TimedMsgPort

COLLABORATORS			
	TITLE : TimedMsgPort		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 6, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

TimedMsgPort

1.1 Implementation notes

```
                The A++ TimedMsgPort & MessageC class
              ------------------------------------------
                 ($Date: 1994/05/09 21:25:11 $)
```

The TimedMsgPort class provides easy usage of EXEC® message passing. Furthermore it prevents deadlocks by applying timeout checking on all sent messages.

The TimedMsgPort class can be utilized by overloading the virtual methods installed for processing of incoming messages and replied messages.

Sent messages are tracked in a sending window of variable size. There can only be as much messages pending as place is available in this sending window.

The MessageC class enhances EXEC® Messages with some useful methods.

```
 How to derive a specialized TimedMsgPort class
 How to send messages

------------------------------------------------------------
 -> Back to the root menu..
```

1.2 timedmsgport_1

First, consider what you want to do with your derived MsgPort class and derive a Message class that meets your needs, i.e. add data members to the class that hold the data you want to transmit.
Let's say, you want to transmit a string to another message port. Here is the necessary Message class:

```
   class StringMsg : public MessageC
   {
      private:
         char *string;
```

```
      public:
         StringMsg(char *initString) : string(initString) {}
   };
```

Of course, the denoted string is not copied during the message passing and must therefore stay valid while the message is away. Copying of referenced data would necessarily be implemented by a more sophisticated Message class.

Now,the following two virtual methods have to be overwritten to define your specialized TimedMsgPort class:

```
   void virtual processMsg(MessageC *incoming);
   void virtual processReply(MessageC *reply);
```

'processMsg' is being called on each received message with the message as argument. Note, that the argument is of type 'MessageC*'. You will have to cast 'MessageC' into the type of your derived Message class.
'processReply' will be invoked on each message being replied to the message port object. That implies that every message arriving at 'processReply' was sent from 'this' object before.

1.3 timedmsgport_2

There are two methods available to send previously created MessageC objects from one TimedMsgPort to another:

```
   TimedMsgPort sourcePort("myPort");

   BOOL sourcePort.sendMsgToPort(
         MessageC *yourMsg,
         TimedMsgPort *destinationPort);

   BOOL sourcePort.sendMsgToPort(
         MessageC *yourMsg,
         char *destinationPortName);
```

Both take the given message and send it to the given destination.
Invalid destination ports can be detected.
While the message is pending do not alter it. The message will be replied and delivered to your sourcePort::processReply() method.