IAR Application Note #13151

Event queue handling for events with parameters

The information in this document is based on version 4.2 of the IAR visualSTATE software. It may also apply to other versions of IAR visualSTATE.

SUMMARY

When an event queue is needed for storing events with parameters, the queue elements differ in size, thus making it difficult to add and retrieve elements. This note describes how to construct and use such a queue.

KEYWORDS

event, event parameter, event queue.

The problem to be solved

Events are typically stored in an event queue external to visualSTATE before being processed by the visualSTATE API. This event queue must be made and maintained by the user. When there are no events with parameters, an ordinary array with element type `SEM_EVENT_TYPE` can be used. However, when events have parameters, the parameters must be stored in the queue along with the event. If the number of and/or the type of the parameters differ, there is no predefined type that can be used as element type for such an array. To be able to use an array as storage for the queue, a common element type must be constructed and code must be written to add and retrieve elements.

Note: In this document, *design* is used to refer to a visualSTATE model/visualSTATE Project.

Solution

First, a common queue element type must be constructed. Then the type must be used as the element type for an event queue, and code must be written for adding and retrieving elements.

Common queue element type

The first step is to construct an element type that for each event can hold the parameters associated with that event. This must be done manually by inspecting all events in the design. As an example, assume that the design contains the following events and associated parameters:

- `e1 (VS_UINT par)`
- `e2 (VS_UINT8 par1, VS_BOOL par2)`
- `e3 ()`

On the basis of the events found and their associated parameters, construct a union for storage of the parameters:

```
typedef union
{
  struct event_e1
  {
    VS_UINT par;
  };
  struct event_e2
  {
    VS_UINT8 par1;
    VS_BOOL par2;
  };
} EventParameters;
```

Then construct the element type for the queue:

```
typedef struct
{
  SEM_EVENT_TYPE event;
  EventParameters parameters;
} QueueElement;
```

Adding and retrieving elements

The next step is to construct a queue with element type `QueueElement`. As a minimum the queue must have a function for adding elements onto the queue (`add`) and a function for retrieving elements from the queue (`retrieve`). The functions could have the following declarations (no error handling):

```
void add(QueueElement const * pQe);

void retrieve(QueueElement * pQe);
```

Note: It is not within the scope of this document to describe how to implement a queue. For a straightforward implementation of a queue, see the visualSTATE sample code. It is located in the `Examples\SampleCode` directory of your visualSTATE installation.

For example, if event `e2` occurs with parameters `par1` equal to 7 and `par2` false, the event is added to a global queue known by the `add` and `retrieve` functions. In the following code it is assumed that the queue can hold at least one more event:

```
QueueElement qe;
qe.event = e2;
qe.parameters.event_e2.par1 = 7;
qe.parameters.event_e2.par2 = 0; /* false value */
add(&qe);
```

When an event in the queue is to be processed by visualSTATE, the event and associated parameters are retrieved from the queue, and the API function `SEM_Deduct` is called (assuming the visualSTATE Basic API is used). In the following code it is assumed that there is at least one event in the queue):

```
QueueElement qe;
unsigned char cc;
retrieve(&qe);
switch (qe.event)
{
case e1:
  cc = SEM_Deduct(qe.event, qe.parameters.event_e1.par);
  break;
case e2:
  cc = SEM_Deduct(qe.event, qe.parameters.event_e2.par1,
    qe.parameters.event_e2.par2);
  break;
default: /* for events without parameters */
```

```
  cc = SEM_Deduct(qe.event);
}
if (cc != SES_OKAY)
  ; /* error handling */
/* continue with main loop */
```

After the call to `SEM_Deduct`, the returned completion code must be checked for errors, and the remaining part of a visualSTATE main loop must be executed (see IAR Application Note *Setting up the visualSTATE main loop with the IAR visualSTATE Basic API*).

Consequences

The outlined approach has some consequences that are difficult to avoid:

- The approach is tailored for a specific design. If events and/or parameters in the design are added, modified or deleted, the code for adding and retrieving events must also be modified. In some cases, the C compiler will not be able to detect the changes (for example addition of a parameter to an event); in this case it is crucial that the user keeps the hand-written code consistent with the design, otherwise run-time errors may occur. Note that this risk also exists when not using an event queue, i.e. when supplying too few parameters to the `SEM_Deduct` function directly.
- The size of the element type used in the queue is the size of the largest struct in the `EventParameters` union. This implies that there will be an overhead of allocated memory in the queue. This can be avoided by having a queue that dynamically creates and destroys elements of the required sizes, but the cost will typically be reduced speed.

Conclusions

The outlined solution is the most straightforward one and it is easy to implement. The drawback is that it relies on the user to write and maintain some amount of code that must be kept consistent with the visualSTATE design. However, when size and speed of the application are of prominent concern, alternative solutions seem to have even larger drawbacks.

References

IAR Application Note *Setting up the visualSTATE main loop with the IAR visualSTATE Basic API*.

visualSTATE sample code, which is located in the `Examples\SampleCode` directory of your visualSTATE installation.

Contact information

SWEDEN: IAR Systems AB
P.O. Box 23051, S-750 23 Uppsala
Tel: +46 18 16 78 00 / Fax: +46 18 16 78 38
Email: info@iar.se

USA: IAR Systems US HQ - West Coast
One Maritime Plaza, San Francisco, CA 94111
Tel: +1 415 765-5500 / Fax: +1 415 765-5503
Email: info@iar.com

USA: IAR Systems - East Coast
2 Mount Royal, Marlborough, MA 01752
Tel: +1 508 485-2692 / Fax: +1 508 485-9126
Email: info@iar.com

UK: IAR Systems Ltd
9 Spice Court, Ivory Square, London SW11 3UE
Tel: +44 20 7924 3334 / Fax: +44 20 7924 5341
Email: info@iarsys.co.uk

GERMANY: IAR Systems AG
Posthalterring 5, D-85599 Parsdorf
Tel: +49 89 900 690 80 / Fax: +49 89 900 690 81
Email: info@iar.de

DENMARK: IAR Systems A/S
Elkjærvej 30-32, DK-8230 Åbyhøj
Tel: +45 86 25 11 11 / Fax: +45 86 25 11 91
Email: info@iar.dk

March 2001.