LSL Sample Cases using EDE

TASKING™
Embedded software development from Altium™

The aim of this document is to demonstrate the usage of the EDE's **Script File** page while using the TriCore toolchain to locate your application. It contains examples that deal with the most commonly used locating aspects. It first lists all questions and continues with the proposed solutions. Studying this document will take about one hour of your time. If your howto is not amongst the ones listed in this document, EDE will probably not support it. In that case please refer to *"LSL Sample Cases using the Control Program"* which demonstrates more complex cases and offers an introduction to the *Linker Script Language*.

While building the sample solutions, always choose the **TC1796B** target:

1. From the **Project menu**, select **Project Options...**
2. Expand the **Processor** entry and select **Processor Definition**.
3. From the **Target processor** dropdown box, select **TC1796B**.

In addition you will need to select the **Script File** project options:

1. From the **Project** menu, select **Project Options...**
2. Expand the **Linker** entry.
3. Expand the **Script File** entry.

   And from here, depending on the example, either of the following pages:

   - **Special Areas**
   - **Defines/Stack/Heap**
   - **Internal Memory**
   - **External Memory**
   - **Sections**
   - **Output Sections**
   - **Reserved**

You are encouraged to verify the results of the locator by checking the map file. Alternatively you can debug the examples using for example CrossView Pro's simulator execution environment.

THE QUESTIONS...

1. How do I locate a function within a specific memory definition?
2. How do I locate a function at an absolute memory offset address?
3. How do I locate a function at an absolute address?
4. I have several functions that must be located within a specific memory definition. Their order must be the order in which they were defined plus that they must be contiguous. How do I do this?
5. How do I locate a variable at a fixed address without using the `__at()` memory qualifier?
6. I know how I can address linker labels for the stack or heap. But how can I change their size and/or start address?
7. Within the memory map of our application we have specific memory parts that must be reserved. No data may be located within this range other than data that is located by means of an absolute address. Is there a way?

THE PROPOSED SOLUTIONS...

QUESTION 1

How do I locate a function within a specific memory definition?

Solution: Assume the following C-module:

```
#include <stdio.h>

extern void _lc_ub__text_test_Sample[];
extern void _lc_ue__text_test_Sample[];

void Sample(void)
{
    printf("_lc_ub__text_test_Sample: 0x%p\n",
            _lc_ub__text_test_Sample);
    printf("_lc_ue__text_test_Sample: 0x%p\n",
            _lc_ue__text_test_Sample);
}

void main(void)
{
    Sample();
}
```

Suppose you want to locate function *Sample* and suppose the C-module is called *test.c* then according to TriCore's section naming convention (See section 3.10, *compiler generated sections* of the User's Manual) *Sample* will be assigned to the following code section:

```
".text.test.Sample"
```

Suppose *Sample* must be located within memory *ext_c* such as defined in the **External Memory** page, then the **Sections** page pictured below will accomplish this:

Space	Sections	Group	Copy	Alloc	Location	Name
linear	.text.test.Sample	unrestricted	NO	extmem	ext_c	

EDE will translate this to a nameless group. In here:

- ***linear*** locates the nameless group within TriCore's 4 GB address space
- ***.text.test.Sample*** is added to the nameless group.
- ***unrestricted*** imposes no restrictions to sections within the nameless group.
- ***NO*** implies a non-copy group.
- ***extmem*** restricts the nameless group to an external memory definition.
- ***ext_c*** identifies which external memory definition.

QUESTION 2

How do I locate a function at an absolute memory offset address?

Solution: Supposing the same test module as with Solution 1 and supposing a memory offset equal to 32k within memory *ext_c*, then the **Sections** page listed below will accomplish this:

Space	Sections	Group	Copy	Alloc	Location	Name
linear	.text.test.Sample	ordered	NO	extmem	ext_c[32k]	

EDE will translate this to a nameless group. In here:

- ***linear*** locates the nameless group within TriCore's 4 GB address space
- ***.text.test.Sample*** is added to the nameless group.
- ***ordered*** restricts sections within the nameless group to retain their defined order.
- ***NO*** implies a non-copy group.
- ***extmem*** restricts the nameless group to an external memory definition.
- ***ext_c[32k]*** locates nameless group to a memory offset of 32k within *ext_c*.

Note that since the nameless group is located to an absolute – and therefore restricted - memory offset address the Group-attribute changed from ***unrestricted*** to ***ordered***.

QUESTION 3

How do I locate a function at an absolute address?

Solution: Supposing the same test module as with Solution 1 and supposing an absolute address of *0xA100C000*, then the **Sections** page listed below will accomplish this:

Space	Sections	Group	Copy	Alloc	Location	Name
linear	.text.test.Sample	ordered	NO	abs	0xA100C000	

EDE will translate this to a nameless group. In here:

- ***linear*** locates the nameless group within TriCore's 4 GB address space
- ***.text.test.Sample*** is added to the nameless group.
- ***ordered*** restricts sections within the nameless group to retain their defined order.
- ***NO*** implies a non-copy group.
- ***abs*** restricts the nameless group to an absolute address.
- ***0xA100C000*** is the absolute address.

QUESTION 4

I have several functions that must be located within a specific memory definition. Their order must be the order in which they were defined plus that they must be contiguous. How do I do this?

Solution: Assume the following C-module called test.c:

```
#include <stdio.h>

extern void _lc_ub__text_test_SampleB[];
extern void _lc_ue__text_test_SampleE[];

void SampleB(void)
{
   printf("_lc_ub__text_test_SampleB: 0x%p\n",
           _lc_ub__text_test_SampleB);
}

void SampleE(void)
{
   printf("_lc_ue__text_test_SampleE: 0x%p\n",
           _lc_ue__text_test_SampleE);
}

void main(void)
{
    SampleB();
    SampleE();
}
```

Suppose you want to locate functions *SampleB* and *SampleE* then from Solution 1 it follows that these are allotted to the sections listed below:

```
".text.test.SampleB"
".text.test.SampleE"
```

Further suppose function *SampleB* must be located before *SampleE* then the **Sections** page listed below will accomplish this:

Space	Sections	Group	Copy	Alloc	Location	Name
linear	.text.test.SampleB;.text.test.SampleE	sequential	NO	extmem	ext_c	

EDE will translate this to a nameless group. In here:

- ***linear*** locates the nameless group within TriCore's 4 GB address space
- ***.text.test.SampleB*** and ***.text.test.SampleE*** are added to the nameless group.
- ***sequential*** resticts the grouped sections to be ordered and contiguous
- ***NO*** implies a non-copy group.
- ***extmem*** restricts the nameless group to an external memory definition.
- ***ext_c*** identifies which external memory definition.

Note the subtle difference between using ***ordered*** and ***sequential***. The first tells something about the order but does not prevent (unrestricted) sections or groups to be inserted because these do not affect the order. The second not only orders the sections but also makes them contiguous i.e. no sections or groups will be inserted. The only exception in case of the latter is that alignment gaps may be replaced my small enough groups or sections.

QUESTION 5

How do I locate a variable at a fixed address without using the `__at()` *memory qualifier?*

Solution: Assume the following C-module called test.c:

```
#include <stdio.h>

extern void _lc_ub__bss_test_MMRDeviceOne[];
extern void _lc_ub__bss_test_MMRDeviceTwo[];

__far unsigned MMRDeviceOne;
__far unsigned MMRDeviceTwo;

void main(void)
{
     while ( _lc_ub__bss_test_MMRDeviceOne != ((void*)0xA1080100) );
     while ( _lc_ub__bss_test_MMRDeviceTwo != ((void*)0xA1080200) );
     puts("memory mapped devices have been properly located");
}
```

In this example memory mapped registers *MMRDeviceOne* and *MMRDeviceTwo* are supposed to reside at addresses *0xA1080100* and *0xA1080200* respectively. If they are not, the program will lock at either the 1st or 2nd while-statement. Locating both memory mapped registers to their designated addresses first of all requires one unique section per data object. From EDE:

1. From the **Project** menu, select **Project Options...**
2. Expand the **C-Compiler** entry and select the **Code Generation** page.
3. Tick the **Generate a section for each data** object checkbox.

This will assure variables *MMRDeviceOne* and *MMRDeviceTwo* will reside on the following sections:

```
".bss.test.MMRDeviceOne"
".bss.test.MMRDeviceTwo"
```

The **Sections** page below locates the memory mapped registers using memory relative addresses to external memory *ext_d* as defined in the **External Memory** page:

Space	Sections	Group	Copy	Alloc	Location
linear	.bss.test.MMRDeviceOne	ordered	NO	extmem	ext_d[0x100]
linear	.bss.test.MMRDeviceTwo	ordered	NO	extmem	ext_d[0x200]

Or alternatively using absolute addresses:

Space	Sections	Group	Copy	Alloc	Location
linear	.bss.test.MMRDeviceOne	ordered	NO	abs	0xA1080100
linear	.bss.test.MMRDeviceTwo	ordered	NO	abs	0xA1080200

QUESTION 6

I know how I can address linker labels for the stack or heap. But how can I change their size and/or start address?

Solution: Assume the following C-module:

```
#include <stdio.h>

extern void _lc_ub_ustack[];
extern void _lc_ue_ustack[];
extern void _lc_ub_heap[];
extern void _lc_ue_heap[];

void main(void)
{
    printf("_lc_ub_ustack = 0x%p\n", _lc_ub_ustack);
    printf("_lc_ue_ustack = 0x%p\n", _lc_ue_ustack);
    printf("_lc_ub_heap = 0x%p\n", _lc_ub_heap);
    printf("_lc_ue_heap = 0x%p\n", _lc_ue_heap);
}
```

Further suppose this application requires a relatively small stack size of 256 bytes which must be offset to 32k of memory definition *ext_d*. The heap - though not used - should be 2k in size and located to an offset of 40k within the same memory. Listed below is an extract from the **Defines/Stack/Heap** page that does just that:

User stack size:	256
User stack start address:	0xA1088000
Heap size:	2k
Heap start address:	0xA108A000

QUESTION 7

Within the memory map of our application we have specific memory parts that must be reserved. No data may be located within this range other than data that is located by means of an absolute address. Is there a way?

Solution: Assume the following C-module called *test.c*:

```
#include <stdio.h>
#include <stdlib.h>

__far volatile unsigned DACSample __at(0xA1080000);

static inline unsigned NewSample(void)
{
    return DACSample = rand();
}

void main(void)
{
    while (1)
    {
        printf("Current DAC Noice Sample: 0x%04X\n", NewSample());
    }
}
```

Suppose that memory range *0xA1080000* through *0xA10800FF* is reserved memory and may not be used for unrestricted relocatable application data. However memory mapped register *DACSample*, having an absolute address, should still be allowed. Listed below is EDE's **Reserved** page that makes this possible:

Name	Size	Address	Allow
mmio	0x100	0xA1080000	absolute

In here:

- ***mmio*** is the name of the reserved group as it will appear in the map file.
- ***0x100*** is the size of the reserved group.
- ***0xA1080000*** is the start address of the reserved group.
- ***absolute*** allows absolute sections to (still) be located within the reserved range.

Using ***none*** rather than ***absolute*** will also exclude absolute sections and, for the above, therefore result in a linker error message.