Here is a list of all modules:

- License Terms and Copyright Information
- Abbreviations and Definitions
- Overview
- Architecture Description
- APP Configuration Parameters
- Enumerations
- Data structures
- Methods
- Usage
- Release History
- INTERRUPT
License Terms and Copyright Information

Copyright (c) 2015, Infineon Technologies AG All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

To improve the quality of the software, users are encouraged to share modifications, enhancements or bug fixes with Infineon Technologies AG (dave@infineon.com).
Abbreviations and Definitions

Abbreviations:

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE™</td>
<td>Digital Application Virtual Engineer</td>
</tr>
<tr>
<td>APP</td>
<td>DAVE Application</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller Unit</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>LLD</td>
<td>Low Level Driver</td>
</tr>
<tr>
<td>SCU</td>
<td>System Control Unit</td>
</tr>
<tr>
<td>IO</td>
<td>Input Output</td>
</tr>
<tr>
<td>NVIC</td>
<td>Nested Vector Interrupt Controller</td>
</tr>
</tbody>
</table>

Definitions:

<table>
<thead>
<tr>
<th>Definitions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singleton</td>
<td>Only single instance of the APP is permitted</td>
</tr>
<tr>
<td>Sharable</td>
<td>Resource sharing with other APPs is permitted</td>
</tr>
<tr>
<td>initProvider</td>
<td>Provides the initialization routine</td>
</tr>
<tr>
<td>Physical connectivity</td>
<td>Hardware inter/intra peripheral (constant) signal connection</td>
</tr>
<tr>
<td>Conditional connectivity</td>
<td>Constrained hardware inter/intra peripheral signal connection</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Indicates consumption of low level (dependent) APPs</td>
</tr>
</tbody>
</table>
INTERRUPT

Overview

The INTERRUPT APP is a system APP. The Cortex-M vector table contains the address of the exception handlers and interrupts service routine (ISR). It allows the user to overwrite the provided default implementation of the interrupt service routine and to set the interrupt priority.

The user needs to provide the implementation of the ISR. The user has also the choice to enable the interrupt at initialization.

The INTERRUPT APP requires the CPU APP to be informed about the number of priority levels and in case of Cortex-M4 also the number of subpriority levels.

The INTERRUPT APP based on the peripheral service request connectivity resolves the NVIC IRQ node to be used.

Supported Devices

1. XMC4800/XMC4700 Series
2. XMC4500 Series
3. XMC4400 Series
4. XMC4300 Series
5. XMC4200 / XMC4100 Series
6. XMC1400 Series
7. XMC1300 Series
8. XMC1200 Series
9. XMC1100 Series

References
1. XMC4800/XMC4700 Reference Manual
2. XMC4500 Reference Manual
5. XMC4200 / XMC4100 Reference Manual
7. XMC1300 Reference Manual
The above diagram represents the internal software architecture of the INTERRUPT APP and its instance exists in a DAVE™ project with fixed attributes as shown. Each instance of this APP consumes one NVIC node in the MCU. The INTERRUPT APP also provides input signal for inter-peripheral connections.

An instantiated APP generates (after code generation) a specific data structure with the GUI configuration. The name of this data structure can be modified by changing the APP instance label (e.g. change label from default INTERRUPT_0 to MY_INTERRUPT).
**Signals:**

The following table describes the list of IO signals for INTERRUPT APP.

**Table 1:** APP Input Output signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Input/Output</th>
<th>Availability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sr_irq</td>
<td>Input</td>
<td>Always</td>
<td>This signal can be used to connect to any other source which can generate interrupt.</td>
</tr>
</tbody>
</table>
INTERRUPT

APP Configuration Parameters

App Configuration Parameters

<table>
<thead>
<tr>
<th>Interrupt Settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable interrupt at initialization</td>
<td></td>
</tr>
<tr>
<td>Interrupt Priority</td>
<td></td>
</tr>
<tr>
<td>Preemption priority</td>
<td>63</td>
</tr>
<tr>
<td>Interrupt handler: UserIRQHandler</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Interrupt Settings
**Enumerations**

<table>
<thead>
<tr>
<th>enum</th>
<th>INTERRUPT_STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INTERRUPT_STATUS_SUCCESS = 0U,</td>
</tr>
<tr>
<td></td>
<td>INTERRUPT_STATUS_FAILURE = 1U</td>
</tr>
</tbody>
</table>

typedef enum INTERRUPT_STATUS INTERRUPT_STATUS_t
Typedef Documentation

typedef enum INTERRUPT_STATUS INTERRUPT_STATUS_t

Initialization status.
Enumeration Type Documentation

enum INTERRUPT_STATUS

Initialization status.

**Enumerator:**

*INTERRUPT_STATUS_SUCCESS*  APP initialization success

*INTERRUPT_STATUS_FAILURE*  APP initialization failure

Definition at line 94 of file INTERRUPT.h.
# INTERRUPT

<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
</tr>
</thead>
</table>

**Data structures**
### Data Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td><code>INTERRUPT</code></td>
<td>This structure holds run-time configurations of <code>INTERRUPT</code> APP. More...</td>
</tr>
<tr>
<td>typedef struct</td>
<td><code>INTERRUPT</code></td>
<td>This structure holds run-time configurations of <code>INTERRUPT</code> APP.</td>
</tr>
</tbody>
</table>
## INTERRUPT

<table>
<thead>
<tr>
<th>Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DAVE_APP_VERSION_t</code></td>
<td><code>INTERRUPT_GetAppVersion</code></td>
<td>(void) Get INTERRUPT APP version.</td>
</tr>
<tr>
<td></td>
<td><code>INTERRUPT_Init</code></td>
<td>(<code>const INTERRUPT_t *const handler</code>) Initializes INTERRUPT APP instance.</td>
</tr>
<tr>
<td><code>__STATIC_INLINE void</code></td>
<td><code>INTERRUPT_Enable</code></td>
<td>(<code>const INTERRUPT_t *const handler</code>) Enables the IRQ.</td>
</tr>
<tr>
<td><code>__STATIC_INLINE void</code></td>
<td><code>INTERRUPT_Disable</code></td>
<td>(<code>const INTERRUPT_t *const handler</code>) Disables the IRQ.</td>
</tr>
<tr>
<td><code>__STATIC_INLINE uint32_t</code></td>
<td><code>INTERRUPT_GetPending</code></td>
<td>(<code>const INTERRUPT_t *const handler</code>) Get the pending IRQ.</td>
</tr>
<tr>
<td><code>__STATIC_INLINE void</code></td>
<td><code>INTERRUPT_SetPending</code></td>
<td>(<code>const INTERRUPT_t *const handler</code>) Set the IRQ to pending state.</td>
</tr>
<tr>
<td><code>__STATIC_INLINE void</code></td>
<td><code>INTERRUPT_ClearPending</code></td>
<td>(<code>const INTERRUPT_t *const handler</code>) Clears the pending status of the IRQ.</td>
</tr>
<tr>
<td><code>__STATIC_INLINE uint32_t</code></td>
<td><code>INTERRUPT_GetActive</code></td>
<td>(<code>const INTERRUPT_t *const handler</code>) Get current running active status of the IRQ. This API is applicable only for XMC4000 devices.</td>
</tr>
</tbody>
</table>
Function Documentation

__STATIC_INLINE void INTERRUPT_ClearPending (const INTERRUPT_t handle)

Clears the pending status of the IRQ.

Parameters:

  handle Constant pointer to constant structure of type INTERRUPT_t

Returns:

  None

Example: Pre-requisite: Instantiate two instances of INTERRUPT APP

```c
#include <DAVE.h>

uint32_t pend_IRQ;
int main(void)
{
  DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
  INTERRUPT_Enable(&INTERRUPT_0);
  while(1)
  {
    return 0;
  }

  void MyISR_handler(void)
  {
    INTERRUPT_Enable(&INTERRUPT_1);
    INTERRUPT_SetPending(&INTERRUPT_1);
    pend_IRQ = INTERRUPT_GetPending(&INTERRUPT_1);
    if(pend_IRQ)
```
Definition at line 324 of file INTERRUPT.h.

References INTERRUPT::node.

__STATIC_INLINE void INTERRUPT_Disable (const INTERRUPT_t *

Disables the IRQ.

Parameters:
  handle Constant pointer to constant structure of type
  INTERRUPT_t

Returns:
  None

Example: Pre-requisite: Instantiate one instance of INTERRUPT

APP

#include <DAVE.h>

int main(void)
{
  DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
  INTERRUPT_Disable(&INTERRUPT_0);
  while(1)
  {}
  return 0;
__STATIC_INLINE void INTERRUPT_Enable ( const INTERRUPT_t * 

Enables the IRQ.

Parameters:

handle Constant pointer to constant structure of type INTERRUPT_t

Returns:
None

Example: Pre-requisite: Instantiate one instance of INTERRUPT_APP

```c
#include <DAVE.h>

int main(void)
{
    DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
    INTERRUPT_Enable(&INTERRUPT_0);
    while(1)
    {
        return 0;
    }
}
```

Definition at line 210 of file INTERRUPT.h.
__STATIC_INLINE uint32_t INTERRUPT_GetActive (const INTERRUPT_t* handle)

Get current running active status of the IRQ. This API is applicable only for XMC4000 devices.

**Parameters:**
- **handle** Constant pointer to constant structure of type INTERRUPT_t

**Returns:**
- uint32_t current active running IRQ node

**Example:** Pre-requisite: Instantiate one instance of INTERRUPT APP

```c
#include <DAVE.h>

int main(void)
{
    uint32_t Status;
    DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
    Status = INTERRUPT_GetActive(&INTERRUPT_0);
    while(1)
    {
        return 0;
    }
}
```

Definition at line 352 of file INTERRUPT.h.

References INTERRUPT::node.
DAVE_APP_VERSION_t INTERRUPT_GetAppVersion ( void )

Get INTERRUPT APP version.

**Returns:**
DAVE_APP_VERSION_t APP version information (major, minor and patch number)

**Description:**
The function can be used to check application software compatibility with a specific version of the APP.

```c
#include <DAVE.h>

int main(void)
{
    DAVE_APP_VERSION_t version;
    DAVE_Init();
    version = INTERRUPT_GetAppVersion();
    if(version.major != 4U)
    {
    }
    while(1)
    {
        return 0;
    }
}
```

Definition at line 79 of file INTERRUPT.c.

__STATIC_INLINE uint32_t INTERRUPT_GetPending ( const INTERRUPT_t )

Get the pending IRQ.

**Parameters:**
**handle** Constant pointer to constant structure of type

**INTERRUPT_t**

**Returns:**

uint32_t IRQ node

**Example:** Pre-requisite: Instantiate one instance of **INTERRUPT**

```c
#include <DAVE.h>

int main(void)
{
    uint32_t Status;
    DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
    Status = INTERRUPT_GetPending(&INTERRUPT_0);
    while(1)
    {
        return 0;
    }
}
```

Definition at line 261 of file **INTERRUPT.h**.

References **INTERRUPT::node**.

**INTERRUPT_STATUS_t INTERRUPT_Init (const INTERRUPT_t *cor**

Initializes **INTERRUPT** APP instance.

**Parameters:**

**handle** Constant pointer to constant structure of type

**INTERRUPT_t**
Returns: `INTERRUPT_STATUS_t`

Example: Pre-requisite: Instantiate one instance of `INTERRUPT APP`

```c
#include <DAVE.h>

int main(void)
{
    DAVE_Init(); // INTERRUPT_Init(&INTERRUPT_0) is called within DAVE_Init()
    while(1)
    {}
    return 0;
}
```

Definition at line 93 of file `INTERRUPT.c`.

References `INTERRUPT::enable_at_init`, `INTERRUPT_Enable()`, `INTERRUPT_STATUS_SUCCESS`, `INTERRUPT::irqctrl`, `INTERRUPT::node`, `INTERRUPT::priority`, and `INTERRUPT::subpriority`.

```c
__STATIC_INLINE void INTERRUPT_SetPending ( const INTERRUPT_t handle )

Set the IRQ to pending state.

Parameters:
- **handle**: Constant pointer to constant structure of type `INTERRUPT_t`

Returns: None
```
Example: Pre-requisite: Instantiate one instance of INTERRUPT APP

```c
#include <DAVE.h>

int main(void)
{
    DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
    INTERRUPT_SetPending(&INTERRUPT_0);
    while(1)
    {
        return 0;
    }
}
```

Definition at line 286 of file INTERRUPT.h.

References INTERRUPT::node.
INTERRUPT

Usage

**INTERRUPT** is a global DAVE™ APP. It is used by other APPs which required interrupt connectivity. For information on how **INTERRUPT** is being used, refer for example to the ERU related APPs help documentation.
# INTERRUPT

## Home

## Release History

Release History

---

## Release History

---
# Data Structures

<table>
<thead>
<tr>
<th>struct</th>
<th>INTERRUPT</th>
<th>This structure holds run-time configurations of INTERRUPT APP. More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT_STATUS_t</td>
<td>INTERRUPT_Init (const INTERRUPT_t *const handler)</td>
<td>Initializes INTERRUPT APP instance.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td>INTERRUPT_Enable (const INTERRUPT_t *const handler)</td>
<td>Enables the IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td>INTERRUPT_Disable (const INTERRUPT_t *const handler)</td>
<td>Disables the IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE uint32_t</td>
<td>INTERRUPT_GetPending (const INTERRUPT_t *const handler)</td>
<td>Get the pending IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td>INTERRUPT_SetPending (const INTERRUPT_t *const handler)</td>
<td>Set the IRQ to pending state.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td>INTERRUPT_ClearPending (const INTERRUPT_t *const handler)</td>
<td>Clears the pending status of the IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE uint32_t</td>
<td>INTERRUPT_GetActive (const INTERRUPT_t *const handler)</td>
<td>Get current running active status of the IRQ. This API is applicable only for XMC4000 devices.</td>
</tr>
</tbody>
</table>
**Function Documentation**

__STATIC_INLINE__ void **INTERRUPT_ClearPending** (const **INTERRUPT_t**)

Clears the pending status of the IRQ.

**Parameters:**

- **handle** Constant pointer to constant structure of type **INTERRUPT_t**

**Returns:**

None

**Example:** Pre-requisite: Instantiate two instances of **INTERRUPT**

```c
#include <DAVE.h>

uint32_t pend_IRQ;
int main(void)
{
    DAVE_Init(); // **INTERRUPT_Init()** is called within **DAVE_Init()**
    **INTERRUPT_Enable**(&**INTERRUPT_0**);
    while(1)
    {
        return 0;
    }
}

void MyISR_handler(void)
{
    **INTERRUPT_Enable**(&**INTERRUPT_1**);
    **INTERRUPT_SetPending**(&**INTERRUPT_1**);
    pend_IRQ = **INTERRUPT_GetPending**(&**INTERRUPT_1**);
    if(pend_IRQ)
```

{
    INTERRUPT_Disable(&INTERRUPT_0);
    INTERRUPT_ClearPending(&INTERRUPT_1);
}

Definition at line 324 of file INTERRUPT.h.

References node.

__STATIC_INLINE void INTERRUPT_Disable ( const INTERRUPT_t * const handle )

Disables the IRQ.

Parameters:
    handle Constant pointer to constant structure of type INTERRUPT_t

Returns:
    None

Example: Pre-requisite: Instantiate one instance of INTERRUPT APP

#include <DAVE.h>

int main(void)
{
    DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
    INTERRUPT_Disable(&INTERRUPT_0);
    while(1)
    
    return 0;
__STATIC_INLINE void INTERRUPT_Enable ( const INTERRUPT_t * const)

Enables the IRQ.

Parameters:

handle Constant pointer to constant structure of type INTERRUPT_t

Returns:

None

Example: Pre-requisite: Instantiate one instance of INTERRUPT APP

```
#include <DAVE.h>

int main(void)
{
    DAVE_Init();  // INTERRUPT_Init() is called within DAVE_Init()
    INTERRUPT_Enable(&INTERRUPT_0);
    while(1)
    {
        return 0;
    }
}
```


## References

node.

Referenced by `INTERRUPT_Init()`.

__STATIC_INLINE uint32_t `INTERRUPT_GetActive` (const `INTERRUPT_t`

Get current running active status of the IRQ. This API is applicable only for XMC4000 devices.

**Parameters:**

- **handle** Constant pointer to constant structure of type `INTERRUPT_t`

**Returns:**

- `uint32_t` current active running IRQ node

**Example:** Pre-requisite: Instantiate one instance of `INTERRUPT APP`

```c
#include <DAVE.h>

int main(void)
{
    uint32_t Status;
    DAVE_Init(); // `INTERRUPT_Init()` is called within `DAVE_Init()`
    Status = `INTERRUPT_GetActive`(&`INTERRUPT_0`);
    while(1)
    {
        return 0;
    }
}
```

Definition at line 352 of file `INTERRUPT.h`.

References node.
__STATIC_INLINE uint32_t INTERRUPT_GetPending ( const INTERRUPT_t *handle )

Get the pending IRQ.

**Parameters:**

- `handle` Constant pointer to constant structure of type `INTERRUPT_t`

**Returns:**

- `uint32_t` IRQ node

**Example:** Pre-requisite: Instantiate one instance of `INTERRUPT_APP`

```c
#include <DAVE.h>

int main(void)
{
    uint32_t Status;
    DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
    Status = INTERRUPT_GetPending(&INTERRUPT_0);
    while(1)
    {
        return 0;
    }
}
```

Definition at line 261 of file `INTERRUPT.h`.

References `node`.
Initializes `INTERRUPT` APP instance.

**Parameters:**
- `handle` Constant pointer to constant structure of type `INTERRUPT_t`

**Returns:**
- `INTERRUPT_STATUS_t`

**Example:** Pre-requisite: Instantiate one instance of `INTERRUPT` APP

```c
#include <DAVE.h>

int main(void)
{
    DAVE_Init(); // INTERRUPT_Init(&INTERRUPT_0) is called within DAVE_Init()
    while(1)
    {
        return 0;
    }
}
```

Definition at line 93 of file `INTERRUPT.c`.

References `enable_at_init`, `INTERRUPT_Enable()`, `INTERRUPT_STATUS_SUCCESS`, `irqctrl`, `node`, `priority`, and `subpriority`.

```c
__STATIC_INLINE void INTERRUPT_SetPending ( const INTERRUPT_t
```

Set the IRQ to pending state.

**Parameters:**
**handle** Constant pointer to constant structure of type **INTERRUPT_t**

**Returns:**
None

**Example:** Pre-requisite: Instantiate one instance of **INTERRUPT_APP**

```c
#include <DAVE.h>

int main(void)
{
    DAVE_Init(); // INTERRUPT_Init() is called within DAVE_Init()
    INTERRUPT_SetPending(&INTERRUPT_0);
    while(1)
    {
        return 0;
    }
}
```

Definition at line 286 of file **INTERRUPT.h**.

References **node**.
## INTERRUPT

Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>INTERRUPT</th>
<th>This structure holds run-time configurations of INTERRUPT APP</th>
</tr>
</thead>
</table>
### INTERRUPT Struct Reference

INTERRUPT | Data structures

---
Detailed Description

This structure holds run-time configurations of INTERRUPT APP.

Definition at line 114 of file INTERRUPT.h.

#include <INTERRUPT.h>
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>const XMC_SCU_IRQCTRL_t</td>
<td>irqctrl</td>
</tr>
<tr>
<td>const IRQn_Type</td>
<td>node</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>priority</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>subpriority</td>
</tr>
<tr>
<td>const bool</td>
<td>enable_at_init</td>
</tr>
</tbody>
</table>
Field Documentation

**const bool** `INTERRUPT::enable_at_init`  
Interrupt enable for Node  
Definition at line `124` of file `INTERRUPT.h`.  
Referenced by `INTERRUPT_Init()`.

**const XMC_SCU_IRQCTRL_t** `INTERRUPT::irqctrl`  
selects the interrupt source for a NVIC interrupt node  
Definition at line `117` of file `INTERRUPT.h`.  
Referenced by `INTERRUPT_Init()`.

**const IRQn_Type** `INTERRUPT::node`  
Mapped NVIC Node  
Definition at line `119` of file `INTERRUPT.h`.  
Referenced by `INTERRUPT_ClearPending()`, `INTERRUPT_Disable()`, `INTERRUPT_Enable()`, `INTERRUPT_GetActive()`, `INTERRUPT_GetPending()`, `INTERRUPT_Init()`, and `INTERRUPT_SetPending()`.

**const uint8_t** `INTERRUPT::priority`  
Node Interrupt Priority  
Definition at line `120` of file `INTERRUPT.h`. 
Referenced by `INTERRUPT_Init()`.

```cpp
const uint8_t INTERRUPT::subpriority
```

Node Interrupt SubPriority only valid for XMC4x
Definition at line 122 of file `INTERRUPT.h`.

Referenced by `INTERRUPT_Init()`.

The documentation for this struct was generated from the following file:

- `INTERRUPT.h`
Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- `enable_at_init` : INTERRUPT
- `irqctrl` : INTERRUPT
- `node` : INTERRUPT
- `priority` : INTERRUPT
- `subpriority` : INTERRUPT
• `enable_at_init` : INTERRUPT
• `irqctrl` : INTERRUPT
• `node` : INTERRUPT
• `priority` : INTERRUPT
• `subpriority` : INTERRUPT
Here is a list of all documented files with brief descriptions:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT.c</td>
<td>[code]</td>
</tr>
<tr>
<td>INTERRUPT.h</td>
<td>[code]</td>
</tr>
</tbody>
</table>
### INTERRUPT

<table>
<thead>
<tr>
<th>Home</th>
<th>File List</th>
<th>Globals</th>
<th>Functions</th>
</tr>
</thead>
</table>

**INTERRUPT.c File Reference**
Detailed Description

Date:
2015-09-18

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file INTERRUPT.c.

#include "interrupt.h"
## Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DAVE_APP_VERSION_t</code></td>
<td><code>INTERRUPT_GetAppVersion</code> (void)</td>
<td>Get <code>INTERRUPT</code> APP version.</td>
</tr>
<tr>
<td><code>INTERRUPT_STATUS_t</code></td>
<td><code>INTERRUPT_Init</code> (const <code>INTERRUPT_t</code> *const handler)</td>
<td>Initializes <code>INTERRUPT</code> APP instance.</td>
</tr>
</tbody>
</table>

Go to the source code of this file.
INTERRUPT

INTERRUPT.h File Reference
Detailed Description

Date:
2015-10-05

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file **INTERRUPT.h**.

```c
#include <xmc_common.h> #include <DAVE_Common.h>
#include <xmc_scu.h>
#include "interrupt_conf.h"
#include "interruptExtern.h"
```
**Data Structures**

| struct INTERRUPT | This structure holds run-time configurations of INTERRUPT APP. More... |
typedef struct INTERRUPT INTERRUPT_t
This structure holds run-time configurations of INTERRUPT APP.
## Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td><code>INTERRUPT_GetAppVersion (void)</code>&lt;br&gt;Get <strong>INTERRUPT</strong> APP version.</td>
</tr>
<tr>
<td>INTERRUPT_STATUS_t</td>
<td><code>INTERRUPT_Init (const INTERRUPT_t *const handler)</code>&lt;br&gt;Initializes <strong>INTERRUPT</strong> APP instance.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td><code>INTERRUPT_Enable (const INTERRUPT_t *const handler)</code>&lt;br&gt;Enables the IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td><code>INTERRUPT_Disable (const INTERRUPT_t *const handler)</code>&lt;br&gt;Disables the IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE uint32_t</td>
<td><code>INTERRUPT_GetPending (const INTERRUPT_t *const handler)</code>&lt;br&gt;Get the pending IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td><code>INTERRUPT_SetPending (const INTERRUPT_t *const handler)</code>&lt;br&gt;Set the IRQ to pending state.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td><code>INTERRUPT_ClearPending (const INTERRUPT_t *const handler)</code>&lt;br&gt;Clears the pending status of the IRQ.</td>
</tr>
<tr>
<td>__STATIC_INLINE uint32_t</td>
<td><code>INTERRUPT_GetActive (const INTERRUPT_t *const handler)</code>&lt;br&gt;Get current running active status of the IRQ. This API is applicable only for XMC4000 devices.</td>
</tr>
</tbody>
</table>

**INTERRUPT_STATUS**

*INTERRUPT_STATUS_SUCCESS*
enum

interrupt_status_failure = 1U

typedef enum interrupt_status_t

Go to the source code of this file.
Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- INTERRUPT_ClearPending() : INTERRUPT.h
- INTERRUPT_Disable() : INTERRUPT.h
- INTERRUPT_Enable() : INTERRUPT.h
- INTERRUPT_GetActive() : INTERRUPT.h
- INTERRUPT_GetAppVersion() : INTERRUPT.c, INTERRUPT.h
- INTERRUPT_GetPending() : INTERRUPT.h
- INTERRUPT_Init() : INTERRUPT.c, INTERRUPT.h
- INTERRUPT_SetPending() : INTERRUPT.h
- INTERRUPT_STATUS : INTERRUPT.h
- INTERRUPT_STATUS_FAILURE : INTERRUPT.h
- INTERRUPT_STATUS_SUCCESS : INTERRUPT.h
- INTERRUPT_STATUS_t : INTERRUPT.h
- INTERRUPT_t : INTERRUPT.h
### INTERRUPT

<table>
<thead>
<tr>
<th>Function</th>
<th>File(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT_ClearPending()</td>
<td>INTERRUPT.h</td>
</tr>
<tr>
<td>INTERRUPT_Disable()</td>
<td>INTERRUPT.h</td>
</tr>
<tr>
<td>INTERRUPT_Enable()</td>
<td>INTERRUPT.h</td>
</tr>
<tr>
<td>INTERRUPT_GetActive()</td>
<td>INTERRUPT.h</td>
</tr>
<tr>
<td>INTERRUPT_GetAppVersion()</td>
<td>INTERRUPT.c, INTERRUPT.h</td>
</tr>
<tr>
<td>INTERRUPT_GetPending()</td>
<td>INTERRUPT.h</td>
</tr>
<tr>
<td>INTERRUPT_Init()</td>
<td>INTERRUPT.c, INTERRUPT.h</td>
</tr>
<tr>
<td>INTERRUPT_SetPending()</td>
<td>INTERRUPT.h</td>
</tr>
</tbody>
</table>
• INTERRUPT_STATUS_t: INTERRUPT.h
• INTERRUPT_t: INTERRUPT.h
• INTERRUPT_STATUS : INTERRUPT.h
• INTERRUPT_STATUS_FAILURE : INTERRUPT.h
• INTERRUPT_STATUS_SUCCESS : INTERRUPT.h
INTERRUPT

INTERRUPT.h

Go to the documentation of this file.

00001
00002 #ifndef INTERRUPT_H
00003 #define INTERRUPT_H
00004
00005 /***************************************************************************/
00006 /* HEADER FILES */
00007 /***************************************************************************/
00008 #include <xmc_common.h>
00009 #include <DAVE_Common.h>
00010
00011 #if (UC_SERIES == XMC14)
00012 #include <xmc_scu.h>
00013 #endif
00014
00015 #include "interrupt_conf.h"
00016
00017 /***************************************************************************/
00018 /* MACROS */
00019 /***************************************************************************/
00020
typedef enum INTERRUPT_STATUS
{
    INTERRUPT_STATUS_SUCCESS = 0U,
    INTERRUPT_STATUS_FAILURE = 1U
} INTERRUPT_STATUS_t;

typedef struct INTERRUPT
{
    #if(UC_SERIES == XMC14)
        const XMC_SCU_IRQCTRL_t irqctrl;
    #endif
    const IRQn_Type node;
    const uint8_t priority;
    #if(UC_FAMILY == XMC4)
        const uint8_t subpriority;
    #endif
    const bool enable_at_init;
} INTERRUPT_t;

* API PROTOTYPES
*****************************************
**************************************************
************************/

```c
#ifdef __cplusplus
extern "C" {
#endif

DAVE_APP_VERSION_t INTERRUPT_GetAppVersion(void);

INTERRUPT_STATUS_t INTERRUPT_Init(const INTERRUPT_t *const handler);

__STATIC_INLINE void INTERRUPT_Enable(const INTERRUPT_t *const handler)
{
    XMC_ASSERT("Handler NULL", (handler != NULL));
    NVIC_EnableIRQ(handler->node);
}

__STATIC_INLINE void INTERRUPT_Disable(const INTERRUPT_t *const handler)
{
    XMC_ASSERT("Handler NULL", (handler != NULL));
    NVIC_DisableIRQ(handler->node);
}

__STATIC_INLINE uint32_t INTERRUPT_GetPending(const INTERRUPT_t *const handler)
{
    XMC_ASSERT("Handler NULL", (handler != NULL));
    return NVIC_GetPendingIRQ(handler->node);
}
```

```c
} #endif
```

__STATIC_INLINE void INTERRUPT_SetPending(const INTERRUPT_t *const handler) {
    XMC_ASSERT("Handler NULL", (handler != NULL));
    NVIC_SetPendingIRQ(handler->node);
}

__STATIC_INLINE void INTERRUPT_ClearPending(const INTERRUPT_t *const handler) {
    XMC_ASSERT("Handler NULL", (handler != NULL));
    NVIC_ClearPendingIRQ(handler->node);
}

#if(UC_FAMILY == XMC4)
__STATIC_INLINE uint32_t INTERRUPT_GetActive(const INTERRUPT_t *const handler) {
    XMC_ASSERT("Handler NULL", (handler != NULL));
    return NVIC_GetActive(handler->node);
}
#endif

#ifdef __cplusplus
}
#endif
#include "interruptExtern.h"
#endif
INTERRUPT.c

Go to the documentation of this file.

```c
00001
00055  /******************************************************************************************
00056  ************************************************************
00057  ************************************************************
00058  * HEADER FILES
00059  ***************************************************************
00060  ***************************************************************
00061  #include "interrupt.h"
00062  ***************************************************************
00063  ***************************************************************
00064  * MACROS
00065  ***************************************************************
00066  * LOCAL DATA
00067  ***************************************************************
00068```

LOCAL ROUTINES

API IMPLEMENTATION

/* API to retrieve the version of the INTERRUPT_APP */

DAVE_APP_VERSION_t INTERRUPT_GetAppVersion(void)
{
    DAVE_APP_VERSION_t version;
    version.major = INTERRUPT_MAJOR_VERSION;
    version.minor = INTERRUPT_MINOR_VERSION;
    version.patch = INTERRUPT_PATCH_VERSION;
    return (version);
}

/* API to initialize the INTERRUPT_APP */

INTERRUPT_STATUS_t INTERRUPT_Init(const INTERRUPT_t *const handler)
{
XMC_ASSERT("INTERRUPT_Init:HandlePtr NULL", (handler != NULL));

#if(UC_FAMILY == XMC4)
  NVIC_SetPriority(handler->node,
        NVIC_EncodePriority(NVIC_GetPriorityGrouping(),
        handler->priority,
        handler->subpriority));
  if (handler->enable_at_init == true) {
    INTERRUPT_Enable(handler);
  }
#else
  NVIC_SetPriority(handler->node, handler->priority);
  #if(UC_SERIES == XMC14)
    XMC_SCU_SetInterruptControl((uint8_t)handler->node,
                  (XMC_SCU_IRQCTRL_t)((handler->node << 8) |
                          handler->irqctrl));
  #endif
  /* Enable the interrupt if enable_at_init is enabled */
  if (handler->enable_at_init == true) {
    INTERRUPT_Enable(handler);
  }
#endif
return (INTERRUPT_STATUS_SUCCESS);
00124 }