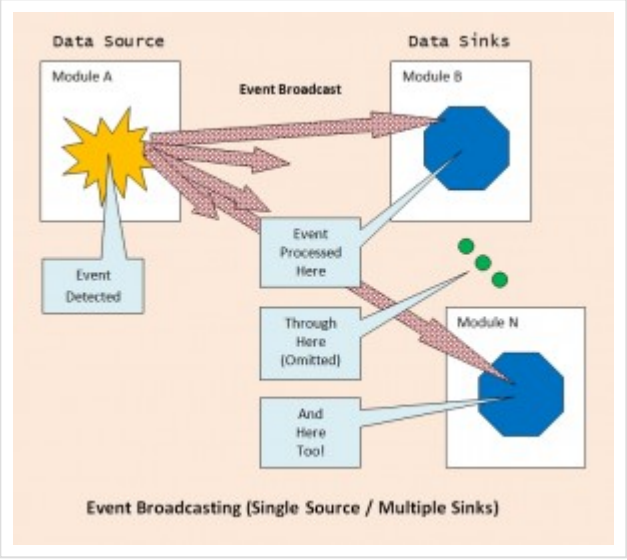
Modules: The Callback Manager

Posted on **August 27, 2012**

A recent series of postings have dealt with modular programming and the transmission of events and data from one module to another. The push and pull strategies were both examined and compared. This article deals with a special case of data flow. The case were there is a single event/data source, and an unknown number (zero or more) of event/data sinks. This is illustrated below:



In this case we can see that there is a single data source and an unknown multitude of sinks. To handle this task, an array of pointers to functions, maintained by the CallBack Manager, is employed. The CallBack manager is a small module that provides the functionality associated with the call back list data structure. Since this data structure is never allocated as a global, static or automatic (stack based) structure, its type definition only specifies a pointer to a structure and not the structure itself. It is shown below:

```
//*********************************************************
// The call back list data structure.
//*********************************************************
```

```
typedef struct
{
    uint8_t         capacity;   // The capacity of the list.
    uint8_t         count;      // The occupancy of the list.
    uint8_t         rfu1;       // Reserved. Padding for now.
    uint8_t         rfu2;       // Reserved. Padding for now.
    emPCCALLBACK    list[0];    // The list of call backs.
} *emPCALLBACKLIST;
```

Since only a pointer is defined, some means of allocating the structure must be provided. In the CallBack manager this is the emCreateCallBackList function:

```
emPCALLBACKLIST emCreateCallBackList(uint8_t capacity);
```

This function definition reveals the first compromise. While the number of event sinks may not be know, the upper limit on the number of sinks must. This function creates a callback list with the specified capacity and returns a pointer to it. If the list could not be allocated, a NULL pointer is returned instead.

The functions used by the CallBack Manager have a standard signature. This is:

```
typedef void (*emPCALLBACKFN)(PV rarg, PV sarg);
```

Where rarg and sarg are generic void pointer arguments that represent parameters determined by the receiver and the sender respectively.

After the list is created, it must be populated, it must be manipulated. This is done with two functions:

```
void emAddCallBack   (emPCALLBACKLIST list,
                      emPCCALLBACK cb,
                      emLISTDIRECTION dir);

void emRemoveCallBack(emPCALLBACKLIST list,
                      emPCCALLBACK cb);
```

The first is used to add functions to the callback list. The dir parameter determines if the callback is added to the first fee slot (emFront) or the last free slot (emBack). This allows a crude sort of priority in the callback list, The second function removes the specified entry from the list.

Finally, there must be some way to invoke the call back list. There are four available functions for this task:

```
void emExecuteCallBack       (emPCALLBACKLIST list);

void emExecuteCallBackPV     (emPCALLBACKLIST list,
                              PV sarg);

void emExecuteTimedCallBack  (emPCALLBACKLIST list,
                              uint32_t timeOutNS);

void emExecuteTimedCallBackPV(emPCALLBACKLIST list,
                              uint32_t timeOutNS,
                              PV sarg);
```

These four functions provide for two orthogonal choices:

1. To send a void pointer sarg or not.
2. To impose a time limit (in nanoseconds) on the call back list or not.

The sarg is useful in cases like an interrupt service routine where it can be used to pass a pointer to a task awoken flag. The time out is useful to prevent the call back from consuming too much time. Note that individual called functions all run to completion, it is the processing of the array that stops when time has expired. This time parameter relies on the timing facilities discussed in A Brief Time Out.

These small number of simple functions summarize the entire CallBack Manager. It is a small module that provides for flexible connections between modules, promoting loosely bound architectures without a great deal of overhead. Soon to come will be an examination of a module that makes strong use of the CallBack Manager to distribute data; look forward to a posting on the A2D Manager.

As always, comments, replies, and suggestions are welcomed.

Peter Camilleri (aka Squidly Jones)

This entry was posted in **Embedded Development** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=1221]** .