IAR Application Note #29242

Setting up the visualSTATE main loop with the IAR visualSTATE Expert API

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

This application note shows an example of a main loop that can be used with visualSTATE code and the visualSTATE Expert API. In most cases you can use this main loop as a common framework in your application and use the parts you need. IAR visualSTATE Expert API should be used for visualSTATE Projects containing multiple visualSTATE Systems.

KEYWORDS

main loop, initialization, loop, deduct, action expressions, visualSTATE Expert API.

General

This application note describes how to write the visualSTATE main loop with the visualSTATE Expert API, including a code example that can be used as a framework for customizing your own visualSTATE main loop for the Expert API. The note also briefly describes the various parts of the main loop framework.

The visualSTATE Expert API are used for visualSTATE Projects containing multiple visualSTATE Systems. For example, a microcontroller used in a stereo that is to control a radio and a CD player could be modeled as a visualSTATE Project named `Stereo`. `Stereo` would contain the visualSTATE Systems `Radio` and `CD_player`.

For a description of visualSTATE Projects and Systems, see *IAR visualSTATE Reference Guide*.

Assumptions and conventions

In this document *visualSTATE main loop* is used to refer to the user-written code that handles the calls to the visualSTATE Expert API.

The examples in this application note are based on designs that do not use event arguments. If you need to use event arguments, refer to IAR Application Note *Event queue handling for events with parameters*.

The examples are based on a System called `vsSystem`. The files should be generated and included in the final application as described in *IAR visualSTATE API Guide*. If your System is named something else than `vsSystem`, replace `vsSystem` with the name of your System. If your action function pointer table is named something else than `VSAction,` replace `VSAction` with the name of the action function pointer table for your System.

The visualSTATE main loop

The visualSTATE main loop serves two purposes:

- It initializes visualSTATE with a number of visualSTATE Expert API calls.
- It calls the visualSTATE Expert API functions that handle the run-time behavior of visualSTATE.

Writing the visualSTATE main loop

Writing of the visualSTATE main loop falls in three steps:

Step 1	You should define some variables that are needed to hold the current visualSTATE event, action function, and a completion code. Normally you can use the same code lines in all cases for defining the variables.
Step 2	You should initialize the visualSTATE run-time model by calling the appropriate visualSTATE Expert API functions. These calls depend on the model you have designed.
Step 3	You should make a loop that keeps calling the visualSTATE Expert API functions that are needed to make your model work at run-time. These calls depend on how you want to access the user-written action functions that are used in the designed model.

In most cases the following basic structure can be used:

```
void vSSystemMainLoop (void)

{
  /* Step 1, Define variables */
  /* Step 2, Initialize */
  /* Step 3, Loop forever */
}
```

The structure matches three steps that are described separately in the following, including small code fragments. After the description of the individual steps, a complete example of code is shown.

Step 1: Defining the visualSTATE variables

The purpose of this step is simply to define the variables that are needed by visualSTATE. Normally four variables must be defined:

- A variable for storing the current completion code. Most visualSTATE Expert API functions return a completion code. For detailed information on completion codes, refer to *IAR visualSTATE API Guide*, or the `SEMLibE.h` header file.
- A variable for storing the current visualSTATE action expression number. The number will later be returned from the visualSTATE API call `SMP_GetOutput(…)` and used afterwards.
- A variable for storing a pointer to a `SEM_CONTEXT` data structure. The pointer will be used by the visualSTATE Expert API. It will be allocated and filled with data required by visualSTATE. The context pointer is needed by most Expert API functions.

- A variable for storing the current event. It will be used for sending the event into visualSTATE and for getting the next event. In this example, the variable is initialized at the definition.

If some other variables are needed for the main loop, you can of course add them to the code.

Defining the visualSTATE variables can be done in the following manner:

```
void vSMainLoop (void)
{
  /* Step 1 */
  /* Define completion code storage */
  unsigned char cc;

  /* Define action expression variable. */
  SEM_ACTION_EXPRESSION_TYPE actionExpressNo;

  /* Define SEM_CONTEXT storage */
  SEM_CONTEXT *pSEMContext;

  /* Define and initialize event variable.
   * In this case the reset event is SE_RESET.
   */
  SEM_EVENT_TYPE eventNo = SE_RESET;
```

The variable types `SEM_EVENT_TYPE` and `SEM_ACTION_EXPRESSION_TYPE` are defined by visualSTATE. They should always be used to contain events and action functions.

Step 2: Initialization

The `SMP_Connect` function creates a `SEM_CONTEXT` structure that is needed by visualSTATE for storing run-time data and initializing the context. This function must be the first visualSTATE Expert API function that is called.

In the Expert API one function handles all the initialization of all other visualSTATE data. The function is generated by the visualSTATE Coder. In this example, the function is named `vsSystemSMP_InitAll`.

If some other variables must be initialized before calling the loop, you can of course add them to the code.

Initialization can be done in the following manner:

```
 /* Step 2 */
 /* Initialize the VS System. */
  if ((cc = SMP_Connect(&pSEMContext, &vsSystem)) != SES_OKAY)
    ; /* error handling */

  /* Initialize all needed functions */
  vsSystemSMP_InitAll(&pSEMContext);
```

Step 3: Creating the loop

The purpose of this step is to run through the needed functions so that visualSTATE will react on the events that occur.

The loop runs forever. First the event is sent to visualSTATE by calling `SMP_Deduct(…)`. This will prepare visualSTATE for other actions that are necessary.

Then the resulting action expressions are retrieved from visualSTATE by continuously calling `SMP_GetOutput` for as long as it returns `SES_FOUND`. Returning `SES_FOUND` indicates that a new action expression has been found as a result of the event. Then the `SMP_TableAction` macro is called with the action expression number as argument. The `SMP_TableAction` macro uses the specified function pointer table to call the right action expression function. When `SMP_GetOutput` has completed finding the action expressions that are a result of the current event, `SES_OKAY` is returned.

The internal next state configuration is updated in visualSTATE by calling `SMP_NextState`.

Finally the next event should be found by your function.

```
  /* Step 3 */
  /* Do forever */
  for (;;)
  {
    /* Deduct the event. */
    if ((cc = SMP_Deduct(pSEMContext, eventNo)) != SES_OKAY)
      ; /* error handling */

    /* Get resulting action expressions and execute them. */
    while ((cc = SMP_GetOutput(pSEMContext, &actionExpressNo)) == SES_FOUND)
      SMP_TableAction(pSEMContext, VSAction, actionExpressNo);
    if (cc != SES_OKAY)
      ; /* error handling */

    /* Change the next state vector. */
    if ((cc = SMP_NextState(pSEMContext)) != SES_OKAY)
      ; /* error handling */

    /* Get next event to process. Write that function yourself */
    /* GetNextEvent(&eventNo); */
  }
}
```

All functions prefixed with `SMP_` are defined in the IAR visualSTATE Expert API.

Error handling

Error handling depends on your target application. Normally you should not get any errors in *Step 3* if you have tested and verified your design with the visualSTATE Validator and Verificator. If errors occur, you could choose to halt the target processor, or show in some manner that a serious error has occurred.

Event queueing

You may use an event queue or some other way to store the event. For detailed information on event queues, refer to IAR Application Note *Event queue handling for events with parameters*, or the visualSTATE sample code located in the `Examples\SampleCode` directory of your visualSTATE installation.

Example of a complete visualSTATE main loop for IAR visualSTATE Expert API

```
void vSSystemMainLoop (void)
{
  /* Step 1, Define variables */
  /* Define completion code storage */
  unsigned char cc;

  /* Define action expression variable. */
  SEM_ACTION_EXPRESSION_TYPE actionExpressNo;

  /* Define SEM_CONTEXT storage */
  SEM_CONTEXT *pSEMContext;

  /* Define and initialize event variable.
   * In this case the reset event is SE_RESET.
   */
  SEM_EVENT_TYPE eventNo = SE_RESET;

  /* Step 2, Initialize */
  /* Initialize the VS System. */
  if ((cc = SMP_Connect(&pSEMContext, &vsSystem)) != SES_OKAY)
    ; /* error handling */

  /* Initialize all needed data */
  vsSystemSMP_InitAll(&pSEMContext);


  /* Step 3, Loop */
  /* Do forever */
  for (;;)
  {
    /* Deduct the event. */
    if ((cc = SMP_Deduct(pSEMContext, eventNo)) != SES_OKAY)
      ; /* error handling */

    /* Get resulting action expressions and execute them. */
    while ((cc = SMP_GetOutput(pSEMContext, &actionExpressNo)) == SES_FOUND)
      SMP_TableAction(pSEMContext, VSAction, actionExpressNo);
    if (cc != SES_OKAY)
      ; /* error handling */

    /* Change the next state vector. */
    if ((cc = SMP_NextState(pSEMContext)) != SES_OKAY)
      ; /* error handling */

    /*
     * Get next event to process.
     * This function must be written by the user.
     */
    /* GetNextEvent(&eventNo); */
  }
}
```

Conclusions

The code shown in *Example of a complete visualSTATE main loop for IAR visualSTATE Expert API* can be used for most applications that use the visualSTATE Expert API. Defining variables, initializing visualSTATE and writing the loop is all you need to do.

References

IAR Application Note *Event queue handling for events with parameters*.

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

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

IAR visualSTATE Expert API is described in *IAR visualSTATE API Guide*.

For a description of visualSTATE Projects and Systems, see *IAR visualSTATE Reference Guide*.

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