NISTIR 5971

StopWatch User's Guide Version 1.0

William F. Mitchell
U. S. Department of Commerce
Technology Administration
National Institute of Standards and Technology
Information Technology Laboratory
Gaithersburg, MD 20899 USA

March 3, 1997

StopWatch User's Guide Version 1.0

William F. Mitchell
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899 USA

March 3, 1997

Abstract

StopWatch is a Fortran 90 module for portable, easy-to-use measurement of execution time. It supports four clocks – wall clock, CPU clock, user CPU clock and system CPU clock – and returns all times in seconds. It provides a simple means of determining which clocks are available, and the precision of those clocks. StopWatch is used by instrumenting your code with subroutine calls that mimic the operation of a stop watch. StopWatch supports multiple watches, and provides the concept of watch groups to allow functions to operate on multiple watches simultaneously.

Contents

Chapter 1

Introduction

StopWatch is a Fortran 90 module for measuring execution time of program segments. Measuring execution time is an important part of software development, especially for benchmarking and performance tuning. Unfortunately, Fortran has never supported the measurement of execution time, except through non-portable vendor extensions. Fortran 90 introduced a subroutine for measuring wall clock time, but overlooked the more desirable CPU time. It is anticipated that the next Fortran standard, Fortran 95, will include a CPU time subroutine, but it does not break the time into "user" and "system" time like many CPU clock routines, and the standard still does not guarantee that either the wall clock or CPU clock routines will necessarily contain clock information. Moreover, direct use of the routines can be unwieldy, requiring multiple variables to keep track of returned values, differencing the returned values, and conversion of the values to useful units.

StopWatch is designed to be a portable, easy-to-use means of measuring execution time. It supports the wall clock, CPU clock, a breakdown of the CPU clock into user and system times, and returns all times in seconds. It provides a simple means of determining which clocks are available, and the precision of those clocks. It is written in a style that allows it to be used with the subset languages ELF90 and F, as well as full Fortran 90 and Fortran 95 compilers.

StopWatch is used by instrumenting your code with subroutine calls that mimic the operation of a stop watch. The primary routines are **start_watch**, **stop_watch**, **reset_watch**, **read_watch** and **print_watch**. StopWatch supports multiple watches, and provides the concept of watch groups to allow functions to operate on multiple watches simultaneously.

Chapter 2

Quick Start

This section provides just enough information to start using the basic features of STOPWATCH. If you run into trouble or want to learn about the advanced features, read the rest of the STOPWATCH User's Guide and the man pages.

1. Select a *makefile* that matches the configuration of your system. The makefile names are of the form *mf.<os>.<compiler>.<cpusec>* where *<os>* is the operating system, *<compiler>* is the Fortran 90 compiler, and *<cpusec>* is the form of subroutine **cpu_second**. If you don't find your system, select a *makefile* for a similar system and modify it. The *makefile* contains examples of how to compile your program along with STOPWATCH.

2. Using an example program as a model (for example, "simple"), modify the *makefile* to compile your program.

3. In each program unit that calls a STOPWATCH subroutine, insert the statement

```
use stopwatch
```

4. Declare one or more variables to be of type `watchtype`, for example

```
type (watchtype) w
```

5. Instrument your code as appropriate with subroutine calls:

```
call create_watch(w)
call start_watch(w)
call stop_watch(w)
call reset_watch(w)
call print_watch(w)
call read_watch(val,w,s)
call destroy_watch(w)
```

   where *s* in **read_watch** is one of the character strings 'cpu', 'user', 'sys', or 'wall', depending on what clock you want to read, and *val* is a real variable (of default kind) in which the clock value is returned.

Chapter 3

Obtaining and Compiling StopWatch

Information on StopWatch is available at the World Wide Web page
`http://math.nist.gov/StopWatch.`

StopWatch can be obtained by anonymous ftp from
`ftp://math.nist.gov/pub/mitchell/stopwatch/stopwatch-x.x.tgz`
where x.x is the version number. This is a gzipped tar file which must be uncompressed with gunzip and expanded by tar.

Untarring the file will create a directory called `stopwatch` with subdirectories `doc` and `src`. `doc` contains the User's Guide in postscript and html formats, man pages for every StopWatch subroutine, and an overview man page. `src` contains the source code for the stopwatch module, example programs, and makefiles.

The makefiles illustrate how to compile StopWatch along with your program. A makefile is provided for several systems; see Table 3.1 for a list of the makefiles and the systems they have been tested on. If your system matches one of these, then you need only modify the makefile to use your Fortran 90 programs instead of the examples. If your system is not listed, you might need to modify one of the makefiles to match your system configuration. You might also need to create a new **cpu_second** subroutine; see section 7. If you succeed in running StopWatch on a different system, you can contribute your *makefile* and/or **cpu_second** by sending email to `william.mitchell@nist.gov`.
Contributions will be made available on the WWW page, so check there first before writing your own.

makefile	computer	operating system	compiler
makelf.bat	Pentium Pro	Windows NT 3.51	Lahey Elf90 v. 2.00c
maksalf.bat	Pentium 90	Windows NT 3.51	Salford FTN90 V2.15
mf.aix.xlf.etime_	IBM RS/6000	AIX 4.1	XLF 4.1
mf.cray.cf90.cray	Cray C90/6256	UNICOS 8.0.3.2	CF90 1.0.3.5
mf.dec.decf90.etime	DEC AlphaServer 2100 5/250	Digital UNIX V4.0	Digital Fortran 90 V4.1
mf.dec.decf90.f95	DEC AlphaServer 2100 5/250	Digital UNIX V4.0	Digital Fortran 90 V4.1
mf.hpux.hpf90.etime	HP 9000/710	HP-UX 10.10	HP Fortran 90 1.0
mf.hpux10.nag.etime	HP 9000/735	HP-UX 10.20	NAGWare F90 2.2 (284)
mf.hpux9.nag.etime	HP 9000/735	HP-UX 9.05	NAGWare F90 2.1 (676)
mf.linuxaout.nag.c1	80486DX-50	Linux 1.2.13	NAGWare F90 2.1
mf.linuxelf.F.c2	80486DX-50	Linux 1.2.13	Imagine1 F Compiler, R.96
mf.mac.absoft.nil	PowerMac 9500/120	MacOS V7.5.3	Absoft F90 V1.0
mf.solaris.fujitsu.etime	Sun SPARC 10	Solaris 2.3	Fujitsu Fortran 90 2.03
mf.solaris.sunsoft.etime	Sun SPARC 10	SunOS 5.4	Sunsoft F90 1.1
mf.sun4.epc.etime	Sun SPARC 10	SunOS 4.1.3	EPC Fortran 90 V. 1.1.2
mf.sun4.nag.etime	Sun SPARC 10	SunOS 4.1.3	NAGWare F90 2.1

Table 3.1: Available makefiles.

Chapter 4

Using StopWatch

The entities in StopWatch that have public accessibility are two derived types and fifteen subroutines. Any program unit that references any of these entities must use the `stopwatch` module, i.e., must contain the statement

```
use stopwatch
```

The derived types are:

- `watchtype` – used for declaring a variable to be a watch
- `watchgroup` – used for declaring a variable to be a handle for a group of watches

These two types have public accessibility, but the internals of the type are private. Any operations performed on a variable of one of these types must be performed by one of the StopWatch subroutines.

This section describes, in general terms, the operations that can be performed by the StopWatch subroutines. The formal interfaces and detailed descriptions of the routines can be found in Section 9.

4.1 Watches, Clocks and Watch Groups

A watch is a variable declared to be of type `watchtype`. It can be passed to subroutines as an actual argument or through modules like any Fortran variable, but can only be operated on by the StopWatch subroutines. Watches must be created by subroutine **create_watch** before they are used. Attempting to use a watch that has not been created will generate a Fortran 90 error, because this amounts to passing a pointer with undefined association status to the Fortran intrinsic function **associated**. Watches must be destroyed when no longer useful. For example, consider a local variable of type `watchtype` in a subroutine. Since the contents of a local variable are lost when the subroutine returns, the watch should be destroyed before returning to the calling program. Failure to destroy watches can lead to a memory leak.

- **create_watch** – creates a watch
- **destroy_watch** – destroys a watch

Watches can optionally be given a name (up to 132 characters) through an optional argument, *name*, in **create_watch**. This name is used in error messages and **print_watch** to identify the watch in the printed output.

Different applications demand different definitions of "time". StopWatch supports four clocks in each watch, with each clock measuring a different concept of time. All of them measure time in seconds.

- user – the amount of CPU time used by the user's program
- sys – the amount of CPU time used by the system in support of the user's program
- cpu – the total CPU time, i.e., user+sys
- wall – the wall clock time, i.e., elapsed real time

It is not required that all clocks be used. A watch can be created with any combination of the four clocks. You can also specify a set of *default clocks* to be used whenever the clocks are not explicitly determined.

Since Fortran 90 does not contain an intrinsic function for CPU time, the implementation of the cpu, sys and user clocks is system dependent. Some implementations may support only cpu and wall, not user and sys. Some implementations may support only wall. Since the Fortran 90 standard requires the existence of a **system_clock** subroutine, but does not require that it provide clock information, it is possible that some implementations might not support wall. Clock availability can be determined by **inquiry_stopwatch** (see Section 4.4). Unavailable clocks will automatically be removed from the set of default clocks, but if a clock that is not available is explicitly requested, a warning message will be generated.

StopWatch supports multiple watches simultaneously. Often it is useful to perform the same operation on several watches. This is essential for correct operation of **pause_watch** and **end_pause_watch** and is convenient for procedures like **read_watch**, **print_watch** and **reset_watch**. To facilitate this, StopWatch supports the concept of *watch groups*. When calling a StopWatch subroutine, a watch group can be specified instead of a watch. The group is referenced by a variable of type `watchgroup`. Watch groups must be created before they are used. Attempting to use a watch group that has not been created will generate a Fortran 90 error, because this amounts to passing a pointer with undefined association status to the Fortran intrinsic function **associated**. Watch groups must be destroyed when no longer useful. The watches themselves are not destroyed, only the grouping of them. Failure to destroy watch groups can lead to a memory leak.

- **create_watchgroup** – creates a new watch group
- **destroy_watchgroup** – destroys a watch group (but not the watches in the group)

Most StopWatch subroutines take *watch* as the first dummy argument, and accept several forms of *watch*. The forms are:

- type (watchtype) watch – a single watch
- type (watchtype) watch(:) – an array of watches
- type (watchgroup) watch – a watch group handle

In most StopWatch routines, an array of watches can be specified by an array constructor in the calling statement, for example:

```
type (watchtype) watch :: w1, w2, w3
call print_watch( (/w1,w2,w3/) )
```

However, this can not be used in routines where *watch* has intent OUT or intent INOUT, because the array constructor is actually an expression, not a list of the variables. Currently this prohibits the use of array constructors in the arguments to the routines **create_watch** and **destroy_watch**.

Most StopWatch subroutines take *clock* as the (optional) second dummy argument to determine which of the four clocks will be affected by the action. *clock* can be one of the character strings 'user', 'sys', 'cpu', or 'wall', or can be an array of such character strings to specify more than one clock. Since *clock* is always intent IN, an array of clock types can be built with an array constructor. However, note that Fortran 90 requires all character strings in such a construction to have the same length. Thus 'sys' and 'cpu' should be padded with a blank, as in:

```
call start_watch(watch, (/'user','sys ','cpu '/) )
```

If the optional argument *clock* is omitted, the current set of default clocks is used. The set of default clocks is set with **option_stopwatch** (see Section 4.4) and initially consists of all available clocks.

4.2 Operations on Watches

StopWatch is used by inserting subroutine calls into your program. These subroutine calls correspond to the actions performed with a common stop watch. The basic operation of a watch involves starting it, stopping it, and resetting it's value to 0.

- **start_watch** – starts an idle watch, like the Start/Stop button on a stop watch
- **stop_watch** – stops a running watch, like the Start/Stop button on a stop watch
- **reset_watch** – sets the clocks on a watch to 0.0, like the Reset button on a stop watch

Of course, running a stop watch is of little use unless you can see what it says. The following routines can be called regardless of whether a watch is running, stopped or paused.

- **read_watch** – returns the current clock value of a watch, like looking at the display of a stop watch
- **print_watch** – prints the current clock value of a watch to an output device. To push the analogy to the limit, imagine a stop watch with a printer attached to it.

read_watch returns the clock value in the first argument. The result variable is either a scalar, a pointer to an array of rank one, or a pointer to an array of rank two depending on whether *watch* and *clock* are scalars or arrays. Unless it is a scalar, the result variable should be deallocated after use to avoid memory leakage.

When measuring CPU time, it is often desirable to not include the time used by certain parts of the code, such as printing or graphics. In a subroutine, you might not know which of the clocks are currently running, so you can not simply stop them before the I/O and start them up again after the I/O. For this, StopWatch provides the pause function.

- **pause_watch** – temporarily suspend any of the specified watches that are running
- **end_pause_watch** – resume suspended watches that were running before **pause_watch** was called

4.3 Operations on Watch Groups

Besides **create_watchgroup** and **destroy_watchgroup**, there are two operations that can be performed on `watchgroup` variables:

- **join_watchgroup** – adds a watch to a watch group
- **leave_watchgroup** – removes a watch from a watch group

4.4 Options and System Inquiries

Subroutines are provided to set several options within StopWatch, to determine the current value of these options, and to determine system dependent values of the implementation.

- **option_stopwatch** – sets options within StopWatch.
- **inquiry_stopwatch** – returns values of options and system dependent values

All arguments to these subroutines are optional. All arguments to **option_stopwatch** are intent IN, and all arguments to **inquiry_stopwatch** are intent OUT. The options that can be set by **option_stopwatch** and read by **inquiry_stopwatch** are:

- *default_clock* – character(len=*) or character(len=*)(:) (must be an array in **inquiry_stopwatch**). Specifies one or more clock types to be used as the default clocks when the clock argument is omitted. Initial default (/'cpu ','user','sys ','wall'/). Unavailable clocks will be automatically dropped from the list.
- *io_unit_print* – integer. Specifies an I/O unit for printed output from routine **print_watch**. Initial default is 6. The specified unit must be open for writing sequential formatted output.
- *io_unit_error* – integer. Specifies an I/O unit for printed error messages. Initial default is 6. The specified unit must be open for writing sequential formatted output.
- *print_errors* – logical. Flag to specify whether or not error messages should be printed. Initial default is .true.
- *abort_errors* – logical. Flag to specify whether or not the program should abort on an error. If the program does not abort, then the requested operation is ignored and execution continues. Initial default is .false.
- *print_form* – character(len=*). Specifies the form for printing time in **print_watch**. Currently all the forms print the time to .01 seconds. The valid values are:
  - 'sec'. Print seconds as a real number. This is the default.
  - 'hh:mm:ss'. Print time as hours, minutes and seconds separated by colons.
  - '[[hh:]mm:]ss'. The same as 'hh:mm:ss' except hours and minutes are printed only if they are nonzero.

In addition, **inquiry_stopwatch** takes the following optional arguments:

- *cpu_avail* – logical. True if the cpu clock is available in the implementation.
- *user_avail* – logical. True if the user clock is available in the implementation.
- *sys_avail* – logical. True if the sys clock is available in the implementation.
- *wall_avail* – logical. True if the wall clock is available in the implementation.
- *cpu_prec* – real. The cpu clock precision in seconds, i.e., the smallest amount of time that the cpu, user and sys clocks can measure.
- *wall_prec* – real. The wall clock precision in seconds.
- *version* – character(len=16). The version number of StopWatch.

Chapter 5

Examples

The StopWatch distribution contains several example programs to demonstrate how to use StopWatch, and to test the installation. These programs are located in the `src` directory. Once you select or create the correct *makefile* you should be able to compile these examples with "make *prog*" where *prog* is the name of the source file without the `.f90` extension.

- *simple.f90* – This is a short example showing the simplest use of StopWatch.
- *advanced.f90* – This example illustrates the use of some of the advanced features of StopWatch, including array arguments and watchgroups.
- *overhead.f90* – This program prints the clock precisions, and measures the amount of time used by calls to StopWatch subroutines. As long as the clock precision is much larger than the overhead of a StopWatch subroutine, StopWatch should not increase the time being measured.
- *testsw.f90* – This is a program that tests most of the functionality of StopWatch.
- *errors.f90* – This is a program that tests many of the error conditions detected by StopWatch.
- *bomb.f90* – This program attempts to make StopWatch crash by using a watch that has not been created. Running this program should indicate how your system handles this error condition, but there is no guarantee that your compiler will handle the Fortran error consistently.

Chapter 6

Trouble Shooting

All StopWatch subroutines take an optional argument *err* as the last dummy argument. This is an INTENT(OUT) integer argument in which a status code is returned. The code is the sum of the values listed below.

Errors can also be determined through printed error messages. An error message will be printed to a specified I/O unit (6 by default) if *print_errors* is TRUE (default is TRUE; see Section 4.4). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

All errors are non-fatal. If *abort_errors* is FALSE (default is FALSE, see Section 4.4) the requested operation is ignored and execution will continue.

The relevant status codes and messages are:

0 – operation successful; no errors.

1 – Watch needs to be created. This occurs when you attempt to use a watch that has been destroyed. Some compilers might also generate this error when you attempt to use a watch that has never been created.

2 – Watch is in the wrong state for this operation. This occurs when you attempt to start a watch that is already running, stop a watch that is not running, etc.

4 – Watch is in an unknown state. This occurs if StopWatch does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in StopWatch.

8 – Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch** (Section 4.4) to determine what clocks are available.

16 – Too many clocks specified. This occurs when the argument *clock* is an array longer than four.

32 – Number of names is not equal to number of watches. This occurs in **create_watch** if the array of watch names is not of the same length as the array of watches.

64 – Character string too long. This occurs when a watch name with more than 132 characters is passed into **create_watch**.

128 – Watch not found in given group. This occurs when you attempt to remove a watch from a group that it does not belong to.

256 – I/O unit is not open for writing. This can occur from **print_watch** or when printing an error message.

512 – Failed to allocate required memory. When a StopWatch routine is called with an array or group of watches, temporary memory is allocated. This error occurs if the allocate statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, destroying local variable watches and groups before returning from a subroutine, and deallocating array results from **read_watch**.

1024 – Error occurred while deallocating memory. This error occurs if the deallocate statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The operation is performed, but be aware that other problems could develop as a result of the deallocate error.

2048 – Illegal output form. This error occurs in **option_stopwatch** or **print_watch** if the given print format is not one of the valid strings listed in section 4.4.

Chapter 7

Subroutine cpu_second

Although Fortran 90 standardized an intrinsic function for wall clock time, it does not include a function for CPU time. At the time of this writing, it is anticipated that a CPU time intrinsic function will be added to the language in Fortran 95. If this happens, then StopWatch can become fully system independent once Fortran 95 compilers are widespread. Meanwhile, StopWatch requires that a system dependent CPU time subroutine be provided by the user. Several versions of this subroutine are included with the StopWatch package. One of these may work on your system. The current versions, systems they have been tested on, and clock precisions are shown in Table 7.1. The computers and version numbers of the operating systems and compilers can be found in Table 3.1. Those based on the Cray routine **second** and the Fortran 95 routine **cpu_second** do not provide the user and sys clocks. The version *cpusec.nil.f* contains no CPU clock information, and can be used on systems where there is no routine to measure CPU time. If this routine is used, only the wall clock will be available.

If none of the **cpu_second** versions work on your system, you will have to write your own. The interface is

```
subroutine cpu_second(cpu,user,sys)
real, intent(OUT) :: cpu, user, sys
```

The first argument is for CPU time in seconds. Where available, the second and third arguments should break down the CPU time into "user" and "system" CPU time. If the underlying system does not provide for a way of accessing the breakdown (i.e., has only CPU time), then return a negative constant in *user* and *sys* (for example, `user=-1.; sys=-1.`). The value returned in *cpu* (and *user* and *sys* where available) should be a nonnegative real number such that the difference between two successive calls is the amount of elapsed CPU time in seconds.

If you write a new version of **cpu_second** because none of the supplied versions worked on your system, please send this information to the author so that it can be included in the next release.

file	basis	OS	compiler	cpu precision	wall precision
cpusec.c1.c	times	Linux	NAGWare F90	1.E-2	1.E+0
cpusec.c2.c	times	Linux	Imagine1 F	1.E-2	1.E+0
cpusec.cray.f90	second	UNICOS	CF90	4.E-6	4.E-9
cpusec.etime.f	etime	HP-UX	NAGWare F90	1.E-2	1.E+0
cpusec.etime.f90	etime	Digital UNIX	Digital Fortran 90	1.E-3	1.E-4
		HP-UX	HP Fortran 90	1.E-2	1.E-3
		Solaris	Fujitsu Fortran 90	1.E-2	1.E-3
		Solaris	Sunsoft F90	9.E-5	1.E-6
		SunOS	EPC Fortran 90	1.E-2	1.E-3
		SunOS	NAGWare F90	1.E-2	2.E-2
cpusec.etime_.f90	etime	AIX	XLF	1.E-2	1.E-2
cpusec.f95.f90	cpu_time	Digital UNIX	Digital Fortran 90	4.E-3	1.E-4
cpusec.nil.f90	none	Windows NT	Lahey Elf90	N/A	1.E-2
		Windows NT	Salford FTN90	N/A	1.E-3
		MacOS	Absoft F90	N/A	1.E-6

Table 7.1: Available versions of **cpu_second** with clock precisions.

Chapter 8

Acknowledgments

I would like to thank:
Ron Boisvert, Roldan Pozo, and Eite Tiesinga for many helpful suggestions.
Karin Remington, Walt Brainard, Neil Campbell, Neil Carlson, Jeroen Groenenboom, Alan Hoffman, Steve Lionel, Christian de Polignac, Mitsu Sakamoto, David Vallance, and Mike Vermeulen for beta testing or otherwise providing assistance.

Chapter 9

Reference Manual

This section contains an alphabetical listing of all StopWatch routines. Each routine is described in detail, along with diagnostics and examples. The information in this section can also be obtained online through the man pages.

CREATE_WATCH

creates and initializes a StopWatch watch

SYNOPSIS

subroutine **create_watch***(watch, clock, name, err)*

      type (watchtype), intent(OUT) :: watch
OR type (watchtype), intent(OUT) :: watch(:)

      character(len=*), optional, intent(IN) :: clock
OR character(len=*), optional, intent(IN) :: clock(:)

      character(len=*), optional, intent(IN) :: name
OR character(len=*), optional, intent(IN) :: name(:)

      integer, optional, intent(OUT) :: err

DESCRIPTION

Creates and initializes the specified clocks of the specified watches. Upon return from **create_watch**, all clocks are not running and have the value 0. All watches must be created before they are used or added to a watch group. In Fortran 90 it is impossible to test whether or not a watch has been created, and using a watch that has not been created may cause the program to crash. It is not an error to create a watch that has already been created, however the prior information *and memory locations* will be lost. Watches should be destroyed (see **destroy_watch**(3)) before they are recreated. Also, local variable watches should be destroyed before returning from a subroutine, to avoid memory leaks.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to create one watch, or an array of type *watchtype* to create several watches.

The optional argument *clock* specifies which clocks to create on the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are created. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

The optional argument *name* allows you to attach a name to the watch. The name is used when printing error messages, or when printing clock values using **print_watch.** If omitted, the name of the watch is 'unnamed watch'. If present, it must be of the same rank and dimension as *watch*. Watch names are limited to 132 characters.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be created.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.
8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.
32 Number of names is not equal to number of watches. This occurs if the array of watch names, *name,* is not of the same length as the array of watches, *watch.*
64 Character string too long. This occurs when a watch name has more than 132 characters. The watch is created, but the name is truncated to the first 132 characters.
512 Failed to allocate required memory. Creating a watch involves allocating memory for it. Also, when **create_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.
1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are created, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by StopWatch, the following problems may arise:

- Since *watch* has intent OUT, you cannot use an array constructor as an actual argument to construct an array of watches. Some compilers will recognize this as a compile time error, but will generate an obscure error message, such as “no specific match for generic name”.
- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu ', and pad watch names so they all have the same length (within an array constructor).

EXAMPLES

```
type (watchtype) w1, w2(3), w3
integer errcode

call create_watch(w1)
call create_watch(w2, name=(/'part 1', 'part 2', 'total '/), err=errcode)
call create_watch(w3, (/'cpu ', 'wall'/), err=errcode)
```

The first call creates the default clocks on a single watch with name 'unnamed watch'. The second call creates the default clocks on three watches given as an array and with names 'part 1', 'part 2', and 'total', and returns a status code. The third call creates one watch with the cpu and wall clocks, the name 'unnamed watch', and returns a status code.

BUGS

None known.

CREATE_WATCHGROUP

creates a StopWatch watch group

SYNOPSIS

subroutine **create_watchgroup***(watch, handle, err)*

```
     type (watchtype), intent(IN) :: watch
  OR type (watchtype), intent(IN) :: watch(:)

     type (watchgroup), intent(OUT) :: handle
     integer, optional, intent(OUT) :: err
```

DESCRIPTION

Creates a new watch group and returns a handle for it. A watch group must be created by this routine before it is passed to any other StopWatch routines. In Fortran 90 it is impossible to test whether or not a watch group has been created, and using a watch group that has not been created may cause the program to crash. It is not an error to create a watch group that has already been created, however the prior information *and memory locations* will be lost. Watch groups should be destroyed (see **destroy_watchgroup**(3)) before they are recreated. Also, local variable watch groups should be destroyed before returning from a subroutine, to avoid memory leaks.

One or more watches may be optionally specified. If *watch* is present, the watch group will initially contain the specified watch(es). If *watch* is omitted, the watch group will initially be empty. Watches can be added and removed from the group with **join_watchgroup** and **leave_watchgroup.** The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to start the group with one watch, or an array of type *watchtype* to start the group with several watches.

The argument *handle* is a variable of type *watchgroup* that will subsequently be used to access the watch group.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch group will not be created.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

- 0 No errors; execution successful.
- 1 Watch needs to be created. This error occurs if you attempt to put a watch that has been destroyed in the group. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.
- 512 Failed to allocate required memory. When a group is created, memory is allocated for the group. Also, when **create_watchgroup** is called with an array of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.
- 1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array of watches. The group is created, but be aware that other problems could develop as a result of the deallocate error.

EXAMPLES

```
type (watchtype) w(3)
type (watchgroup) g1, g2
integer errcode

call create_watchgroup(handle=g1)
call create_watchgroup(w, g2, err=errcode)
```

The first call creates an empty group *g1.* The second call creates the group *g2* with three watches, and returns a status code.

BUGS

It cannot be determined whether or not a watch variable has been created (passed as an argument to **create_watch).** If a watch that has never been created is passed into **create_watchgroup,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

DESTROY_WATCH

destroys a StopWatch watch

SYNOPSIS

subroutine **destroy_watch***(watch, clock, err)*

```
    type (watchtype), intent(INOUT) :: watch
 OR type (watchtype), intent(INOUT) :: watch(:)

    character(len=*), optional, intent(IN) :: clock
 OR character(len=*), optional, intent(IN) :: clock(:)

    integer, optional, intent(OUT) :: err
```

DESCRIPTION

Destroys the specified clocks of the specified watches. If the watch has no remaining clocks after the specified clocks are destroyed, then the watch is destroyed and associated memory freed. To avoid memory leaks, watches should be destroyed when no longer useful, before being recreated, and before returning from a subroutine in which the watch is a local variable.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to destroy one watch, or an array of type *watchtype* to destroy several watches.

The optional argument *clock* specifies which clocks to destroy on the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are destroyed. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition.

Otherwise, the program will continue execution but the watch(es) will not be destroyed.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.

1 Watch needs to be created. This error occurs if you attempt to destroy a watch that has already been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.

8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.

512 Failed to allocate required memory. When **destroy_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.

1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating the memory for the watch or temporary memory used for an array or group of watches. The watches are destroyed, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by StopWatch, the following problems may arise:

- Since *watch* has intent OUT, you cannot use an array constructor as an actual argument to construct an array of watches. Some compilers will recognize this as a compile time error, but will generate an obscure error message, such as "no specific match for generic name".
- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
integer errcode

call destroy_watch(w1)
call destroy_watch(w2, (/'sys ', 'user'/), err=errcode)
```

The first call destroys the default clocks on a single watch. Assuming the default clocks have not changed since the watch was created, the watch will be destroyed. The second call destroys the sys and user clocks on three watches given as an array and returns a status code. Assuming the watch also had the cpu or wall clock, the watches are not destroyed.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **destroy_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

DESTROY_WATCHGROUP

destroys a StopWatch watch group

SYNOPSIS

subroutine **destroy_watchgroup** *(handle, err)*

type (watchgroup), intent(INOUT) :: handle
integer, optional, intent(OUT) :: err

DESCRIPTION

Destroys a watch group. Only the group is destroyed, not the watches in the group. To avoid memory leaks, watch groups should be destroyed when no longer useful, before being recreated, and before returning from a subroutine in which the watch group is a local variable.

The argument *handle* is a variable of type *watchgroup* that is the handle for the group to be destroyed.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch group will not be created.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.
1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating memory used for the group. The group is destroyed, but be aware that other problems could develop as a result of the deallocate error.

EXAMPLES

```
type (watchgroup) g1, g2
integer errcode

call destroy_watchgroup(g1)
call destroy_watchgroup(g2, errcode)
```

The first call destroys the group *g1.* The second call destroys the group *g2* and returns a status code.

BUGS

None known.

END_PAUSE_WATCH

resumes a paused STOPWATCH watch

SYNOPSIS

subroutine **end_pause_watch***(watch, clock, err)*

```
    type (watchtype), intent(IN) :: watch
 OR type (watchtype), intent(IN) :: watch(:)
 OR type (watchgroup), intent(IN) :: watch

    character(len=*), optional, intent(IN) :: clock
 OR character(len=*), optional, intent(IN) :: clock(:)

    integer, optional, intent(OUT) :: err
```

DESCRIPTION

Resumes the running status of the specified clocks of the specified watches that have previously been paused (see **pause_watch**(3)). Pausing is useful when you want to temporarily stop the clocks to avoid timing a small segment of code, for example printed output or graphics, but do not know which watches or clocks are running. When **pause_watch** is called, the information about which of the clocks were running is maintained, so that a subsequent call to **end_pause_watch** will restart only those clocks that were running.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to resume one watch, an array of type *watchtype* to resume several watches, or a variable of type *watchgroup* (see **stopwatch**(3)) to resume the watches in a group.

The optional argument *clock* specifies which clocks to resume on the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are resumed. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be resumed.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.
1 Watch needs to be created. This error occurs if you attempt to resume a watch that has been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.
2 Watch is in the wrong state for this operation. This occurs when you attempt to resume a watch that is currently running.
4 Watch is in an unknown state. This occurs if StopWatch does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in StopWatch.
8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.
512 Failed to allocate required memory. When **end_pause_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.
1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are resumed, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by StopWatch, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call end_pause_watch(w1)
```

```
call end_pause_watch(w2, err=errcode)
call end_pause_watch(g1, (/'cpu ', 'wall'/), errcode)
```

The first call resumes the default clocks on a single watch. The second call resumes the default clocks on three watches given as an array and returns a status code. The third call resumes the cpu and wall clocks on the watches in the group *g1,* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **end_pause_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

INQUIRY_STOPWATCH

returns StopWatch options and system dependent values

SYNOPSIS

subroutine **inquiry_stopwatch***(default_clock, io_unit_print, io_unit_error, print_errors, abort_errors, print_form, cpu_avail, user_avail, sys_avail, wall_avail, cpu_prec, wall_prec, version, err)*

```
character(len=*), optional, intent(OUT) :: default_clock(4)
integer, optional, intent(OUT) :: io_unit_print, io_unit_error
logical, optional, intent(OUT) :: print_errors, abort_errors
character(len=*), optional, intent(OUT) :: print_form
logical, optional, intent(OUT) :: cpu_avail, user_avail, sys_avail, wall_avail
real, optional, intent(OUT) :: cpu_prec, wall_prec
character(len=16), optional, intent(OUT) :: version
integer, optional, intent(OUT) :: err
```

DESCRIPTION

Returns the value of StopWatch options and other system and implementation dependent values. All arguments are optional and have intent OUT.

The following arguments can be set by **option_stopwatch**. See **option_stopwatch**(3) for further details on their meaning. *default_clock* is the set of clocks that are used when the *clock* argument is omitted in a call to a StopWatch routine. *io_unit_print* returns the unit for output from subroutine **print_watch.** *io_unit_error* returns the unit for any error messages printed by StopWatch. If *print_errors* is TRUE, then an error message will be printed to *io_unit_error* whenever an error condition occurs. If *abort_errors* is TRUE, then the program will terminate when an error condition occurs. *print_form* is the format used by **print_watch**(3) when the *form* argument is omitted.

The remaining arguments return system information that can not be changed.

Since an interface to the CPU clock is not part of the Fortran 90 standard, the availability of clocks and clock precisions are implementation dependent. Not all clocks are available in all implementations. The logical arguments *cpu_avail, user_avail, sys_avail* and *wall_avail* return TRUE if the respective clock is available in this implementation.

The precision (the shortest time interval that can be measured) of the clocks also varies between implementations. The real variables *cpu_prec* and *wall_prec* return the precision of the CPU and wall clocks, in seconds. It is assumed that the user and sys clocks have the same precision as the CPU clock. If the CPU clock is not available, then *cpu_prec* will return 0., and similar for the wall clock.

The character string *version* returns the version number of StopWatch.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the requested value(s) might not be returned.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

- 0 No errors; execution successful.
- 512 Failed to allocate required memory. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.

EXAMPLES

```
logical user_is_there
real cpu_prec

call inquiry_stopwatch(user_avail=user_is_there)
call inquiry_stopwatch(cpu_prec=cpu_prec)
```

The first call determines if the user clock is available in this implementation. The second call determines the shortest time that can be measured by the CPU clock.

BUGS

None known.

JOIN_WATCHGROUP

adds a StopWatch watch to a watch group

SYNOPSIS

subroutine **join_watchgroup***(watch, handle, err)*

```
     type (watchtype), intent(IN) :: watch
  OR type (watchtype), intent(IN) :: watch(:)

     type (watchgroup), intent(INOUT) :: handle
     integer, optional, intent(OUT) :: err
```

DESCRIPTION

Adds the specified watch(es) to the specified watch group. The watch(es) and group must have been previously created with **create_watch** and **create_watchgroup.**

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to add one watch, an array of type *watchtype* to add several watches.

The watch group is specified by *handle,* a variable of type *watchgroup.*

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be added to the group.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.

1 Watch needs to be created. This error occurs if you attempt to add a watch that has been destroyed to a group. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.

512 Failed to allocate required memory. Memory is allocated in the group when a watch is added. Also, when **join_watchgroup** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.

1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are added to the group, but be aware that other problems could develop as a result of the deallocate error.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call join_watchgroup(w1, g1)
call join_watchgroup(w2, g1, errcode)
```

The first call adds the watch *w1* to watch group *g1.* The second call adds three watch to *g1* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **join_watchgroup,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

LEAVE_WATCHGROUP

removes a StopWatch watch from a watch group

SYNOPSIS

subroutine **leave_watchgroup***(watch, handle, err)*

```
    type (watchtype), intent(IN) :: watch
 OR type (watchtype), intent(IN) :: watch(:)

    type (watchgroup), intent(INOUT) :: handle
    integer, optional, intent(OUT) :: err
```

DESCRIPTION

Removes the specified watch(es) from the specified watch group.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to remove one watch, or an array of type *watchtype* to remove several watches.

The watch group is specified by *handle,* a variable of type *watchgroup.*

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be removed from the group.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.

128 Watch not found in given group. This occurs when you attempt to remove a watch from a group that it does not belong to. One cause of this is if you destroy a watch and later try to remove it from a group.

512 Failed to allocate required memory. When **leave_watchgroup** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.

1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches or the memory used for an entry in the group. The watches are removed from the group, but be aware that other problems could develop as a result of the deallocate error.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call leave_watchgroup(w1, g1)
call leave_watchgroup(w2, g1, errcode)
```

The first call removes the watch *w1* from watch group *g1.* The second call removes three watch from *g1* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **leave_watchgroup,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

OPTION_STOPWATCH

sets StopWatch options

SYNOPSIS

subroutine **option_stopwatch** *(default_clock, io_unit_print, io_unit_error, print_errors, abort_errors, print_form, err)*

```
    character(len=*), optional, intent(IN) :: default_clock(:)
OR  character(len=*), optional, intent(IN) :: default_clock

    integer, optional, intent(IN) :: io_unit_print, io_unit_error
    logical, optional, intent(IN) :: print_errors, abort_errors
    character(len=*), optional, intent(IN) :: print_form
    integer, optional, intent(OUT) :: err
```

DESCRIPTION

Sets options that control the behavior of StopWatch. All arguments are optional and have intent IN, with the exception of the status code *err* which has intent OUT. These options are global in nature, and remain in effect until another call to **option_stopwatch** changes them.

The argument *default_clock* determines what clocks will be used for all subsequent operations in which the *clock* argument is omitted. This allows you to specify what clocks you are interested in once and for all, and not have to specify those clocks with every subroutine call. The initial default value is (/'cpu ', 'user', 'sys ', 'wall'/), i.e., all clocks. However, if any clocks are not available in the implementation, they will be automatically removed from the list of default clocks.

Printed output can be redirected to any valid I/O unit number. *io_unit_print* determines the unit for output from subroutine **print_watch.** *io_unit_error* determines the unit for any error messages printed by StopWatch. When an I/O unit is reset by one of these variables, the unit must already be open for writing. The initial default is 6 for both I/O units, which is standard output on many systems.

What to do when an error occurs is controlled by the two logical variables *print_errors* and *abort_errors.* If *print_errors* is TRUE, then an error message will be printed to *io_unit_error* whenever an error condition occurs. In all cases where an error can be detected, the program can continue to execute, although the behavior of StopWatch might not be as expected. If *abort_errors* is TRUE, then the program will terminate when an error condition occurs. The initial defaults are TRUE for *print_errors* and FALSE for *abort_errors.*

The argument *print_form* determines the form for printing time when *form* is omitted in **print_watch.** Currently all the forms print the time to .01 seconds. The valid values for *print_form* are:
'sec', seconds
'hh:mm:ss', colon separated hours, minutes and seconds
'[[hh:]mm:]ss', same as 'hh:mm:ss' except hours and minutes are printed only if nonzero

The default value is 'sec'.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

The relevant status codes and messages are:

0 No errors; execution successful.
8 Invalid clock type. This occurs if *default_clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.
16 Too many clocks specified. This occurs when the argument *default_clock* is an array longer than four.
256 I/O unit is not open for writing. The I/O unit requested for *io_unit_print* or *io_unit_error* is not open for writing.
512 Failed to allocate required memory. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.
1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating memory. Be aware that other problems could develop as a result of the deallocate error.
2048 Illegal output form. This error occurs if *print_form* is not one of the strings listed above.

In addition to the run time diagnostics generated by StopWatch, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
call option_stopwatch(default_clock='cpu', abort_error=.true.)
call option_stopwatch(io_unit_print=11, io_unit_error=12)
```

The first call sets the default clock to be the cpu clock and says to terminate the program if an error occurs. The second call reassigns the I/O units.

BUGS

None known.

PAUSE_WATCH

pauses a StopWatch watch

SYNOPSIS

subroutine **pause_watch***(watch, clock, err)*

```
    type (watchtype), intent(IN) :: watch
 OR type (watchtype), intent(IN) :: watch(:)
 OR type (watchgroup), intent(IN) :: watch

    character(len=*), optional, intent(IN) :: clock
 OR character(len=*), optional, intent(IN) :: clock(:)

    integer, optional, intent(OUT) :: err
```

DESCRIPTION

Pauses the specified clocks of the specified watches. This is useful when you want to temporarily stop the clocks to avoid timing a small segment of code, for example printed output or graphics, but do not know which watches or clocks are running. When **pause_watch** is called, the information about which of the clocks were running is maintained, so that a subsequent call to **end_pause_watch** will restart only those clocks that were running. Watches that are paused can not be started, stopped, reset, or paused again until they are resumed by **end_pause_watch.** However, they can be read and printed.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to pause one watch, an array of type *watchtype* to pause several watches, or a variable of type *watchgroup* (see **stopwatch**(3)) to pause the watches in a group.

The optional argument *clock* specifies which clocks to pause on the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are paused. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be paused.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

- 0 No errors; execution successful.
- 1 Watch needs to be created. This error occurs if you attempt to pause a watch that has been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.
- 2 Watch is in the wrong state for this operation. This occurs when you attempt to pause a watch that is currently paused.
- 4 Watch is in an unknown state. This occurs if StopWatch does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in StopWatch.
- 8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.
- 512 Failed to allocate required memory. When **pause_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.
- 1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are paused, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by StopWatch, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call pause_watch(w1)
```

```
call pause_watch(w2, err=errcode)
call pause_watch(g1, (/'cpu ', 'wall'/), errcode)
```

The first call pauses the default clocks on a single watch. The second call pauses the default clocks on three watches given as an array and returns a status code. The third call pauses the cpu and wall clocks on the watches in the group *g1,* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **pause_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

PRINT_WATCH

prints the current value of a StopWatch watch

SYNOPSIS

subroutine **print_watch***(watch, clock, title, form, err)*

type (watchtype), intent(IN) :: watch
OR type (watchtype), intent(IN) :: watch(:)
OR type (watchgroup), intent(IN) :: watch

character(len=*), optional, intent(IN) :: clock
OR character(len=*), optional, intent(IN) :: clock(:)

character(len=*), optional, intent(IN) :: title, form

integer, optional, intent(OUT) :: err

DESCRIPTION

Prints the specified clocks of the specified watches. A title line is printed followed by two lines for each watch. The first contains the name of the watch, which was defined in **create_watch**(3) and maintained internally, and the second contains the values of the specified clocks. Output is written to a user specified I/O unit (see **option_stopwatch**(3)) which is 6 by default. Clocks can be printed regardless of whether they are running, stopped or paused.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to print one watch, an array of type *watchtype* to print several watches, or a variable of type *watchgroup* (see **stopwatch**(3)) to print the watches in a group.

The optional argument *clock* specifies which clocks to print from the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are printed. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

The optional argument *title* is a character string to be printed before printing the watch values. If omitted, the string "Times printed by StopWatch:" is printed.

The optional argument *form* determines the form for printing time. Currently all the forms print the time to .01 seconds. The valid values are:
'sec', seconds
'hh:mm:ss', colon separated hours, minutes and seconds

'[[hh:]mm:]ss', same as 'hh:mm:ss' except hours and minutes are printed only if nonzero

If omitted, the current default form is used. The default form is initially 'sec' and can be reset by **option_stopwatch**(3).

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be printed.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

- 0 No errors; execution successful.
- 1 Watch needs to be created. This error occurs if you attempt to print a watch that has been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.
- 4 Watch is in an unknown state. This occurs if StopWatch does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in StopWatch.
- 8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.
- 256 I/O unit is not open for writing. The I/O unit to which **print_watch** expects to write is not open for writing. The I/O unit number is set by *io_unit_print* in **option_stopwatch** and is 6 by default.
- 512 Failed to allocate required memory. When **print_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.
- 1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are printed, but be aware that other problems could develop as a result of the deallocate error.
- 2048 Illegal output form. This error occurs if *form* is not one of the strings listed above.

In addition to the run time diagnostics generated by StopWatch, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call print_watch(w1)
call print_watch(w2, title='Array of 3 watches', err=errcode)
call print_watch(g1, (/'cpu ', 'wall'/), errcode)
```

The first call prints the default clocks from a single watch, and the default title. The second call prints the default clocks on three watches given as an array and the title "Array of 3 watches", and returns a status code. The third call prints the cpu and wall clocks on the watches in the group *g1,* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **print_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

READ_WATCH

reads the values from a STOPWATCH watch

SYNOPSIS

subroutine **read_watch***(read_result, watch, clock, err)*

```
    real, intent(OUT) :: read_result
 OR real, pointer :: read_result(:)
 OR real, pointer :: read_result(:,:)

    type (watchtype), intent(IN) :: watch
 OR type (watchtype), intent(IN) :: watch(:)

    character(len=*), optional, intent(IN) :: clock
 OR character(len=*), optional, intent(IN) :: clock(:)

    integer, optional, intent(OUT) :: err
```

DESCRIPTION

Returns the value of the specified clocks from the specified watches. The result is returned in *read_result.* Clocks can be read regardless of whether they are running, stopped or paused.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to read one watch, or an array of type *watchtype* to read several watches. *watch* can not be a *watchgroup* because there is no natural order of the watches in the group to use in constructing an array for the result.

The optional argument *clock* specifies which clocks to read from the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are read. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

The type of *read_result* must agree with the form of the arguments *watch* and *clock:*

- If *watch* is a scalar and *clock* is a scalar, then *read_result* must be a real scalar.
- If *watch* is an array and *clock* is a scalar, then *read_result* must be a pointer to a rank 1 real array. The i^{th} entry of the result is the specified clock value on *watch(i).*
- If *watch* is a scalar and *clock* is either an array or omitted, then *read_result* must be a pointer to a rank 1 real array. The i^{th} entry of the result is the value in *clock(i)* on the specified *watch.* In the case that *clock* is omitted, note that the default clocks

- specify the contents of the result, and the default clocks can be determined using **inquiry_stopwatch**(3).
- If *watch* is an array and *clock* is either an array or omitted, then *read_result* must be a pointer to a rank 2 real array. The $(i,j)^{\text{th}}$ entry of the result is the value in *clock(j)* on *watch(i)*.

If *read_result* is a pointer to an array, it will be allocated by **read_watch**, and should be deallocated after use to avoid memory leakage.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be read.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.

1 Watch needs to be created. This error occurs if you attempt to read a watch that has been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.

4 Watch is in an unknown state. This occurs if STOPWATCH does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in STOPWATCH.

8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.

512 Failed to allocate required memory. When **read_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.

1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are read, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by STOPWATCH, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
real x
real, pointer ::  y(:), z(:,:)
integer errcode

call read_watch(x, w1, 'user')
call read_watch(y, w1, err=errcode)
call read_watch(z, w2, (/'cpu ', 'wall'/), errcode)
deallocate(y, z)
```

The first call reads the user clock on a single watch. The second call reads the default clocks on a single watch and returns a status code. y is allocated with dimension equal to the number of default clocks. The third call reads the cpu and wall clocks from three watches given as an array and returns a status code. The deallocate statement frees the memory allocated in *read_watch.*

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **read_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

RESET_WATCH

resets a StopWatch watch to 0.0

SYNOPSIS

subroutine **reset_watch***(watch, clock, err)*

```
     type (watchtype), intent(IN) :: watch
  OR type (watchtype), intent(IN) :: watch(:)
  OR type (watchgroup), intent(IN) :: watch

     character(len=*), optional, intent(IN) :: clock
  OR character(len=*), optional, intent(IN) :: clock(:)

     integer, optional, intent(OUT) :: err
```

DESCRIPTION

Resets the specified clocks of the specified watches to 0. Clocks can be reset regardless of whether they are running or not.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to reset one watch, an array of type *watchtype* to reset several watches, or a variable of type *watchgroup* (see **stopwatch**(3)) to reset the watches in a group.

The optional argument *clock* specifies which clocks to reset on the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are reset. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be reset.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

- 0 No errors; execution successful.
- 1 Watch needs to be created. This error occurs if you attempt to reset a watch that has been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.
- 2 Watch is in the wrong state for this operation. This occurs when you attempt to reset a watch that is currently paused.
- 4 Watch is in an unknown state. This occurs if StopWatch does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in StopWatch.
- 8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.
- 512 Failed to allocate required memory. When **reset_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.
- 1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are reset, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by StopWatch, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call reset_watch(w1)
call reset_watch(w2, err=errcode)
call reset_watch(g1, (/'cpu ', 'wall'/), errcode)
```

The first call resets the default clocks on a single watch. The second call resets the default clocks on three watches given as an array and returns a status code. The third call resets the cpu and wall clocks on the watches in the group *g1,* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **reset_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

START_WATCH

starts a STOPWATCH watch

SYNOPSIS

subroutine **start_watch***(watch, clock, err)*

```
    type (watchtype), intent(IN) :: watch
 OR type (watchtype), intent(IN) :: watch(:)
 OR type (watchgroup), intent(IN) :: watch

    character(len=*), optional, intent(IN) :: clock
 OR character(len=*), optional, intent(IN) :: clock(:)

    integer, optional, intent(OUT) :: err
```

DESCRIPTION

Starts the specified clocks of the specified watches. Any time previously accumulated in the clock is NOT cleared before starting. (Use **reset_watch** to clear accumulated time.)

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to start one watch, an array of type *watchtype* to start several watches, or a variable of type *watchgroup* (see **stopwatch**(3)) to start the watches in a group.

The optional argument *clock* specifies which clocks to start on the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are started. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be started.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.

1 Watch needs to be created. This error occurs if you attempt to start a watch that has been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.

2 Watch is in the wrong state for this operation. This occurs when you attempt to start a watch that is currently running or paused.

4 Watch is in an unknown state. This occurs if StopWatch does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in StopWatch.

8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.

512 Failed to allocate required memory. When **start_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.

1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are started, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by StopWatch, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call start_watch(w1)
call start_watch(w2, err=errcode)
call start_watch(g1, (/'cpu ', 'wall'/), errcode)
```

The first call starts the default clocks on a single watch. The second call starts the default clocks on three watches given as an array and returns an status code. The third call starts the cpu and wall clocks on the watches in the group *g1,* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **start_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.

STOP_WATCH

stops a STOPWATCH watch

SYNOPSIS

subroutine **stop_watch***(watch, clock, err)*

```
     type (watchtype), intent(IN) :: watch
 OR  type (watchtype), intent(IN) :: watch(:)
 OR  type (watchgroup), intent(IN) :: watch

     character(len=*), optional, intent(IN) :: clock
 OR  character(len=*), optional, intent(IN) :: clock(:)

     integer, optional, intent(OUT) :: err
```

DESCRIPTION

Stops the specified clocks of the specified watches.

One or more watches must be specified. The argument *watch* can be a single variable of type *watchtype* (see **stopwatch**(3)) to stop one watch, an array of type *watchtype* to stop several watches, or a variable of type *watchgroup* (see **stopwatch**(3)) to stop the watches in a group.

The optional argument *clock* specifies which clocks to stop on the specified watch(es). If omitted, the current default clocks (see **option_stopwatch**(3)) are stopped. If present, *clock* must be a character string containing 'cpu', 'user', 'sys', or 'wall', or an array of such character strings.

DIAGNOSTICS

If present, the optional intent OUT integer argument *err* returns a status code. The code is the sum of the values listed below.

An error message will be printed to a specified I/O unit (unit 6 by default) if *print_errors* is TRUE (default is TRUE). The error message contains more detail about the cause of the error than can be obtained from just the status code, so you should set *print_errors* to TRUE if you have trouble determining the cause of the error.

If *abort_errors* is TRUE (default is FALSE), the program will terminate on an error condition. Otherwise, the program will continue execution but the watch(es) will not be stopped.

See **option_stopwatch**(3) for further information on *print_errors, abort_errors* and I/O units.

The relevant status codes and messages are:

0 No errors; execution successful.

1 Watch needs to be created. This error occurs if you attempt to stop a watch that has been destroyed. The watch must first be created again. See also the comment about watches that have never been created in the BUGS section.

2 Watch is in the wrong state for this operation. This occurs when you attempt to stop a watch that is currently paused or not running.

4 Watch is in an unknown state. This occurs if StopWatch does not recognize the state (running, stopped, etc.) that the watch is in. This error should not occur, and indicates an internal bug in StopWatch.

8 Invalid clock type. This occurs if *clock* is present and one of the specified clocks is not supported by the implementation. See **inquiry_stopwatch**(3) to determine what clocks are available.

512 Failed to allocate required memory. When **stop_watch** is called with an array or group of watches, temporary memory is allocated. This error occurs if the Fortran **allocate** statement returns a nonzero status indicating that memory could not be allocated. Avoid memory leaks by always destroying watches and groups before recreating them, and destroying local variable watches and groups before returning from a subroutine.

1024 Error occurred while deallocating memory. This error occurs if the Fortran **deallocate** statement returns a nonzero status while deallocating temporary memory used for an array or group of watches. The watches are stopped, but be aware that other problems could develop as a result of the deallocate error.

In addition to the run time diagnostics generated by StopWatch, the following problem may arise:

- In Fortran 90, the character strings in an array constructor must all have the same length. Pad three letter clock names with a blank on the right to make a four character string, for example, 'cpu '.

EXAMPLES

```
type (watchtype) w1, w2(3)
type (watchgroup) g1
integer errcode

call stop_watch(w1)
call stop_watch(w2, err=errcode)
call stop_watch(g1, (/'cpu ', 'wall'/), errcode)
```

The first call stops the default clocks on a single watch. The second call stops the default clocks on three watches given as an array and returns a status code. The third call stops the cpu and wall clocks on the watches in the group *g1,* and returns a status code.

BUGS

It cannot be determined whether or not a watch variable or watch group has been created (passed as an argument to **create_watch** or **create_watchgroup**). If a watch or watch group that has never been created is passed into **stop_watch,** it might generate a Fortran error due to passing a pointer with undefined association status to the Fortran intrinsic function **associated.** Some compilers will allow this as an extension to the Fortran 90 standard and recognize that the pointer is not associated, in which case the "Watch needs to be created" error message is generated.