

# Using extended keywords for C compilers in a visualSTATE design

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 describes why you may want to use extended keywords in your visualSTATE design. It shows an example of how extended keywords can be used for achieving some specific run-time behavior in code generated by IAR visualSTATE.

## KEYWORDS

keywords, extended keywords, processor, run-time behavior, C compiler.

## The problem to be solved

In some situations it may be necessary or desirable to use extended keywords with the ANSI C code generated by IAR visualSTATE Coder. The keywords may be needed in order to place some data in specific memory areas, or because the visualSTATE-generated code does not fit into the memory areas normally used.

Many C compilers offer extensions to ANSI C in order to achieve some specific behavior in the target processor. It is not desirable to modify the IAR visualSTATE auto-generated C files manually because the manual changes will be overwritten by the next visualSTATE code generation. Instead you should take advantage of the support for extended keywords in IAR visualSTATE.

## Solution

This application note describes how to set up extended keywords for a visualSTATE design, and explains why you may need extended keywords.

The example shown here is based on the `AVSystem` example that is included with the IAR visualSTATE software, but the solution can be applied to any visualSTATE model.

For detailed information on the extended keywords supported by your C compiler, refer to your C compiler documentation.

## Setting up extended keywords for visualSTATE designs

Extended keywords for visualSTATE code generation are set up in IAR visualSTATE Navigator as follows:

- 1) Launch the Navigator and open the appropriate visualSTATE Project. Select the visualSTATE System for which to generate code. On the Navigator menu select **View > Properties**. See *Figure 1*.

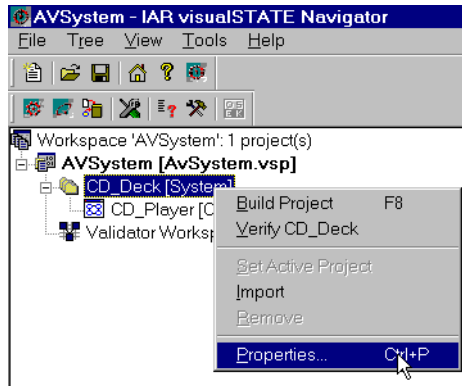


Figure 1: Opening Build properties in IAR visualSTATE Navigator

- 2) A Properties dialog box will be displayed. Click the Build tab to open a dialog box for specifying IAR visualSTATE Coder options. Select the **Extended keywords** option. See *Figure 2*.

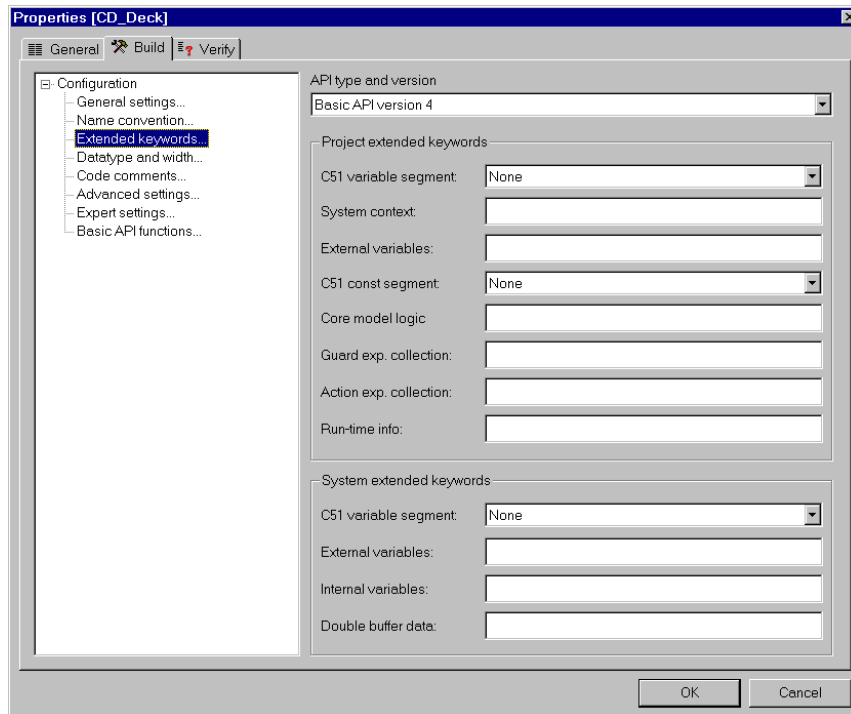


Figure 2: Dialog box for specifying extended keywords

**Note:** For each of the possible extended keyword settings described in the following, it is stated whether the setting applies to variable data or constant data. If you specify an extended keyword for variable data, you should only use keywords that are intended for data in the writable memory areas of your hardware.

## visualSTATE Project extended keywords

Each visualSTATE Project may have its own extended keywords settings. See *Figure 2*.

### C51 variable segment

Here you specify where to place visualSTATE variable data for a C51 processor. The *variable data* comprise the **System context** and **External variables** (see *Figure 2*). These options are described on page 3 of this document.

The settings specified in **C51 variable segment** become the default settings for the System context and external variables.

#### Description of C51 variable segment options

- None:** No C51 keywords are used. Select this value if the intended target does not support C51 keywords.
- DATA segment:** Coder-generated variable data are stored in DATA.
- IDATA segment:** Coder-generated variable data are stored in IDATA.
- PDATA segment:** Coder-generated variable data are stored in PDATA.
- XDATA segment:** Coder-generated variable data are stored in XDATA.

If you specify a setting in **System context** or a keyword in **External variables**, this setting will take precedence.

#### Example

If you specify **DATA** in **C51 variable segment**, and **\_\_flash** in **External variables**, the external variables will have the keyword `__flash`, and the **DATA** keyword will be ignored. The **DATA** keyword will still be used for the System context.

### System context (variable data)

You can specify an extended keyword string that will be used for the System context variable(s). The System context is for internal usage by visualSTATE. In the visualSTATE Expert API, it is used for the `SEM_CONTEXT` data structure.

### External variables (variable data)

Here you can specify an extended keyword string that will be used for external variables.

### C51 const segment

(See *Figure 2*) Here you specify where to place visualSTATE constant data for a C51 processor. The *constant data* comprise the core model logic, the guard expression collection, the action expression collection and the run-time information (described below).

The settings specified in **C51 const segment** become the default settings for the constant data.

### **Description of C51 const segment options**

**None:** No C51 keywords are used. Select this value if the intended target does not support C51 keywords.

**CODE segment:** Coder-generated constant data are stored in CODE.

If you specify a setting for one of the constant data items described in the following this setting will take precedence.

### **Example**

If you specify **CODE** for the C51 const segment and **\_\_flash** for the core model logic, the core model logic will have the keyword `__flash` and the **CODE** keyword will be ignored. The **CODE** keyword will still be used for the three other constant data items.

### **Core model logic (constant data)**

You can specify an extended keyword string that will be used for the core model logic struct variable(s). The core model logic is the data that represents most of your visualSTATE design. The data are only used by visualSTATE.

### **Guard exp. collection (constant data)**

You can specify an extended keyword string that will be used for the guard expression collection variable(s).

### **Action exp. collection (constant data)**

You can specify an extended keyword string that will be used for the action expression collection variable(s).

### **Run-time info (constant data)**

You can specify an extended keyword string that will be used for the run-time info struct variable. The run-time info struct contains the digital signature for the visualSTATE Project.

## **visualSTATE System extended keywords**

Each System may have its own extended keyword settings. See *Figure 2*.

### **C51 variable segment**

System extended keywords, correspond to the Project extended keyword described above.

### **External variables**

System extended keywords, correspond to the Project extended keyword described above.

### **Internal variables (variable data)**

You can specify an extended keyword string that will be used for the internal variables in the System.

### **Double buffer data (variable data)**

You can specify an extended keyword string that will be used for the double buffer variable in the System.

## **Examples of extended keywords for C compilers**

`__tiny`  
`__far`  
`__huge`  
`__flash`  
`__farflash`  
`__eeprom`  
`near`  
`far`  
`huge`

Refer to your compiler documentation for a complete list of extended keywords supported by your compiler.

## **Example: Using extended keywords with Atmel AT90S8515 processor**

If you use the Atmel AT90S8515 processor, the visualSTATE generated core model logic is placed in RAM by default.

Atmel AT90S8515 has 512 bytes of RAM available. This may not be enough RAM to handle the code and data needed in an application for that target. As mentioned above, the visualSTATE core model logic is constant data. Thus it would be preferable to store this constant data somewhere else than in the RAM of the processor.

The IAR C compiler EWAVR for Atmel AVR supports the extended keyword `__flash` (see *Figure 3*). If this keyword is specified for an object, the object is placed in code space. The object must be declared as constant.

IAR Application Note #35915  
Using extended keywords for C compilers in a visualSTATE design

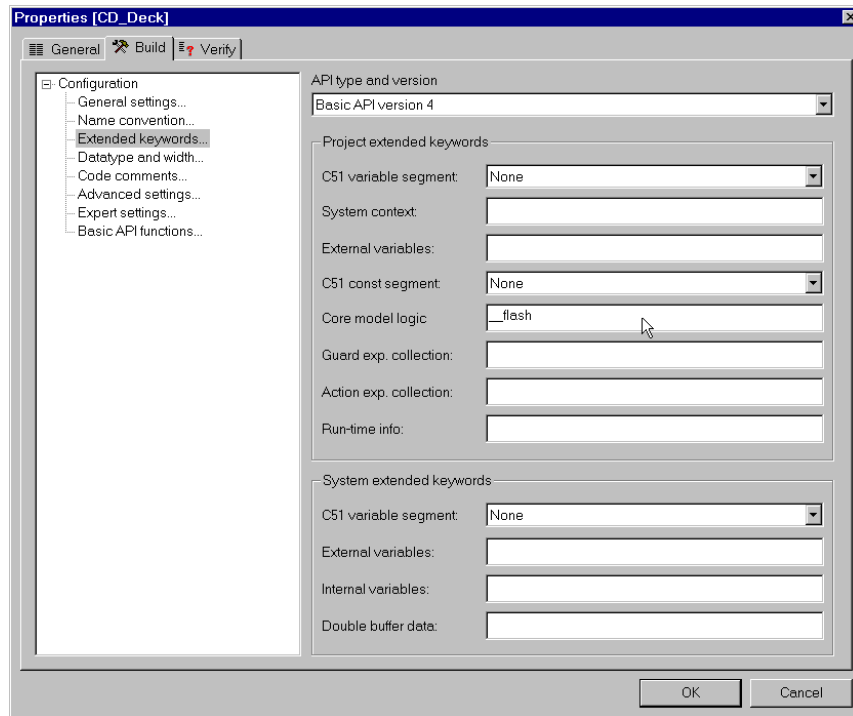


Figure 3: Specifying extended keyword `__flash` for visualSTATE core model logic

In Figure 3, the extended keyword `__flash` has been specified.

When you have specified the extended keywords, code-generate the visualSTATE model as you would usually do.

In this example, code is generated for the AVSystem example included in the visualSTATE software.

The Coder-generated file `CD_Deck.c` contains the following code for the core model logic. The `VS_DATA` structure is the core model logic.

```
...
/*
 * VS System Data Declaration and Initialization.
 *
 * VS System Informations:
 * - Rule data format number: 1
 */
VSDATA __flash VS =
{
...

```

The Coder-generated `SEMBDef.h` file also reflects the specified keyword:

```
...
/*
 * Type qualifier macros
 */
#define VS_TQ_RULEBASE          __flash
...
/*
 * Data External Declaration.

```

```
*/  
extern VSDATA __flash VS;  
...
```

The `VS_TQ_RULEBASE` type qualifier macro must be used in your code if you want direct access to names or explanations of the variables at run-time. If you specify the pointers as shown below, you will be able to access the data, even if the `__flash` keyword is changed later in the development process.

```
void WriteExplanationForState (SEM_EXPLANATION_TYPE nStateNo)  
{  
    unsigned char cc;  
    char VS_TQ_RULEBASE *pExpl;  
  
    if ((cc = SEM_ExplAbs(STATE_TYPE, nStateNo, &pExpl)) != SES_OKAY)  
        ; /* Error handling */  
    /* Use the pointer to write the explanation to e.g. a display */  
    ...  
}
```

For detailed information about the `SEM_ExplAbs` function, refer to *IAR visualSTATE API Guide*.

## Conclusions

The use of extended keywords may be necessary in order to fit the code, constant data and variable data into your target processor. `visualSTATE` offers an easy and flexible method for specifying just the extended keywords you need for the various parts of the code and data generated by `visualSTATE`. `visualSTATE` keyword settings should be used in preference to editing the `visualSTATE` generated code manually.

The keywords specified in `visualSTATE` will automatically be placed in the right position of the generated code each time it is generated.

It is easy to access parts of the `visualSTATE`-generated code. To achieve access to the code, define the variables for accessing the `visualSTATE` data, and set up the keywords using the predefined type qualifier macros supported by `visualSTATE`.

## References

For information on extended keyword settings, refer to *IAR visualSTATE User Guide*.

For a list of supported keywords for C compilers, refer to your C compiler documentation.

IAR Application Note #35915  
Using extended keywords for C compilers in a visualSTATE design

---

**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

---

© Copyright 2001 IAR Systems. All rights reserved.

The information in this document is subject to change without notice and does not represent a commitment on any part of IAR Systems. While the information contained herein is assumed to be accurate, IAR Systems assumes no responsibility for any errors or omissions.

visualSTATE is a registered trademark of IAR Systems. IAR visualSTATE RealLink, IAR Embedded Workbench and IAR MakeApp are trademarks of IAR Systems. Microsoft is a registered trademark, and Windows is a trademark of Microsoft Corporation. All other product names are trademarks or registered trademarks of their respective owners.

March 2001.

---