

WIND RIVER

# Wind River® Simics® Eclipse

USER'S GUIDE

4.6

<i>Revision</i>	4081
<i>Date</i>	2012-11-16

Copyright © 2010–2012 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, Simics, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. The Wind River logo is a trademark of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:  
[www.windriver.com/company/terms/trademark.html](http://www.windriver.com/company/terms/trademark.html)

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation at the following location:  
`installDir/LICENSES-THIRD-PARTY/`.

Wind River may refer to third-party documentation by listing publications or providing links to third-party Web sites for informational purposes. Wind River accepts no responsibility for the information provided in such third-party documentation.

#### **Corporate Headquarters**

Wind River  
500 Wind River Way  
Alameda, CA 94501-1153  
U.S.A.

Toll free (U.S.A.): 800-545-WIND  
Telephone: 510-748-4100  
Facsimile: 510-749-2010

For additional contact information, see the Wind River Web site:  
[www.windriver.com](http://www.windriver.com)

For information on how to contact Customer Support, see:  
[www.windriver.com/support](http://www.windriver.com/support)

# Contents

<b>1</b>	<b>Getting Started</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Launching Eclipse . . . . .	6
1.2.1	Advanced Launch Option . . . . .	8
1.3	Running the Simulation . . . . .	10
1.4	Checkpointing . . . . .	12
1.5	Reverse Execution . . . . .	13
<b>2</b>	<b>User Interface</b>	<b>17</b>
2.1	Basics . . . . .	17
2.1.1	Projects . . . . .	17
2.1.2	Perspectives . . . . .	18
2.2	Before Launch . . . . .	18
2.2.1	Checkpoints . . . . .	18
2.2.2	Scripts . . . . .	19
2.3	Running . . . . .	20
2.3.1	The Simics Command Line Console . . . . .	20
2.3.2	Running Several Simulations . . . . .	20
2.3.3	Reverse Execution and the Timeline . . . . .	20
2.3.4	Stopping . . . . .	21
2.3.5	Saving Checkpoints . . . . .	22
2.4	Additional Views . . . . .	22
2.4.1	Target Info View . . . . .	22
2.4.2	Graphical Timeline View . . . . .	23
2.4.3	Target Memory View . . . . .	24
2.4.4	Memory Spaces View . . . . .	25
2.4.5	Device Registers View . . . . .	26
2.5	Diagnosing problems . . . . .	26
<b>3</b>	<b>Simics Eclipse DML Editor</b>	<b>28</b>
3.1	Enable Emacs Key-Bindings . . . . .	28
3.2	Creating a new Simics Project . . . . .	28
3.3	Connect to an Existing Simics Workspace . . . . .	32
3.4	Creating a new DML Device . . . . .	32
3.4.1	Using the Outline View . . . . .	32

3.4.2	Using the Problems View . . . . .	35
3.5	Indexing and Searching . . . . .	35
3.6	Building the device . . . . .	37
<b>Index</b>		<b>40</b>

# Chapter 1

## Getting Started

### 1.1 Introduction

The Simics Eclipse Tools provide a graphical user interface to Simics for the open source development environment Eclipse. It provides a graphical user interface for Simics with tools to run and analyze simulations. This document will show you how to use Simics Eclipse.

---

**Note:** In order to use the Simics Eclipse Tools described in this document, you first need to have a working installation of *Wind River Simics 4.6*. Please make sure that you have the following packages installed:

- Simics-Base (package number 1000)
- Firststeps (package number 4005)

Refer to the *Installation Guide* for instructions on how to obtain and install these packages. Before using the Simics Eclipse Tools, make sure that you can launch Simics from outside Eclipse to ensure that everything is installed properly.

For instructions on how to obtain and install Eclipse and the Simics Eclipse Tools see the chapter *Installing the Simics Eclipse Tools* in the *Installation Guide*. That chapter also list the versions of the Eclipse Platform and other Eclipse features that are compatible with the Simics Eclipse Tools.

---

Before we start our first simulation there are a few Simics Eclipse concepts that we need to understand. Some of these concepts come from Simics and some from Eclipse in general. There is also some concepts which have different names in Simics and Eclipse.

Simics is a system level instruction set simulator, which simulates one or more *target machines* (or *target architectures*) while Simics itself is running on a *host machine*. Physical properties of the target machine, such as CPU-model and frequency, memory size, etc. are described by a *configuration*. The internal state of the target, such as memory and register contents, can also be stored in a *checkpoint*.

Simics's configuration system is object-oriented, and target machine configurations are usually made up of *components*. A component is typically the smallest hardware unit that can be used when configuring a real machine, and examples include motherboards, PCI cards,

hard disks, and backplanes. Components are usually implemented in Simics using several *configuration objects*.

---

**Note:** To learn more about *configurations*, *checkpoints*, *components*, and *configuration objects* see the chapter *Configuration and Checkpointing* in the *Hindsight User's Guide*.

---

The Eclipse environment is organized into one or several *projects*, corresponding to directories on your hard drive. The projects live in an Eclipse *workspace*. The workspace stores settings that are not directly associated with a certain project. In a multi user setting, each user would usually have her own workspace where all her projects are stored.

---

**Note:** Do not confuse the *Eclipse workspace* with the *Simics workspace*. A Simics workspace corresponds to an Eclipse project. An Eclipse workspace may contain several projects (Simics workspaces). Compare with the *Installation Guide*, section *Associate Eclipse with a Simics Base installation*.

---

In Eclipse, projects are viewed through *perspectives*, which basically are pre-defined collections of *views*. Views are components on the screen marked by a tab with their name. A perspective is intended for a certain kind of activity such as writing code or debugging, and you can change between perspectives at any time. During the First Steps tutorial you will mostly work with the Simics perspective.

---

**Note:** To learn more about *workspaces*, *projects*, *perspectives*, and *views*, see Chapter 2.1.

---

In Simics you will often work with one or more *consoles*. First of all there is the *Simics console* in which you can enter commands to Simics, through the command line interface (CLI). You also get output from the simulation in the Simics console. To learn more about the Simics command line interface see the chapter *Script Support in Simics* in the *Hindsight User's Guide*.


For each machine you are simulating you will usually also have either a *text-console* or a *graphics-console* representing a terminal attached to the simulated machine. If the simulation is running you can enter commands to the simulated machine in this console, just as you would at a terminal connected to a real machine.

## 1.2 Launching Eclipse

Start Eclipse by running the `eclipse` program.

After the Eclipse splash screen has been displayed, you are asked for a workspace directory. It should look like in Figure 1.1.

Use the **Browse** button to select any empty directory to be used as the Eclipse workspace. Simics Eclipse will use this directory to store user preferences and files.

Once the Eclipse environment has completed loading, you can use the new Simics Session button  to create a Simics project and launch a simulation session.

First, we need to create a project for the simulation. This is done in the dialog window, shown in figure 1.2.

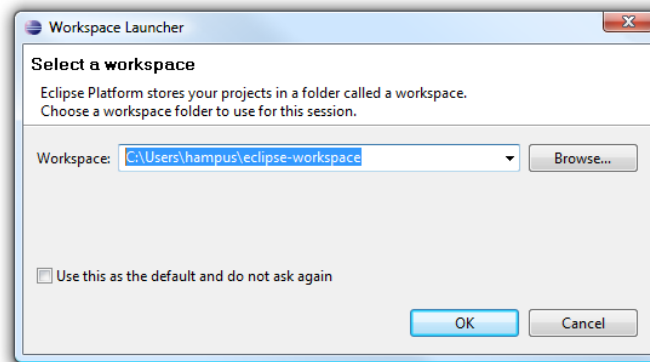


Figure 1.1: The Workspace Directory prompt

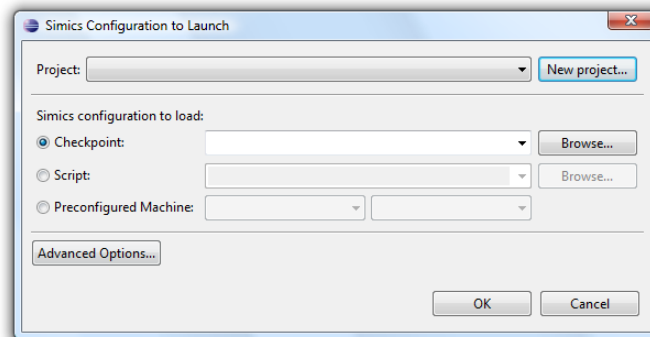


Figure 1.2: The New Simics Session Dialog

Click the button *New Project...* and type in the name **SimicsProject** in the *Name* field, as seen in figure 1.3. Click OK to continue.

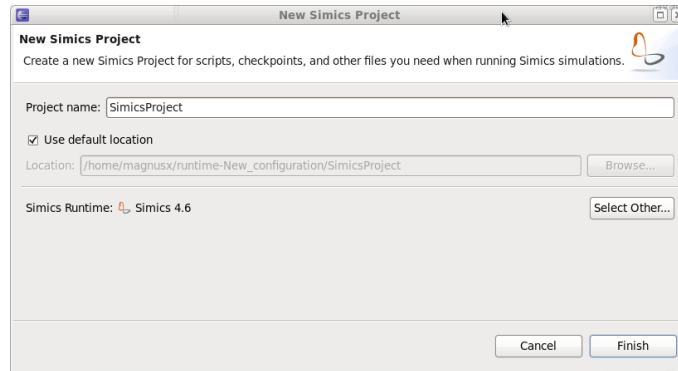


Figure 1.3: The New Simics Session Dialog: Select Project

Next, we select *Preconfigured machine*, *mpc8641-simple*, and *firststeps* and click the OK button, as seen in figure 1.4.

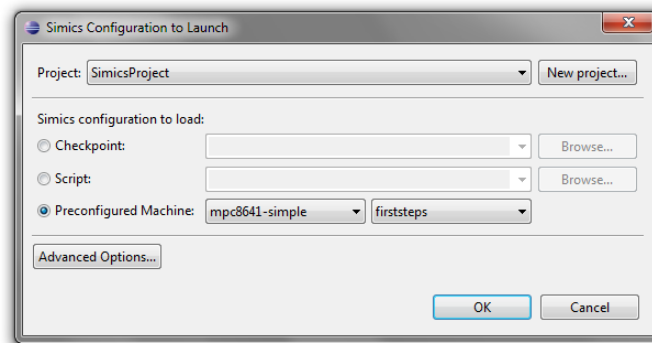


Figure 1.4: The New Simics Session Dialog: With MPC8641 First Steps selected.

An empty terminal window will pop up, which corresponds to *mpc8641d\_simple*'s console. The simulation starts in suspended mode, but in the section "Running the simulation" you will learn how to start the simulation.

### 1.2.1 Advanced Launch Option

When pressing the "Advance Option" button the following alternatives appear:

CPU mode determines whether to run Simics in the *standard mode* or *stall mode* (slower, but offers more profiling options). For details, see the section Simulation Modes in the *Hindsight User's Guide*.

By default, reverse execution support is enabled. If you are not interested in the ability to reverse the simulation, uncheck the checkbox marked `Allow reverse execution`. You can, however, also activate or disable it later (see section 2.3.3).



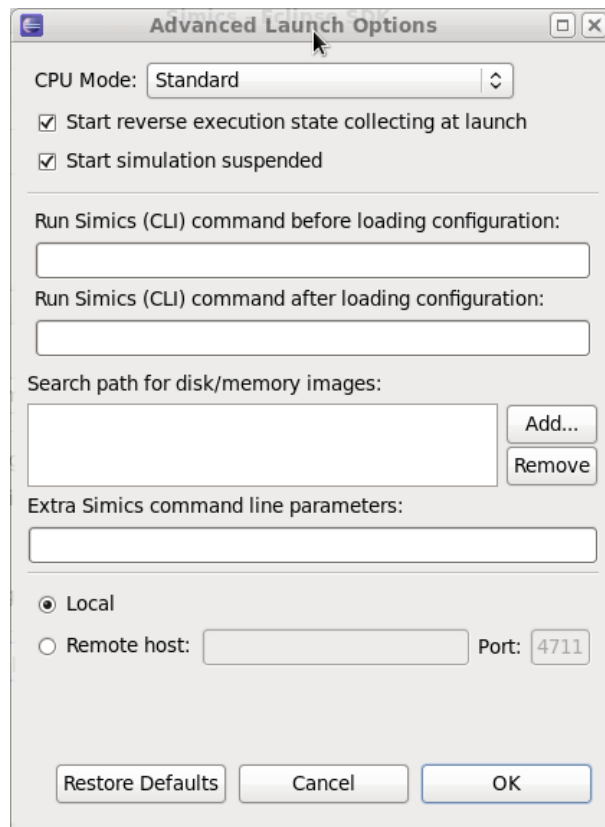


Figure 1.5: The advanced launch options dialog.

By default, the simulation is suspended immediately after launch. Uncheck `Start simulation suspended` if you prefer to have the simulation running directly from launch.

The `Run Simics (CLI)` command... input boxes allow for entering command line (CLI) commands that are run immediately after starting Simics or after loading the configuration (using one of the three methods above), respectively. You can separate several commands by semicolon ("`;`"), but if you have many commands to perform, the preferred method is to put them into a script file and use **run-command-file** `<file>`.

If you have disk dump files (`.craff`) located in non-standard directories, you can add them to the Simics search path using the `Search path for dumps` list.

Extra Simics command line parameters is for specifying command line arguments for which there are no graphical user interface equivalent. For a list of startup options see the chapter *Using Simics from the Command Line* in the *Hindsight User's Guide*.

The semantic for writing extra Simics command are to use double-quotes around arguments or parts of arguments containing spaces and if you want to pass double quotes in an argument you should escape it by prefixing it with a `\`.

Example of semantic for extra Simics commands:

- `-e $sim=ics or -e $sim="ics"`
- `-e "$simics=\"A simulator for the future\""`

and if you want to add more commands on a line separate them with white space


- `-e $sim=ics -e "$simics=\"A nice simulator\""`


When launching a simulation, an instance of the Simics executable is started. On Windows or if you use the standard installation location on Linux, Eclipse automatically locates the Simics installation. If you wish to use another installation, you can point out the Simics installation to use under **Window** → **Preferences** → **Simics**. However, do make absolutely sure that the Simics installation is compatible with the Eclipse environment version you are using. Otherwise, you risk experiencing hard-to-diagnose problems.

The Simics installation to use to run Simics sessions can be configured separately for each Simics project. Right-click on the project and select **Properties** → **Simics**.

## 1.3 Running the Simulation

Once you have launched the *First Steps* machine as described in the previous section you are ready to run the simulation.

The simulation can be controlled by the buttons that are located in the *Simics Control* view. The resume button have the icon . Click it to start the simulation.

In the virtual machine's console, you will first see the boot-loader *U-boot* load the compressed Linux kernel and a RAM disk. After that you will see the Linux kernel boot. You can pause the simulation at any time by clicking the suspend button , and continue by clicking resume again.

After some time Linux will be up and running and when the simulation arrives at the login prompt a script takes over to login in and configure the network. After that the

### 1.3. Running the Simulation

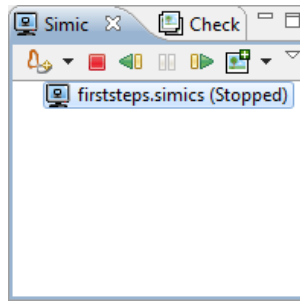


Figure 1.6: The Simics Control View

simulation keeps running and you can now try typing a few commands at the target prompt, e.g. the following commands (a sample output can be seen in Figure 1.7).

```
~ # pwd
/root
~ # ls /
bin          home        lib          proc         sys
dev          host        linuxrc      root         usr
etc          init        lost+found  sbin         var
~ #
```

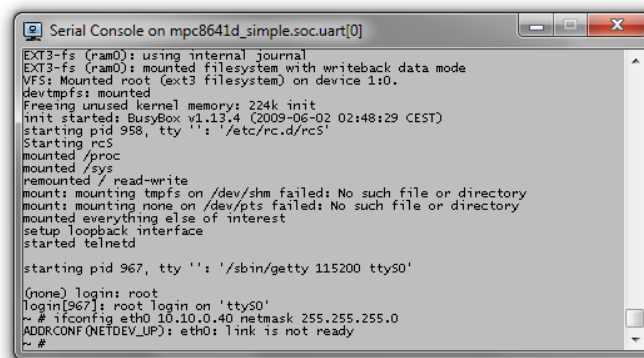



Figure 1.7: A booted MPC8641 Simple

---

**Note:** If the terminal console does not respond, make sure that the simulator is actually running.

---

You can end, or *terminate* the session by clicking , but do not do that right now.

Note that there is a distinct difference between *suspending the simulation*, which stops the target machine, and *terminating the session*, which unloads the configuration. A suspended

simulation can be resumed, but once a session has been terminated, it cannot be resumed again (unless a checkpoint has been saved as we shall see in section 1.4).

## 1.4 Checkpointing

In order to avoid booting First Steps every time, we use the facility known as *configuration checkpointing* or simply *checkpointing*. This enables Simics to save the entire state of a simulated machine in a portable format to be loaded at a later time.

To write a checkpoint, click the Save Checkpoint button in the *Control view*. You will be prompted for a name to use for the checkpoint. Name it “After boot.ckpt”.

The state is now saved, and you can see it in the *Checkpoint Explorer* view (Figure 1.8). Now terminate the simulation (■). All that remains of the simulation is the checkpoint we just created.

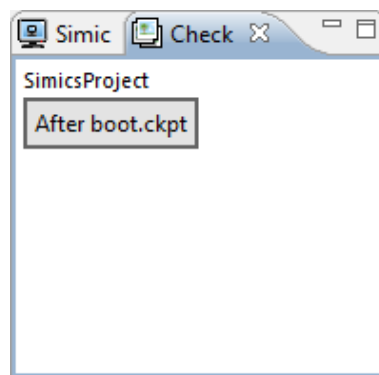


Figure 1.8: Checkpoint Explorer

---

**Note:** Optionally, you can also close the Eclipse environment. When you start it again in the same workspace, all checkpoints created should appear in the Checkpoint Explorer and also in the directory Simics Checkpoints in the Simics Projects view.

---

Double-click on the checkpoint in the Checkpoint Explorer to restore the simulation. In the virtual machine’s console, you should be able to see that it has already booted. The simulation will automatically be resumed.

Run the simulation for a while, enter some commands in the virtual machine’s console e.g.:

```
~ # cd /var
/var # ls
log  www
```

Now create a new checkpoint (**Run** → **Write Checkpoint**). Name it “a little later.ckpt”. If you look at the Checkpoint Explorer, you should see *a little later.ckpt* connected

to *After boot.ckpt*. In other words, Checkpoint Explorer displays the relationships between checkpoints.

You can try to load the *After boot.ckpt* checkpoint again (double-click on it), run it for a while and perhaps enter some commands like:

```
~ # cd /sys
/sys # ls
block      class      devices    fs          module
bus        dev        firmware   kernel
/sys #
```

Create another checkpoint, and call it e.g. “*alternative future.ckpt*”. It should appear connected to *After boot.ckpt*, but below the *a little later.ckpt* checkpoint. It should look like in Figure 1.9.

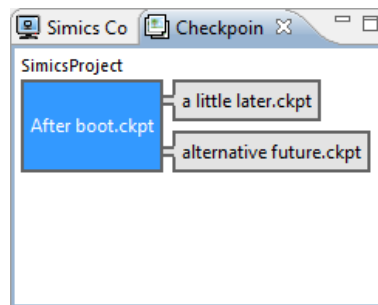


Figure 1.9: Several checkpoints

---

**Note:** You can read more about checkpointing in the chapter Configuration and Checkpointing in the *Hindsight User's Guide*.

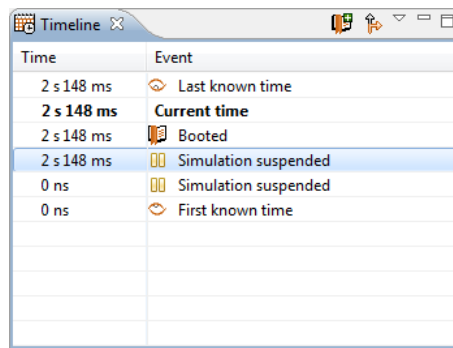
---

## 1.5 Reverse Execution

*Reverse execution* is a feature of Simics which allows you to reverse the time of the simulation and run the simulation backward. This makes it possible to deterministically move backward and forward in time. When moving forward you can either replay the same inputs as last time or supply new inputs. This is best understood by looking at an example, so let's try it out.

To demonstrate reverse execution, we will use a view called the *Timeline View*. To open it, select **Window** → **Show View** → **Other...** and select *Timeline*.

**Note:** By default, Simics Eclipse enables reverse execution at launch time when creating a configuration, this way the simulation can be reversed all the way back to the beginning of the session. Disabling reverse execution however allows the simulation to run slightly faster. If you prefer to not have reverse execution loaded by default you can change this when creating the new session (click the button *Advanced Options...*).



Time	Event
2 s 148 ms	Last known time
<b>2 s 148 ms</b>	<b>Current time</b>
2 s 148 ms	Booted
2 s 148 ms	Simulation suspended
0 ns	Simulation suspended
0 ns	First known time

Figure 1.10: The Timeline view

When reverse execution is enabled, you can right-click on any entry and select **Skip to** to move in time. This can in some cases be time-consuming. That is why it is also possible to create *bookmarks*. A time bookmark appear in the list with the name of your choice. It is usually much faster to skip to a bookmark than to other events. However, bookmarks are not stored in checkpoints. Thus, when a simulation is terminated, all bookmarks are erased. Create a bookmark now by right-clicking on **Current time** → **Bookmark Current Point in Time** Name it *Booted*.

Make sure the simulation is running (current time should be increasing), and in the virtual machine's console, enter the following command:

```
~ # cat /etc/fstab
proc          /proc  proc      defaults      0      0
sysfs         /sys   sysfs     defaults      0      0
simicsfs      /host  simicsfs  rw,noauto,nocache 0      0
tmpfs         /dev/shm tmpfs     defaults      0      0
none          /dev/pts devpts    defaults      0      0

~ #
```

Right-click on the bookmark **Booted** → **Skip to**, see Figure 1.11. The output from **cat** has disappeared! Move forward in time again by selecting **Run** → **Resume**. The **cat** command will be replayed.

Now, let's see how we can use reverse execution. Enter these commands in the virtual machine's console:

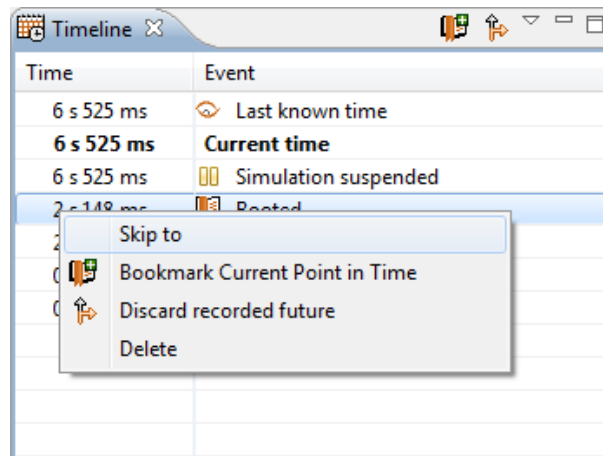


Figure 1.11: Skip to event in time

```

~ # rm /bin/ls
~ # ls /
-sh: ls: not found

```

The program `ls` has been removed. You can no longer list files. Let's use reverse execution to "recover" `ls`.

Use the *Booted* bookmark to move back in time again, but do not resume the simulation just yet. If you do, the `rm` command will be replayed and the file `/bin/ls` lost again. What we need to do is erase all knowledge about the future. Right click on **Current time** → **Discard recorded future**, as shown in Figure 1.12. All events in the future will now be erased.

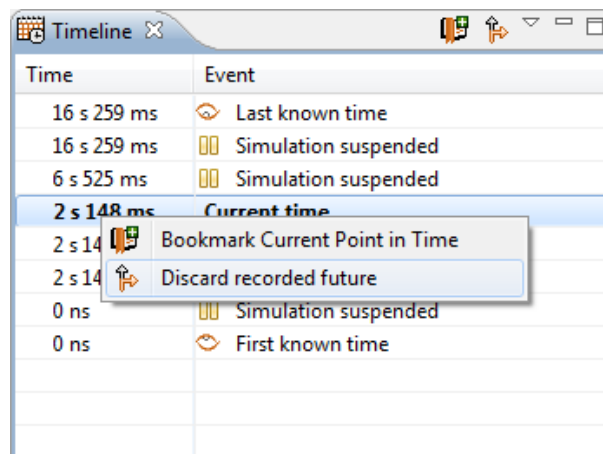


Figure 1.12: Clear recorder

Resume the simulation and enter the following command:

```
~ # ls /  
bin      home      lib        proc       sys  
dev      host       linuxrc    root       usr  
etc      init       lost+found sbin       var  
~ #
```

**ls** works again!



## Chapter 2

# User Interface

Simics provides two user environments. You will usually use the Eclipse environment, a graphical user interface based on the open source development environment Eclipse. It provides an easy-to-use interface to most functionality and allows for integration with other Eclipse-based development tools, thus making it possible to perform all software development activities from a single integrated environment.

There is also a command line interface (CLI) environment that is more light-weight, can be used for batch-mode simulation and might be preferred by advanced users familiar with previous versions of Simics.

This section focuses on use of the Eclipse environment, at each step explaining what command line commands can be used to perform the same actions.

Simics Eclipse uses a graphical user interface based on the open source development environment Eclipse. It provides an easy-to-use interface to most functionality and allows for integration with other Eclipse-based development tools, thus making it possible to perform all software development activities from within it.

## 2.1 Basics

### 2.1.1 Projects

All data used in the Eclipse environment is organized into *projects*, corresponding to directories on your hard drive. The first thing to do when starting to use Simics is to create a new project.

Projects are organized into *workspaces*, storing settings that are not directly associated with a certain project. A workspace has to be selected when starting Eclipse, though the most common setting is to automatically use the last opened workspace. Projects are usually created directly under the workspace directory, though you can choose to create them in another directory.

You usually keep several projects open, meaning you are not “in” any particular project, but you can only use one single workspace at a time.

There are different kinds of projects. The most commonly used together with Simics are the C/C++ project and the Simics project. The former should be used when performing software development in C or C++ and the latter is intended for using Simics as an emulator

rather than as a debugger. The functionality available using a Simics project is, however, a strict subset of that available in the C/C++ project.

### 2.1.2 Perspectives



Projects are viewed through *perspectives*, basically pre-defined collections of *views*. Views are components on the screen marked by a tab with their name. A perspective is intended for a certain kind of activity such as writing code or debugging and you can change between perspectives at any time.

Thus, the Debug perspective has views for showing the values of variables in the program being debugged, while the Simics perspective has a view showing all of your Simics checkpoints.

Perspectives can be customized; you can move views by dragging their tabs, close them or add views to any perspective using **Window** → **Show View**. **Window** → **Reset Perspective** returns to the original set and arrangement of views.

The most important perspectives are:

- The *Simics perspective* offering all tools needed to launch simulations and manage checkpoints and configurations.
- The *Debug perspective* offering the tools needed for debugging a running Simics session, showing the running CPUs, their current stack traces, a disassembly and register view, breakpoints etc. You usually switch to the Debug perspective after having launched a simulation and switch back to the C/C++ perspective or the Simics perspective after terminating it.
- The *C/C++ perspective* offering tools for developing C code.

You can switch freely between perspectives using the  symbol always available just below the menu bar to the right of the Eclipse window. Immediately to the left of the  symbol, the name of the current perspective is displayed.

## 2.2 Before Launch

You usually start out in the Simics perspective or the C/C++ perspective. These perspectives have a Navigator view (called “Simics Projects” in the Simics perspective and “C/C++ Projects” when in the C/C++ perspective) on the right-hand side of the screen, allowing you to manage the resources of the projects in the current workspace.

There are basically two sorts of Simics-specific resources (files) that can be used to start a simulation: *checkpoints* and *scripts*.

### 2.2.1 Checkpoints

Checkpoints represent a “frozen” simulation. They are saved as several files containing data for internal Simics objects representing devices in the simulation.

When in the Simics or C/C++ perspectives, there is an extra entry in Simics Projects view and the C/C++ Projects view under the project called `Simics checkpoints` listing the checkpoints in the project, irrespective of the directory they are located in.

Note that the files Eclipse displays in the Projects view are not automatically synchronized with the file system, so if you move, create or delete checkpoints outside Eclipse, it will still reflect the old state. Right-click and select `Refresh` or press `F5` to fetch the current status from the file system.

When you save several checkpoints in a simulation, only the differences to the previous checkpoint are stored, to keep file size down. Therefore the new checkpoint is dependent on the previous one; if the previous is deleted, the new checkpoint cannot be started.

The Checkpoint Explorer view, available as a tab next to the C/C++ projects view in the C/C++ perspective, shows a graphical view of all checkpoints in your workspace as well as their dependencies. The Checkpoint Explorer is also available by default in the Simics perspective and can be added to any perspective using **Window** → **Show view** → **Other** → **Simics** → **Checkpoint Explorer**.

To launch a simulation from a checkpoint just double-click it, or select it and press `Enter`. You can also click it with the right mouse button and select **Run As** → **Simics Simulation**. This will look for an existing launch configuration launching the checkpoint and run it, if found. If there is no such launch configuration, a new one will be created with the name of the checkpoint. If the checkpoint is dependent upon another checkpoint and a launch configuration is present for that checkpoint, the settings of that launch configuration will be copied to the new configuration. The launch configuration will then be run.

---

**Note:** Checkpoints can be loaded from the command line using the **read-configuration** command or using the `-c` command line option when starting, e.g. **simics -c mycheckpoint**.

---

A checkpoint may be deleted by clicking it with the right mouse button and selecting “Delete”. This will delete all its component files. If there are other checkpoints that are dependent on the selected checkpoint, you will be asked whether you want to delete them all. You cannot delete a single checkpoint that has dependencies, since it would make the dependent checkpoints stop working.

You can also rename a checkpoint by right-clicking and selecting “Rename”. As well as renaming all component files, this modifies the references in any dependent checkpoints to point to the new name.

---

**Note:** There is no command line command to delete or rename checkpoints automatically.

---


### 2.2.2 Scripts

Simics scripts are text files containing Simics command line commands that can be used to set up a configuration or run specific command sequences. Simics scripts have the extension `.simics`.

You can create a Simics script from within the Eclipse environment using **File** → **New...** (followed by **Other unless you are in the Simics perspective**) → **Simics Script**, which will open a text editor for writing the script code.

If the script is supposed to start a new simulation, you can start it by right-clicking on the script in the “Navigator” or “C/C++ Projects” views. Select **Run as... → Simics Debug Session** from the pop-up menu.

It is assumed that scripts run this way set up a configuration, since the Eclipse environment cannot work without a configuration being loaded. You will get an error message if a configuration has not been set up when the script terminates.

To run a script not setting up a configuration, launch a simulation and, in the Debug view, click the run script button  and select the script you want to run.

---

**Note:** On the command line, use the **run-command-file** command to run a script when Simics has started. You can start Simics with the script to run as a command line parameter, e.g. **simics myscript.simics**. The scripts can perform any command; it is not assumed that they set up a configuration.

---

## 2.3 Running

Once you have launched the simulation, Simics offers a number of possibilities that are not available when running a program natively.

Most of these are available from the Simics perspective, but you might need to switch to the Debug perspective for more advanced features and those related specifically to debugging.

### 2.3.1 The Simics Command Line Console


When a simulation is launched, the console view (available in all perspectives) opens, showing Simics status messages and allowing you to enter command line commands. It can be used much like the command line version of Simics. Note that the pointer needs to be at the end of the console for you to be able to enter a command.

### 2.3.2 Running Several Simulations

You can launch several simulations concurrently from the Eclipse environment. In the Debug view of the Debug perspective, the running simulations are listed and you are able to switch between them. From the Simics perspective, you can switch between different simulations with the Simics Control view or by clicking the downward-pointing arrow in the console toolbar.

### 2.3.3 Reverse Execution and the Timeline

Simics offers *reverse execution* functionality. It uniquely allows you reverse the time of the simulation and run it backwards on most targets.

Reverse execution is enabled by default when running the simulation. Disabling reverse execution will make the simulation run slightly faster and can be done in the launch configuration or by clicking the reverse execution icon  in the Debug view toolbar available in the Debug perspective.

---

**Note:** Enabling and disabling reverse execution by command line commands is done by setting a bookmark (**bookmark label**) and removing all bookmarks (**delete-bookmark -all**), respectively.

---

The timeline view, available in the Simics and debug perspectives, displays a log of events in your current simulation session and is especially useful in conjunction with reverse execution, since it enables you to conveniently jump to specific points in time. In the timeline, events such as breakpoints or watchpoints being triggered, stepping and suspending the simulation, are displayed.

A special entry is the one representing the current time. Note that it need not be the last entry if you have been running the simulation backwards.

Clicking on the `Steps` header switches between displaying the target time and showing the number of steps run on each CPU.


---

**Note:** Though the time is identical on all CPUs, they need not have run the same number of steps since they are not actually run in parallel, but in a round-robin fashion and some CPUs have thus run farther than the other.

---


To hide some types of events, e.g. because they are too frequent, click the downward-pointing arrow in the timeline view, select `Filters...` and uncheck the types you are not interested in. This setting is saved in your workspace and persists if restarting Eclipse.

When turning on reverse execution, the events `Last known time` and `First known time` are added at the current time. Running the simulation will make `Last known time` shadow the current time. These entries indicate the time range you can run reverse execution within, i.e. the farthest you have ever run the simulation and the moment you turned on reverse execution, respectively.



You can move the simulation to the state it was in at the time of an event by double-clicking on it or right-clicking and selecting `Skip to`. This can be a slow operation; to speed it up, set a bookmark when the event occurs using the add bookmark button .

Adding a bookmark also enables you to name an arbitrary point in time. Since bookmarks are specific to a certain debugging session, which cannot be saved, they are cleared when terminating the session.

When using a bookmark to skip to a future time, the simulation will run at its normal speed. The simulation does not stop on breakpoints when skipping.

When running the simulation backwards and then moving forwards again, key strokes and network traffic that it received the first time it was run, will be replayed until you reach the `last known time` event. If you wish to bypass this and enable new input, press the discard future data button , clearing recorded input for the time following the current time. `Last known time` will move to the current time and all future events are cleared, since the simulated machine might follow a different route when you run it with different input.

### 2.3.4 Stopping

You can suspend a simulation using the pause button  or terminate it using the stop button . Both are available in the console view and in the Debug view in the Debug perspective.


Note that when running a multi-CPU machine or when using multiple debugging contexts, the current CPU or context might not be automatically selected and you might need to expand it in the Debug view to see the stack trace.

---

**Note:** Suspending using the command line is done using the command **stop**. Terminate a simulation using **quit**. This exits Simics when using the command line version. In the Eclipse environment, it terminates the current simulation.

---

### 2.3.5 Saving Checkpoints

The state of the simulation can be saved as a checkpoint using the write checkpoint button  in the Debug view in the Debug perspective or by using **Run** → **Write checkpoint**. In the dialog that opens, you can enter a name to save the checkpoint in the project directory or specify a subdirectory (e.g. *directory/checkpoint\_name* ).

---

**Note:** Save checkpoints using the command line interface with the command **write-configuration**.

---

## 2.4 Additional Views

Simics also provides some additional views which can be used to get additional details about the system. These views can be used to complement both the debugger and the configuration browser.

### 2.4.1 Target Info View

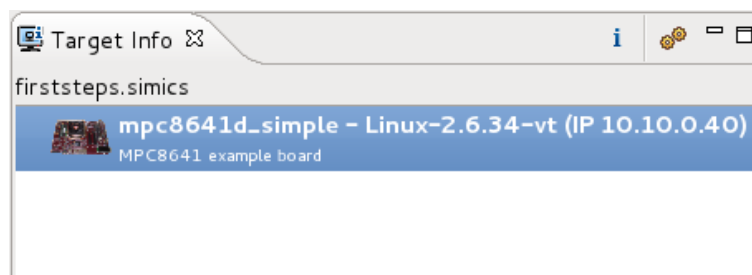


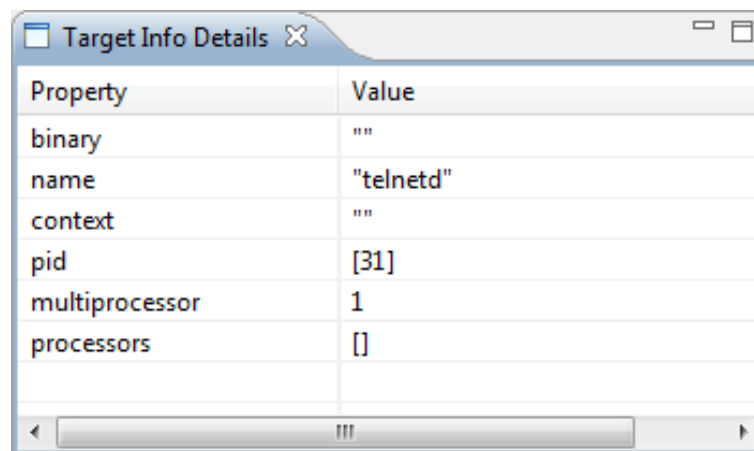
Figure 2.1: The Target Info view

The Target Info view provides an overview of the machines in the current Simics session. For each machine the view shows a short description of the machine, its name and usually what software it runs. The view also shows a graphical image for the machine and some high level properties, like the kind of system, its processors, and memory. The exact properties shown depends on the machine.

If you have Simics Analyzer and the target system has been configured with OS awareness, the view can also show information about the software running on the target as a tree of software nodes for each target machine with OS awareness configured. A node can for example be a hypervisor, kernel, process, or thread. The structure of this information depends on the target system, but there are some part which are common among all software nodes. By default this information is not shown. To show it press the Enable/disable display of software button in the view's toolbar. This can lower the performance of the simulation.

Each software node is shown as two lines. The first lines gives the name of the software node, as provided by the OS awareness system. If OS awareness provides additional identifiers used by the target system to identify the node, these are shown after the name, in parenthesis.

The second line of a software node shows which processors the node is currently active on. If the node is not active on any processors at the moment, this line will be empty.



Property	Value
binary	""
name	"telnetd"
context	""
pid	[31]
multiprocessor	1
processors	[]

Figure 2.2: The Target Info Details view

You can also view more details about an item in the Target Info view in the Target Info Details view. Open the details view by double-clicking an item, or by clicking the Show Details button in the view's toolbar.

The process info is updated when the simulation pauses.

You can read more about Simics Analyzer and OS awareness in the *Analyzer User's Guide*.

### 2.4.2 Graphical Timeline View

The Graphical Timeline view shows the software and processors in the system and what software has been running where and when. It requires Simics Analyzer with OS awareness configured to function properly. Without this it will not show any interesting data.

The view shows the structure of the system at the left. The rest of the view shows the data recorded for the parts of the system. The top of the view contains a ruler, showing the currently visible time range. The time is measured since the start of the recording.

To zoom in on a particular time range select it by clicking and dragging across the time range you are interested in and press the Zoom in on Selection button in the view's toolbar.

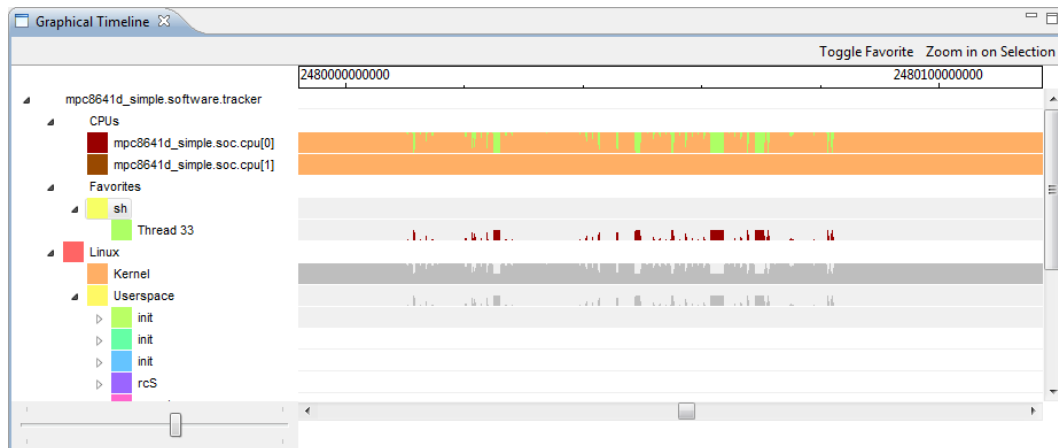


Figure 2.3: The Graphical Timeline view

You can also use the slider in the lower left of the view too zoom in and out, and the bottom scroll bar to scroll to different points in time.

Since the view keeps track of all software which has run during the time it has been recording, the tree with software can get very long. To only focus on a subset of the software you can make them favorites. To make a software item a favorite select it and press the Toggle Favorite button in the view's toolbar. To remove a favorite select it and press the same button again.

### 2.4.3 Target Memory View

The Target Memory view shows the contents of the memories in the current Simics session. It complements the standard memory view provided in Eclipse. While the standard view shows the memory of the active context, this view shows the current contents of all the memories in the system.

The memory contents is presented in two columns: a hexadecimal column, which shows the contents of the memory in hexadecimal notation, and an ASCII column, which shows the contents in ASCII notation. Values which cannot be represented in ASCII are displayed as full stop (.) characters.

If a region of the memory does not contain valid data the view will display special markers instead:

\*

The address is outside memory.

?

The view is unable to read the value at the address.

-

The virtual address could not be translated into a physical address.



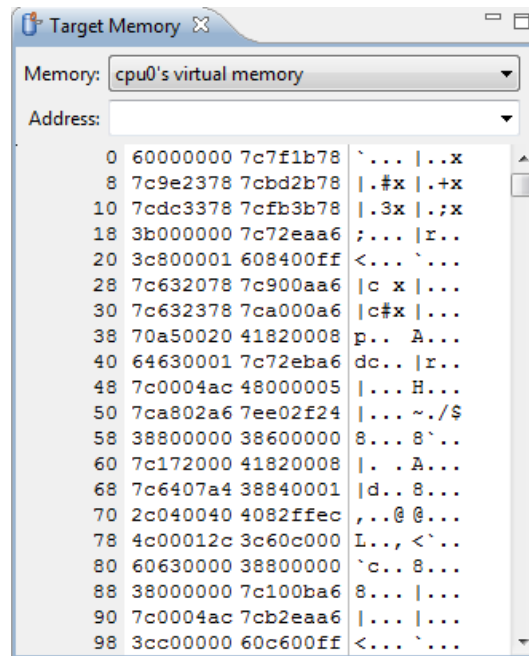


Figure 2.4: The Target Memory view

You can select what memory to view in the drop down at the top of the view, and jump to a particular address by typing it into the address field and pressing return. To write an address in hexadecimal notation, prefix it with 0x, for example write 0x100 to jump to address 256. The view also supports basic keyboard navigation to move between addresses.

You can also edit the content of the memory by navigating to the correct address, either with the mouse or the keyboard and typing the new values. The hexadecimal column accepts hexadecimal values and the ASCII column accepts ASCII values. You can jump between the columns with the tab key, or the mouse.

#### 2.4.4 Memory Spaces View

The Memory Spaces view shows the memory spaces in the current Simics session and their structure. The left part of the view is a tree which shows all the memory spaces in the system and how they are connected to each other and the right part is a table which shows the devices mapped into the memory space. Select a memory space in the tree to the left to inspect it in the table to the right.


In the tree control with the memory spaces, the left part of the tree, the roots of the trees are 1) the memory spaces which are physical memories of processors in the system, 2) memory spaces not mapped into any other memory spaces, 3) and finally, just to make sure all memory spaces are shown, all memory spaces not reachable from any of the first two sets. There is no division between the three categories of roots, but they are sorted. First all the physical memories are shown, then unmapped memory spaces and finally the rest. Each memory space is present as a child of every memory space it is mapped into. If you

Base	Device	Offset	Length	Target
0x0	sdram	0x0	0x8000000	
0xff660000	hfs0	0x0	0x10	
0x140000200	uart0	0x0	0x8	
0x140000300	uart1	0x0	0x8	
0x140000400	iic0	0x0	0x2	
0x140000402	iic0	0x2	0x2	
0x140000404	iic0	0x4	0x1	
0x140000405	iic0	0x5	0x1	
0x140000406	iic0	0x6	0x1	
0x140000407	iic0	0x7	0x1	

Figure 2.5: The Memory Spaces view

have cycles between the memory spaces the cycle will be broken the second time the same memory space is present in a branch from the root. This node in the tree will not have any children, but you can still see all the devices mapped into it in the right part of the window.

The table with devices mapped into the memory space lists all the banks mapped into the memory space. A bank in this case can be either a device, a port of a device or a function of a device. Each line contains the start address of the bank, the name of the bank, the size of the window, the offset in the bank where this window is placed and finally, if the mapping is a translation mapping, the target of the translation.

You can open the current memory space at the address of the selected line in the map table by pressing the  icon in the Memory Spaces view's toolbar.

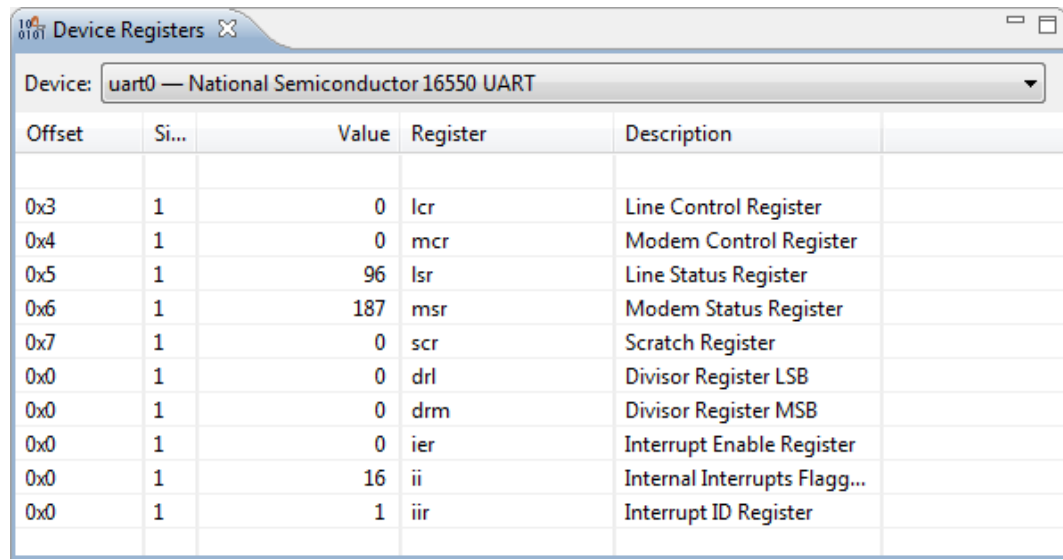
### 2.4.5 Device Registers View

The Device Registers View lists all DML devices in the target system which have device registers made visible. At the top of the view, a drop down menu is available which can be used to select a device. The table below it will then list names, descriptions, values and other information such as size and offset of each register in the device.

The values of a device registers can also be edited from the Device Registers View. To edit a device register, select the register value and press the “Enter” key to start editing the value. Once done, press “Enter” again to conform the change and write the new value to the model, or press “Escape” to cancel.

## 2.5 Diagnosing problems

If you experience problems using Simics, you might be helped by the detailed logging information available in the log file called **simics\_eclipse.log** and located in your workspace directory. It contains a highly technical and detailed account of the internal events during your session. Please attach the log files when filing a problem report.



Offset	Si...	Value	Register	Description
0x3	1	0	lcr	Line Control Register
0x4	1	0	mcr	Modem Control Register
0x5	1	96	lsr	Line Status Register
0x6	1	187	msr	Modem Status Register
0x7	1	0	scr	Scratch Register
0x0	1	0	drl	Divisor Register LSB
0x0	1	0	drm	Divisor Register MSB
0x0	1	0	ier	Interrupt Enable Register
0x0	1	16	ii	Internal Interrupts Flagg...
0x0	1	1	iir	Interrupt ID Register

Figure 2.6: The Device Registers view

The log file is only available in the Eclipse environment, not using the command line version of Simics.

## Chapter 3

# Simics Eclipse DML Editor

Simics Eclipse provide an integrated editor and build tools for the *DML* language, used for building device models for Simics. This editor provides a number of features intended to simplify device development such as a built-in parser that instantly finds and indicates syntax errors. These features are presented in this chapter.

---

**Note:** The DML editor is distributed as the separate feature *Simics DML Editor for Eclipse*. Please ensure that you have this feature installed before following the instructions in this section. For more information, refer to the chapter *Installing the Simics Eclipse Tools* in the *Installation Guide*.

Also note that the *Model Builder* package (1010) and associated license is required in order to use the DML Editor. Please refer to the *Installation Guide* for instructions on how to obtain and install this package.

---

### 3.1 Enable Emacs Key-Bindings

If you are an Emacs user, the first thing you probably want to do is enable Emacs key bindings in Eclipse. To do this, select **Window** → **Preferences...** and set the key bindings in the section *General / Keys* as shown in Figure 3.1.

You also need to specify the location of the Simics executable to launch in the preferences, if you have not already done so. See figure 3.2.

Next, open the Simics perspective as shown in Figure 3.3.

### 3.2 Creating a new Simics Project

Create a new Simics project by selecting **File** → **New...** → **Simics Project**. In the dialog, specify a name for the project.

This will create a project directory as a sub directory of the workspace and will configure this directory as a Simics workspace and add some files to the project, see Figure 3.6.

### 3.2. Creating a new Simics Project

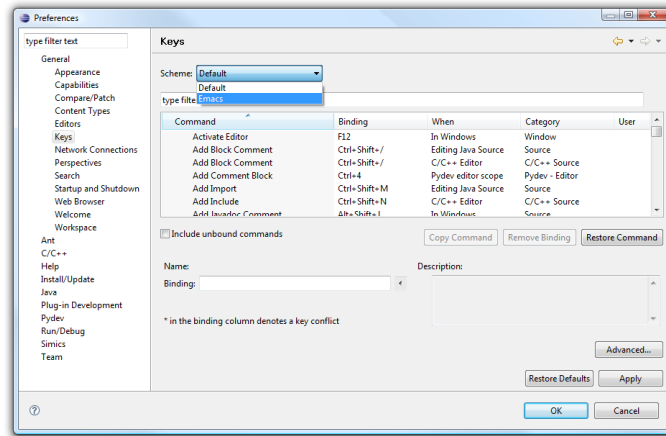


Figure 3.1: Selecting Emacs key-bindings

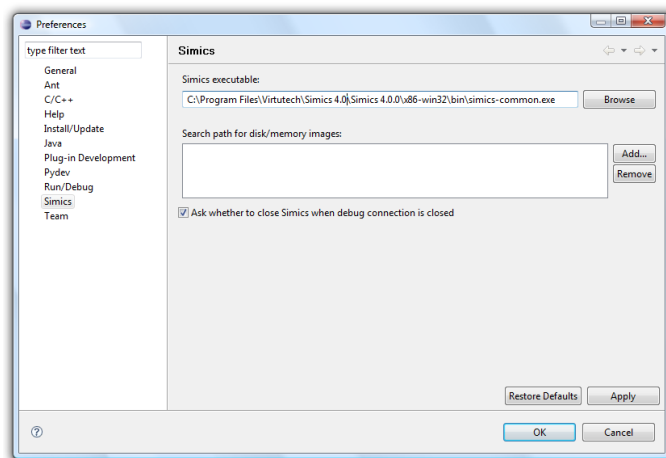


Figure 3.2: Selecting Simics executable

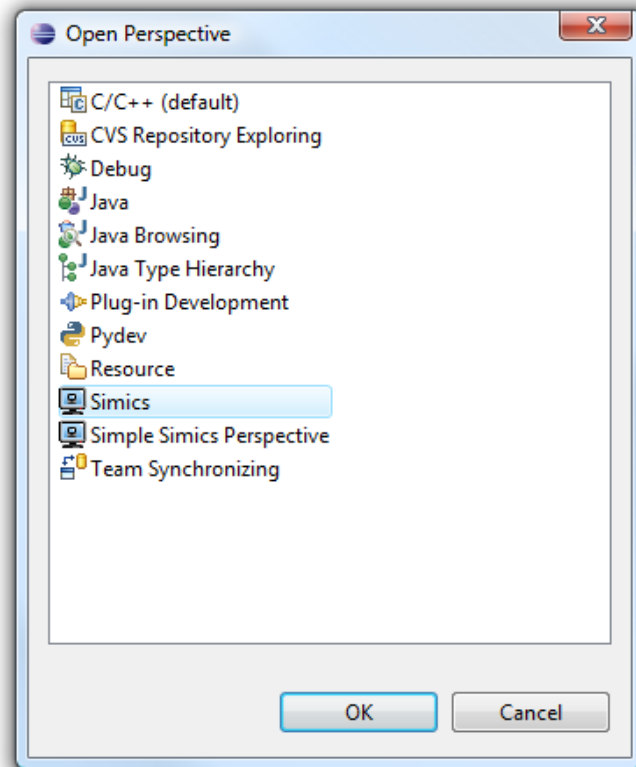


Figure 3.3: Selecting Simics perspective

### 3.2. Creating a new Simics Project

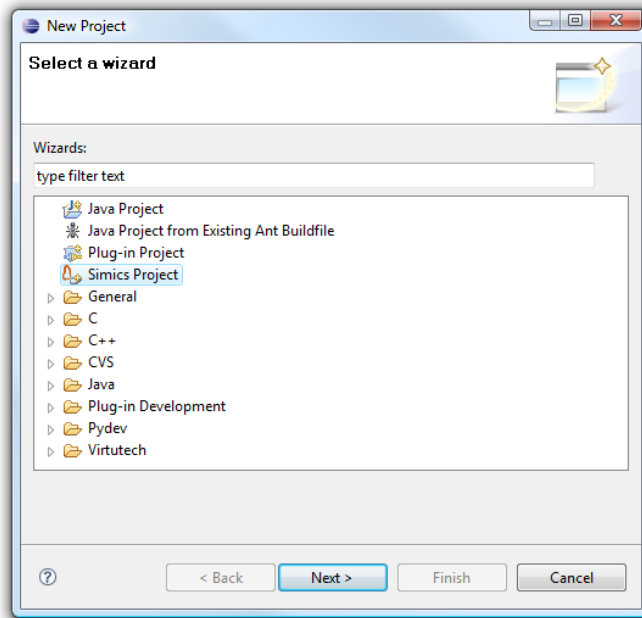


Figure 3.4: Creating a new Simics project

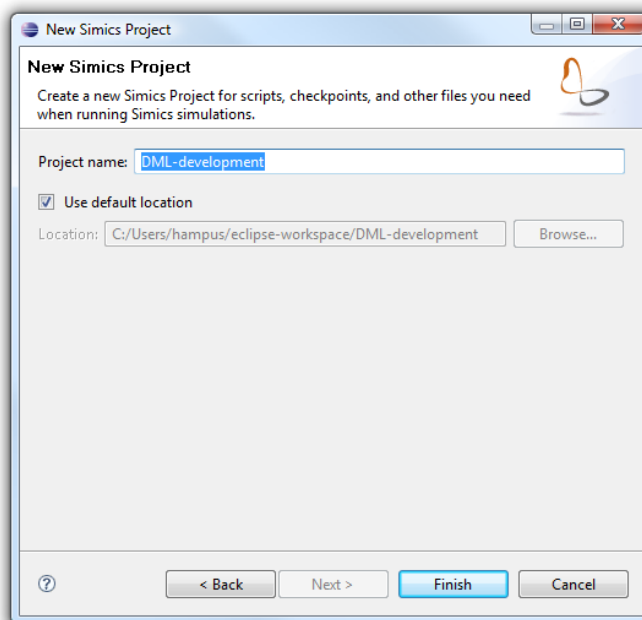


Figure 3.5: New Simics Project Wizard

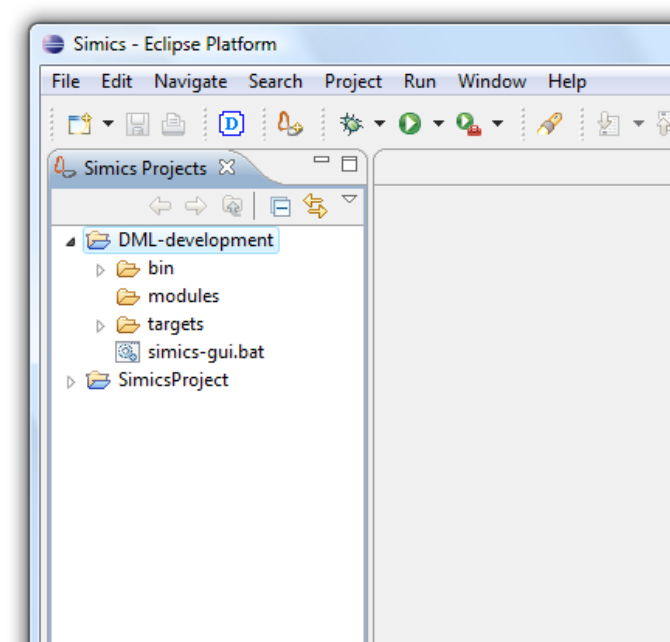


Figure 3.6: The new Project is configured as a Simics Workspace

## 3.3 Connect to an Existing Simics Workspace

If you have an existing Simics workspace, you can create a Simics project which references it. Start like when creating a new Simics project by selecting **File** → **New...** → **Simics Project**. In the dialog, specify a name for the project, uncheck the "Use default location" checkbox and fill in the location of your existing Simics workspace in the "Location:" field. You can also browse to the workspace by clicking the "Browse..." button. See Figure 3.7.

This will create a project in the Eclipse workspace, which references the existing Simics workspace, see Figure 3.8.

## 3.4 Creating a new DML Device

Next, select **File** → **New...** → **DML Model**, as shown in Figure 3.9, to add a new DML model to the project.

Specify a name for the device and select the project to add it to.

You can now edit and build the device in Eclipse.

### 3.4.1 Using the Outline View

The Eclipse DML editor parses the source code and creates a tree of the elements in the source. This can be useful for viewing or navigating the source. If the outline view is not shown, select **Window** → **Show View...** → **Outline** to open it. Figure 3.11 shows the outline view, displaying the outline of a sample DML source file.



### 3.4. Creating a new DML Device

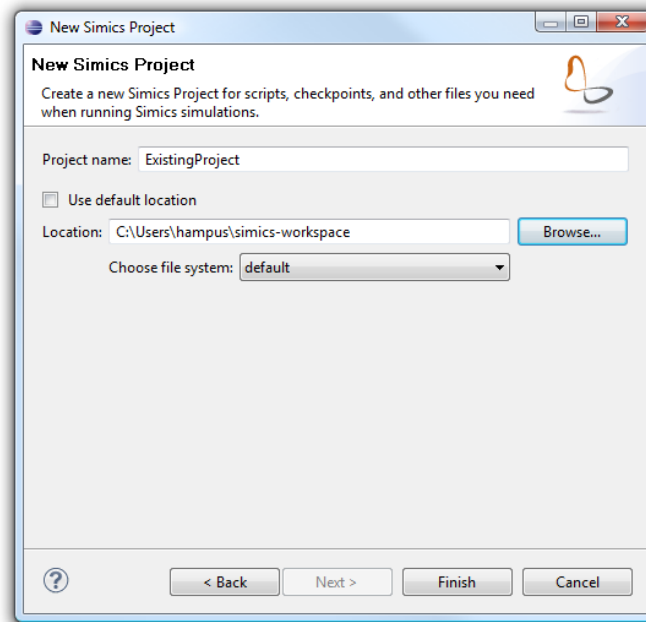


Figure 3.7: New Simics Project Wizard

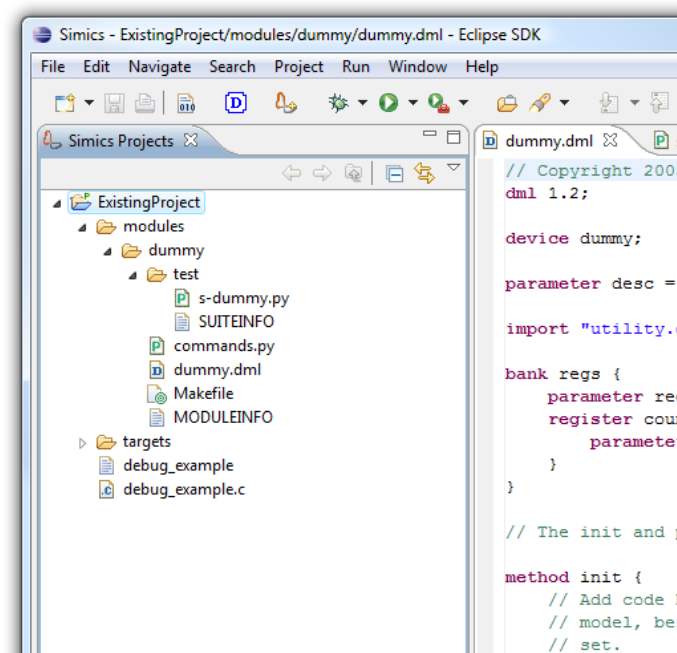


Figure 3.8: The new project refers to the existing Simics workspace

### 3.4. Creating a new DML Device

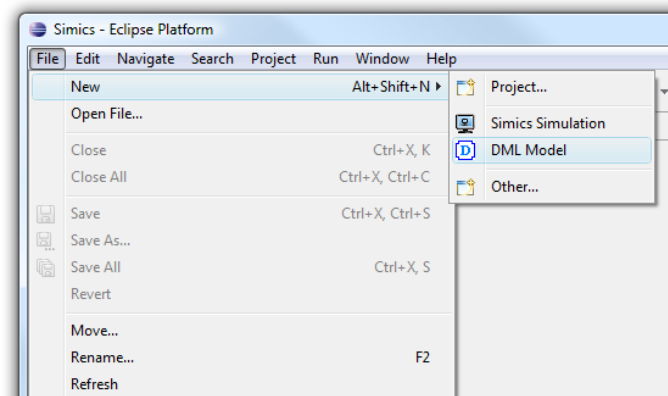


Figure 3.9: Creating a new DML Model

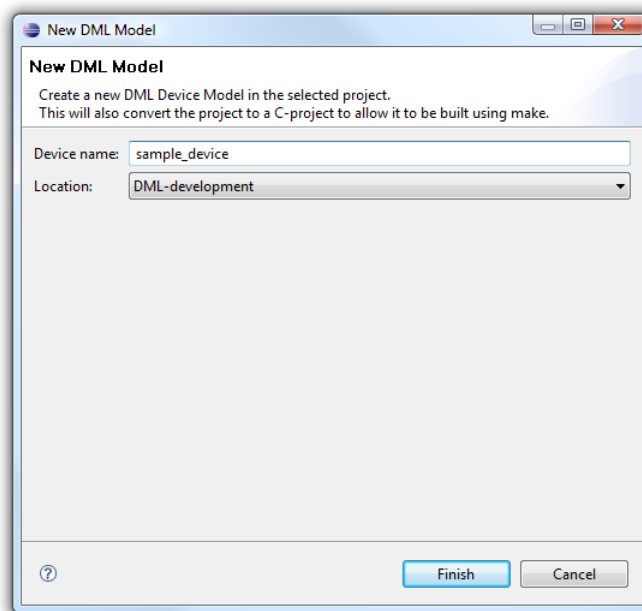


Figure 3.10: Creating a new DML Model 2

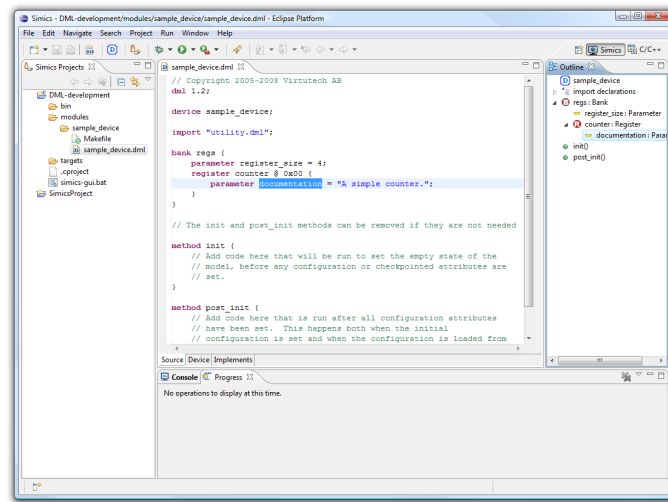


Figure 3.11: The Outline view

### 3.4.2 Using the Problems View

Eclipse will also use the parser to find syntax errors in the source. These will be displayed in the margin next to the source code and also in the problems view. To show the problems view, select **Window** → **Show View...** → **Problems**.

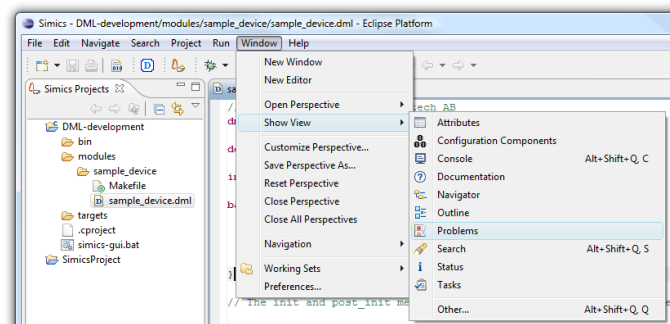


Figure 3.12: Opening the Problems view

The parser will indicate any parse errors in the source file using red markers in the margin and squiggly lines, as shown in Figure 3.14.

## 3.5 Indexing and Searching

The DML editor also constructs an index with all defined variables/types and references to them. Using this index the editor is able to indicate when undefined variables or types are used. In Figure 3.15 the type `BOOLEAN` is defined, but the type `BOOLEEN` is not.

### 3.5. Indexing and Searching

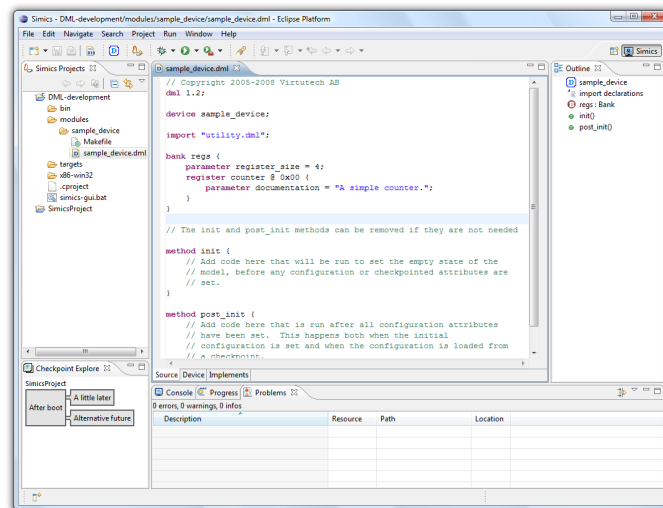


Figure 3.13: Opening the Problems view

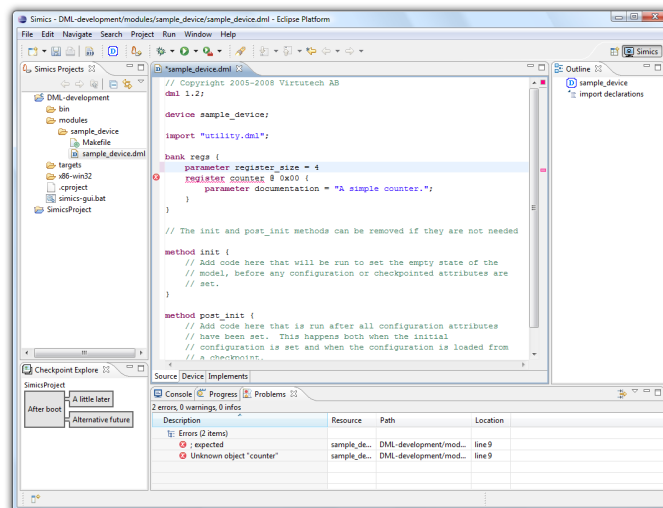


Figure 3.14: Syntax errors are displayed in the source editor and in the problems view

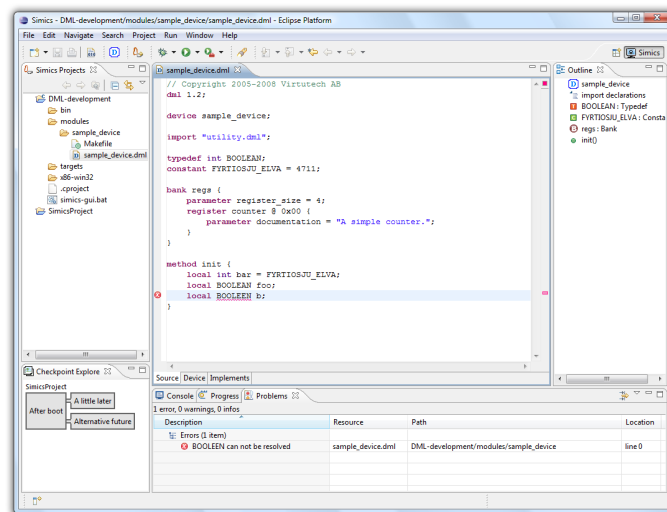


Figure 3.15: The undefined type BOOLEAN is marked

The indexer will keep track of nested scopes and mix-ins (via the “is” keyword). In Figure 3.16, the variable `tp` inherited from the template `t` can be used in the bank `b`.

The index can also be used search for the declaration of a variable, just left-click on a variable reference and select “Open Declaration”, this will select the place where the variable is declared. See Figure 3.17.

The indexing and search feature also works in imported files.

## 3.6 Building the device

To build the device, select “Build Project” from the “Project” menu.

This will launch an external make process, and build the “all” target. Output from the make process appears in the build console view, as shown in Figure 3.20.

### 3.6. Building the device

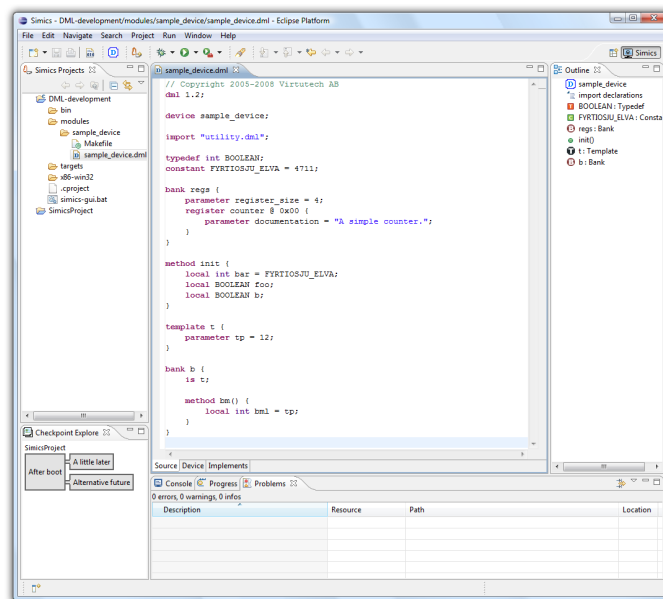


Figure 3.16: The variable `tp` inherited from the template `t` can be used in the bank `b`

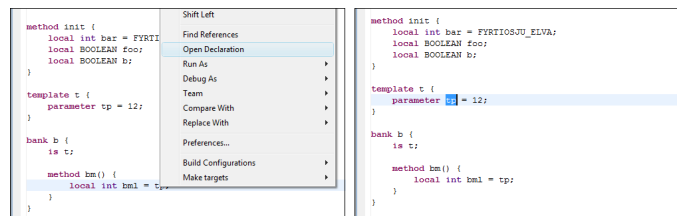


Figure 3.17: Looking up the declaration of the parameter `tp`

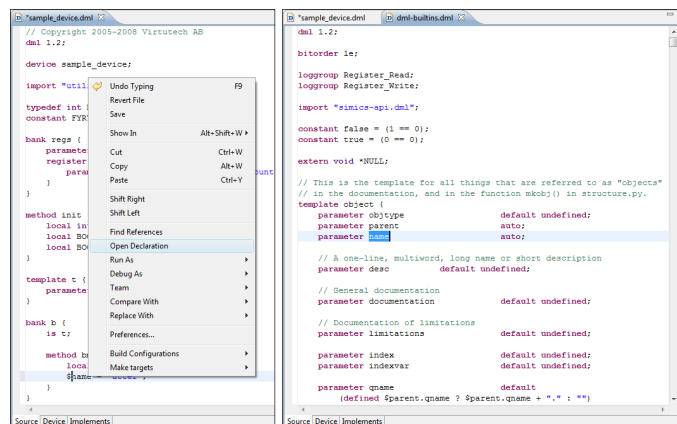


Figure 3.18: Looking up the declaration of an imported parameter

### 3.6. Building the device

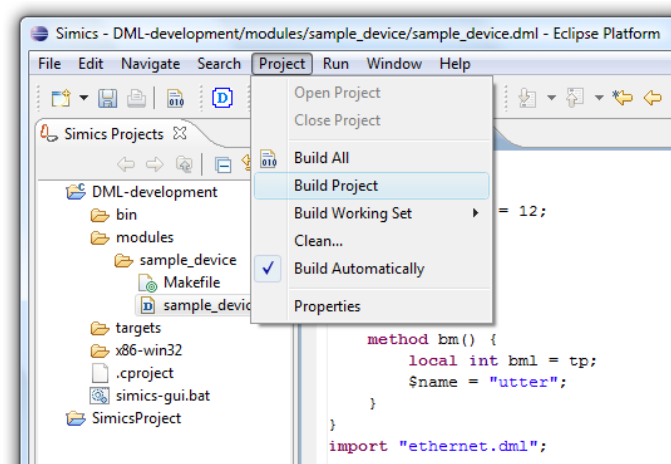


Figure 3.19: Building the device

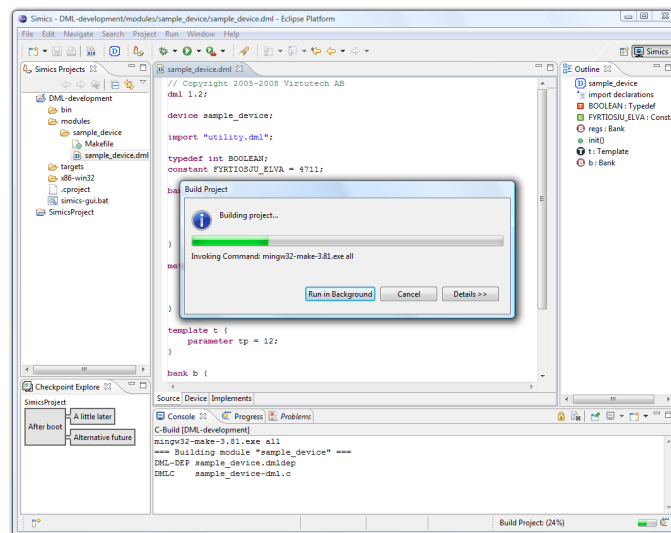


Figure 3.20: Building the device 2

# Index

## C

- C/C++ perspective, [18](#)
- C/C++ project, [17](#)
- checkpoint, [5](#)
- Checkpoints, [18](#)
- command line interface, [17](#)
- components, [5](#)
- configuration, [5](#)
- configuration objects, [6](#)
- console, [6](#)

## D

- Debug perspective, [18](#), [20](#)

## E

- Eclipse environment, [17](#)

## H

- host machine, [5](#)

## L

- launch configuration, [19](#)

## P

- perspectives, [6](#), [18](#)
- projects, [6](#), [17](#)

## R

- reverse execution, [20](#)

## S

- scripts, [19](#)
- Simics perspective, [18](#), [20](#)
- Simics project, [17](#)

## T

- target architectures, [5](#)
- target machines, [5](#)

- timeline view, [21](#)

## V

- views, [6](#), [18](#)

## W

- workspace, [6](#), [17](#)