparent_union`) are not recognized by the compiler and result in an error. Part of the type safety offered by AVIX is implemented through this keyword.
- Passing structs by value creates a large stack overhead. C30 allows structs to be passed by value. Doing so with C30 version 3.02 results in a very large stack overhead and thus a waste of memory.

### AVIX-32

- **MPLAB™ 8.02 simulator not working:**

The MPLAB™ 8.02 simulator for PIC32 contains a number of Microchip confirmed bugs leading to the simulator not working for a number of PIC32 machine instruction sequences. These machine instruction sequences are valid sequences and are generated by the C32 compiler. In a number of places in AVIX-32 these instruction sequences occur, leading to an AVIX-32 based application not working in the simulator. As a result, an AVIX-32 based application can only be tested on hardware. Advised is to use an Explorer16 board with a PIC32 PIM or the PIC32 starter kit. Unknown is whether version 8.00 and 8.01 of MPLAB™ are hurt by the same bug but due to stability issues related to other bugs, using the PIC32 simulator from these versions of MPLAB™ is not advised.

- **C32 generates wrong Interrupt Service Routine code for priority 7**

Both C32 version 1.00 and 1.01 generate faulty code for a priority 7 Interrupt Service Routine (Microchip confirmed). As a result when using a priority 7 interrupt based on the standard C32 Interrupt Service Routine construct your application will fail. When needing priority 7 Interrupt Service Routines, use can be made of the AVIX-32 supplied macro `avixDeclareISRShadow` which has the added advantage of using the AVIX-32 system stack.

This page is intentionally left blank

## Summary 2.01

This section contains release notes for AVIX version 2.0.1 which is the successor to version 1.3.2

## New Features 2.01

- **Memory management:**  
Starting with this release, AVIX offers memory management. Pools of memory blocks can be created with a user specified number of memory blocks where each block in a pool has a user specified fixed size. Memory pools are very flexible. They can be shared between threads and Interrupt Service Routines. More details can be found in the user manual.
- **Message content type void\* added to allow pointer values to be used as message content:**  
To be able to send pointer values using messages two functions are added to the message mechanism to set a pointer value in the message body (avixMsg\_PutPtr) and retrieve a pointer value from a message body (avixMsg\_GetPtr).

## Improvements 2.01

The following improvements have been made:

- **Scheduler performance increase**  
The core of the scheduler has undergone a number of performance enhancements, leading to a significant increase of the schedule speed. The number of cycles taken by the scheduler core is decreased from ~100 to ~90.
- **Interrupt handling performance increase**  
The DIH (Deferred Interrupt Handler) mechanism has undergone a number of performance enhancements leading to a significant increase in the performance of interrupt handling.
- **Pipe handling performance increase**  
Internal enhancements have been made to the pipe mechanism leading to significant less overhead when using pipes from Interrupt handlers with small data amounts.

## Changes 2.01

This release contains a number of changes, some of which affect the API. The API related changes are minor and require a minimal effort to change existing code in order to adhere to the new interface.

- **Pipes extended with user data pointer**  
Pipes are extended with the possibility to add a pointer to user specified data which will be passed to the pipe callback function. Because of this change the interface of the pipe create function and the user supplied pipe callback function are extended:

```
extern tavixPipeId avixPipe_Create
(
    tavixKernelObjectName  pName,
    unsigned int            sizeInBytes,
    tavixPipeCallback       pCallback,
    void*                   pUserData    /* Added argument ! */
);

typedef void (*tavixPipeCallback)
(
    tavixPipeEvent event,
    int             nrBytes,
    void*           pUserData    /* Added argument ! */
);
```

*If any of these functions is used in your code, your code must be adapted to the new interface. When creating a pipe, for argument pUserData value NULL can be passed in which case the behavior is the*

same as in the previous release.

- Distribution** **model**  
 Changes have been made to the distribution model. Starting with this release, the Basic and Standard distribution are not built for a specific controller but, like the Extended Release, are controller neutral. Through configuration settings all three distribution types (Basic, Standard and Extended) can be used with any of the four controller families (24F, 24H, 30F or 33F). The difference between the three distributions is in the maximum number of kernel objects that can be created. Details can be found in the user manual. As a result of this change one of the error symbols has changed. Symbol:

AVIXE\_SYSTEM\_DEMO\_VERSION\_LIMIT\_REACHED

is no longer available and is replaced with:

AVIXE\_SYSTEM\_DISTRIBUTION\_LIMIT\_REACHED

*When symbol AVIXE\_SYSTEM\_DEMO\_VERSION\_LIMIT\_REACHED is used in your code, this must be replaced by AVIXE\_SYSTEM\_DISTRIBUTION\_LIMIT\_REACHED.*

- Initial semaphore value changed from greater or equal one ( $\geq 1$ ) to greater or equal zero ( $\geq 0$ )**  
 In the previous release the initial value a semaphore is created with had to be greater or equal to one. Since for semaphores it is desirable to be able to create one with an initial value greater or equal to zero, the semaphore API is changed likewise. As a result of this change and in order to be more in line with other error codes, one of the error symbols has changed. Symbol:

AVIXE\_SEMAPHORE\_INITIAL\_VALUE\_LESS\_OR\_EQUAL\_ZERO

is no longer available and is replaced with:

AVIXE\_SEMAPHORE\_INVALID\_ARGUMENT\_VALUE

*When symbol AVIXE\_SEMAPHORE\_INITIAL\_VALUE\_LESS\_OR\_EQUAL\_ZERO is used in your code, this must be replaced by AVIXE\_SEMAPHORE\_INVALID\_ARGUMENT\_VALUE.*

- Library name extended to reflect version number:**  
 The name of the library in the distribution is extended with a four digit number reflecting the version of the distribution. This makes it easier to recognize the library and helps in preventing errors when multiple versions are installed. The library name format is:

AVIXLibrary\_<v>.a where <v> is the placeholder for the version number.

For version 2.01 of AVIX the library name is: AVIXLibrary\_0201.a

When moving to a new version of the library you must change your project settings or build script accordingly.

## Known Issues 2.01

- PR0029:** dsPIC30 and dsPIC33 offer DSP instructions. When using those instructions with modulo addressing or bit-reversed addressing, AVIX will crash. This will be fixed in a forthcoming version. As a workaround, with version 2.01 you should not use modulo or bit reversed addressing modes.

Regarding usage of DSP instructions: AVIX does not allow more than one thread to use DSP instructions. This is by design and based on the fact that dsPIC30 and dsPIC33 do not allow the DSP state to be fully restored. Since this limitation is based on a hardware restriction there is no solution for this. In a forthcoming version the documentation will be enhanced accordingly.

## Fixed Issues 2.01

- **PR008: AVIXMutex.h is not self contained:**  
When including AVIXMutex.h without first including AVIXGeneric.h, an error is generated on a bool data type being unknown. This type is defined in AVIXGeneric.h. To solve this problem, AVIXMutex.h is changed to first include AVIXGeneric.h.
- **PR010: avixDeclareISR macro does not work when first statement after variable declaration**  
When avixDeclareISR is used immediately following a data declaration, the compiler might issue an error depending on the level of optimization. The macro is changed such that this error does no longer occur and avixDeclareISR can be used at any place in a source file.
- **PR017: Error in message mechanism leads to each subsequent message to be larger than the previous:**  
An error in the message mechanism caused each subsequent newly created message to be larger than the previous. This is fixed according the specification such that each allocated message has a body size as configured in AVIXSystemSettings.h

This problem occurred for newly created messages only. Once allocated and reused, a message retains its size.

This problem might need special attention. Messages used in your pre-2.01 application can contain more information than specified by the configured message size. As a result of this fix, when putting information in messages the amount of data is now correctly checked against the configured value which might lead to errors. If this occurs, the code where data is written to or read from a message must be checked to verify the amount of information is according your design. If so you might have to adjust the configuration value specifying the message body size.

*This problem is considered critical and therefore it is strongly advised to replace your current version of AVIX with this new version.*

- **PR018: AVIXSemaphore.h is not self contained:**  
When including AVIXSemaphore.h without first including AVIXGeneric.h, an error is generated on a bool data type being unknown. This type is defined in AVIXGeneric.h. To solve this problem, AVIXSemaphore.h is changed to first include AVIXGeneric.h.
- **PR019: Function avixThread\_Create uses wrong value for maximum priority check:**  
Function avixThread\_Create checks the priority specified for the newly created thread. The value for the maximum priority is defined in the configuration file AVIXSystemSettings.h. An error is fixed that the value specified by the user is not taken into account. The value used for avixThread\_Create was fixed on a maximum priority of 31, regardless the value specified by the user in AVIXSystemSettings.h. This could lead to two types of problems:
  - The user specified a value higher than 31 in AVIXSystemSettings.h: Regardless this setting, the priority for a newly created thread is restricted to a maximum value of 31, higher values lead to an error.
  - The user specified a value lower than 31 in AVIXSystemSettings.h: Creating a thread with a priority higher than the value specified by the user was possible, leading to memory corruption.

*This problem is considered critical and therefore it is strongly advised to replace your current version of AVIX with this new version.*