SignalResponder

COLLABORATORS			
	TITLE : SignalResponder		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 19, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

SignalResponder

1.1 Implementation notes

```
                          The A++ SignalResponder class
                 ------------------------------------
                    ($Date: 1994/05/08 12:58:34 $)
```

The A++ Library SignalResponder class handles the per task signal set.
A SignalResponder object can be used to execute a user defined action on the arrival of one specific signal it is waiting for.

Several SignalResponder objects can be created to wait for several signals.
More than one SignalResponder object can be attached to one signal bit.
SignalResponder objects are priority sorted. The priority controls the order in which they are tested against the received signal set.
They become active by calling the method "SignalResponder::WaitSignal()" which is static, i.e. needs no object to be called on. This method returns after one signal has been received and processed by each dedicated Signal-Responder.

```
               Dedicate a SignalResponder to a user break (CTRL-c)
                 ----------------------------------------------------------
 -> Back to the root menu..
```

1.2 signalresponder_1

How to dedicate a SignalResponder to a user break (CTRL-c):

You just have to derive your special SignalResponder class to overwrite the virtual callback method "void SignalResponder::actionCallback()".

Say, your class looks like this:

```
   class MySRSP : public SignalResponder
   {
      private:
```

```
      BOOL running;  // indicates a received user break to object users
   public:
      MySRSP() : SignalResponder(SIGBREAKB_CTRL_C,0)
      { running = TRUE; }
      //  overload the virtual 'signal received' action callback method.
      void actionCallback()
      {
         cout << "**Break\n";
         running = FALSE;  // end WaitSignal loop
      }
      BOOL hasNotOccured() { return running==TRUE; }
};
```

This example goes along with a main loop like this:

```
main()
{
   //.. initialise something, create some objects
   MySRSP ctrlCBreak();

   while ( ctrlCBreak.hasNotOccured() )
   {
      SignalResponder::WaitSignal();   // receive next signal and process it ↩
          ..
   }
}
```