DOCUMENTATION for SES module

Adam Braun, Ed Carryer, Kevin Wooley
Rev : January 9, 2002

Purpose of Module

This module provides a system to automate a software events and services protocol. Up to 128 event and service function pairs can be registered with the system using 2 different scheduling algorithms to handle the events.

INTERFACE

Defined Constants

Standard Parameters –- These definitions are used to share data between an Event checking routine and the corresponding service routine.

EVENT_PARAM –- standard event checker parameter, use this in your function prototype
SERVICE_PARAM –- standard service routine parameter, use this in your function prototype
SET_SHARED_VAR_TO(z) – set the variable shared by the event checker and service routine to the value z.
GET_SHARED_BYTE() – return the low byte value of the shared variable
GET_SHARED_WORD() – return the 2 byte (integer size) value of the shared variable.

Scheduling Algorithms -- These constants are used to specify the scheduling for the handling of events. The priority of events is based on the order in which they are registered.

SES_ROUND_ROBIN -- In this algorithm a full pass through all the pairs is made (in order of priority) and every event that is detected is serviced. When the pass is completed, it is restarted with the highest priority pair if any of the events were detected.
SES_PRIORITY -- In this algorithm the events are serviced in order of priority until an event is detected. After the event is serviced, the servicing restarts with the highest priority pair. This process continues until a full pass is made with no events being serviced.

Service Timing -- These constants are used to specify the timing period for the SES_TimeToService function. They are used to approximately control the time between event services in a foreground loop. This is to allow the program to perform other tasks in the background loop if necessary.

SES_NO_UPDATE -- No service timer. SES_TimeToService is always true.
SES_4MS_UPDATE -- Update every 4 ms.
SES_8MS_UPDATE -- Update every 8 ms.
SES_16MS_UPDATE -- Update every 16 ms.
SES_32MS_UPDATE -- Update every 32 ms.

Data Types

uchar -- typedef to be unsigned char
schar -- typedef to be signed char
uint -- typedef to be unsigned int
sint -- typedef to be signed int

Module Functions

SES_Init

PROTOTYPE : uchar SES_Init(uchar aScheduleType, uchar aTimePeriod)
CONTENTS : This is the initialization routine for the SES functions.
PARAMETERS :
    aScheduleType -- The scheduling algorithm type.
    aTimePeriod -- The timing for servicing.
RETURNS :
    OK_OPERATION == The setup was done successfully.
    ERR_BADSCHEDULE == The scheduling value was invalid.
    ERR_BADTIMEPERIOD == The time period was invalid.

If any of the ERR_xxx returns occur, then no action is taken in the module.

SES_Register

PROTOTYPE :
    uchar SES_Register(char (*aEvent)(void**), void (*aService)(void*))
CONTENTS : This will register an event routine and an associated service routine. The priority of the pairs is descending in the order they were registered.
PARAMETERS :
    aEvent -- The event detection routine to register.
    aService -- The associated service routine to register.
RETURNS :
    OK_OPERATION == The registration was successful.
    ERR_NOTINSTALLED == The module was not installed with SES_INIT
    ERR_TOOMANYEVENTS == There are already the maximum number of event/service pairs registered.

If any of the ERR_xxx returns occur, then no action is taken in the module.

SES_TimeToService

PROTOTYPE : uchar SES_TimeToService(void)
CONTENTS : This will return a flag indicating whether it is time to service the events or not. It the time schedule is set to SES_NO_UPDATE.
PARAMETERS :
    none
RETURNS :
    Flag for whether the time has expired or not. (1 == time to service).

SES_HandleEvents

PROTOTYPE : void SES_HandleEvents(void)
CONTENTS : This will run the handle events loop to process the events and service functions using the scheduling algorithm set in the SES_Init function. It will return when a full pass through the event/service pairs finds no events to be serviced.
PARAMETERS :
    none
RETURNS :
    nothing

SES_End

```
PROTOTYPE   : void SES_End(void)
CONTENTS    : This will end the SES system.
PARAMETERS  :
        none
RETURNS     :
        nothing
```

CONSTRAINTS/NOTES

1. Once an event has been registered, it can not be de-registered.

2. SES_Init must be called before the module becomes active. If any other SES functions are called before the module is initialized, they will have no effect.

3. The event/service priorities are based on the order they are registered. Pairs that are registered earlier will have priority over those that are registered later.

4. It is the users responsibility to ensure that the event function returns the correct value. A non-zero return value will be interpreted as indicating that an event has occurred. The void ** parameter (EVENT_PARAM) that is passed to the event function is de-referenced and passed to the service function to enable data transfer from one to the other.

THEORY OF OPERATION

This code operates using an array of function pointer pairs. As the event/service functions are registered, the addresses of the functions are entered into the next available array location.

When the SES_HandleEvents function is called, it starts at the top of the array and executes the event functions in order until one of them returns with a non-zero value. It then executes the associated service function. When this is completed, it will either start at the top of the list or continue with the next event/service pair depending on the scheduling algorithm selected at initialization. This continues until an entire passed is made through the array with no events being serviced.