

How to use unsafe pointers

version	1.1.1
scope	Example. This code is provided as example code for a user to base their code on.
description	How to use unsafe pointers
boards	Unless otherwise specified, this example runs on the SliceKIT Core Board, but can easily be run on any XN device by using a different XN file.

An `unsafe` pointer type is provided for compatibility with C and to implement dynamic, aliasing data structures (for example linked lists). This is not the default pointer type and the onus is on the programmer to ensure memory safety for these types.

An `unsafe` pointer is opaque unless accessed in an `unsafe` region. A function can be marked as `unsafe` to show that its body is an `unsafe` region:

```
unsafe void f(int * unsafe x) {
    // We can dereference x in here,
    // but be careful - it may point to garbage
    printintln(*x);
}
```

Unsafe functions can only be called from `unsafe` regions. You can make a local `unsafe` region by marking a compound statement as `unsafe`:

```
void g(int * unsafe p) {
    int i = 99;
    unsafe {
        p = &i;
        f(p);
    }
    // Cannot dereference p or call f from here
}
```

These regions allow the programmer to manage the parts of their program that are safe by construction and the parts that require the programmer to ensure safety.

Within `unsafe` regions, `unsafe` pointers can be explicitly cast to `safe` pointers - providing a contract from the programmer that the pointer can be regarded as `safe` from then on.

It is undefined behavior for an unsafe pointer to be written from one task and read from another.



Copyright © 2013, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the "Information") and is providing it to you "AS IS" with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.