



How to Create an ETR Configuration Tab in the DS-5 DTSL Options Dialog

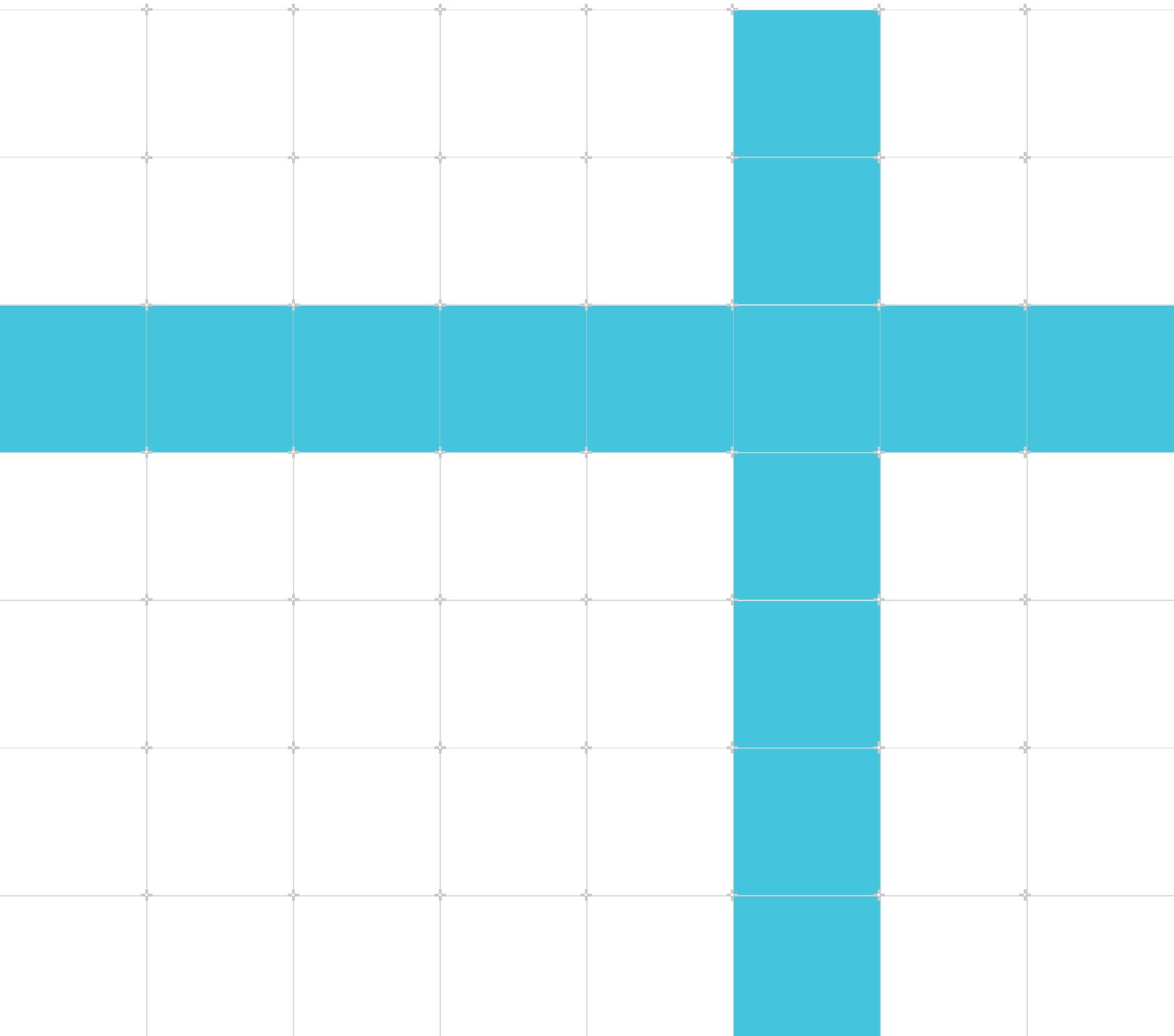
Version 1.0

Non-Confidential

Copyright © 2019 Arm Limited (or its affiliates).
All rights reserved.

Issue 00

102663_0100_00_en



How to Create an ETR Configuration Tab in the DS-5 DTSL Options Dialog

Copyright © 2019 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-00	4 November 2019	Non-Confidential	First release

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly

or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2019 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview.....	6
2. Adding a tab to the DTSL Options dialog.....	8
3. Adding controls.....	9
4. Using the controls values.....	11
5. Summary.....	12

1. Overview

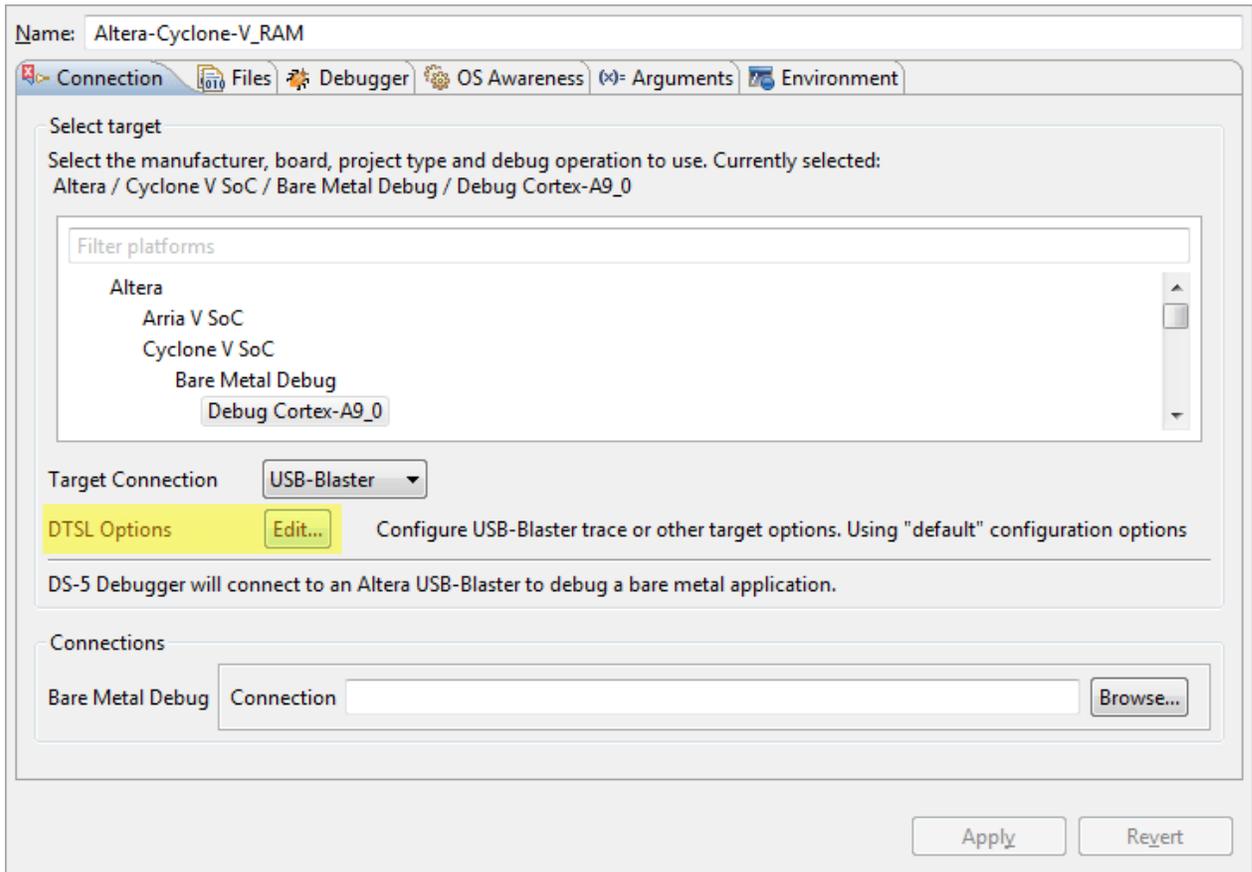
Learn how to create and customize an Embedded Trace Router (ETR) tab in the DTSL options dialog in DS-5 Development Studio using a simple Python script.

The controls in the DTSL Options dialog enable you to customize a debug session - configuring debug and trace according to the needs of a particular debug session. The controls and their supporting functionality are all implemented using a simple Python script. In this tutorial, learn how to extend a Python script to add the ability to configure the area of system memory used by the TMC-ETR for storage of trace data.

The Arm CoreSight Trace Memory Controller (TMC) device plays a crucial role in trace capture and storage in many modern Arm-based designs. When implemented as an Embedded Trace Router (ETR), the TMC stores trace data in the system memory of the target platform. Because memory maps can change and the location and amount of memory available to the TMC might not be constant across debug sessions, we have added a tab to the DTSL Options dialog which contains a number of controls specifically targeted at memory configuration for the TMC. We have also scripted the functionality that takes user-entered values from these controls, and use them to configure the TMC.

This example shows how to build the ETR tab in the existing Altera Cyclone V SoC platform configuration from scratch.

Figure 1-1: Debug configuration menu showing the DTSL scripting button



2. Adding a tab to the DTSL Options dialog

Every DTSL script can provide a `getOptionList()` method to return an array of objects representing the controls in the DTSL Options dialog. Because this method needs to be called before any DTSL objects have been instantiated (i.e. before the debug session has started), the method must be declared as static. For a simple target platform the `getOptionList()` method can return an array of controls directly, but for a more complex platform this can lead to a method that is large, complicated and difficult to read. The Altera Cyclone V SoC platform needs a variety of controls spread across a number of tabs, so for this platform the `getOptionList()` method simply calls a number of other methods, each of which returns the control objects for a single tab in the DTSL Options Dialog:

```
@staticmethod
def getOptionList():
    return [
        DTSLv1.tabSet("options", "Options", childOptions=[
            DtslScript.getOptionCrossTriggerTabPage(),
            DtslScript.getOptionTraceBufferTabPage(),

            DtslScript.getOptionCortexA9TabPage(),
            DtslScript.getOptionSTMTabPage(),
            DtslScript.getOptionETFTabPage()
        ])
    ]
```

We can add a new method for ETR control, which just returns an object for an empty tab:

```
@staticmethod
def getOptionList():
    return [
        DTSLv1.tabSet("options", "Options", childOptions=[
            DtslScript.getOptionCrossTriggerTabPage(),
            DtslScript.getOptionTraceBufferTabPage(),

            DtslScript.getOptionCortexA9TabPage(),
            DtslScript.getOptionSTMTabPage(),
            DtslScript.getOptionETFTabPage(),
            DtslScript.getOptionETRTabPage()
        ])
    ]

@staticmethod
def getOptionETRTabPage():
    return DTSLv1.tabPage("etrtab", "ETR", childOptions=[])
```

Every DTSL control takes at least two parameters. The first is a string that identifies the control to DTSL, and is used when retrieving the user-assigned value of the control. The second parameter is the display name of the control, as it will appear in the DTSL Options dialog. Additional attributes are added to the control as `name=value` pairs; here we are using the `childOptions` attribute to pass an array of controls that will appear inside the tab. This array is currently empty and we now need to add these controls.

3. Adding controls

We need a master checkbox that controls configuration of the ETR memory buffer, and use the `defaultValue` attribute to ensure that it is unchecked by default. The checkbox will guard access to all of its child controls - unless the box is checked the child controls will appear greyed in the DTSL Options dialog and their values cannot be changed:

```
@staticmethod
def getOptionETRTabPage():
    return DTSLv1.tabPage("etrtab", "ETR", childOptions=[
        DTSLv1.booleanOption('etrBuffer',
            'Configure the system memory trace buffer',
            defaultValue = False,
            childOptions = [
                ]
            )
    ])

```

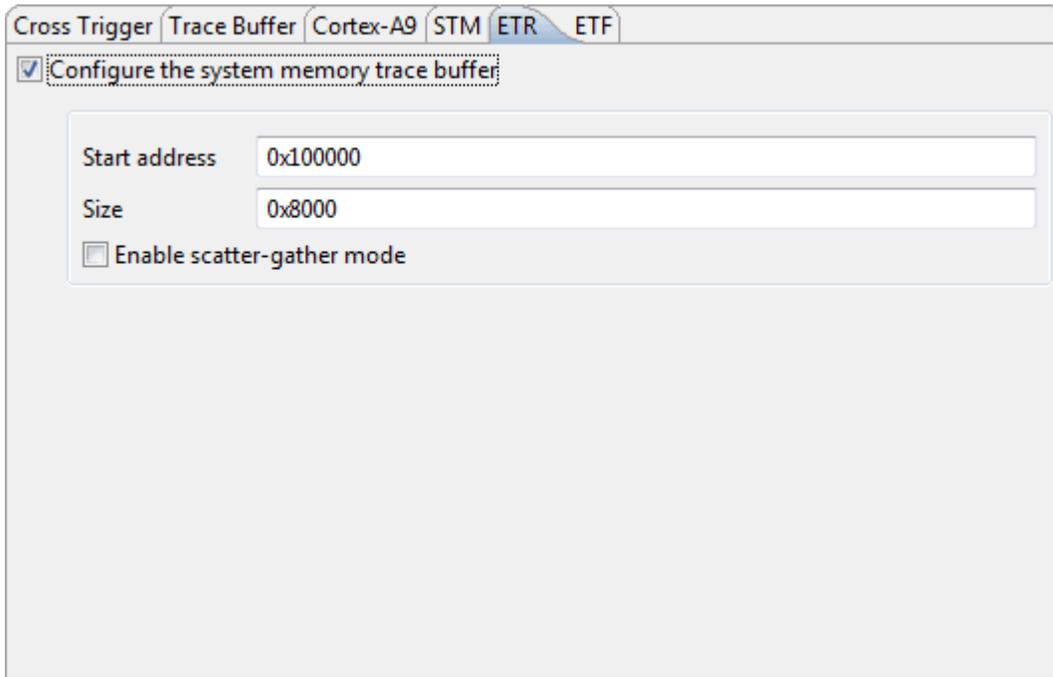
Now we can add the controls we need to configure the ETR memory buffer: edit boxes to control the start and the size of the buffer, and another checkbox to enable scatter-gather mode:

```
@staticmethod
def getOptionETRTabPage():
    return DTSLv1.tabPage("etrtab", "ETR", childOptions=[
        DTSLv1.booleanOption('etrBuffer',
            'Configure the system memory trace buffer',
            defaultValue = False,
            childOptions = [
                DTSLv1.integerOption('start', 'Start address',
                    description='Start address of the system memory trace
buffer',
                    defaultValue=0x00100000,
                    display=IIntegerOption.DisplayFormat.HEX),
                DTSLv1.integerOption('size', 'Size',
                    description='Size of the system memory trace buffer in
bytes',
                    defaultValue=0x8000,
                    display=IIntegerOption.DisplayFormat.HEX),
                DTSLv1.booleanOption('scatterGather', 'Enable scatter-gather
mode',
                    defaultValue=False,
                    description='When enabling scatter-gather mode, the start
address of the on-chip trace buffer must point to a configured scatter-gather
table')
            ]
        )
    ])

```

As well as default values for each of the controls, we've used the `description` attribute to add a tooltip that will appear when the mouse is hovered over the control. We've also ensured that the edit boxes will deal only in hexadecimal numbers. The code that we've added is enough to create the new control tab and populate it with the controls that we need:

Figure 3-1: DTSL options dialog showing a custom ETR tab added using Python scripting



4. Using the controls values

So far we have added code to display additional controls in the DTSL Options dialog, but we haven't added any code to deal with the current values of the controls. The existing `optionValuesChanged()` method uses the `getOptionValue()` method to retrieve the current value of a control, using the DTSL identification string passed as the control's first parameter. We can add a section of code to the `optionValuesChanged()` method to retrieve the current values of the controls in our new tab, and pass them to existing methods provided by the object created to configure and manage the ETR device:

```
def optionValuesChanged(self):
    # Set up the ETR buffer
    configureETRBuffer = self.getOptionValue("options.etrtab.etrBuffer")
    if configureETRBuffer:
        scatterGatherMode =
self.getOptionValue("options.etrtab.etrBuffer.scatterGather")
        bufferStart = self.getOptionValue("options.etrtab.etrBuffer.start")
        bufferSize = self.getOptionValue("options.etrtab.etrBuffer.size")
        self.ETR.setBaseAddress(bufferStart)
        self.ETR.setTraceBufferSize(bufferSize)
        self.ETR.setScatterGatherModeEnabled(scatterGatherMode)
```

5. Summary

That's it - that's all the code we need to add. This is a very simple example, adding a small number of controls to leverage some existing DTSL functionality, but the theory scales very well and can be used to add large numbers of tabs and controls with complex underlying functionality. The flexibility and extensibility can be used to give users the controls they need to customize a debug session and make the best of the capabilities of the target platform.