xTimeComposer supports the following pragmas.

`#pragma unsafe arrays`
: (XC Only) This pragma disables the generation of run-time safety checks that prevent indexing an invalid array element within the scope of the next `do`, `while` or `for` statement in the current function; outside of a function the pragma applies to the next function definition.

`#pragma loop unroll (n)`
: (XC only) This pragma controls the number of times the next `do`, `while` or `for` loop in the current function is unrolled. `n` specifies the number of iterations to unroll, and unrolling is performed only at optimization level 01 and higher. Omitting the `n` parameter causes the compiler to try and fully unroll the loop. Outside of a function the pragma is ignored. The compiler produces a warning if unable to perform the unrolling.

`#pragma stackfunction n`
: This pragma allocates `n` words (`int`s) of stack space for the next function declaration in the current translation unit.

`#pragma stackcalls n`
: (XC only) This pragma allocates `n` words (`int`s) of stack space for any function called in the next statement. If the next statement does not contain a function call then the pragma is ignored; the next statement may appear in another function.

`#pragma ordered`
: (XC only) This pragma controls the compilation of the next `select` statement. This select statement is compiled in a way such that if multiple events are ready when the select starts, cases earlier in the select statement are selected in preference to ones later on.

`#pragma select handler`
: (XC only) This pragma indicates that the next function declaration is a select handler. A select handler can be used in a select case, as shown in the example below.

```
#pragma select handler
void f(chanend c, int &token, int &data);

...
select {
  case f(c, token, data):
    ...
    break;
}
...
```

The effect is to enable an event on the resource that is the first argument to the function. If the event is taken, the body of the select handler is executed before the body of the case.

The first argument of the select handler must have transmissive type and the return type must be `void`.

If the resource has associated state, such as a condition, then the select will not alter any of that state before waiting for events.

`#pragma fallthrough`
(XC only) This pragma indicates that the following switch case is expected to fallthrough to the next switch case without a break or return statement. This will suppress any warnings/errors from the compiler due to the fallthrough.

`#pragma xta label "name"`
This pragma provides a label that can be used to specify timing constraints.

`#pragma xta endpoint "name"`
(XC only) This pragma specifies an endpoint. It may appear before an input or output statement.

`#pragma xta call "name"`
(XC only) This pragma defines a label for a (function) call point. Use to specify a particular called instance of a function. For example, if a function contains a loop, the iterations for this loop can be set to a different value depending on which call point the function was called from.

`#pragma xta command "command"`
(XC only) This pragma allows XTA commands to be embedded into source code. All commands are run every time the binary is loaded into the XTA. Commands are executed in the order they occur in the file, but the order between commands in different source files is not defined.

`#pragma xta loop (integer)`
(XC only) This pragma applies the given loop XTA iterations to the loop containing the pragma.