Partner “How-To”

Contents

[Introduction 2](#_Toc427585466)

[The .i79 file 3](#_Toc427585467)

[Syntax 3](#_Toc427585468)

[Sections and parameters 3](#_Toc427585469)

[The FILEFORMAT section 3](#_Toc427585470)

[The CHIP section 3](#_Toc427585471)

[The EMUPARAMS section 5](#_Toc427585472)

[The CORE section 5](#_Toc427585473)

[The DDF FILE section 6](#_Toc427585474)

[The LINKER FILE section 6](#_Toc427585475)

[The FLASH LOADER section 6](#_Toc427585476)

[C-SPY Macros 7](#_Toc427585477)

[Customized reset strategy as C-SPY macro 8](#_Toc427585478)

[The flash programming process in even more detail 11](#_Toc427585479)

[The Linker configuration file editor 14](#_Toc427585480)

# Introduction

This document provides information for device support developers, together with the following documents found in the arm\doc folder of a EWARM installation:

EWARM\_DDFFormat.pdf  
EWARM\_HeaderFormat.pdf  
EWARM\_HeaderTemplate.pdf  
EWARM\_IjetConnectMacros.pdf  
FlashLoaderGuide.ENU.pdf  
EWARM\_DebuggingGuide.ENU.pdf

TBD: Is this relevant?:  
core\ide\Doc\IAR Project Connection.pdf

TBD, consolidate all relevant docs into one, with content:  
The Linker configuration file editor  
.i79 file  
Flashloader Guide  
 The flash programming process in even more detail  
DDF format  
C-SPY macros  
I-jet connect macros

# The .i79 file

## Syntax

[<section>]

<key>=<value>

// Comment

A <section> includes a logical group of parameters. Multiple sections may be specified within one i79 file where every section name must be unique. A section may be mandatory or optional.

<key> is the name of a parameter and must be unique within the <section>. A section may contain multiple parameters. A parameter may have more than one instance if explicitly said so, otherwise it shall occur only once or not at all. A parameter may be mandatory or optional.

<value> is the value of the parameter.

## Sections and parameters

### The FILEFORMAT section

[FILEFORMAT] Mandatory

rev Mandatory The format version of the content of this file.  
1.6 is the latest format (November 2014). Example:  
rev=1.6

### The CHIP section

[CHIP] Mandatory Device and board specifics

name Mandatory Device name, specified by the vendor. This name must be unique within the scope of EWARM. Example:  
name=ACME123ABC

endiansupport Optional Endian supported by this device. Available values:  
le\_be8 Little and BE8  
le\_be32 Little and BE32  
le Little  
be8\_be32 BE8 and BE32  
be8 BE8  
be32 BE32  
le\_be8\_be32 All options supported  
Default: le\_be8\_be32  
Example:  
endiansupport=be8

thumbsupport Optional Specifies whether the THUMB instruction set is supported or not. Available values are true or false. Default: true

armsupport Optional Specifies whether the ARM instruction set is supported or not. Available values are true or false. Default: true

fpu Optional Specifies supported FPU coprocessor. Available values: VFPv2, VFP9-s, VFPv3, VFPv3\_d16, VFPv4\_sp, VFPv4, VFPv4\_d16, VFPv5\_sp, VFPv5\_d16, None. Default: None. Example:  
fpu=VFPv4

simd Optional Specifies SIMD (NEON) support. Available values are true or false. Default: false.

IElfToolPostProcess Optional This parameter is used if it is mandatory to run ielftool as a final step when building an application for a device, for instance to insert a checksum which is validated by a boot loader. Example:  
IElfToolPostProcess=--checksum \_\_vector\_table+0x1c:4,sum32:2;\_\_vector\_table-\_\_vector\_table+0x1B

#### Debugger generic parameters

jtag Optional Specifies JTAG support. Available values are true or false. Default: true

cjtag Optional Specifies CJTAG support. Available values are true or false. Default: false

rtck Optional Specifies RTCK support. Available values are true or false. Default: true

swd Optional Specifies SWD support. Available values are true or false. Default: true for Cortex cores, false otherwise.

swo\_traced0 Optional Specifies support for SWO on the traceD0 pin. Available values are true or false. Default: false

swo\_clock Optional SWO clock frequency in MHz, given as integer or float.

DebuggerExtraOption Optional Note, multiple entries are supported for this option.  
Extra debugger options can be provided in this parameter, in the same manner as in the project settings in the Embedded Workbench:  
“Project->Options->Debugger->Extra Options”.  
Example:  
DebuggerExtraOption=--drv\_vector\_table\_base=0

DeviceMacros Optional Note, multiple entries are supported for this option.  
This parameter specifies a C-SPY macro file, with full path, that will be loaded throughout the complete debug session, including the flash loader phase. Example:  
DeviceMacros=$TOOLKIT\_DIR$\config\debugger\acme\acme123xxx.dmac

#### Debugger I-jet/JTAGjet parameters

MtbRegBase Optional Base of MTB registers. Example:  
MtbRegBase=0xf0000000

MtbDwtRegBase Optional Base of MTB DWT registers. Example:  
MtbDwtRegBase=0xf0001000

MtbMinRamAddr Optional Lower limit of MTB buffer RAM. Example:  
MtbMinRamAddr=0x20000000

MtbMaxRamAddr Optional Upper limit of MTB buffer RAM. Example:  
MtbMaxRamAddr=0x20005FFF

ProbeConfigFile Optional The name, with full path, of the file that specifies the configuration of the debug system on the board/device. Example:  
ProbeConfigFile=$TOOLKIT\_DIR$\config\debugger\acme\acme123.ProbeConfig

ProbeScriptFile Optional The name, with full path, of the file that provides scripts for customized reset strategies. (This file format is deprecated and will eventually be replaced by C-SPY macros provided in files specified by the DeviceMacros parameter.) Example:  
ProbeScriptFile=$TOOLKIT\_DIR$\config\debugger\acme\acme123.ProbeScript

#### Debugger J-Link/J-Trace parameters

JLinkDeviceName Optional The name of the device in the J-Link realm, if it differs from the name specified by the vendor (see [CHIP].name above).

JLinkScriptFile Optional The name, with full path, of the file that contains J-Link specific scripts for this device. Example:  
JLinkScriptFile=$TOOLKIT\_DIR$\config\debugger\acme\acme123.JLinkScript

#### Debugger XDS100/200/ICDI parameters

XdsDeviceName Optional A device identifier that is used as prefix in the name of the XDS .dat file found in $TOOLKIT\_DIR$\config\debugger\TexasInstruments\xds.

### The EMUPARAMS section

[EMUPARAMS] Optional I-jet/JTAGjet low-level parameters

<emukey>=<value> Optional Parameters for configuration of low level mechanisms in the jet driver. Description of available parameters and corresponding value ranges is provided when needed.  
Examples:  
Device=Vybrid  
L2CC=0x3FFFF000

### The CORE section

[CORE] Mandatory

name Mandatory Specifies the core of the device. Currently available values (November 2014): ARM1020E, ARM1022E, ARM1026EJ-S, ARM10E, ARM1136J, ARM1136J-S, ARM1136JF, ARM1136JF-S, ARM1156T2-S, ARM1156T2F-S, ARM1176J, ARM1176J-S, ARM1176JF, ARM1176JF-S, ARM1176JZ, ARM1176JZ-S, ARM1176JZF, ARM1176JZF-S, ARM710T, ARM720T, ARM740T, ARM7EJ-S, ARM7TDMI, ARM7TDMI-S, ARM920T, ARM922T, ARM926EJ-S, ARM940T, ARM946E-S, ARM966E-S, ARM968E-S, ARM9E, ARM9E-S, ARM9TDMI, Cortex-M0, Cortex-M0+, Cortex-M1, Cortex-Ms1, Cortex-M3, Cortex-M4, Cortex-M4F, Cortex-M7, Cortex-R4, Cortex-R4F, Cortex-R5, Cortex-R5F, Cortex-R7, Cortex-R7F, Cortex-A5, Cortex-A5F, Cortex-A7, Cortex-A8, Cortex-A9, Cortex-A15, SC000, SC100, SC300, XScale, XScale-IR7.  
Example:  
name=Cortex-M3

### The DDF FILE section

[DDF FILE] Mandatory

name Mandatory Specifies the device description file (ddf) for this device, with path relative to “$TOOLKIT\_DIR$\config\debugger”. Example:  
name=acme\acme123xxx.ddf

### The LINKER FILE section

[LINKER FILE] Optional

name Optional Specifies the default linker configuration file (icf) for this device, with full path. Example:  
name=$TOOLKIT\_DIR$\config\linker\acme\acme123xxx.icf

### The FLASH LOADER section

[FLASH LOADER] Optional

big Optional Specifies the flash loader board file, prepared for big endian, for this device, with full path. Example:  
big=$TOOLKIT\_DIR$\config\flashloader\acme\acme123xxxBE.board

little Optional Specifies the flash loader board file, prepared for little endian, for this device, with full path. Example:  
little=$TOOLKIT\_DIR$\config\flashloader\acme\acme123xxxLE.board

# C-SPY Macros

Read more about C-SPY macro functions in general, and about setup macro functions (a.k.a. hook macros) in particular, in the C-SPY Debugging Guide (EWARM\_DebuggingGuide.ENU.pdf).

C-SPY macros are provided in macro files in three different scopes:

* The device support, typically through the DeviceMacros parameter in arm\config\devices\SomeVendor\SomeDevice.i79
* The flash loader, typically in arm\config\flashloader\SomeVendor\SomeDevice.flash
* The application, typically in "Project->Options->Debugger->Setup macros", or in “View->Macros->Macro Registration”, or from a setup macro file through the \_\_registerMacroFile() macro.

The application and flash loader scopes don't interfere with each other. C-SPY first loads the flash loader macros, executes the flash loading, and then unloads the flash loader macros before the application macros are loaded. Also, the name spaces don't interfere, so even if they were loaded on top of each other, it would work.

However, the application and device support scopes interfere. These macro files are loaded together, and previously the only namespace available was execUser...(). So if for instance execUserReset() was defined both in the application macro file, and in the device support macro file, C-SPY would have picked one of them and skipped the other.

In current EWARM, each hook macro execUser...() is accompanied by another hook macro \_ExecDevice...(). The new name space, \_ExecDevice, is meant to be used in the device support scope, and the execUser name space shall as always be used in the application scope. C-SPY first calls \_ExecDevice...() and then execUser...(), for each defined hook macro, so the application macro may override anything done in the corresponding device support macro.

Probe and core connection hook macros, currently only available in the I-jet/JTAGjet C-SPY driver:  
The three (or actually six) new hook macros, execUserProbeConnect, execUserProbeReset, and execUserCoreConnect (\_ExecDeviceProbeConnect, \_ExecDeviceProbeReset and \_ExecDeviceCoreConnect), are more or less mostly relevant in the device support scope (we think). But if it is needed in the application scope (i.e. different needs in different applications), then the execUser name space is better to use.  
Read more in EWARM\_IjetConnectMacros.pdf.

…TBD A special note on trace initialization macros… execConfigureTraceETM() and execConfigureTraceSWO() … (not described anywhere else in user doc or infocenter…)

…Same TBD applies to getPersistentQuickLaunchMacros() as well

## Customized reset strategy as C-SPY macro

Here is a short description of how to implement customized reset strategies in C-SPY macros.

The description below only applies to C-SPY macro files which are provided with the device support through the .i79 parameter DeviceMacros. Flashloader macros specified in .flash files, and application macros specified in "Project->Options->Debugger->Setup->Setup macros" are NOT within the scope of this functionality.

Specify which C-SPY driver(s) a .dmac file is relevant for:

See for example: arm\config\debugger\TexasInstruments\AM1xxx.dmac

The first line in this .dmac file specifies that this complete .dmac file will only be passed to C-SPY when the I-jet driver is chosen:

//@DRIVER=ijet

If the .dmac file is relevant both for ijet and cmsisdap, then add ",cmsisdap" to the list:

//@DRIVER=ijet,cmsisdap

Notes:

- //@DRIVER=... must be present on the very first line of the .dmac file, otherwise it will be ignored.

- No white space characters or comments are allowed behind //@ (as the .dmac "parser" is very basic).

- The C-SPY driver identity name range is the same as for the C-SPY macro \_\_driverType(driver\_id), for example "ijet" and "cmsisdap".

- This feature may be used for any C-SPY driver.

A customized reset strategy in a C-SPY macro:

See for example: arm\config\debugger\TexasInstruments\AM1xxx.dmac

ResetAndStopAtUser()

//@RESET\_DRIVER=ijet

//@reset\_index=7

//@reset\_default=true

{

\_\_message("Reset and stop after bootloader");

\_\_probeCmd("emu jtagheader=reset:0");

\_\_probeCmd("emu jtagheader=trst:0");

\_\_probeCmd("emu jtagheader=trst:c");

\_\_probeCmd("emu jtagheader=reset:r");

\_\_probeCmd("reset default");

//Run through the bootloader and stop at beginning of user code

//Change this address if appropriate

\_\_probeCmd("go till 0x60000004");

}

The three lines between the macro name and the first "{" specifies that this is a customized reset strategy:

|  |  |
| --- | --- |
| //@RESET\_DRIVER=ijet | Specifies that this is a reset strategy for ijet only |
| //@reset\_index=7 | The index of the reset strategy, read more below |
| //@reset\_default=true | This reset strategy is tagged as default in the drop-down list in EW project options |

If this reset stratetgy is relevant for cmsis-dap as well, then the first line shall be

//@RESET\_DRIVER=ijet,cmsisdap

Pre-defined reset strategy index range, with strategy names "hard coded" in EWARM (the same index range is used in .ProbeScript files):

1 - "Software"

2 - "Hardware"

3 - "Core"

4 - "System"

5 - "Custom"

6 - "Reset by watchdog or reset register"

7 - "Reset and halt after bootloader"

8 - "Reset and halt before bootloader"

9 - "Connect during reset"

The first four reset strategies have built-in implementations in the C-SPY driver for ijet and cmsisdap, but it is possible to replace them with customized strategies in C-SPY macros.

The body of the macro in the example above contains the same type of low level I-jet macros that are found in .ProbeScript files, wrapped in the C-SPY macro \_\_probeCmd(...). But, as it is a regular C-SPY macro, you may of course utilize the normal C-SPY macro language to the full extent as well.

Notes:

- The name of a C-SPY macro that implements a reset strategy can be anything, it is not a fixed set of macro names. So the name "ResetAndStopAtUser()" could as well be "FooBarBig()" or anything, as long as it is unique within the scope of one C-SPY session.

- A C-SPY macro that implements a reset strategy does not take any parameters.

- No white space characters or comments are allowed behind "//@" (as the .dmac "parser" is very basic).

- The C-SPY driver identity name range is the same as for the C-SPY macro \_\_driverType(driver\_id), for example "ijet" and "cmsisdap".

- Customized reset strategies as C-SPY macros are only supported in the ijet and cmsisdap C-SPY drivers.

# The flash programming process in even more detail

The following process description is an elaboration of the corresponding description in FlashLoaderGuide.ENU.pdf. Also compare the section “Debugging code in flash” in the Debugging Guide.

1. The application to be downloaded to flash memory exists as an image file. A .board file is read by C-SPY and specifies one or more flash loading passes, one for each flash memory device on the board.
   1. Load the macro file(s) specified in the .i79 file, parameter(s) DeviceMacros. This(these) macro file(s) is(are) loaded throughout the complete C-SPY session.
2. For each pass, a specific address range (or subset) of the original image file is specified. The image file (ELF) is split accordingly into a separate image file ("simple" format) for each pass. If there is only one pass, the original image file is used as is.
3. Each pass specifies a flash device file (a .flash file) which, among other things, designates a specific flash loader.
4. C-SPY downloads the flash loader of the current pass into RAM.
   1. Load the macro file of the current pass in C-SPY, macro file specified in .flash file
   2. If this is the first pass:  
      Connect  
      In jet driver (see EWARM\_IjetConnectMacros.pdf):  
       - execUserProbeConnect()   
       - execUserCoreConnect()  
      Reset according to WTD (driver specific)  
      In jet driver (see EWARM\_IjetConnectMacros.pdf):  
       - execUserProbeReset()  
       - execUserCoreConnect()
   3. execUserFlashInit()
   4. Download the flash loader executable in RAM
   5. execUserFlashPreReset()
   6. "Target reset", soft reset (i.e. not according to setting in WTD)
   7. execUserFlashReset()
5. If the pass specifies an offset, all records from the image file are relocated accordingly.
6. C-SPY sets PC to FlashInit (or technically, to a label that will subsequently call FlashInit).
7. Parameters and data are written to the RAM buffer.
8. Execution is started, FlashInit is executed, and C-SPY regains control when execution hits a special breakpoint. FlashInit has the opportunity to override some information from the .flash file, such as the page size and block layout.
9. C-SPY partitions the data from the image file into suitable pieces with respect to the page and block layout of the flash memory, and to the size of the RAM buffer.
10. Before a certain block can be written to for the first time, the block must first be erased. In this case, the procedure continues with the next step. Otherwise, the procedure continues at step 13.
11. The RAM parameters are assigned the size of the block and its address.
12. C-SPY sets PC to FlashErase and starts the execution. When the function is done, the breakpoint is hit.
13. C-SPY writes some of the data to the RAM buffer.
14. C-SPY sets PC to FlashWrite and starts the execution. When the function is done, the breakpoint is hit.
15. If there is more data, the procedure returns to step 10.  
    If not:
    1. C-SPY sets PC to FlashChecksum and starts the execution. When the function is done, the breakpoint is hit.
    2. C-SPY sets PC to FlashSignoff and starts the execution. When the function is done, the breakpoint is hit.
    3. execUserFlashExit()
    4. Unload the C-SPY macro file of the current pass, see step 4a
16. If there are more passes, the procedure returns to step 3.
17. The debug information that corresponds to the final application is read.  
    But first:
    1. If "download only"  
       then disconnect and skip 17b-20  
       else:
    2. Load the macro file(s) specified in the project, in  
       "Project->Options->Debugger->Setup->Setup macros"  
       Note, macro file(s) loaded in step 1a is(are) still loaded.
    3. Reset according to setting in WTD
    4. \_ExecDevicePreload() / execUserPreload()
    5. Now, at last: The debug information that corresponds to the final application is read.
    6. Verify download (if checked in "Project->Options->Debugger->Download")
    7. \_ExecDevicePreReset() / execUserPreReset()
    8. "Target reset", soft reset (i.e. not according to setting in WTD).
    9. \_ExecDeviceReset() / execUserReset()
    10. \_ExecDeviceSetup() / execUserSetup()
18. C-SPY sets PC to the start address of the final application.
19. Optionally, execution starts at main:  
    Set breakpoint at main and start execution  
    \_ExecDeviceExecutionStarted() / execUserExecutionStarted()  
    \_ExecDeviceExecutionStopped() / execUserExecutionStopped()
20. Run application and debug.  
    For every go->break:  
    \_ExecDeviceExecutionStarted() / execUserExecutionStarted()  
    \_ExecDeviceExecutionStopped() / execUserExecutionStopped()
21. Terminate C-SPY  
    \_ExecDeviceExit() / execUserExit()

# The Linker configuration file editor

“Project->Options->Linker->Config->Linker configuration file->Edit” 🡺 “Linker configuration file editor”

The “Linker configuration file editor” field definitions are found in xml file specified in the project .icf file:

/\* IcfEditorFile="$TOOLKIT\_DIR$\config\ide\IcfEditor\a\_v1\_1.xml" \*/

Each xml file has actually two representations in the $TOOLKIT\_DIR$\config\ide\IcfEditor directory, one per selectable language, “ENU” or “JPN”. The “Linker configuration file editor” automatically picks one of the two xml files according to the chosen language.

The field symbols in the xml file must correspond to symbols in the project .icf file, placed between ”/\*###ICF###” and ”###ICF###\*/”.

Example of a “Linker configuration file editor” field definitions file, a\_v1\_1.ENU.xml:

<?xml version="1.0" encoding="UTF-8"?>

<icf\_annotations version="1.0">

<tab name="Vector Table">

<item name=".intvec start" type="Number">

<symbol>\_\_ICFEDIT\_intvec\_start\_\_</symbol>

<tooltip>The vector table start address</tooltip>

</item>

</tab>

<tab name="Memory Regions">

<itemHeaders>

<header>Start:</header>

<header>End:</header>

</itemHeaders>

<item name="IROM1" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_IROM1\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_IROM1\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the internal IROM1 region</tooltip\_1>

<tooltip\_2>The end address of the internal IROM1 region</tooltip\_2>

</item>

<item name="IROM2" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_IROM2\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_IROM2\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the internal IROM2 region</tooltip\_1>

<tooltip\_2>The end address of the internal IROM2 region</tooltip\_2>

</item>

<item name="EROM1" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_EROM1\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_EROM1\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the external EROM1 region</tooltip\_1>

<tooltip\_2>The end address of the external EROM1 region</tooltip\_2>

</item>

<item name="EROM2" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_EROM2\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_EROM2\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the external EROM2 region</tooltip\_1>

<tooltip\_2>The end address of the external EROM2 region</tooltip\_2>

</item>

<item name="EROM3" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_EROM3\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_EROM3\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the external EROM3 region</tooltip\_1>

<tooltip\_2>The end address of the external EROM3 region</tooltip\_2>

</item>

<item name="IRAM1" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_IRAM1\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_IRAM1\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the internal IRAM1 region</tooltip\_1>

<tooltip\_2>The end address of the internal IRAM1 region</tooltip\_2>

</item>

<item name="IRAM2" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_IRAM2\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_IRAM2\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the internal IRAM2 region</tooltip\_1>

<tooltip\_2>The end address of the internal IRAM2 region</tooltip\_2>

</item>

<item name="ERAM1" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_ERAM1\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_ERAM1\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the external ERAM1 region</tooltip\_1>

<tooltip\_2>The end address of the external ERAM1 region</tooltip\_2>

</item>

<item name="ERAM2" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_ERAM2\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_ERAM2\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the external ERAM2 region</tooltip\_1>

<tooltip\_2>The end address of the external ERAM2 region</tooltip\_2>

</item>

<item name="ERAM3" type="Range">

<symbol\_1>\_\_ICFEDIT\_region\_ERAM3\_start\_\_</symbol\_1>

<symbol\_2>\_\_ICFEDIT\_region\_ERAM3\_end\_\_</symbol\_2>

<tooltip\_1>The start address of the external ERAM3 region</tooltip\_1>

<tooltip\_2>The end address of the external ERAM3 region</tooltip\_2>

</item>

</tab>

<tab name="Stack/Heap Sizes">

<item name="CSTACK" type="Number">

<symbol>\_\_ICFEDIT\_size\_cstack\_\_</symbol>

<tooltip>The size of the application stack</tooltip>

</item>

<item name="SVC\_STACK" type="Number">

<symbol>\_\_ICFEDIT\_size\_svcstack\_\_</symbol>

<tooltip>The size of the supervisor stack</tooltip>

</item>

<item name="IRQ\_STACK" type="Number">

<symbol>\_\_ICFEDIT\_size\_irqstack\_\_</symbol>

<tooltip>The size of the stack for the interrupt handler</tooltip>

</item>

<item name="FIQ\_STACK" type="Number">

<symbol>\_\_ICFEDIT\_size\_fiqstack\_\_</symbol>

<tooltip>The size of the stack for the fast interrupt handler</tooltip>

</item>

<item name="UND\_STACK" type="Number">

<symbol>\_\_ICFEDIT\_size\_undstack\_\_</symbol>

<tooltip>The size of the stack for the undefined instruction handler</tooltip>

</item>

<item name="ABT\_STACK" type="Number">

<symbol>\_\_ICFEDIT\_size\_abtstack\_\_</symbol>

<tooltip>The size of the stack for the data abort handler</tooltip>

</item>

<item name="HEAP" type="Number">

<symbol>\_\_ICFEDIT\_size\_heap\_\_</symbol>

<tooltip>The size of the heap</tooltip>

</item>

</tab>

</icf\_annotations>

Example of an .icf file using a\_v1\_1.xml:

/\*###ICF### Section handled by ICF editor, don't touch! \*\*\*\*/

/\*-Editor annotation file-\*/

/\* IcfEditorFile="$TOOLKIT\_DIR$\config\ide\IcfEditor\a\_v1\_1.xml" \*/

/\*-Specials-\*/

define symbol \_\_ICFEDIT\_intvec\_start\_\_ = 0x00800000;

/\*-Memory Regions-\*/

define symbol \_\_ICFEDIT\_region\_IROM1\_start\_\_ = 0x00800000;

define symbol \_\_ICFEDIT\_region\_IROM1\_end\_\_ = 0x0087FFFF;

define symbol \_\_ICFEDIT\_region\_IROM2\_start\_\_ = 0x00FE0000;

define symbol \_\_ICFEDIT\_region\_IROM2\_end\_\_ = 0x00FFFFFF;

define symbol \_\_ICFEDIT\_region\_EROM1\_start\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_EROM1\_end\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_EROM2\_start\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_EROM2\_end\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_EROM3\_start\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_EROM3\_end\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_IRAM1\_start\_\_ = 0x00000000;

define symbol \_\_ICFEDIT\_region\_IRAM1\_end\_\_ = 0x0000FFFF;

define symbol \_\_ICFEDIT\_region\_IRAM2\_start\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_IRAM2\_end\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_ERAM1\_start\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_ERAM1\_end\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_ERAM2\_start\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_ERAM2\_end\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_ERAM3\_start\_\_ = 0x0;

define symbol \_\_ICFEDIT\_region\_ERAM3\_end\_\_ = 0x0;

/\*-Sizes-\*/

define symbol \_\_ICFEDIT\_size\_cstack\_\_ = 0xA00;

define symbol \_\_ICFEDIT\_size\_svcstack\_\_ = 0x80;

define symbol \_\_ICFEDIT\_size\_irqstack\_\_ = 0x80;

define symbol \_\_ICFEDIT\_size\_fiqstack\_\_ = 0x80;

define symbol \_\_ICFEDIT\_size\_undstack\_\_ = 0x80;

define symbol \_\_ICFEDIT\_size\_abtstack\_\_ = 0x80;

define symbol \_\_ICFEDIT\_size\_heap\_\_ = 0x200;

/\*\*\*\* End of ICF editor section. ###ICF###\*/

define symbol WORK\_FLASH\_start = 0x0E000000;

define symbol WORK\_FLASH\_end = 0x0E01FFFF;

define symbol AXI\_IROM1\_start = 0x01000000;

define symbol AXI\_IROM2\_start = 0x01FE0000;

define symbol \_\_SDR\_offset = 0x00000000;

define symbol \_\_BDR\_offset = 0x00000040;

define symbol \_\_WDR\_offset = 0x00000060;

define symbol \_\_FLASH\_MARKERS\_start = 0x00FF0000;

define memory mem with size = 4G;

define region IROM\_region = mem:[from \_\_ICFEDIT\_region\_IROM1\_start\_\_ to \_\_ICFEDIT\_region\_IROM1\_end\_\_]

| mem:[from \_\_ICFEDIT\_region\_IROM2\_start\_\_ to \_\_ICFEDIT\_region\_IROM2\_end\_\_];

define region EROM\_region = mem:[from \_\_ICFEDIT\_region\_EROM1\_start\_\_ to \_\_ICFEDIT\_region\_EROM1\_end\_\_]

| mem:[from \_\_ICFEDIT\_region\_EROM2\_start\_\_ to \_\_ICFEDIT\_region\_EROM2\_end\_\_]

| mem:[from \_\_ICFEDIT\_region\_EROM3\_start\_\_ to \_\_ICFEDIT\_region\_EROM3\_end\_\_];

define region IRAM\_region = mem:[from \_\_ICFEDIT\_region\_IRAM1\_start\_\_ to \_\_ICFEDIT\_region\_IRAM1\_end\_\_]

| mem:[from \_\_ICFEDIT\_region\_IRAM2\_start\_\_ to \_\_ICFEDIT\_region\_IRAM2\_end\_\_];

define region ERAM\_region = mem:[from \_\_ICFEDIT\_region\_ERAM1\_start\_\_ to \_\_ICFEDIT\_region\_ERAM1\_end\_\_]

| mem:[from \_\_ICFEDIT\_region\_ERAM2\_start\_\_ to \_\_ICFEDIT\_region\_ERAM2\_end\_\_]

| mem:[from \_\_ICFEDIT\_region\_ERAM3\_start\_\_ to \_\_ICFEDIT\_region\_ERAM3\_end\_\_];

define region WORK\_FLASH\_region = mem:[from WORK\_FLASH\_start to WORK\_FLASH\_end];

define block CSTACK with alignment = 8, size = \_\_ICFEDIT\_size\_cstack\_\_ { };

define block SVC\_STACK with alignment = 8, size = \_\_ICFEDIT\_size\_svcstack\_\_ { };

define block IRQ\_STACK with alignment = 8, size = \_\_ICFEDIT\_size\_irqstack\_\_ { };

define block FIQ\_STACK with alignment = 8, size = \_\_ICFEDIT\_size\_fiqstack\_\_ { };

define block UND\_STACK with alignment = 8, size = \_\_ICFEDIT\_size\_undstack\_\_ { };

define block ABT\_STACK with alignment = 8, size = \_\_ICFEDIT\_size\_abtstack\_\_ { };

define block HEAP with alignment = 8, size = \_\_ICFEDIT\_size\_heap\_\_ { };

initialize by copy { readwrite };

if (isdefinedsymbol(\_\_USE\_DLIB\_PERTHREAD))

{

// Required in a multi-threaded application

initialize by copy with packing = none { section \_\_DLIB\_PERTHREAD };

}

do not initialize { section .noinit };

place at address mem:\_\_ICFEDIT\_intvec\_start\_\_ { readonly section .intvec,

readonly section .BOOTSTART };

place at address mem:\_\_FLASH\_MARKERS\_start+\_\_SDR\_offset { section .SDR };

place at address mem:\_\_FLASH\_MARKERS\_start+\_\_BDR\_offset { section .BDR };

place at address mem:\_\_FLASH\_MARKERS\_start+\_\_WDR\_offset { section .WDR };

place in IROM\_region { readonly };

place in EROM\_region { readonly section application\_specific\_ro };

place in IRAM\_region { readwrite,

block CSTACK, block SVC\_STACK, block IRQ\_STACK, block FIQ\_STACK,

block UND\_STACK, block ABT\_STACK, block HEAP };

place in ERAM\_region { readwrite section application\_specific\_rw };

define exported symbol AXI\_IROM1\_BASE = AXI\_IROM1\_start;

define exported symbol AXI\_IROM2\_BASE = AXI\_IROM2\_start;

With the .icf file above, the “Linker configuration file editor” will look like:

  