

QCVS Hardware Device Tree Editor User Guide



Contents

- Chapter 1 Hardware Device Tree Editor..... 3**
- 1.1 Introduction..... 3
- 1.2 Using Hardware Device Tree Editor..... 4
 - 1.2.1 Create a new project..... 4
 - 1.2.2 Import device tree files..... 9
 - 1.2.2.1 Standard DTS files..... 9
 - 1.2.2.2 C-preprocessing support..... 9
 - 1.2.3 Modify hardware device trees..... 12
 - 1.2.3.1 GUI editor..... 12
 - 1.2.3.2 Text editor..... 20
 - 1.2.4 Perform validation..... 23
 - 1.2.4.1 Syntax validation..... 23
 - 1.2.4.2 Property constraints..... 24
 - 1.2.5 Search in hardware device trees..... 26
 - 1.2.6 Generate device tree blob..... 28
- 1.3 Known issues and limitations..... 29

Chapter 1

Hardware Device Tree Editor

This document introduces the Hardware Device Tree component of QorIQ Configuration and Validation Suite (QCVS). The document describes how to create a new QorIQ configuration project and use the Hardware Device Tree component.

NOTE

The Hardware Device Tree component of QCVS is only available for projects created with B, P, or T family of processors.

This chapter is divided into the following sections:

- [Introduction](#) on page 3
- [Using Hardware Device Tree Editor](#) on page 4
- [Known issues and limitations](#) on page 29

1.1 Introduction

The hardware device tree editor is a GUI editor that allows you to view and manage the structure of a hardware device tree.

A hardware device tree is a tree data structure with nodes and properties that describes the physical devices in a system, such as Direct Memory Access, Universal Serial Bus Interface, Frame Manager, and Security Monitor.

The ePAPR standard describes the logical structure of the hardware device tree and specifies a base set of required nodes and properties. This set is minimal, but complete enough to boot a simple operating system.

The following are some basic terms related to the hardware device trees:

- Device Tree Syntax (DTS) is the textual representation of a hardware device tree, which is input to a DTC
- Device Tree Compiler (DTC) is an open source tool used to create DTB files from DTS files
- Device Tree Blob (DTB) is a compact binary representation of the hardware device tree that is input to U-Boot and operating system

You can find the details of hardware device trees in:

- Power.org™ Standard for Embedded Power Architecture™ Platform Requirements (ePAPR), version 1.0
- Booting the Linux/ppc kernel without Open Firmware
 - 2005 Benjamin Herrenschmidt <benh at kernel.crashing.org>, IBM Corp.
 - 2005 Becky Bruce <becky.bruce at freescale.com>, Freescale Semiconductor, FSL SOC and 32-bit additions.
 - 2006 MontaVista Software, Inc., Flash chip node definition.

Hardware device trees represent one of the most difficult configuration elements on QorIQ family because of the complexity of the processors and its format. Presently, the modification of hardware device trees is done through text editors. A dedicated device tree tool in the form of a GUI editor is required that represents its tree like structure graphically and facilitates its handling.

1.2 Using Hardware Device Tree Editor

The Hardware Device Tree Editor provides you a graphical interface to edit the standard ePAPR device trees and helps you to configure the processor during the bootstrap processes.

The Hardware Device Tree Editor is used to:

- Define the devices that need initialization
- Support the U-Boot's plug and play functionality
- Discover additional capabilities through PCI bus
- Configure cache memory, CPU cores and specific MMU configuration of the processor
- Check for syntactic and semantic errors and data validation

The *dts* format is complex for inexperienced users to read and modify as it is a textual representation of the hardware device tree where each node has a different set of properties and values describing the characteristics of the device. The Hardware Device Tree Editor offers support for both interpreting and writing hardware device trees.

This section contains the following subsections:

- [Create a new project](#) on page 4
- [Import device tree files](#) on page 9
- [Modify hardware device trees](#) on page 12
- [Perform validation](#) on page 23
- [Search in hardware device trees](#) on page 26
- [Generate device tree blob](#) on page 28

1.2.1 Create a new project

The hardware device tree project is created using the New QorIQ Configuration Project wizard.

You can import an existing hardware device tree file or generate a default one.

To create a new QorIQ configuration project for configuring hardware device tree, follow these steps:

1. Choose **File > New > QorIQ Configuration Project**. Follow the steps in the **New QorIQ Configuration Project** wizard.
2. Enter project name and click **Next**.
3. Select the required target SoC and click **Next**.
4. Select **Device Tree Editor** in the **Toolset selection** page.

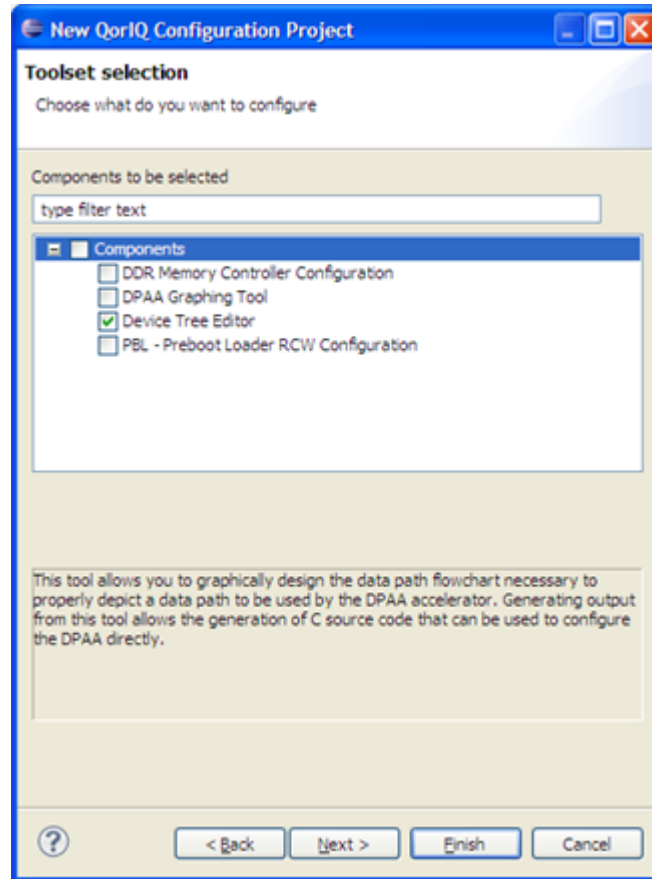


Figure 1. Select Device Tree Editor

5. Click **Next**. The **Device Tree Configuration** screen appears where you can select from three actions:
 - Import configuration from an existing device tree file: Allows you to import device tree configuration from an existing device tree file (.dts). The file that you import is validated before you proceed to next step. The file that you import must have .dts extension and must be compatible with the selected SoC. A compatible file is a .dts file which specifies the "model" property containing the selected SoC. The figure below shows an example where the imported device tree file does not match with the chosen SoC.

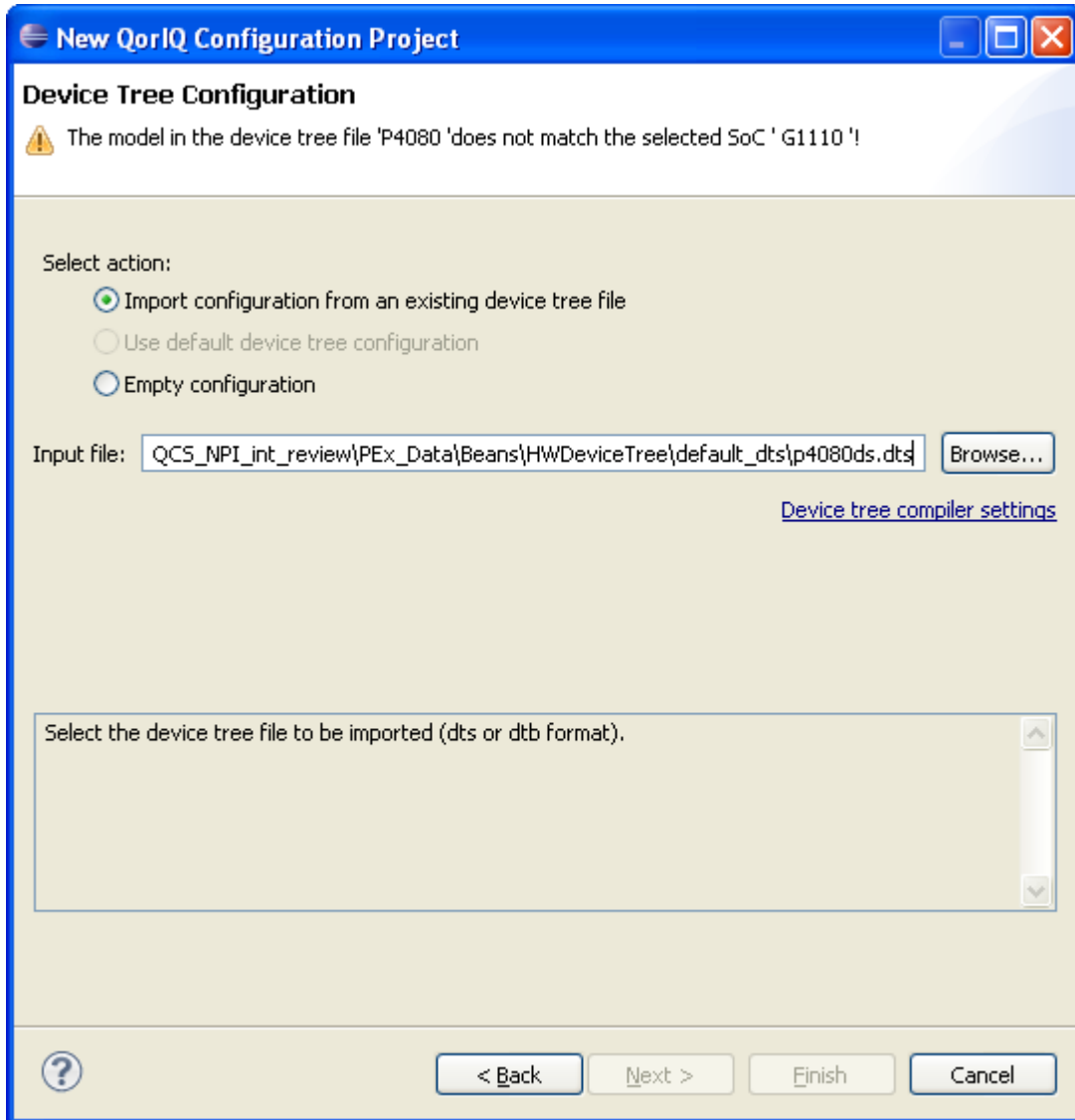


Figure 2. Import configuration from device tree file

- Use default device tree configuration (default option): The default hardware device trees are *.dts* files from existing BSPs, as shown in the figure below

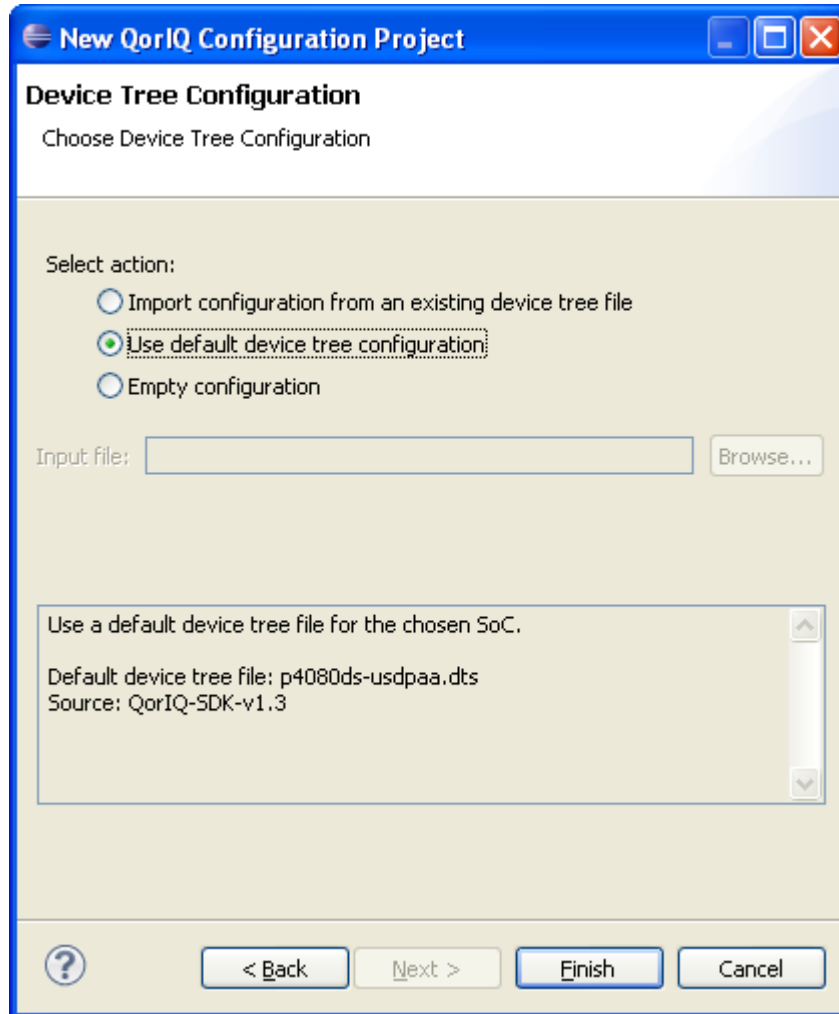


Figure 3. Default device tree file configuration

- Empty configuration: Allows you to create your own configuration of a hardware device tree with an empty tree data structure with no nodes
6. If no default hardware device tree exists for the selected SoC, the associated option, **Use default device tree configuration**, becomes disabled.
 7. Click **Finish**.

The project is created, and the hardware device tree (**HWDeviceTree**) component is added to the project under the **Components** folder, as shown in the figure below. In addition, the data from the device tree file is loaded into the component.

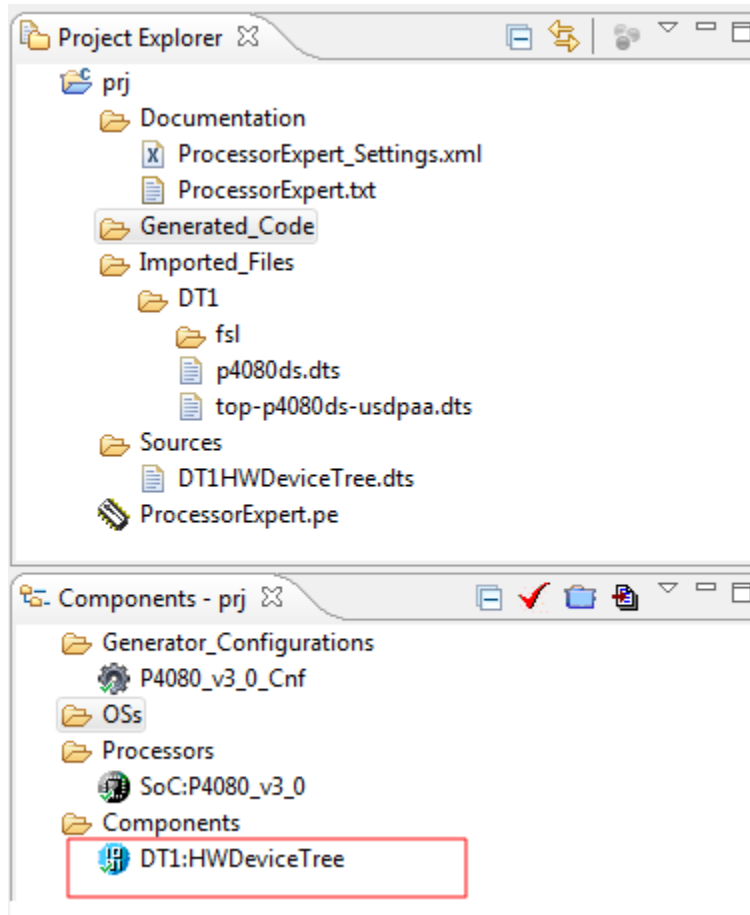


Figure 4. Hardware Device Tree project

This operation may take time depending on the size of the file. A dialog appears as shown below on the screen indicating the progress of the procedure.

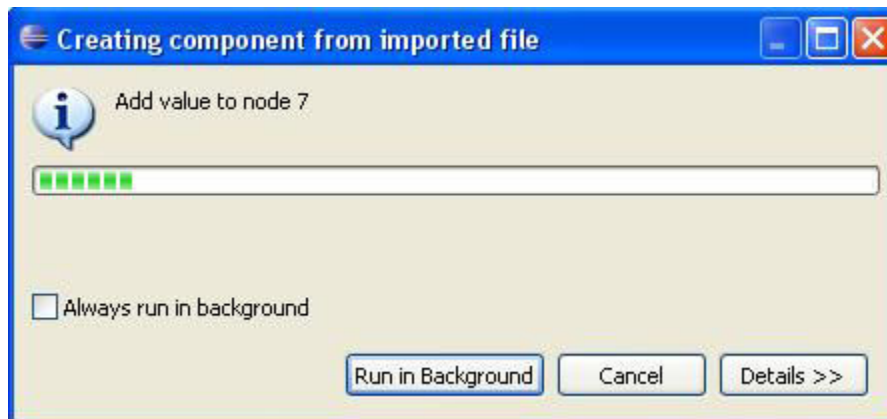


Figure 5. Hardware Device Tree progress monitor

The original file used to create the HWDeviceTree component is added to the **Imported_Files** folder. A new device tree file is generated under the **Sources** folder based on the imported file. At this moment, the two files are identical. You will work with the generated one and the former will remain untouched. There can be more than one **HWDeviceTree** component per project.

Follow these tips while working on the hardware device tree project:

- Enable **Project > Build Automatically** setting to always have everything in sync.
- Save your session before leaving the application in case you still need it. When you close the **Processor Expert**, a pop-up dialog appears allowing you to save the changes.
- When **HWDeviceTree** component is removed from the project, the generated device tree file is also removed.

1.2.2 Import device tree files

The Hardware Device Tree Editor allows you to import an existing device tree file.

To import a device tree configuration from an existing device tree file, click the **Import** button (as shown below) and select a `.dts` file compatible with the existing SoC. A compatible file is a `.dts` file that specifies the "model" property containing the selected SoC. A minimal validation is performed for the imported files.

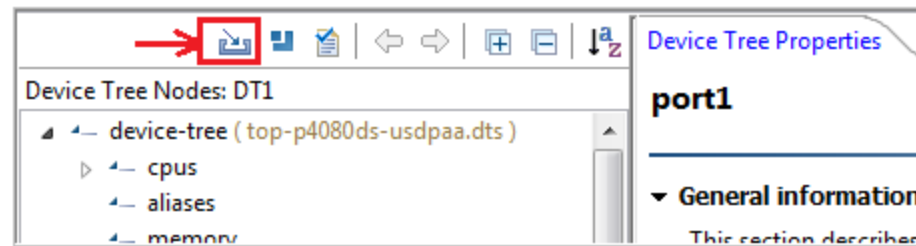


Figure 6. Import new device tree

This section contains the following subsections:

- [Standard DTS files](#) on page 9
- [C-preprocessing support](#) on page 9

1.2.2.1 Standard DTS files

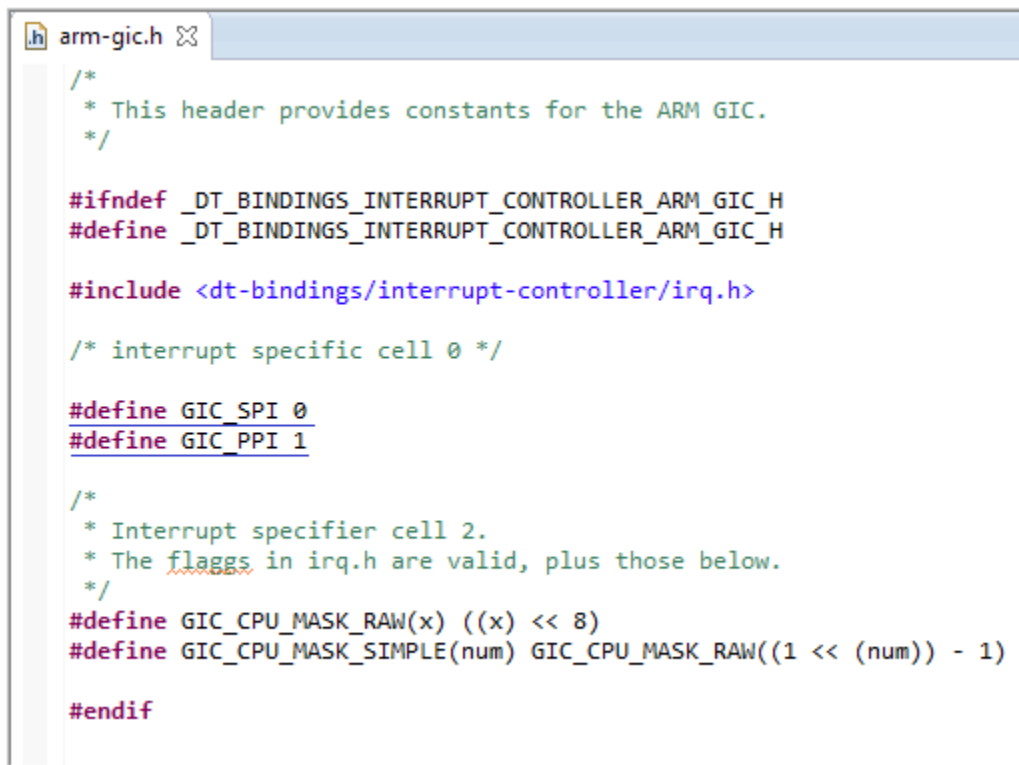
A standard DTS file is a file that contains the textual representation of a device tree, which is input to a DTC.


The standard DTS files can also be used as an input to DTC, without any processing by the Hardware Device Tree tool.

1.2.2.2 C-preprocessing support

Some of the DTS files may contain C-preprocessor syntax that is not supported by the DTC tool.

This syntax is supported in the preprocessing phase by the Hardware Device Tree tool. The `#define` directives or other C-preprocessor syntax should be defined in the included C-header files. During import, the identifiers are replaced with some defined values.



```
.h arm-gic.h 
/*
 * This header provides constants for the ARM GIC.
 */

#ifndef _DT_BINDINGS_INTERRUPT_CONTROLLER_ARM_GIC_H
#define _DT_BINDINGS_INTERRUPT_CONTROLLER_ARM_GIC_H

#include <dt-bindings/interrupt-controller/irq.h>

/* interrupt specific cell 0 */

#define GIC_SPI 0
#define GIC_PPI 1

/*
 * Interrupt specifier cell 2.
 * The flags in irq.h are valid, plus those below.
 */
#define GIC_CPU_MASK_RAW(x) ((x) << 8)
#define GIC_CPU_MASK_SIMPLE(num) GIC_CPU_MASK_RAW((1 << (num)) - 1)

#endif
```

Figure 7. C-header file that defines directives

```
ls1021a.dtsi
#include "skeleton.dtsi"
#include <dt-bindings/interrupt-controller/arm-gic.h>
#include <dt-bindings/dma/ls1021a-edma.h>

/ {
    compatible = "fsl,ls1021a";
    interrupt-parent = <&gic>;

    aliases {
    };

    cpus {
    };

    timer {
        compatible = "arm,armv7-timer";
        interrupts = <GIC PPI 13 0xf08>,
                    <GIC PPI 14 0xf08>,
                    <GIC PPI 11 0xf08>,
                    <GIC PPI 10 0xf08>;
    };

    pmu {
        compatible = "arm,cortex-a7-pmu";
        interrupts = <0 138 0x04>, <0 139 0x04>;
    };

    soc {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "simple-bus";
        interrupt-parent = <&gic>;
    };
}
```

Figure 8. Original DTS files that contain C-preprocessor syntax



```
DT1HWDeviceTree.dts
/dts-v1/;

/ {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "fsl,ls1021a";
    interrupt-parent = <&gic>;
    model = "LS1021A QDS Board";

    chosen {
        bootargs = "earlyprintk initrd=initramfs.cpio root=/dev/ram0 rw console=ttyS0,115200";
    };

    aliases {
    };

    memory {
    };

    cpus {
    };

    timer {
        compatible = "arm,armv7-timer";
        interrupts = <1 13 0xf08>;
        interrupt-parent = <&gic>;
    };

    pmu {
        compatible = "arm,cortex-a7-pmu";
    };
};
```

Figure 9. Generated DTS file input to DTC

1.2.3 Modify hardware device trees

You can work on the hardware device tree component and modify it along with its properties using the GUI editor as well as the text editor.

This section contains the following subsections:

- [GUI editor](#) on page 12
- [Text editor](#) on page 20

1.2.3.1 GUI editor

The GUI editor represents the hardware device tree component graphically.

When the **HWDeviceTree** component is selected in the **Project Explorer**, the **Component Inspector** opens displaying the GUI editor and the hardware device tree properties.

This view is split into two parts:

- A tree structure that handles the nodes
- The **Device Tree Properties** view that handles the properties

The figure below shows the Device Tree GUI editor.

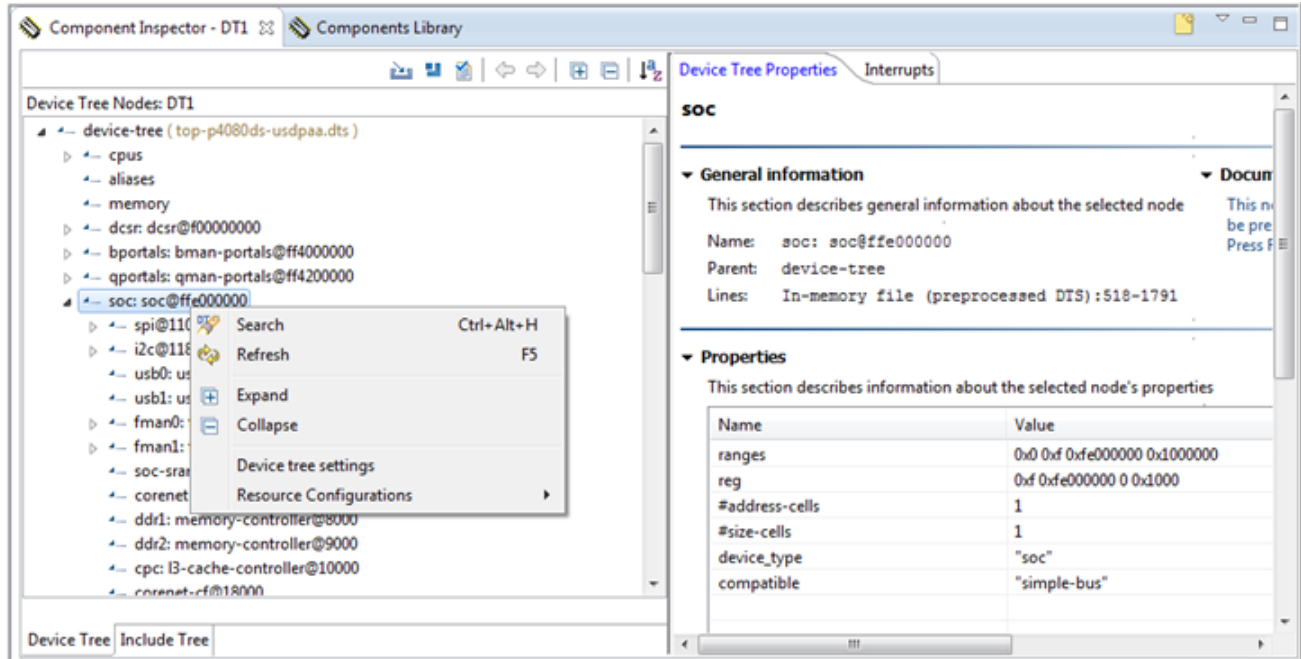


Figure 10. Device Tree GUI editor - DeviceTree and Device Tree Properties views

The hardware device tree nodes are represented in a tree structure. Each node selection makes the corresponding properties display in the **Device Tree Properties** view. The **Device Tree Properties** view displays a list of properties along with the short details of the node. You can activate the dynamic context help by pressing F1 key after selecting the required node for which you want to view the dynamic context help.

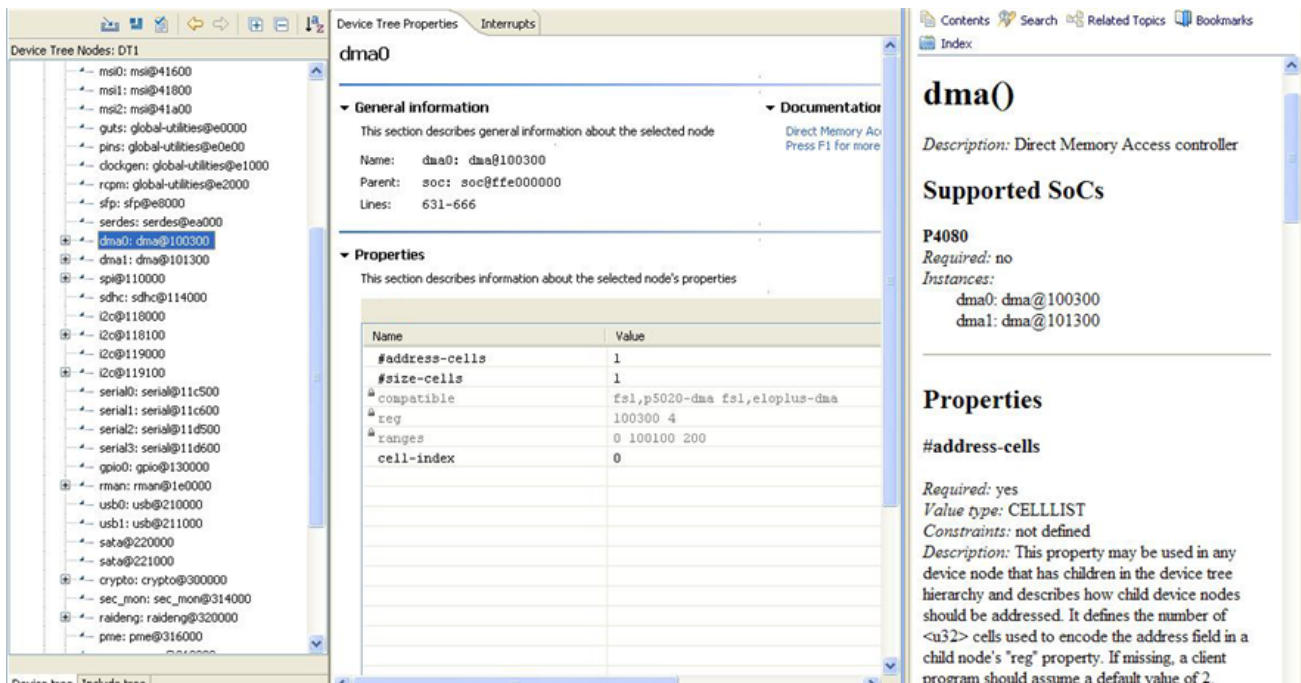


Figure 11. Device Tree Properties Custom View

The **Properties** table contains two columns: **Name** and **Value**. All values in the **Value** column are discarded of any type-specific symbols. For example, <0x1 0x2> cell list appears as 0x1 0x2, that is without the angular brackets. A string list

appears without the quotation marks. This is applicable in the graphical editor only, the text editor displays all values in the complete form. The supported property types are shown in the table below.

Table 1. Supported property types

Type	Example
BYTELIST	[AB CD 01]
CELLLIST	<0x1 0x2 0x800 &mpic>
EMPTY	empty value
PHANDLE	&mpic
STRINGLIST	"fs1,p5020", "fs1,p4080"
U32	<0x1>
STRING	memory
EMPTY	no value

You can perform the following actions in the GUI editor using the **View** toolbar.



Figure 12. View toolbar

- Import a new device tree file
- Include a device tree file
- Perform validation against the hardware device tree
- Tree navigation using **Go back** and **Go forward** icons from the **View** toolbar
- Expand/collapse nodes from the toolbar or using the context menu
- Sort the nodes in ascending/descending order
- Context help support for hardware device tree nodes:

Select the required node in the tree structure and open the Eclipse context help by pressing *F1* on Windows and *Ctrl+F1* on Linux host. The nodes that have no documentation available are marked in the **Device Tree Properties** view.

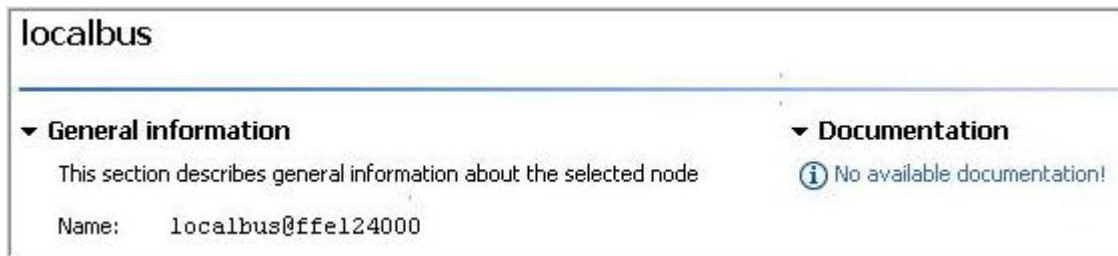


Figure 13. Node with no documentation available

You can edit existing documentation or add missing documentation by performing the following steps:

1. Edit existing file or create a new html file at the location `<layout>eclipse/plugins/com.freescale.processorexpert.doc.qoriq.dt/html` keeping the same format; the name of the file must be identical with the name of the node, that is `<node-name>.html`.

2. If you are creating a new html file, add a new entry in `<layout>eclipse/plugins/com.freescale.processorexpert.doc.qoriq.dt/contexts.xml` for the new file.
3. Close the Eclipse IDE.
4. Open Command Prompt and launch Eclipse using the following command:

```
>eclipse.exe --clean
```

Following are major components of Device Tree GUI editor:

- [Include tree](#) on page 15
- [Interrupts view](#) on page 16
- [Memory Map view](#) on page 17

1.2.3.1.1 Include tree

The **Include tree** tab in the **Component Inspector** represents the hierarchy of the included hardware device tree files.

The **Include tree** tab allows easy navigation among all hardware device tree fragments: dts or dtsti.

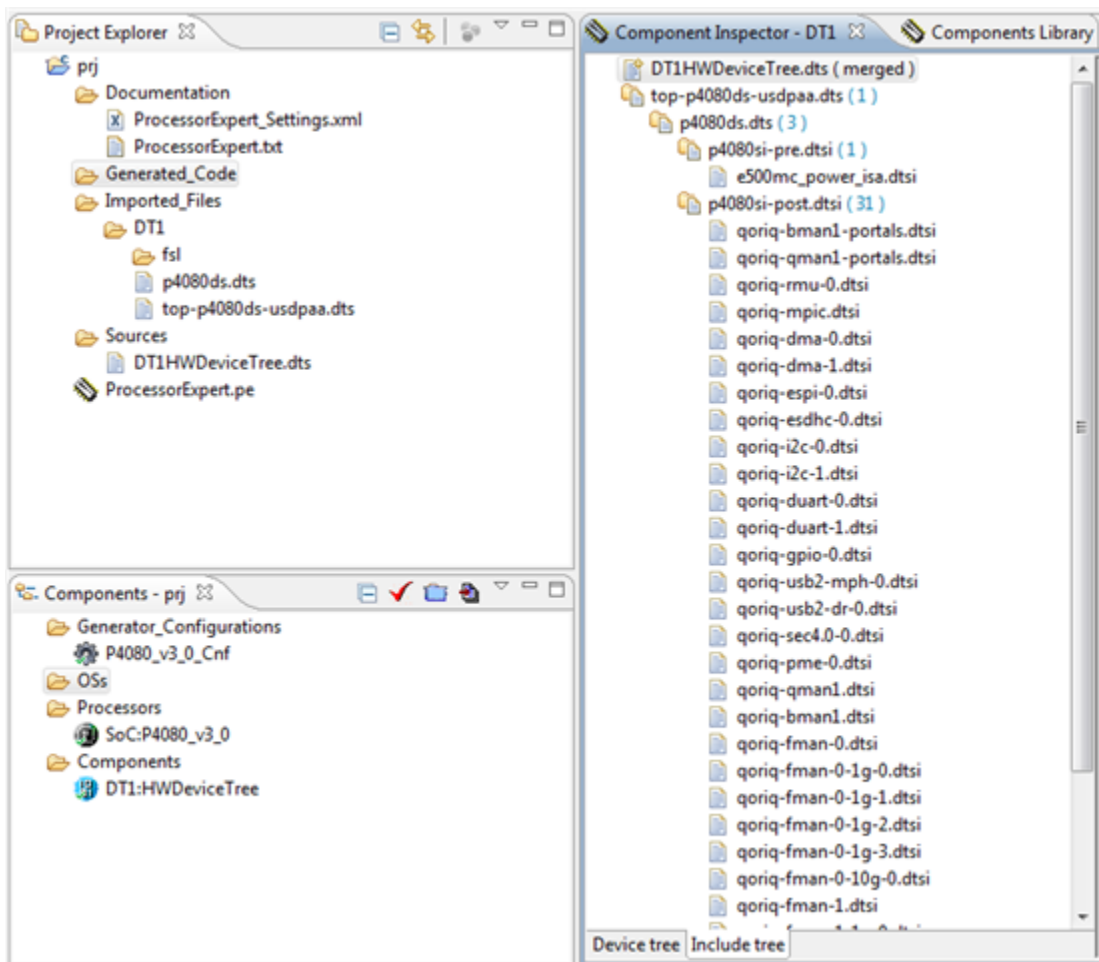


Figure 14. Include tree tab in Component Inspector

The **Include tree** tab displays three types of files:

- Merged hardware device tree file, which represents the all-in-one file merging all nodes and properties from multiple hardware device trees in a single file
- Top-level hardware device tree file, which is the first dts including other hardware device tree(s)

- Included files representing fragments with /include/ statement, usually dtsi files

1.2.3.1.2 Interrupts view

The **Interrupts** view of the hardware device tree contains a logical interrupt tree that represents the hierarchy and routing of the interrupts in the platform hardware.

You can view this interrupt tree in the **Component Inspector** when selecting an existing hardware device tree component.

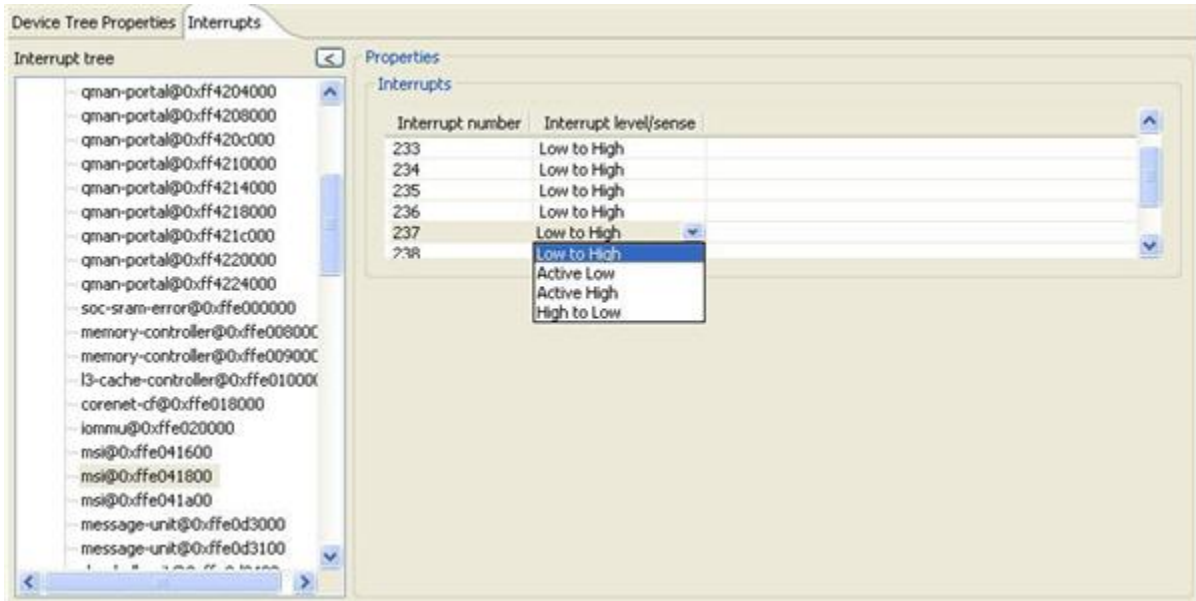


Figure 15. Interrupts view in Component Inspector

The left side of the **Interrupts** view displays the actual representation of the **Interrupt tree**, starting from the root interrupt controller. When you select an element in the **Interrupt tree**, a table appears on the right side listing the interrupt sources for the selected hardware device tree node. Each row in the table provides a user-friendly view of a decoded interrupt specifier, and each column in the table represents a specific cell of the interrupt specifier.

You can edit the **Interrupts** table for each node selected from the hardware device tree. An interrupt tree may contain special nodes called interrupt nexuses that map from one interrupt domain to another interrupt domain. When you select an interrupt nexus node in the **Interrupt tree**, the **Domain map** table appears below the **Interrupts** table on the right side. The **Domain Map** table allows you to perform interrupt domain mapping for the selected interrupt nexus.

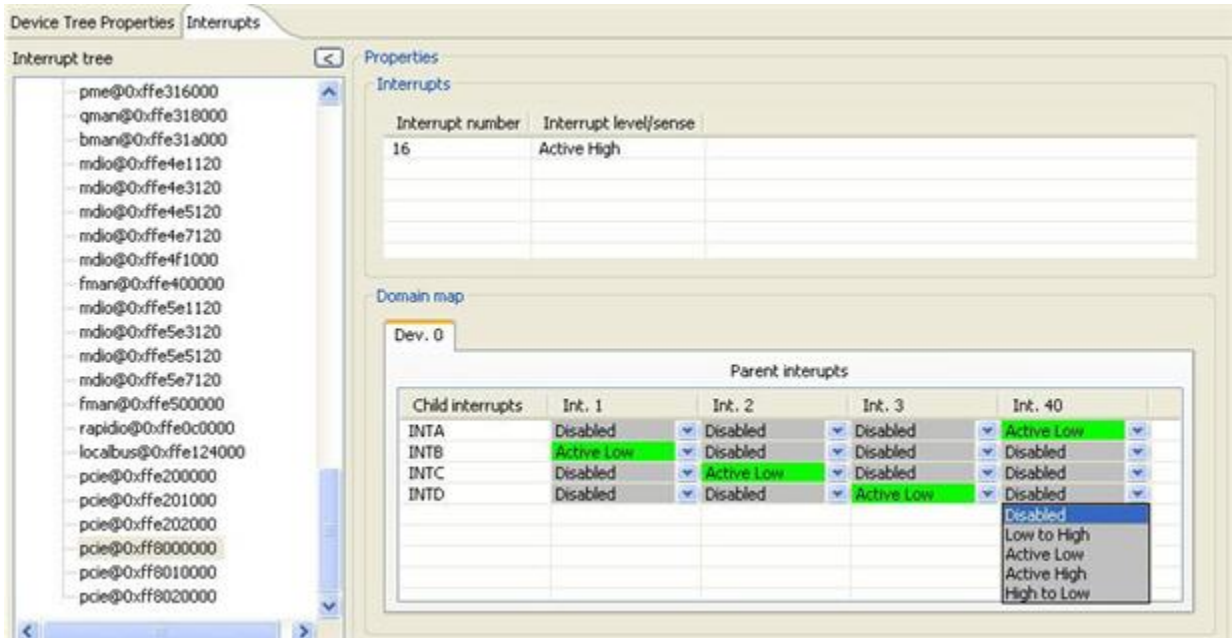


Figure 16. Interrupts tree - Nexus nodes and domain map

1.2.3.1.3 Memory Map view

The Memory Map view displays the decoded memory blocks associated with a hardware device component.

You can view a hardware device tree as a representation of different Local Access Windows (LAWs) within a device. For example, according to the P4080 Reference Manual, you can define the p4080 address map using a set of maximum 32 LAWs. An example of such window is the Configuration Control and Status Register (CCSR) area defined by soc node in the hardware device tree. All LAWs can be relocated within the entire address space of the platform. Each of these LAWs maps a programmable 4 KB to 64 GB region of the local 36-bit address space to a specified target interface, such as DDR Controller, Localbus, and PCI Express Controller.

```
/{
  model = "fsl,P4080DS";
  compatible = "fsl,P4080DS";
  #address-cells = <2>;
  #size-cells = <2>;
  ...
  pci2: pci@ffe202000 {
    compatible = "fsl,p4080-pcie";
    device_type = "pci";
    #interrupt-cells = <1>;
    #size-cells = <2>;
    #address-cells = <3>;
    reg = <0xf 0xfe202000 0 0x1000>;
    bus-range = <0x0 0xff>;
    ranges = <0x02000000 0 0xe0000000 0xc
0x40000000 0 0x200000000x01000000 0 0x00000000
0xf 0xf8020000 0 0x00010000>;
    ...
  }
  ...
};

// e500mc core has 36-bit physical addressing
```

Figure 17. LAW defined inside Hardware Device Tree

Each hardware device tree node having `reg` and `ranges` properties defines memory ranges inside or outside CCSR window. The following figures show an example of such node.

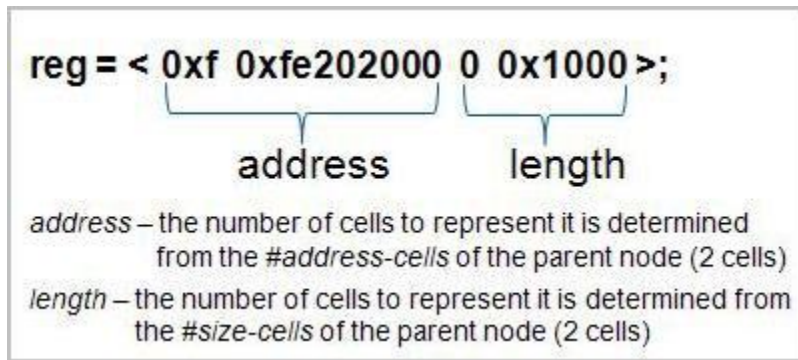


Figure 18. `reg` property definition inside Hardware Device Tree

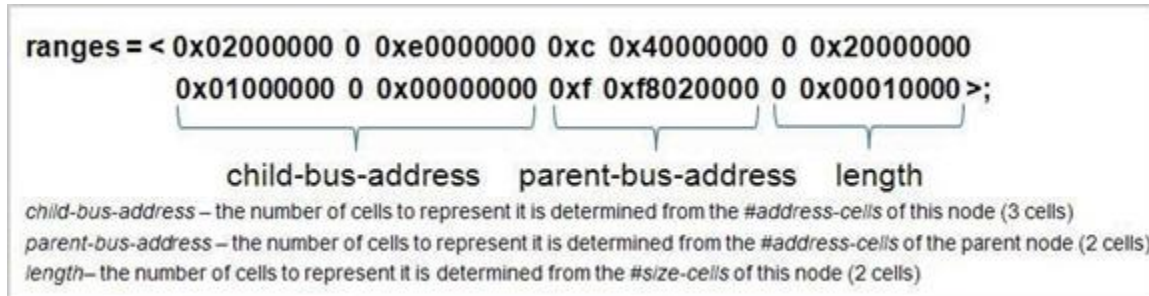


Figure 19. ranges property definition inside Hardware Device Tree

In (Figure 17. LAW defined inside Hardware Device Tree on page 18), `pci2` has one block of memory mapped register starting at `F_FE20_2000` with size 4-Kbyte in the SoC. There are two `pci2` ranges (see Figure 19. ranges property definition inside Hardware Device Tree on page 19) within the entire address space which can be accessed by a load or store operation. The first `pci2` memory block starts at `0xC_4000_0000` with size of 512-MByte; the second `pci2` block starts at address `0xF_F802_0000` with size of 64-Kbyte.

Once all memory blocks are decoded and gathered, they can be visually represented in the **Memory Map** view. The **Memory Map** view pops-up automatically with the created memory block areas when a hardware device tree component is selected in the **Component Inspector**. The **Memory Map** view displays its content created on the basis of DTS file that comes with the component. Block highlighting is available in the **Memory Map** view. Moreover, selecting a block in the **Memory Map** view triggers a node selection in the graphical hardware device tree.



Figure 20. Hardware Device Tree Memory Map view

1.2.3.2 Text editor

The text editor is the textual interpretation of the hardware device tree component.

The text editor of **HWDeviceTree** component involves syntax highlighting and provides support of expanding/collapsing nodes. The hardware device tree GUI editor and the text editor are synchronized. Therefore, when you select a node in the GUI editor, the node is selected automatically in the text editor also. The vice versa is also true. The figure below shows that after selecting the SoC node in the GUI editor, the corresponding entry in the text editor is automatically selected.

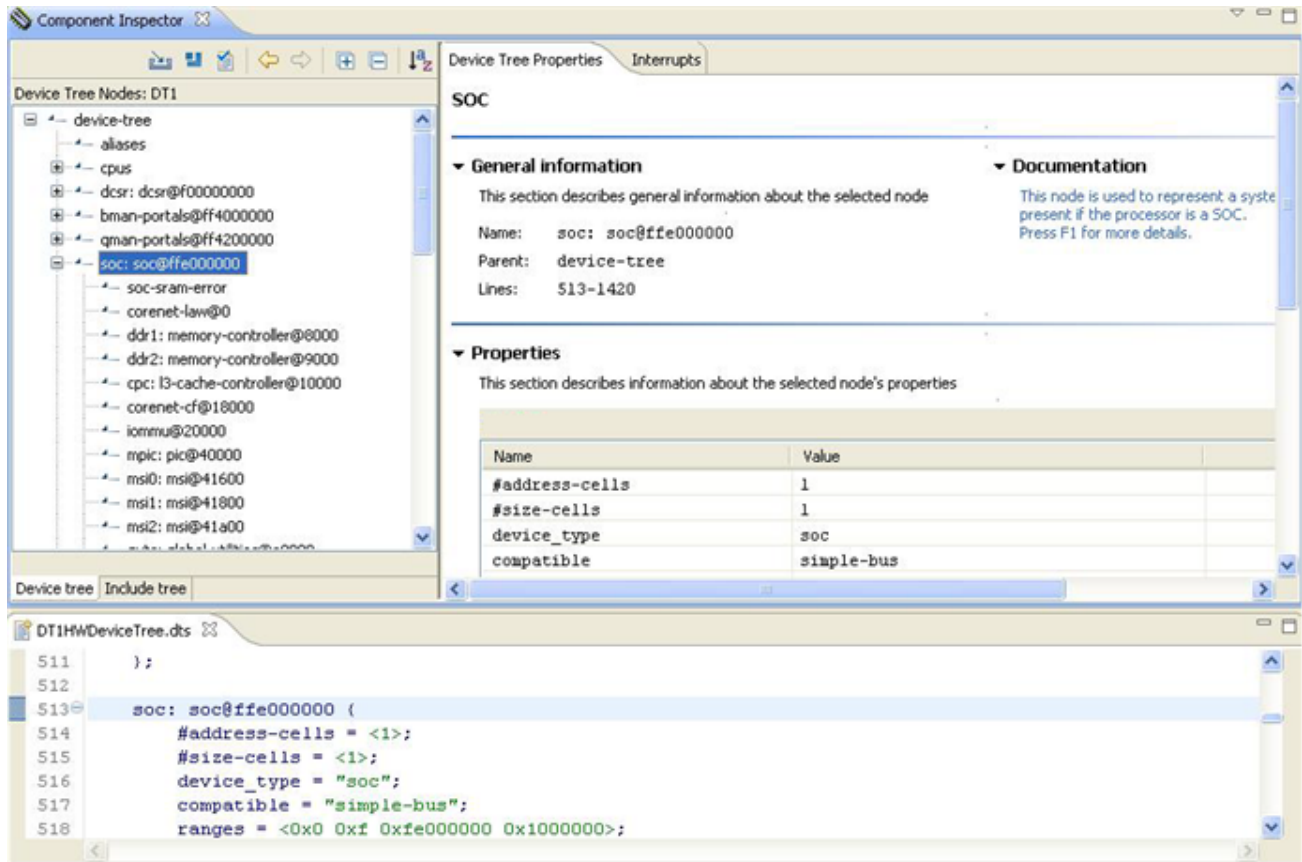


Figure 21. Hardware Device Tree text editor

If inclusions exist, the merged file, `DT1HWDeviceTree.dts`, will have hovering support in the text editor. When mouse cursor hovers over properties and nodes, a tooltip appears displaying initial locations of the files in the **Include Tree**.

The merged file is read-only; it gets updated based on other files. Therefore, you can edit only the top-level and the included files to automatically propagate changes in the final merged file.

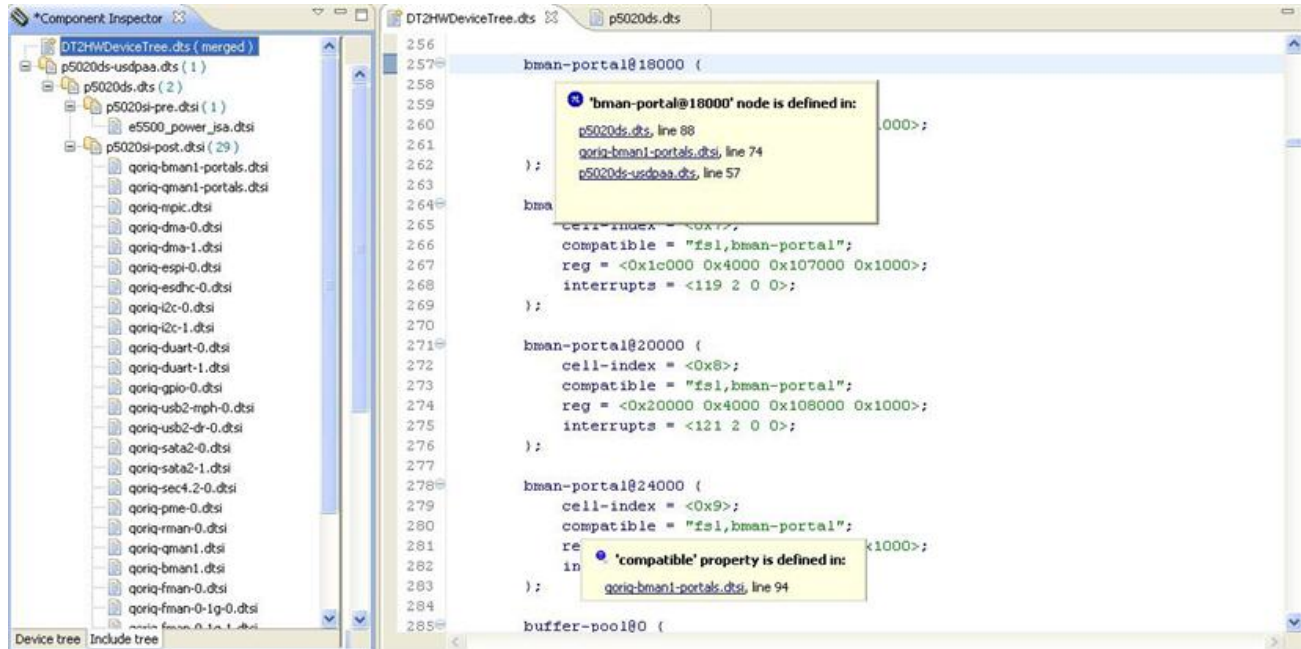


Figure 22. Origin of included files

Hyperlink detection is also supported for /include/ declarations and hardware device tree references. Use CTRL+left click combination on these statements to change the context in the referred file or node.

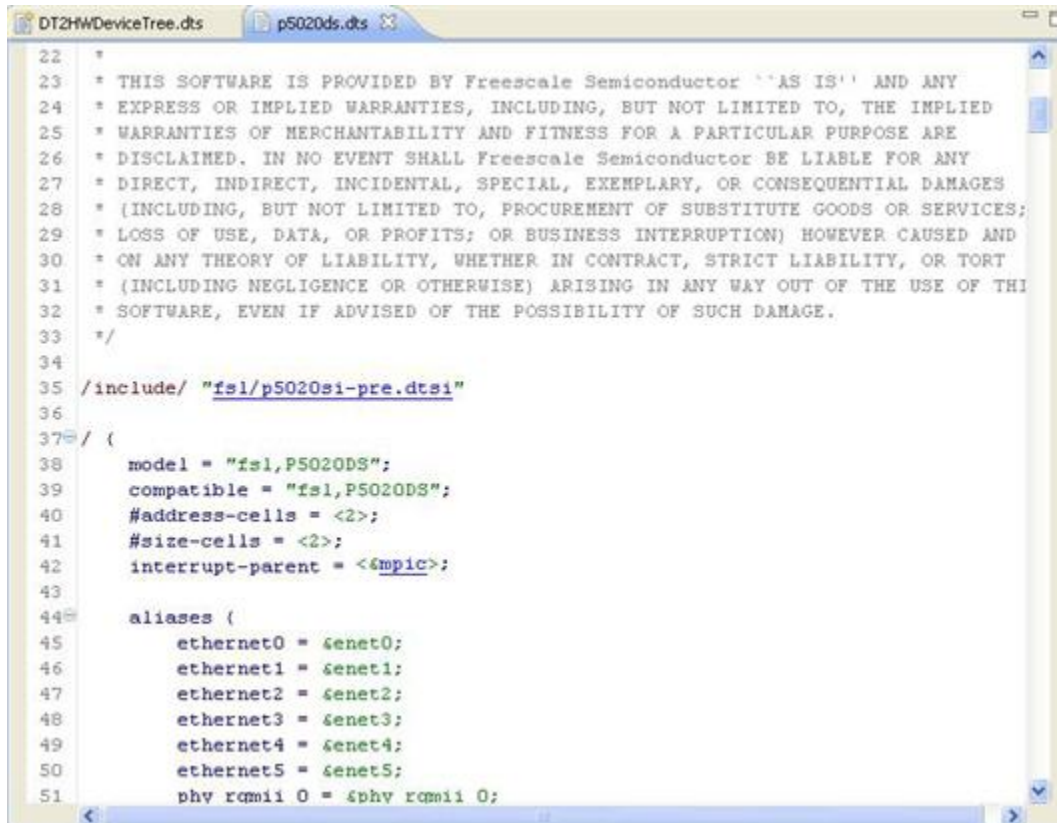


Figure 23. Hyperlink detection

Tips:

- To see the line numbers in the text editor, choose the **Window > Preferences > General > Editors > Text Editors > show line numbers** option.
- Use the workbench views as per your project needs and customize the perspective. To save the customized perspective, choose **Window->Save Perspective As**.

1.2.4 Perform validation

The Hardware Device Tree Editor allows you to verify the syntax of a hardware device tree and check if it conforms to the set of defined constraints.

There are two approaches to display errors: using markers in the **Eclipse Problems** view or using decorators in the **Properties custom** view table.

This section contains the following subsections:

- [Syntax validation](#) on page 23
- [Property constraints](#) on page 24

1.2.4.1 Syntax validation

You can check errors and validate syntax in the text editor using markers.

Using markers, the error checking is performed for the following:

- Syntax errors (inappropriate format)
- More than one root node
- Duplicate node labels
- Undefined node reference
- Node properties that do not preced sub-nodes

In case of inaccuracies, the text editor displays the markers on the incorrect lines. The markers are also visible in the **Problems** view.

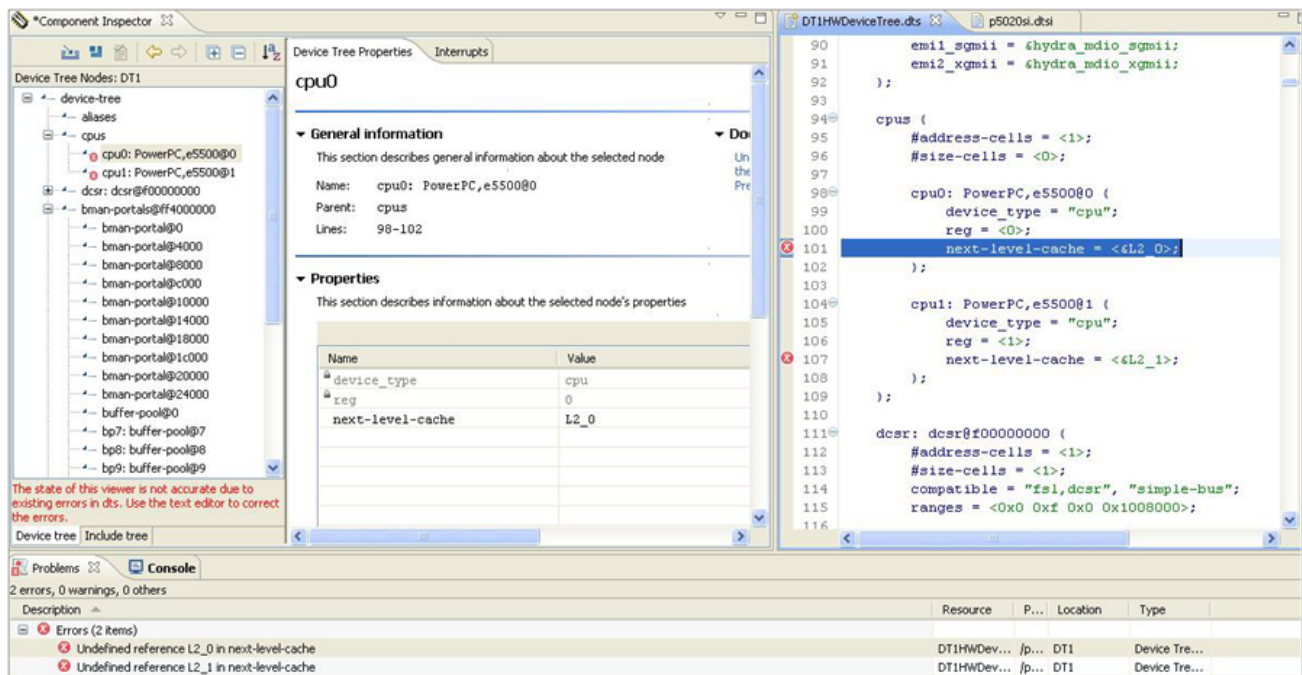


Figure 24. Error detection using markers

You can turn on/off the validation against the device tree bindings in the **Device Tree Settings** page (Figure 31. [Device Tree Settings page](#) on page 28). If it is enabled, the validation starts automatically whenever changes occur in the hardware device tree. You can also perform validation manually using the third option in the **View** toolbar (Figure 12. [View toolbar](#) on page 14).

1.2.4.2 Property constraints

You can set property constraints of the hardware device tree node using XML binding.

Each hardware device tree node is completed and distinguished by an XML "binding". The binding describes which properties are required, which are optional, what each property means and what constraints are to be met. The repository of hardware device tree bindings is stored in `\ProcessorExpert\Beans\HWDeviceTree\dts_bindings`.

The xml binding keeps information about the list of devices supported by the node, node instances, sub-nodes, properties, constraints, and example. The following figure shows the structure of a node and a property.



Figure 25. XML node and property schema

Following are the types of constraints that can be assigned to a property:

- `<type>` - specifies the type of a property value
- `<editable>` - specifies whether or not the property can be modified
- `<min-size>`, `<max-size>` - specifies the minimum and maximum number of items allowed in a property cell
- `<value-list>` - specifies a set of allowed values for a property
- `<range>` - bounds a numeric value between minimum and maximum values; it can also force a property value to match a regular expression.

Take an example where you have the `dma-channel@0` node as follows:

```
dma-channel@0 {
compatible = "fsl,p5020-dma-channel", "fsl,eloplus-dma-channel";
reg = <0x0 0x80>;
cell-index = <0>;
interrupts = <28 2 0 0>;
};

dma-channel@100 {
compatible = "fsl,p5020-dma-channel", "fsl,eloplus-dma-channel";
reg = <0x100 0x80>;
cell-index = <2>;
```



```
interrupts = <30 2 0 0>;
};
```

Assume you want to define the following constraints:

- restrict the type of interrupts property to EMPTY for all dma-channel instances
- for dma-channel@0, define reg property
 - to have at least 3 items in its value
 - first item to be in [0xA, 0xC] range
 - second item to be in [0x0, 0x400] range
- for dma-channel@100
 - compatible property to be non-editable
 - to have exactly 3 items in its value

Then the dma-channel.xml file will be modified as shown in the following figure. Using this representation, you can compare the values set against the device tree bindings.

```
<property>
  <name>reg</name>
  <type>CELLLIST</type>
  <required>yes</required>
  <constraints>
    <set min-size="3" instance="dma-channel@0">
      <set-item index="0">
        <range>
          <min-value>0xA</min-value>
          <max-value>0xC</max-value>
          <regexp>.*</regexp>
        </range>
      </set-item>
      <set-item index="1">
        <range>
          <min-value>0x0</min-value>
          <max-value>0x400</max-value>
        </range>
      </set-item>
    </set>
    <set min-size="3" max-size="3" instance="dma-channel@100"/>
    <editable>true</editable>
  </constraints>
</property>
<property>
  <name>interrupts</name>
  <type>EMPTY</type>
  <required>no</required>
</property>
```

Figure 26. Example of defining a constraint

Non-editable properties are disabled in the graphical editor, as shown in the following figure. They can be changed only through the text editor.

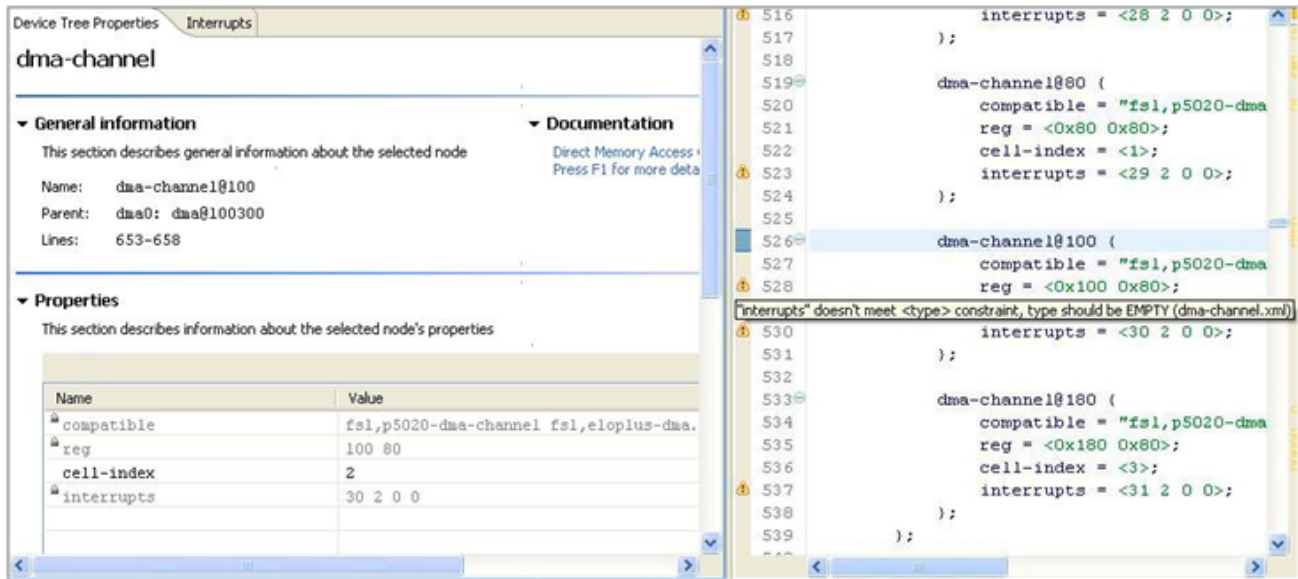


Figure 27. Hardware Device Tree property constraints for non-editable properties

Setting these constraints produce the following warnings in the **Problems** view:

- "interrupts" does not meet <type> constraint for all dma-channel instances from all dma nodes (4 instances x 2 nodes)
- "reg" does not meet <min-size> constraint at dma-channel@0 and dma-channel@100 instances (there are 2 items and the constraint requires minimum 3)
- "reg" does not meet <min-value> constraint at index 0 at dma-channel@0 instances: item at index 0 is 0x0 and the constraint requires a value between 0xA, 0xC

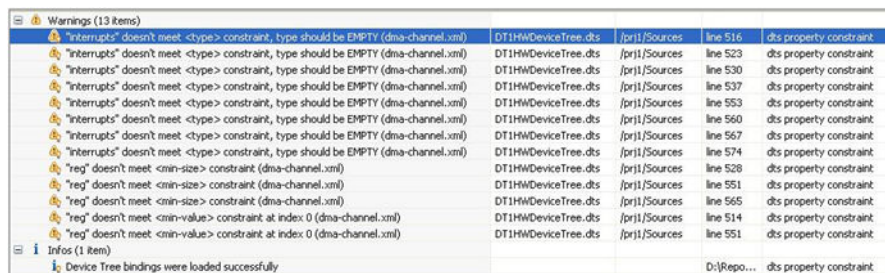


Figure 28. Hardware Device Tree property constraint warnings

1.2.5 Search in hardware device trees

Use the **Search** view to search for the required information in the project's device tree files.

The **Search** view can be opened in three ways:

- Select **Search > Search > Device Tree Search** in the main menu bar.
- **Search** option from the context menu of the hardware device tree graphical editor.
- Use the shortcut CTRL+ALT+H.

There are three options that specify the criteria based on which you can search the text:

- Case sensitive
- Whole word
- Regular expression

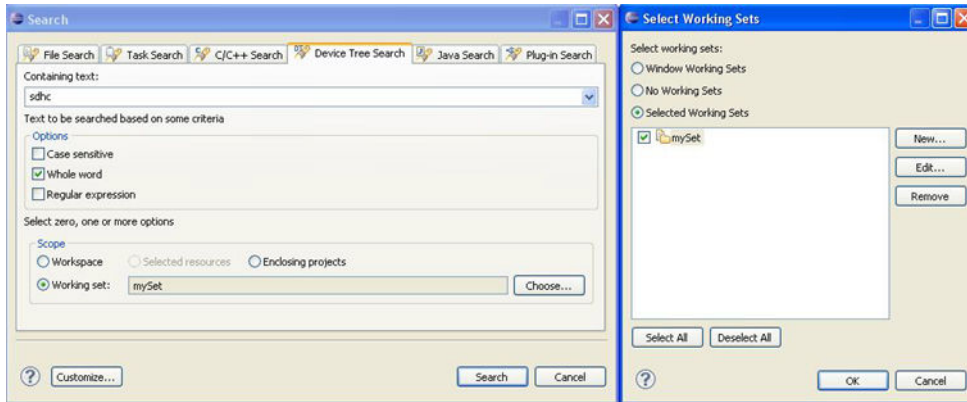


Figure 29. Hardware Device Tree Search view

By default, searching is performed in the whole workspace and all encountered dts files. To restrict searching to a particular project or working set, use the **Scope** feature in the **Search** dialog, which provides the following options:

- **Workspace** - is the default option; when selected, searches in all open projects
- **Selected resources** - searches in the selected resource if applicable; it is a hardware device tree file or contains hardware device tree files
- **Enclosing projects** - searches in the selected project only
- **Working set** - allows you to create a custom working set with the desired resources

The results appear in the **Search** view.

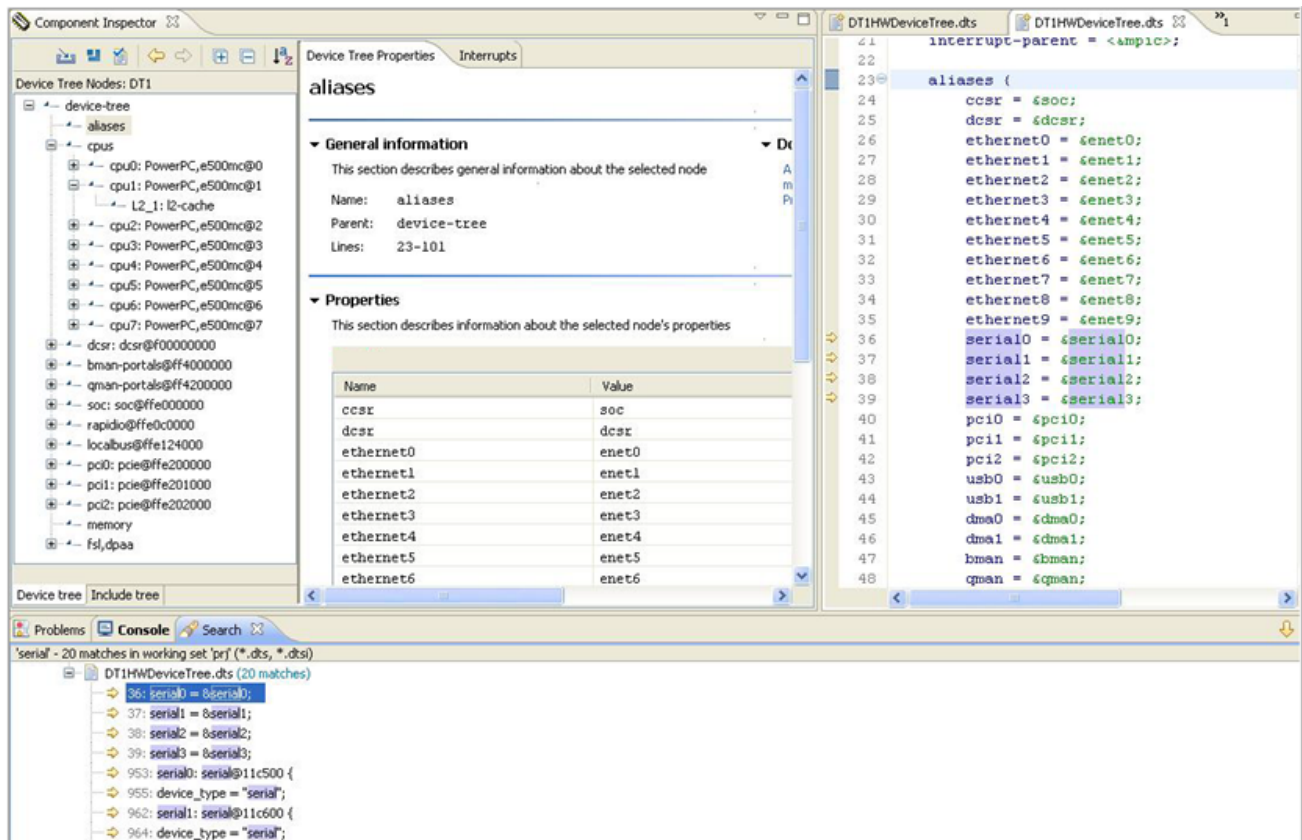


Figure 30. Device Tree Search view results

1.2.6 Generate device tree blob

The Hardware Device Tree Editor enables you to generate device tree binaries if a device tree compiler is available.

You can find the DTC settings in the **Device Tree Compiler** page of the **Preferences** dialog, as shown below, which appears on selecting **Window > Preferences > Processor Expert > Device Tree Settings**.

You can change the default settings. If this functionality is not required, it can be switched off by clearing the **Configure arguments to generate device trees** checkbox.

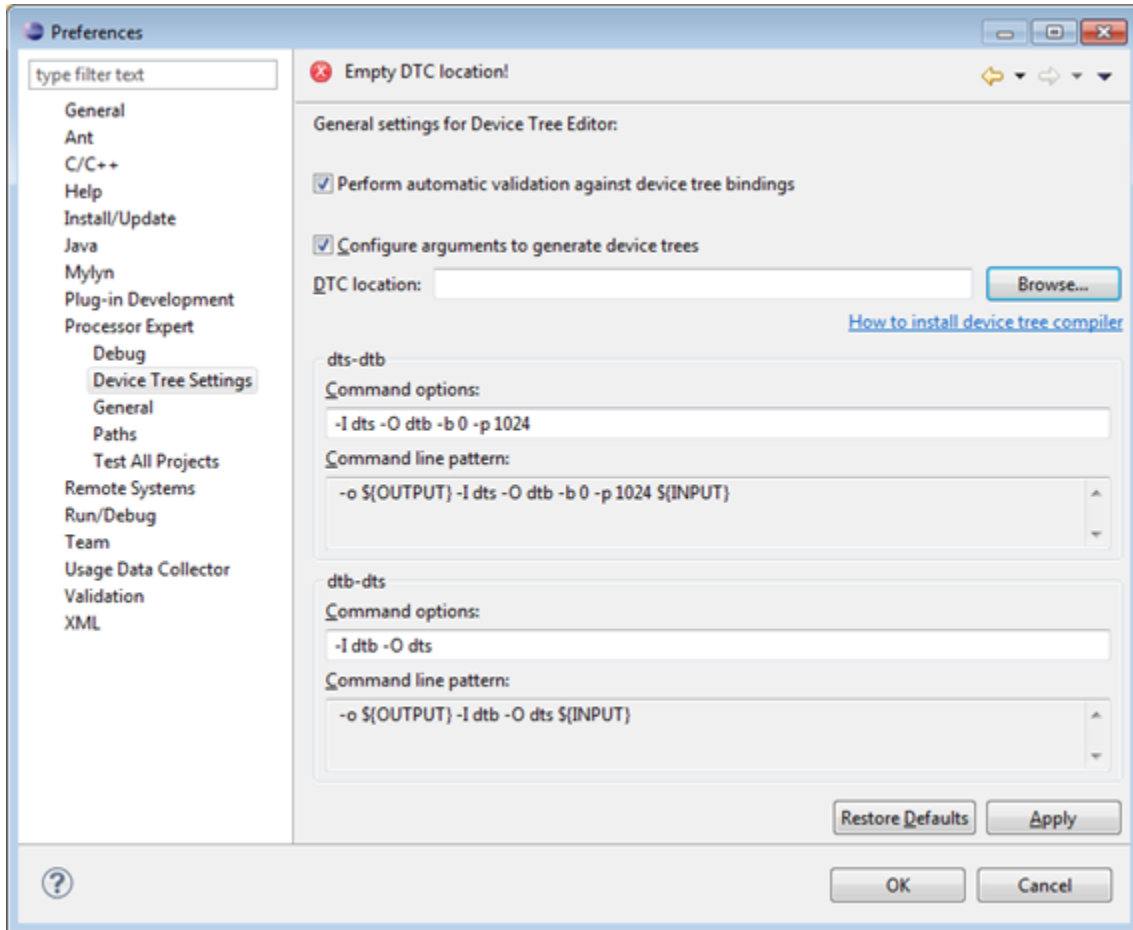


Figure 31. Device Tree Settings page

To generate DTB files, select **Project > Generate Processor Expert Code** from the IDE menu bar. The code is generated per project, therefore, the device tree blobs are created for all encountered device tree sources in the opened projects. The resultant device tree blob files are added to the **Generated_Code** folder, as shown in the following figure.

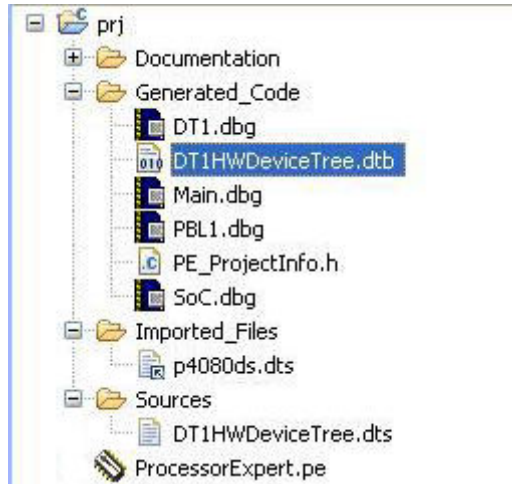


Figure 32. Generate DTB file

1.3 Known issues and limitations

To find a list of known issues and limitations, see the `ReadMe.html` file.

How To Reach Us**Home Page:**

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, Freescale, the Freescale logo, CodeWarrior, QorIQ, and Processor Expert are trademarks of NXP B.V. All other product or service names are the property of their respective owners. All rights reserved.

© 2017 NXP B.V.

