



SSDPlayer Visualization Platform

Version 1.3.1 User's Guide

Gala Yadgar, Roman Shor, Eitan Yaakobi, and Assaf Schuster
September 2021

Computer Science Department
Technion – Israel Institute of Technology
Haifa, Israel

Preface

SSDPlayer is an open source graphical tool for visualizing data layout and movement on storage devices. It is designed to give a better understanding of how data gets from one place to another and why. Users define the device they wish to examine, and the workload on which it operates. SSDPlayer then illustrates the device state after each step in the workload. This illustration forms a “video” of the data movements that take place during execution.

This guide explains how to download and execute SSDPlayer, how to define the devices and their features, and how to collect and format the input workload for SSDPlayer. It also provides details on the device-management options currently supported by SSDPlayer.

Document History

<i>Version</i>	<i>Date</i>	<i>Description</i>	<i>Development</i>	<i>Authors</i>
1.0	19 Jun. 2015	Initial version	Roman Shor	Gala Yadgar
1.1	Feb. 2016	Addition of RAID support	Or Mauda Roman Shor	Or Mauda Gala Yadgar
1.2	Jun. 2016	Addition of Zoom levels and breakpoints	Dolev Hadar Roe Matza Roman Shor	Dolev Hadar Roe Matza Gala Yadgar
1.2.0	Feb 2017	Addition of Info screen and error messages, bug fixes	Roe Matza Roman Shor	Roe Matza Gala Yadgar
1.2.1	May 2017	Addition of wear-awareness block allocation, bug fixes	Roman Shor	Roman Shor Gala Yadgar
1.3.0	April 2021	Addition of CLI, speedup, absolute GC threshold, hot/cold write amplification info	Lior Zelikman	Lior Zelikman Gala Yadgar
1.3.1	Sep. 2021	Ignore empty input lines and extra fields, handle multiple-page requests	Lior Zelikman	Gala Yadgar

Table of Contents

- Preface.....1
- Document History.....2
- Introduction.....7
 - 1.1. What Is SSDPlayer?.....7
 - 1.2. What SSDPlayer Is Not.....7
- 2. Flash Terms and Concepts.....8
 - 2.1. Flash Elements: Cells, Pages, and Blocks.....8
 - 2.2. Flash Operations: Read, Write, and Erase.....8
 - 2.3. Flash Translation Layer (FTL).....8
 - 2.4. Garbage Collection.....9
- 3. SSDPlayer Components..... 10
 - 3.1. SSD..... 10
 - 3.1.1. Physical Device..... 10
 - 3.1.2. Manager..... 10
 - 3.2. Input Workload..... 10
 - 3.2.1. Workload Generators..... 11
 - 3.2.2. Input Trace Files..... 11
- 4. Downloading and Installing SSDPlayer..... 15
 - 4.1. System Requirements..... 15
 - 4.2. Downloading SSDPlayer..... 15
 - 4.3. Installing SSDPlayer..... 15
- 5. Running SSDPlayer in default (GUI) mode..... 16
 - 5.1. Start SSDPlayer..... 16
 - 5.2. Choose Manager..... 16
 - 5.3. Choose Input Workload..... 17
 - 5.3.1. Choose Workload Generator..... 17
 - 5.3.2. Choose Input Trace File..... 18
 - 5.4. Start the Simulation..... 20
 - 5.4.1. Play..... 20
 - 5.4.2. Next Frame..... 20
 - 5.5. Stopping the Simulation..... 21
 - 5.5.1. Pause..... 21
 - 5.5.2. Stop..... 21
 - 5.6. Highlight Stripes (RAID Managers)..... 22

5.6.1.	Open the Stripes Info Window	22
5.6.2.	Show another stripe	22
5.6.2.1.	Specify a stripe to highlight	22
5.6.2.1.1.	Specify a physical page	23
5.6.2.1.2.	Specify a logical page.....	23
5.6.2.1.3.	Specify a parity page.....	23
5.6.3.	Remove stripe.....	24
5.7.	Manage breakpoints.....	24
5.7.1.	Defining Breakpoints	24
5.7.1.1.	Add a new breakpoint	25
5.7.1.2.	Enable/Disable breakpoint	26
5.7.1.3.	Edit an existing breakpoint	27
5.7.1.4.	Remove an existing breakpoint.....	27
5.7.2.	Breakpoint Hits	28
5.8.	Zoom in and out	29
5.8.1.	Choosing a zoom level	29
5.8.2.	<i>Detailed</i> zoom level	30
5.8.3.	<i>Pages</i> zoom level	30
5.8.4.	<i>Blocks</i> zoom level.....	30
5.8.4.1.	Valid count.....	30
5.8.4.2.	Erase count	31
5.8.4.3.	Average temperature	31
5.8.4.4.	Average write level.....	31
5.8.4.5.	RAID parity.....	31
5.8.5.	<i>Small Blocks</i> zoom level.....	32
5.9.	View Information.....	32
5.9.1.	Open the information screen	32
5.9.2.	Saving the simulation state	33
5.10.	Speedup with sampling rate.....	33
5.10.1.	Open the sampling rate screen	33
5.10.2.	Set the sampling rate.....	33
6.	Running SSDPlayer in CLI mode.....	34
6.1.	Command line parameters.....	34
6.1.1.	-C <config file name>	34
6.1.2.	-M <manager name>	34

6.1.3.	-F <trace file name>.....	34
6.1.4.	-G (use workload generator)	34
6.1.4.1.	Required parameters.....	34
6.1.4.2.	Optional parameters	34
6.1.5.	-O <output file name>	34
6.1.6.	-help.....	35
6.2.	Examples.....	35
6.3.	Error messages	35
7.	Understanding the SSDPlayer Display	36
7.1.	Physical Device Display.....	36
7.2.	Histograms.....	37
7.2.1.	Write Amplification	37
7.2.2.	Writes Per Erase	37
7.2.3.	Valid Histogram	38
7.2.4.	HotCold Write Amplification (HotCold Manager)	38
7.2.5.	Partition Distribution (HotCold Manager).....	39
7.2.6.	Write Level Distribution (Reusable Manager).....	39
7.2.7.	Block State Distribution (Reusable Manager)	39
7.2.8.	Valid 1 and Valid 2 Histograms (Reusable Manager)	40
7.2.9.	Parity Update Overhead Histogram (RAID Managers).....	40
7.3.	Stripe Highlighting (RAID Managers).....	40
7.4.	Information Screen.....	41
8.	Editing the Configuration File.....	43
8.1.	Physical Parameters	43
8.2.	Visual Parameters.....	44
	Manager Parameters.....	46
8.2.1.	Defining Colors	46
8.2.2.	Parameters In Common for All Managers.....	46
8.2.3.	Greedy Manager.....	47
8.2.4.	HotCold Manager	47
	Reusable Manager.....	48
8.2.5.	HotCold-Reusable Manager	48
8.2.6.	Reusable visualization Manager	48
8.2.7.	Parameters In Common for All RAID Managers	49
7.3.9	RAID Visualization Manager	49

9.	Editing the breakpoints configuration file.....	50
10.	Supported Managers.....	51
10.1.	Greedy Manager.....	51
10.2.	HotCold Manager.....	51
10.3.	Reusable Manager.....	52
10.4.	HotCold-Reusable Manager.....	53
10.5.	RAID Managers.....	54
10.5.1.	RAID 1 Manager.....	54
10.5.2.	RAID 5 Manager.....	54
10.5.2.	RAID 6 Manager.....	54
10.6.	Reusable visualization Manager.....	55
10.7.	RAID Visualization Manager.....	55
A.	FAQ.....	56
a.	Q: I double click on SSDPlayer.jar but nothing happens. What's wrong?.....	56
b.	Q: I run the HotCold manager with the Zipf workload, but all the pages are red. Why?.....	56
c.	Q: How did you create the online demos?.....	56
d.	Q: I have a cool idea how to improve SSDPlayer, can you do it?.....	56
e.	Q: Where can I get the source code and Programmer's Guide?.....	56
f.	Q: I found a bug, how do I report it?.....	56
B.	Copyright Notice.....	57
C.	Citation.....	57

Introduction

1.1. What Is SSDPlayer?

SSDPlayer is an open source graphical tool for visualizing data layout and movement on flash devices. It is designed to give a better understanding of how data gets from one place to another and why.

SSDPlayer supports two modes of operation. In *simulation* mode, it simulates the chosen device on a raw I/O [trace](#) or on a synthetic workload generated by the built in [workload generator](#), illustrating the SSD state at each step. This illustration forms a “video” of the data movements that take place during execution. This [mode](#) is useful for testing and analyzing various features without, or before, implementing them in a full-scale simulator or hardware platform.

In *visualization* mode, SSDPlayer illustrates operations that were performed on an upstream simulator or device. The input in this mode is an [output trace](#) generated by a simulator, hardware evaluation platform, or a host level FTL, describing the basic operations that were performed on the flash device—writing a logical page to a physical location, changing block state, etc. This mode is useful for illustrating processes that occur in complex research and production systems, without porting their entire set of features into SSDPlayer.

The SSDPlayer display is organized into chips, planes, blocks and pages, as specified by the user at startup. Colors and textures are used to represent page and block properties, such as data ‘temperature’ or valid page count. A page’s properties and state determine its fill color, texture, and frame color. A block’s properties determine its background and frame colors. Users can control all the display parameters by editing the configuration file before starting SSDPlayer.

1.2. What SSDPlayer Is Not

SSDPlayer is not a performance simulator. You can use it to see how data moves, but not how much time it takes. Increased, unexpected and inefficient data movement will harm your device’s performance – SSDPlayer will help you detect such movements and understand what causes them. SSDPlayer does not (currently) perform latency or throughput calculations. The speed of the simulation depends on the strength of the machine you use to run SSDPlayer, not on the simulated device.

SSDPlayer is not a device analyzer. You cannot use SSDPlayer to see what’s going on inside the commercial device plugged into your machine. Data movement in SSDs is controlled by internal mapping and maintenance mechanisms. Currently, manufacturers do not expose the full details of the mechanisms they use, so they cannot be visualized by an external tool.

2. Flash Terms and Concepts

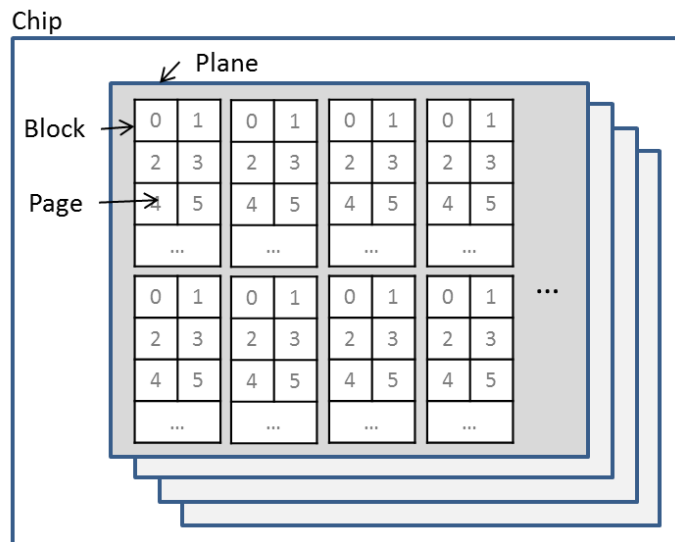
2.1. Flash Elements: Cells, Pages, and Blocks

A flash memory chip is built from floating-gate cells that can be programmed to store a single bit, two bits, or three bits, according to the flash technology used. Cells are organized into pages, which are equivalent to sectors in hard disk drives – this is the smallest unit that can be read or written. Typical page sizes are between 2KB and 16KB.

Note: SSDPlayer displays entire pages, regardless of the technology used to program their cells.

Pages are grouped into blocks, which are the unit of erasure (explained below). Blocks typically contain 64 to 384 pages.

Within the chip, blocks are divided into two or more planes, which are managed and accessed independently. Planes within a chip can operate concurrently, performing independent operations such as Read, Write, and Erase.



2.2. Flash Operations: Read, Write, and Erase

A *write* operation applies voltage to the cells of a page. This modifies their state to store either 0 or 1. The write operation is sometimes referred to as "program". A *read* operation probes the cells of a page to test whether their value is 0 or 1.

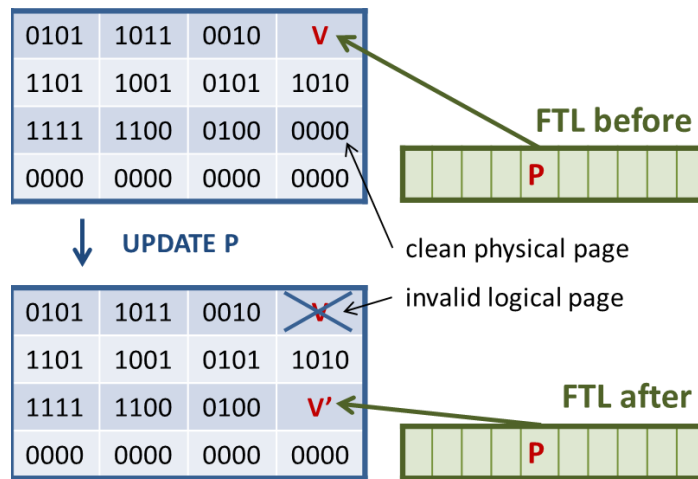
In order to modify the content of a page, an *erase* operation must be applied to the entire block containing that page. An erase operation resets all the pages in the block to their initial state, and the data written on them is lost. Erases are typically an order of magnitude slower than the reads and writes.

2.3. Flash Translation Layer (FTL)

To avoid the delay of an erase operation every time the content of a page is modified, the firmware of the SSD includes a mapping between logical pages to physical ones. The *flash translation layer (FTL)* is the part of the firmware responsible for this mapping.

The basic mapping works as follows:

- The FTL maintains a page mapping table. Each entry in the table corresponds to a logical page. The value of that entry is a pointer to the physical page that stores the data of this logical page.
- The operating system sends an I/O write request to logical page p . This is equivalent to a logical block address (LBA) of a hard disk drive.
 - If this is the first time p is written, then the data is programmed onto a clean physical page.
 - If p is already mapped to some physical page, then the data on this physical page is marked as invalid, and p is written on a new, clean, physical page. This process is called an *out-of-place write*.
- The FTL updates the mapping table entry for p to point to the new physical page.



2.4. Garbage Collection

SSDs contain *overprovisioned* space – extra capacity that is used for out-of-place writes. When the extra capacity is full of invalid pages, some of them must be erased so that additional writes can be performed. The *garbage collection mechanism* is the part of the firmware in charge of this process.

The basic garbage collection works as follows:

- When the number of clean blocks drops below a threshold, this triggers the garbage collection process. This threshold is typically 1% to 5% of the SSD's capacity.
- The garbage collection mechanism chooses a *victim* block to be erased. This choice depends on the garbage collection algorithm. A commonly used algorithm is the *greedy* one. It chooses the block with the smallest number of valid pages as victim.
- The valid pages from the victim block are copied to a new, clean, block. The mapping table is updated with their new location. This process is called *moving*.
- The victim block is erased. The block is now clean and can be used for writing new pages.

3. SSDPlayer Components

SSDPlayer provides visual representation of the behavior of the SSD on a given workload. An SSD includes a physical flash device and its firmware. In SSDPlayer, the firmware is represented by a *manager*. The manager includes the mapping and garbage collection mechanisms. The workload represents the set of applications that run in the system and use the SSD. It includes a set of I/O write commands (in *simulation* mode) or FTL commands (in *visualization* mode). The screen displays the state of the SSD after each input command from the workload.

3.1.SSD

An SSD includes a physical device and a manager. You can specify the parameters of the physical device and the manager by [editing the configuration file](#) before starting SSDPlayer. You can choose a manager from the menu after starting SSDPlayer.

3.1.1. Physical Device

A physical device is defined by the number of flash chips, the number of planes per chip, the number of blocks per plane and the number of pages per block. Edit the configuration file to specify these parameters before starting SSDPlayer.

3.1.2. Manager

A manager is the combination of policies or algorithms used to manage the physical device. It includes the mapping by the FTL, the garbage collection mechanism and possibly additional optimizations and features. This version of SSDPlayer includes an implementation of several managers, described in Section 0.

You can choose which manager to run from the manager menu after you start SSDPlayer. Some managers depend on parameters from the configuration file. You can specify those before you start SSDPlayer. If you wish to implement an alternative manager, please consult the Programmer's Guide.

3.2.Input Workload

The workload is the input of SSDPlayer. It represents the set of operations that are performed on the device and change its state throughout the simulation.

In simulation mode, the workload is the set of I/O write commands that the top level application and operating system send to the device. You can provide the workload to SSDPlayer as an input trace file by choosing it in the trace menu. Alternatively, you can ask SSDPlayer to generate a synthetic workload for you, by choosing one of the available distributions in the generator menu.

In visualization mode, the workload is the set of FTL commands that were performed by the simulator or platform which are visualized by SSDPlayer. You should generate a workload trace by executing the simulator or platform. You can then specify this trace as input to SSDPlayer by choosing it in the trace menu.

3.2.1. Workload Generators

The workload generator generates a series of write commands to the SSD's pages according to the distribution you choose. The generator sends these commands directly to SSDPlayer for simulation. You can choose the distribution from the generator menu, and then specify the distribution parameters in the dedicated window.

The current version of SSDPlayer implements Uniform and Zipf distributions. If you wish to implement an alternative distribution, please consult the Programmer's Guide.

3.2.1.1. Uniform

The Uniform workload generator generates write commands that are uniformly distributed across the device. In other words, all the logical pages are written with the same probability.

When you choose the Uniform generator you have to specify two parameters:

- The number of requests to generate.
- The random seed: this is a parameter used for initializing the process of generating random numbers. You can ensure that two executions run on the exact same workload by specifying the same seed in both of them.

3.2.1.2. Zipf

The Zipf workload generator generates write commands that are *skewed*: some logical pages are written more frequently than others: the frequency of access to block i is proportional to i/α^i for α close to 1. This distribution is commonly used to represent realistic workloads such as those of web servers, file servers, etc.

When you choose the Zipf generator you have to specify three parameters:

- The number of requests to generate.
- The random seed: this is a parameter used for initializing the process of generating random numbers. You can ensure that two executions run on the exact same workload by specifying the same seed in both of them.
- The exponent: this parameter specifies how skewed the workload is. A higher exponent means that the popular pages are more popular.

Note: the exponent must be greater than zero.

3.2.2. Input Trace Files

You can use a trace of a real workload as an input for SSDPlayer, to visualize the behavior of the SSD on your own data. The format and content of the trace file depend on the mode in which you run SSDPlayer.

3.2.2.1. Simulation Mode: Raw I/O Trace

In a raw I/O trace, each line represents an I/O command that the operating system sent to the SSD. You should generate the trace in advance and use it as input in the execution of SSDPlayer. SSDPlayer currently supports a basic trace format and an extended format with *temperature tags*. If you wish to extend SSDPlayer to support additional formats, please consult the Programmer's Guide.

3.2.2.1.1. Raw I/O Trace Generation

You can generate the trace in one of several ways:

1. Collect the trace by running an application or a benchmark in your own system, and log all I/O commands in a separate file.
2. Download a trace from a public repository, such as [SNIA IOTTA](#).
3. Use your own workload generator to create a synthetic trace that represents your workload.

Note: make sure that the trace you generate, downloaded, or collected is in the format used by SSDPlayer.

3.2.2.1.2. Raw I/O Trace Format

The basic trace format is similar to the default trace format of [DiskSim](#)¹. The trace lines include several fields that are not in use in this version of SSDPlayer, and are included for compatibility.

	Field	Values	Meaning	Notes
1	Request arrival time	Number (Up to 6 decimal digits)	Time in Milliseconds from start of trace	<i>Not in use</i>
2	Device number	Integer	For systems that have more than one storage device	<i>Not in use</i>
3	Page number	Integer	First page written by the request. Must be smaller than device size	
4	Request size	Integer	Number of pages. For I/O operations on several pages	
5	I/O command	"R": Read "W": Write	Currently only write is supported	
6	Temperature	Integer	For defining data "hotness". Range is specified in configuration file	<i>Optional field, ignored if not in use by manager</i>

Example input line: 2.371000 0 1 8 W 2

Explanation of example: This request arrived 2.371 milliseconds after the beginning of the simulation. It is a write request to page number 1, which has a temperature of 2. SSDPlayer ignores the arrival time (2.371000) and device number (0). All managers except HotCold ignore the temperature (2).

¹ John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger. *The DiskSim simulation environment version 4.0 reference manual*, May 2008, Carnegie Mellon University, Pittsburgh, PA.

You can find complete sample traces in the *traces* directory.

3.2.2.2. *Visualization Mode: FTL Command Trace*

In an FTL command trace, each line represents an action performed by the FTL of your simulator or platform. You should generate the trace in advance and use it as input in the execution of SSDPlayer. SSDPlayer currently supports a basic set of FTL commands and two write levels. If you wish to extend SSDPlayer to support additional commands, please consult the Programmer's Guide.

3.2.2.2.1. *FTL Command Trace Generation*

Follow these steps to generate an FTL command trace:

1. Modify your platform or simulator to log all FTL operations in a dedicated log file. Use the trace format below for the operations you log. You only need to do this once for every system.
2. Execute your platform or simulator with the workload you want to examine. You can use your own applications, a benchmark suite or tool, or some other synthetic workload.
3. You can repeat step 2 to generate logs for different workloads. You do not have to repeat step 1.

3.2.2.2.2. *FTL Command Format*

SSDPlayer currently supports a basic set of FTL operations, which are detailed below. The FTL operations are performed on one or more entities, whose definitions depend on the manager in use.

Reusable visualization Manager Entities

- Logical page: the integer representing this page in the raw I/O command.
- Physical page: the tuple <plane,block,page> that identifies the physical location of the page.
- Block: the pair <plane,block> that identifies the physical location of the block.

RAID Visualization Manager Entities

The RAID manager supports multiple chips, which are part of the definition of physical pages and blocks. In addition, it distinguishes between data and parity logical writes.

- Logical page: the integer representing this data page in the raw I/O command.
- Parity page: the pair <stripe,parity number> that identifies the stripe and parity number within this stripe.
- Physical page: the tuple <chip,plane,block,page> that identifies the physical location of the page.
- Block: the tuple <chip,plane,block> that identifies the physical location of the block.

Commands in common to all Visualization managers

Command	Format	Meaning
Change State	S <s>	Change the state of block to <s>. <s> can be [C]Clean, [A1]Active1, [U]Used, [Re]Recycled, [A2]Active2, or [Ru]Reused.
Begin garbage collection	B 	Begin garbage collection in block .
Erase	E 	Erase block and change its state to [C]Clean.
End garbage collection	G 	End garbage collection in block .

Reusable Visualization Manager Commands

Command	Format	Meaning
1 st Write	W 1 <lp> <pp>	Write logical page <lp> in 1 st write, using physical page <pp>. If an old copy of <lp> exists, mark it as invalid. The difference between 1 st and 2 nd writes is explained in Section 10.3.
2 nd Write	W 2 <lp> <pp ₁ > <pp ₂ >	Write logical page <lp> in 2 nd write, using physical pages <pp ₁ > and <pp ₂ >. If an old copy of <lp> exists, mark it as invalid. The difference between 1 st and 2 nd writes is explained in Section 10.3.
Move	M 1 <lp> <pp>	Move logical page <lp> to physical page <pp> as a first write, and mark the old copy as invalid. Move is called only during garbage collection.

RAID Visualization Manager Commands

Command	Format	Meaning
Write data	W 1 <lp> <pp> D <stripe>	Write logical data page <lp> on physical page <pp>, and mark it as belonging to stripe <stripe>. If an old copy of <lp> exists, mark it as invalid.
Write parity	W 1 <lp> <pp> P <stripe>	Write logical parity page <lp> on physical page <pp>, and mark it as belonging to stripe <stripe>. If an old copy of <lp> exists, mark it as invalid.
Move	M 1 <lp> <pp> P/D <stripe>	Move logical page <lp> to physical page

<pp> and mark it as a data (D) or parity (P) page belonging to stripe <stripe>. The old copy is marked as invalid but belongs to stripe <stripe> until its block is erased. Move is called only during garbage collection.

If you wish to extend SSDPlayer to support additional commands or alternative formats, please consult the Programmer's Guide.

4. Downloading and Installing SSDPlayer

4.1. System Requirements

SSDPlayer is a Java application. In order to execute it, you must have Java Runtime Environment (JRE) version 1.7 or higher installed on your computer.

4.2. Downloading SSDPlayer

Download SSDPlayer at the [SSDPlayer Home Page](#).

- Windows Users
 - Download SSDPlayer_v1.3.1.zip
 - Extract its content by right clicking and choosing the "Extract" option of your compression software (for example, WinZip or WinRAR).
- Linux and Mac Users
 - Download SSDPlayer_v1.3.1.tar.gz.
 - Extract its content by typing "tar -xzf SSDPlayer_v1.3.1.tar.gz" in the command line.

4.3. Installing SSDPlayer

After you extract the zipped files, your directory will include:

1. The java executable *SSDPlayer.jar*.
2. The *resources* directory, with a default configuration file: *ssd_config.xml*.
3. The *traces* directory with some sample traces to get you started.
4. The copyright notice.

No further installation is required. You can edit *resources/ssd_config.xml* to modify the default configuration parameters, or start SSDPlayer immediately.

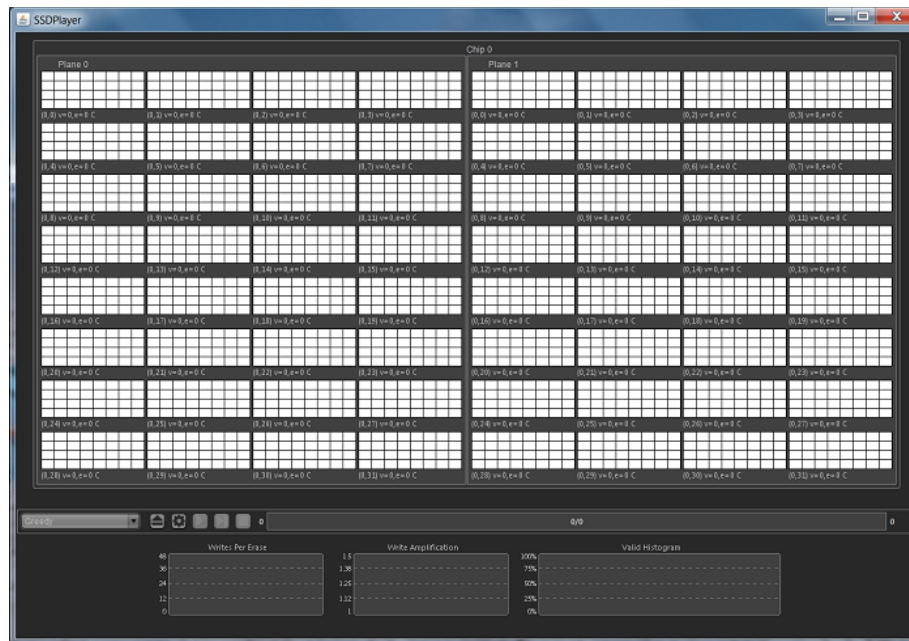
5. Running SSDPlayer in default (GUI) mode

After you download and unzip the SSDPlayer files you can immediately start running it using the sample trace files or the built in workload generators.

5.1. Start SSDPlayer

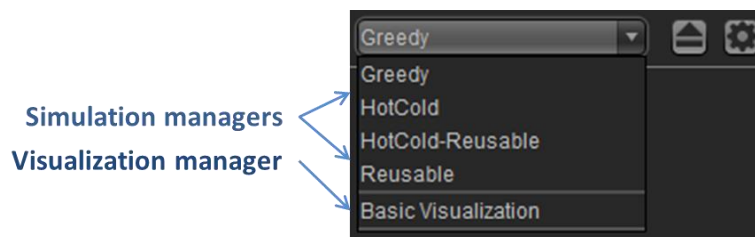
- Windows Users
 - Execute SSDPlayer by double clicking on SSDPlayer.jar.
- Linux and Mac Users
 - Execute SSDPlayer by typing "java -jar SSDPlayer.jar" in the command line.

The initial screen displays the SSD that is defined by the physical parameters in the configuration file. At this point all the blocks are clean, and all the pages are filled with the clean color (white is the default).



5.2. Choose Manager

The default manager is the *Greedy* manager. You can use this manager, or choose from the set of supported managers displayed in the manager menu.



Managers in the top part of the menu run in simulation mode. Managers in the bottom part of the menu run in visualization mode.

5.3. Choose Input Workload

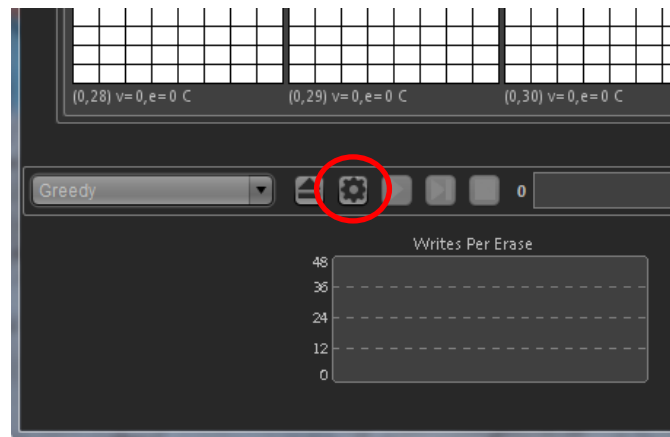
SSDPlayer can run on two input types. It can read input from a trace file (in simulation and in visualization mode), or it can generate its own workload (in simulation mode only).

5.3.1. Choose Workload Generator

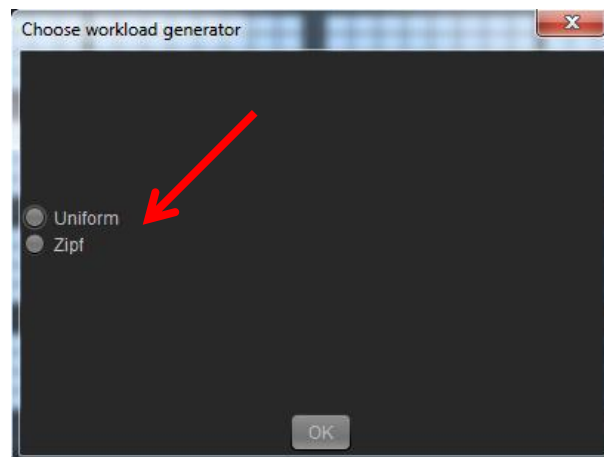
In simulation mode, you can ask SSDPlayer to generate a random workload for you.

Note: this option is not available in visualization mode. If you chose a visualization manager skip to step 5.3.2.

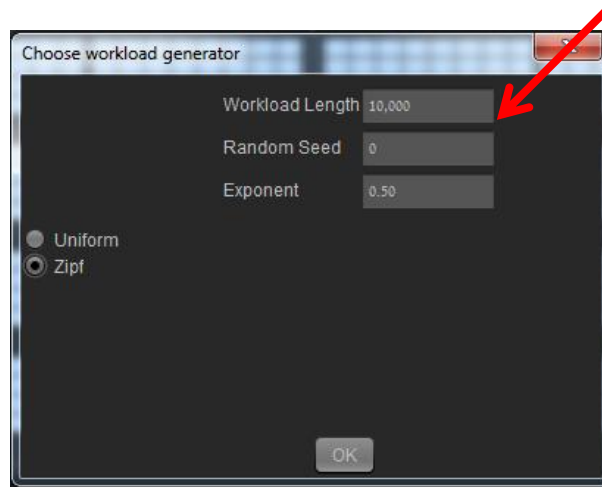
5.3.1.1. Press the generator button



5.3.1.2. Choose one of the available workload generators.



5.3.1.3. Edit the distribution parameters (or use the default ones) and press OK.

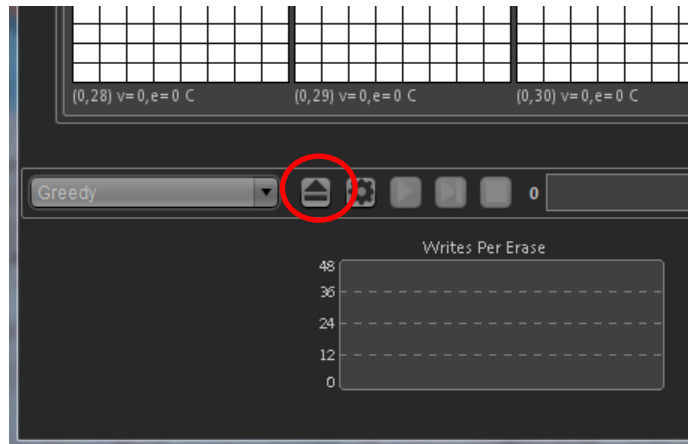


5.3.1.4. Continue to step 5.4.

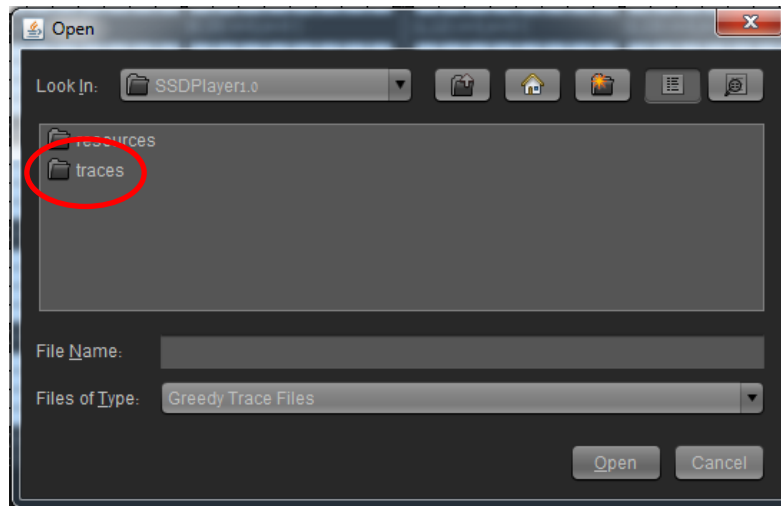
5.3.2. Choose Input Trace File

If you generated your own workload traces, or wish to experiment with the sample traces in the SSDPlayer distribution:

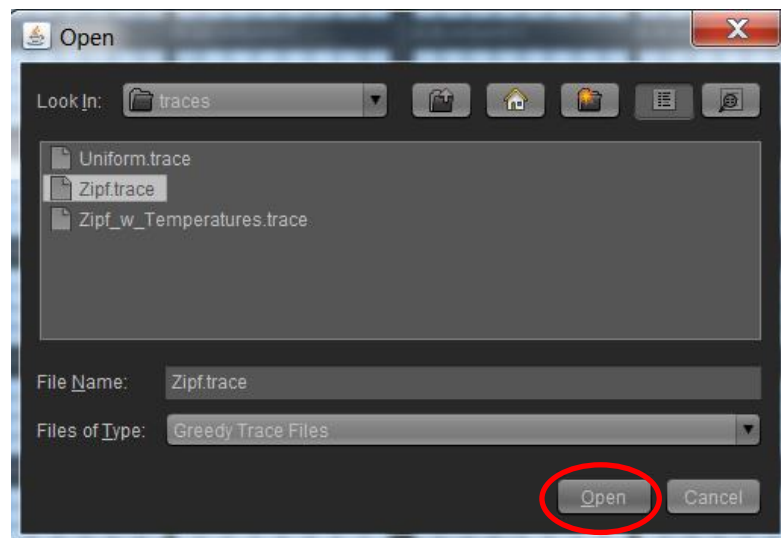
5.3.2.1. Press the *open trace* button.



5.3.2.2. Browse the directory menu to find your trace directory.



5.3.2.3. Double click on the name of the trace file you want to execute, or mark it and press Open.

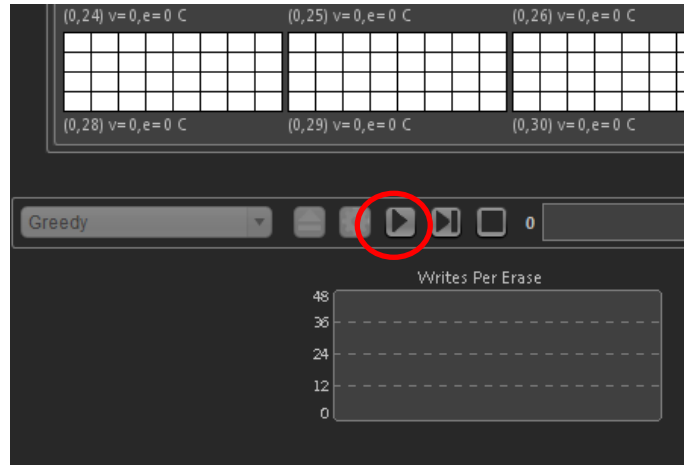


Note: you will only see the trace files of the type suitable for the manager you chose.

5.4. Start the Simulation

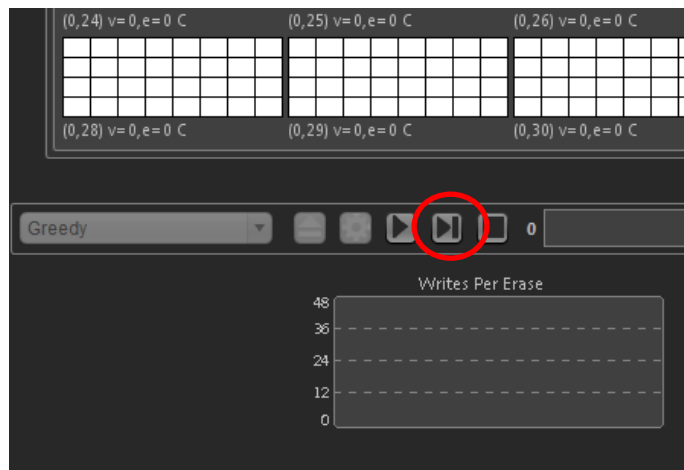
5.4.1. Play

If you press *play*, the simulation will run continuously until the end of the trace file or generated workload, or until you stop it.



5.4.2. Next Frame

If you press *next frame*, SSDPlayer will read one input line from the file or generator, and update the display to show the outcome of this line. You must continue to press this button to advance the simulation.

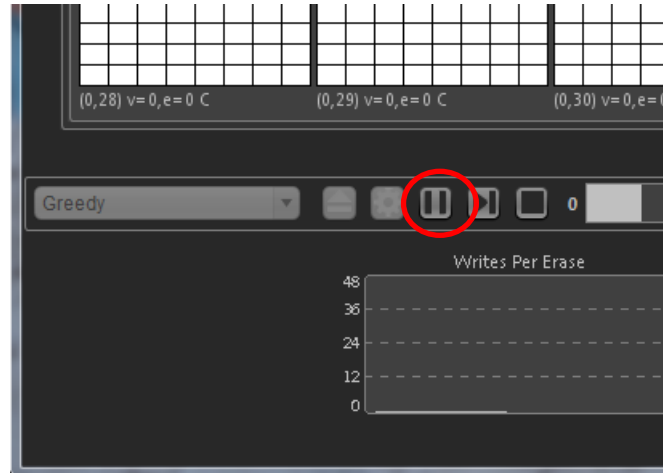


Note: at any point, instead of pressing *next frame* you can press *play* to run the simulation continuously. The simulation will continue from the current device state and the next input line.

5.5. Stopping the Simulation

5.5.1. Pause

Press *pause* if you wish to pause the simulation and continue later on from the same point. The continuous run will stop and the state of the device will 'freeze'.

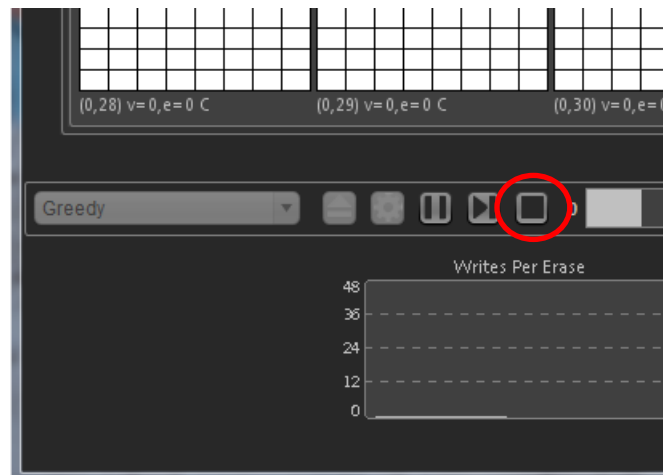


You can continue running the simulation from the same input and state by pressing *play* or *next frame*.

Note: you cannot start a new simulation when the current simulation is paused. If you wish to start over, press *stop*.

5.5.2. Stop

Press *stop* if you wish to end the simulation of this workload. You will then be able to examine the state of the device at the point your simulation stopped. You can now start a new simulation by going back to steps 5.2 or 5.3.



Note: you cannot continue the simulation from where you stopped it. You must start a new simulation by going back to steps 5.2 or 5.3. Next time, if you wish to be able to continue the simulation, press *pause*.

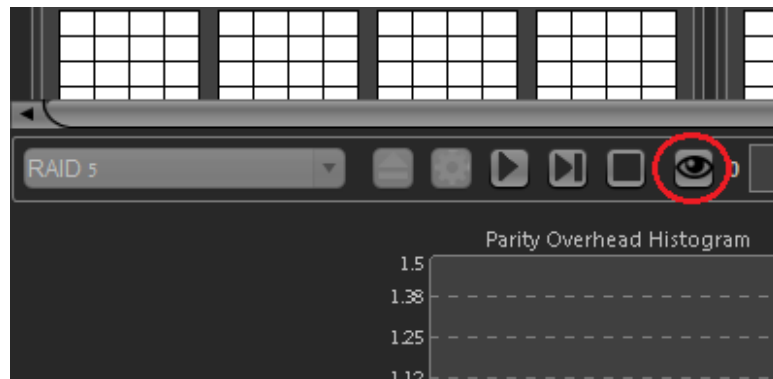
5.6.Highlight Stripes (RAID Managers)

The RAID managers allow you to highlight all the pages in a specified stripe so that you can follow their movements easily. All the data and parity pages belonging to this stripe will be marked with a colored page frame. The parameters in the configuration file (Section 0) determine whether invalidated pages (that have not yet been erased) are also highlighted, and the frame color for each stripe.

5.6.1. Open the Stripes Info Window

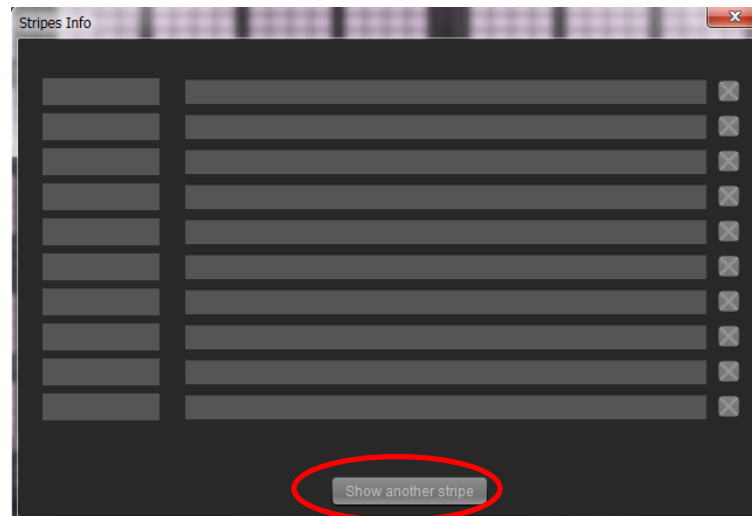
Press *Show Stripes Info* to highlight a stripe, or to view the pages included in the highlighted stripes. The *Stripes Info* window will open.

Note: This button is enabled only in the RAID managers and only when the simulation is paused.



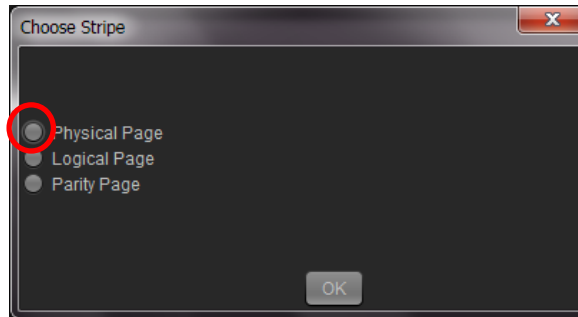
5.6.2. Show another stripe

Press *Show another stripe* to specify a stripe for highlighting. The *Choose Stripe* window will open.



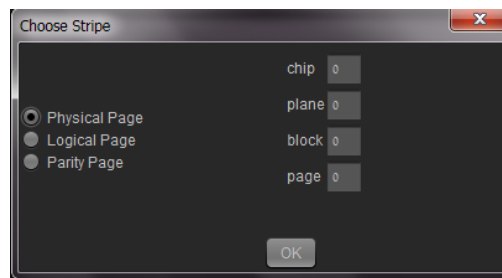
5.6.2.1. Specify a stripe to highlight

Specify the stripe to highlight by naming one of the pages it contains. This can be either a physical page, a logical data page, or a parity page. Choose the specification method by marking it on the *Choose Stripe* window.



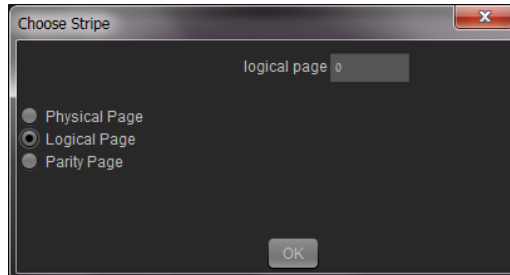
5.6.2.1.1. Specify a physical page

A physical page is identified by its chip, plane, block, and page numbers. Fill those in the corresponding slots and press *OK*.



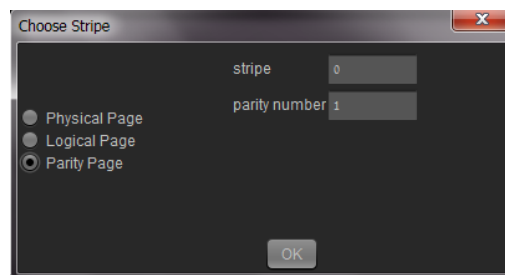
5.6.2.1.2. Specify a logical page

Fill the logical page number you wish to specify and press *OK*.



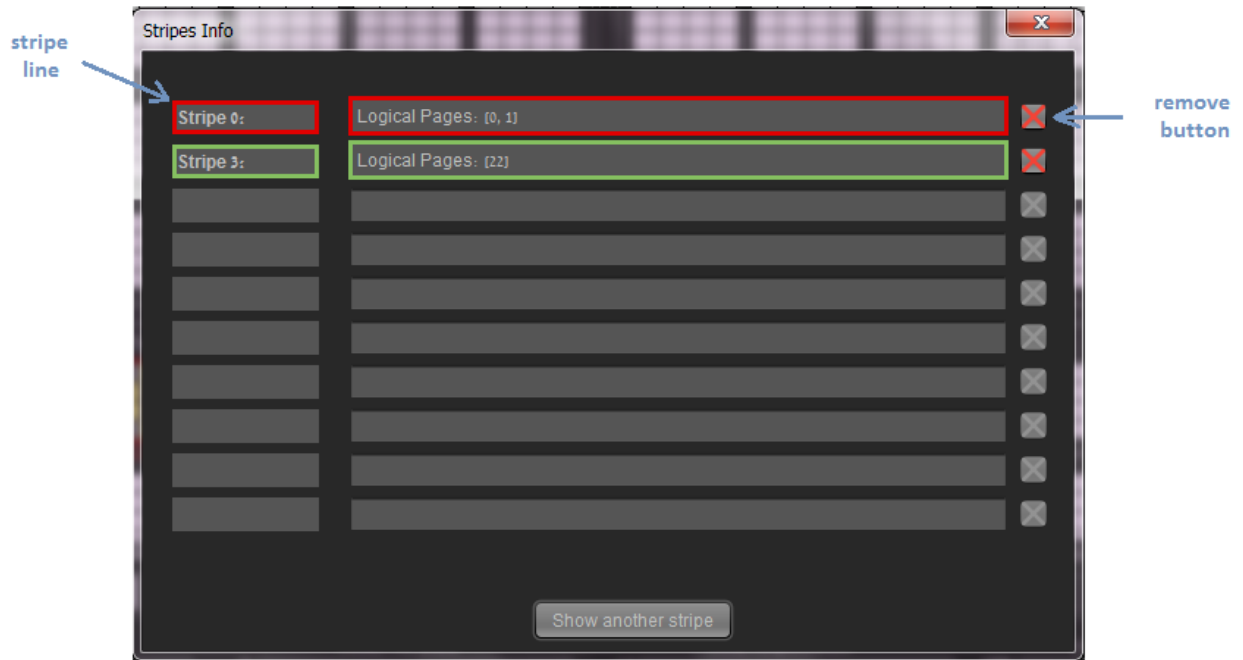
5.6.2.1.3. Specify a parity page

A parity page is identified by its stripe and parity numbers. Fill those in the corresponding slots and press *OK*.



You will return to the *Stripes Info* window, where the stripe you specified will be highlighted by a bold, colored frame, and all the logical pages in it will be listed. The same color will be used on the main simulation view for the page frames of all the pages belonging to this stripe (Section 7.3).

Note: You may choose to highlight a stripe that was not fully written when the simulation was paused, i.e., only some of this stripe's pages were written. In this case, the logical pages that have already been written will appear in the *Stripes Info* window and will be highlighted on the main simulation view. Additional pages belonging to this stripe will be highlighted when they are written, and will appear in the *Stripes Info* window in the next time it is opened.



5.6.3. Remove stripe

To remove a selected stripe, press on the *Remove* button next to the corresponding stripe. The stripe will be removed from the *Stripes Info* window, and its pages will not be highlighted anymore.

Note: SSDPlayer allows you to highlight up to 10 stripes simultaneously.

5.7. Manage breakpoints

Breakpoints in SSDPlayer are similar to debugging breakpoints. They define conditions that cause the simulation to pause. You can define multiple breakpoints, specifying several different conditions.

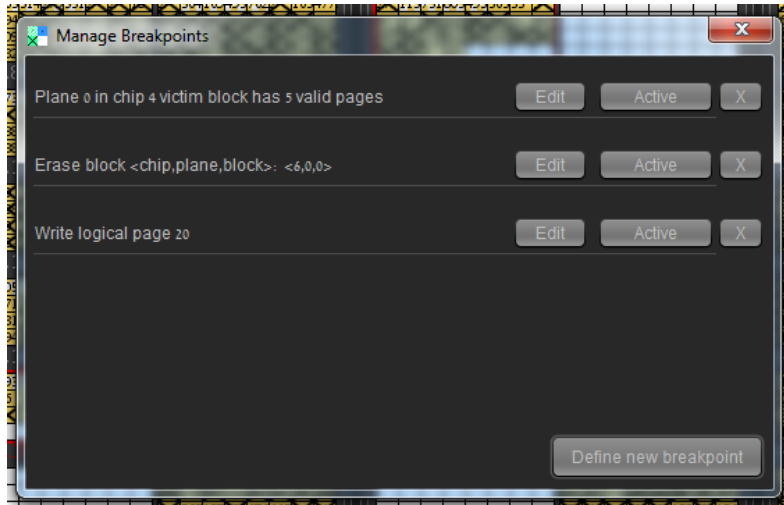
You can define breakpoints manually, at any time during the simulation. You can also define breakpoints in a special [configuration file](#) that is loaded before the simulation starts.

5.7.1. Defining Breakpoints

Click the stopwatch button to open the *Manage Breakpoints Window*.



Note: This button is active only when the simulation is paused.

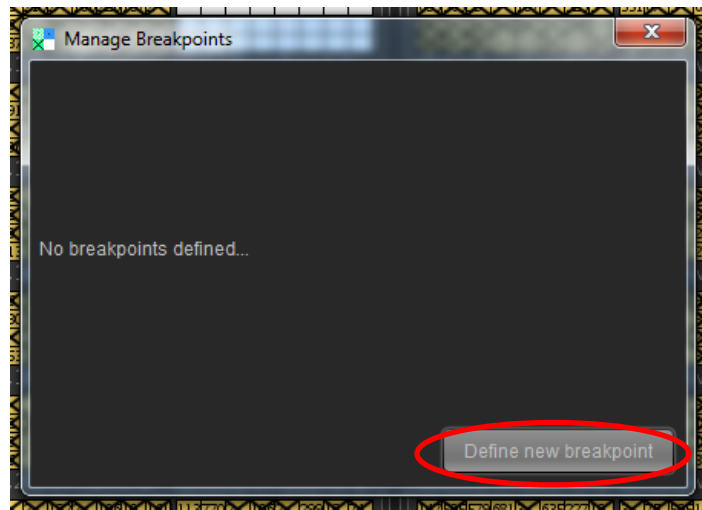


The window allows adding, removing, and editing breakpoints.

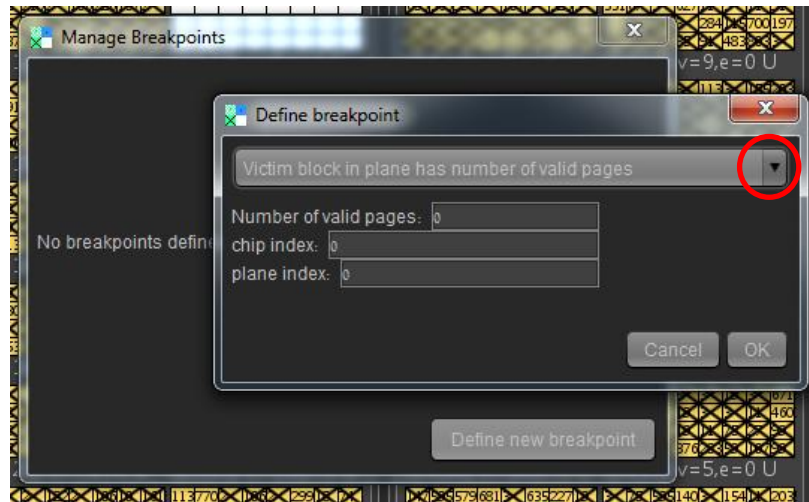
When it is opened, the window will show existing breakpoints: those that were defined earlier in this window and those defined in the breakpoints file.

5.7.1.1. Add a new breakpoint

Click the 'Define new breakpoint' button.



The 'define breakpoint' window will open

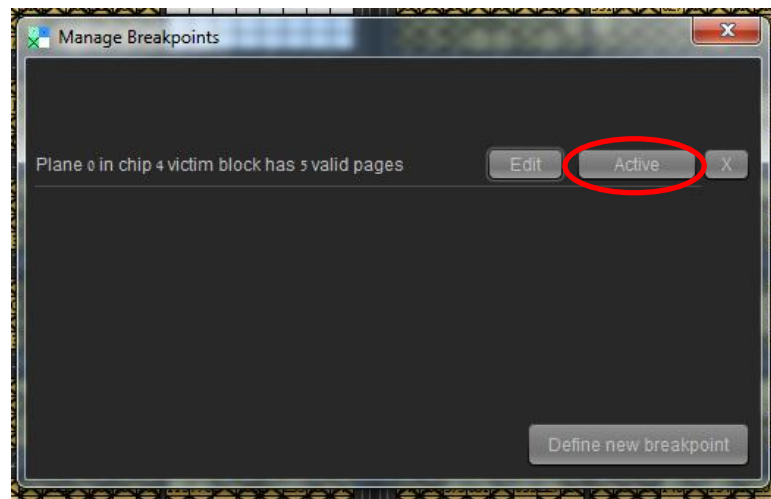


Choose a breakpoint type from the scroll down menu. The relevant parameters for this type will appear. Edit the parameters you are interested in and press OK.

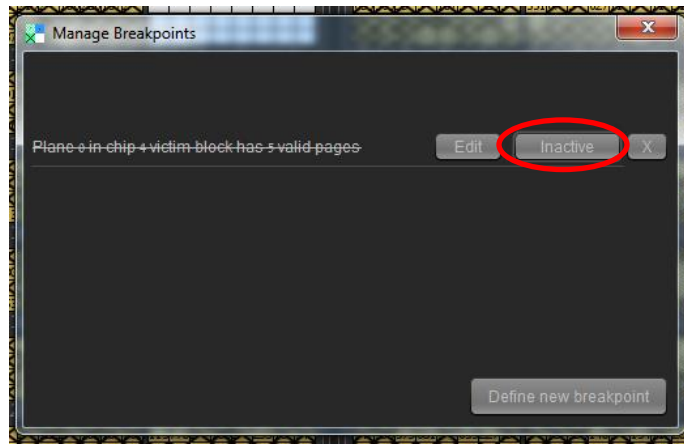
Note: Entering illegal parameters for a breakpoint will report an error and discard the breakpoint.

5.7.1.2. Enable/Disable breakpoint

In the 'manage breakpoints' window, click the 'Active' button next to an active breakpoint to disable it.

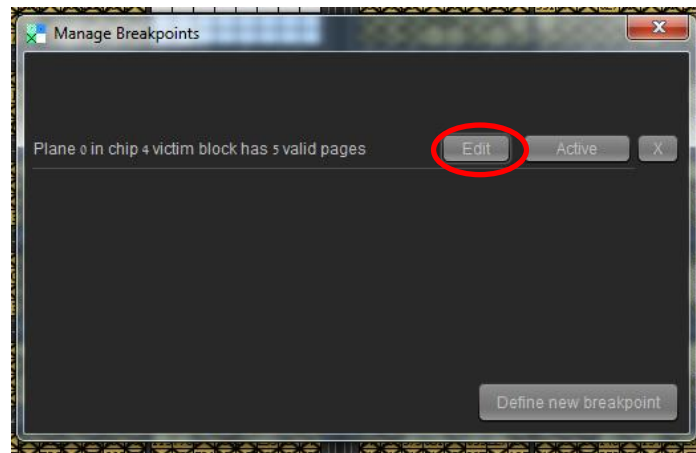


An inactive breakpoint will be crossed out. You can enable it by clicking the 'Inactive' button next to it.



5.7.1.3. Edit an existing breakpoint

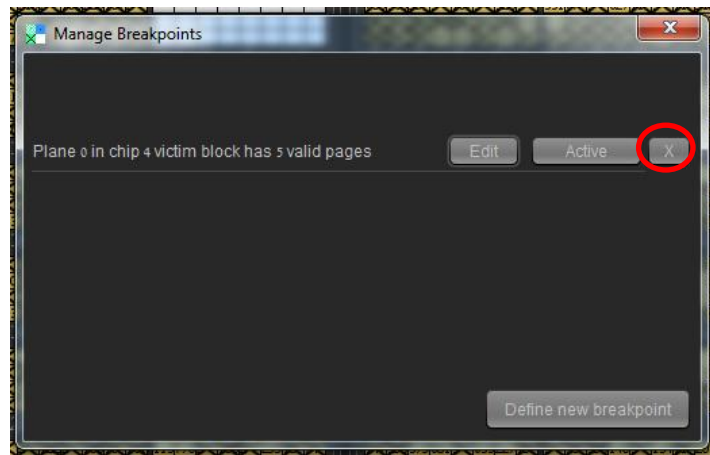
Click the 'Edit' button in the 'manage breakpoints' window.



Change the breakpoint type or its parameters and click OK.

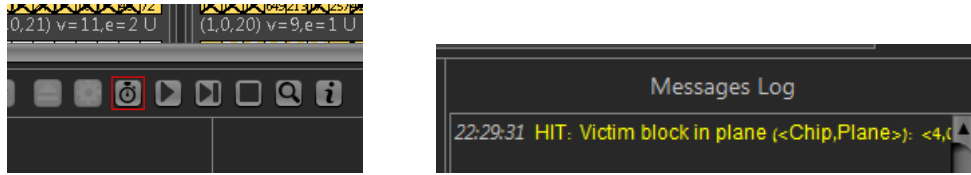
5.7.1.4. Remove an existing breakpoint

Click the 'X' button next to the breakpoint you wish to remove.

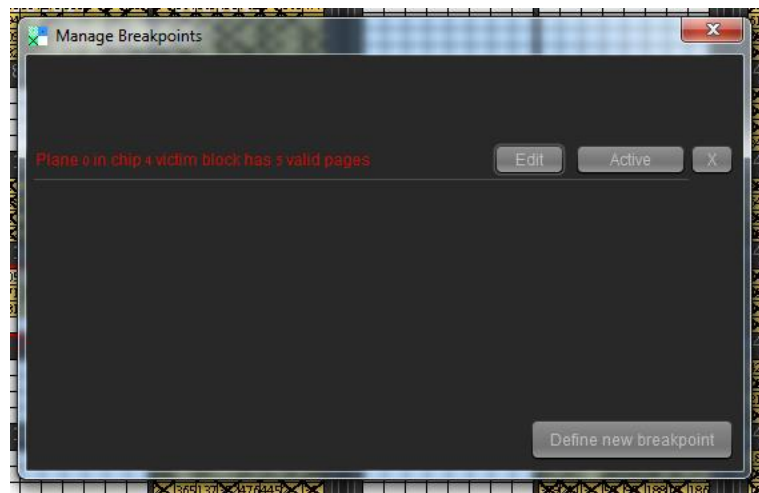


5.7.2. Breakpoint Hits

A breakpoint hit means that the condition defined in one of the breakpoints is true. When this happens, an appropriate message will appear in the message log, the simulation will pause and the breakpoints button will be outlined in red.



If you click the breakpoints button the 'manage breakpoints window' will open and highlight in red all the breakpoint that were hit. You can edit them or remove them if you like.



You can continue the simulation just like after a regular pause.

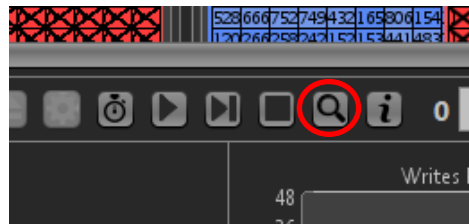
5.8. Zoom in and out

The different zoom levels in SSDPlayer allow you to choose the level of detail presented on the display. They are useful for simulating large devices that cannot be viewed entirely on the screen in the default level of detail.

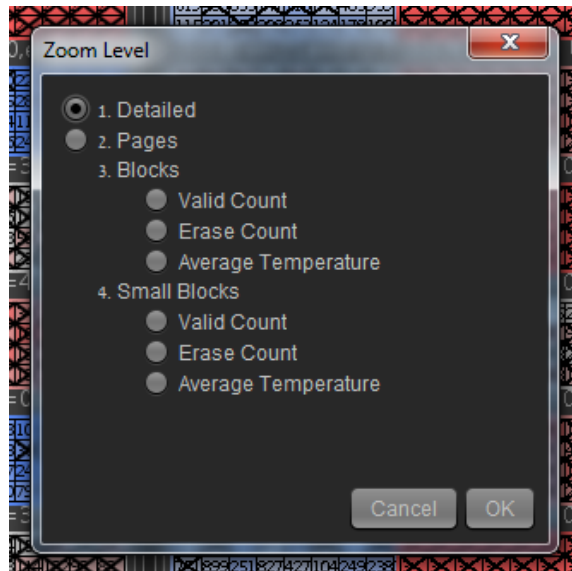
Note: Some zoom level options are only available in specific managers.

5.8.1. Choosing a zoom level

Click the magnifying glass button to open the *Zoom Level Window*.



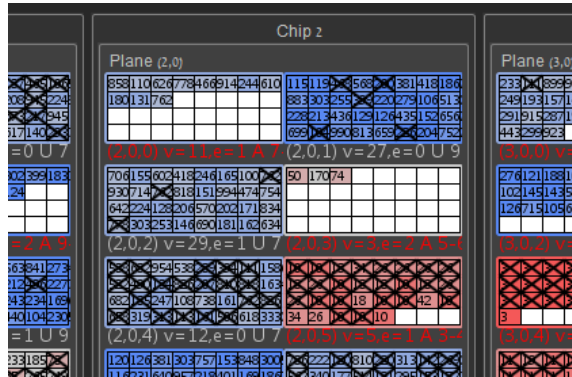
Note: Clicking this button pauses the simulation.



The 'zoom level window' shows the available zoom levels for the current manager. Changing the zoom level will update the device's display accordingly.

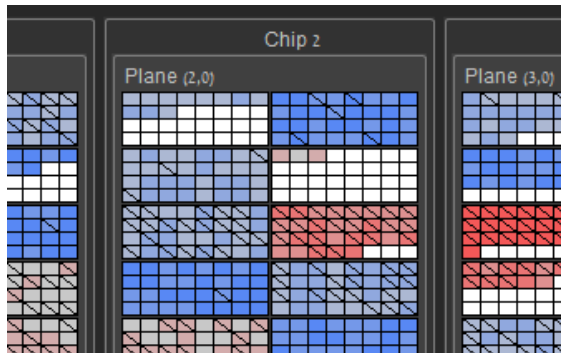
5.8.2. Detailed zoom level

This is the standard zoom level. Page numbers and counters are displayed, written pages are colored and deleted pages are crossed.



5.8.3. Pages zoom level

In this level counters are removed, the sizes of pages and the spacing between them is reduced to half of the original. Invalid pages are marked with a thin line instead of the current bold cross, and the "moved" page pattern is converted to a lighter shade of the original page.

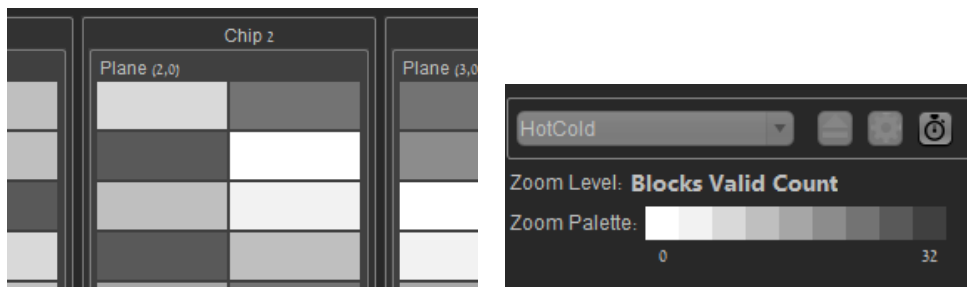


5.8.4. Blocks zoom level

This is an aggregated zoom level. Pages are no longer visible; the color of the block represents the state of its pages.

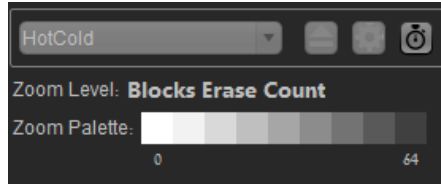
5.8.4.1. Valid count

The color of the block represents its number of valid pages.



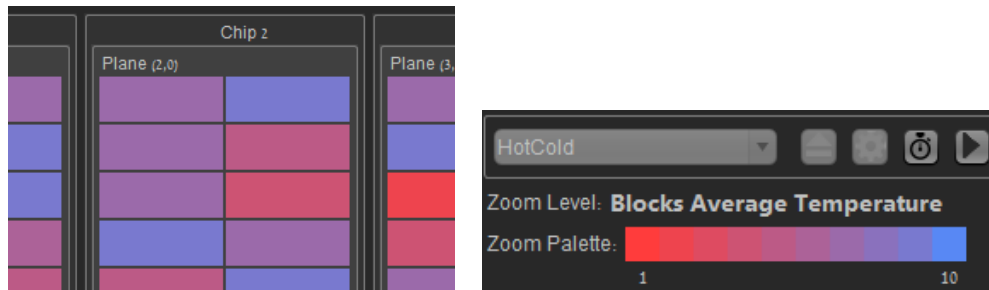
5.8.4.2. Erase count

The color of the block represents the number times is has been erased.



5.8.4.3. Average temperature

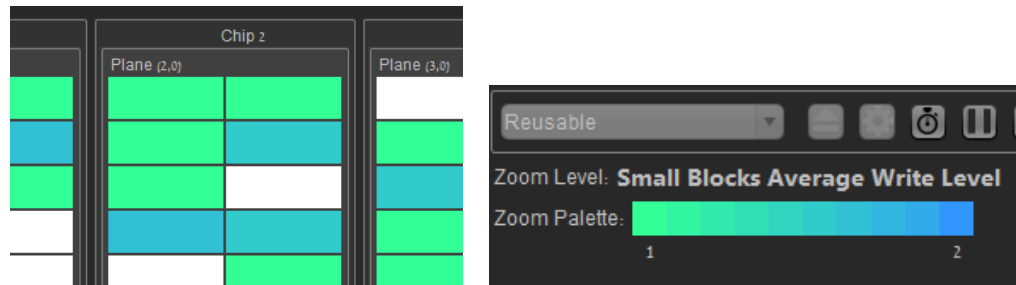
The color of the block represents the average temperature of its pages.



Note: this zoom level is available only in the HotCold manager.

5.8.4.4. Average write level

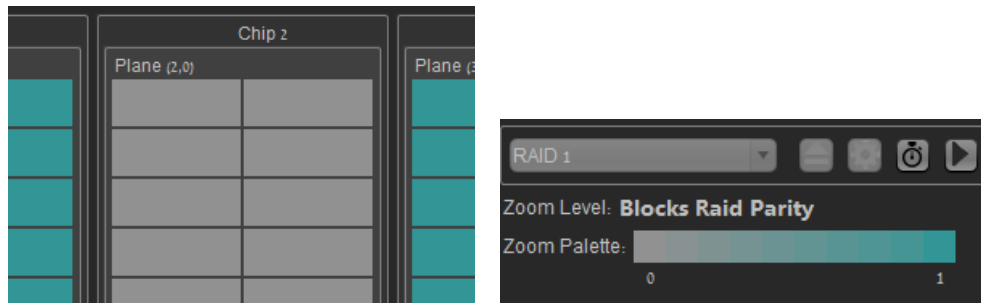
The color of the block represents the average write level of its pages.



Note: this zoom level is available only in the Reusable manager.

5.8.4.5. RAID parity

The color of the block represents the portion of parity and data pages in it.



Note: this zoom level is available only in the RAID managers.

5.8.5. Small Blocks zoom level

This is an aggregated zoom level which is identical to the Blocks zoom level in every aspect except that the blocks are smaller. For a detailed description of the options in this level see Section 5.8.4.

5.9.View Information

You can view the internal simulation state in detail in the information screen. This screen contains the state of every entity in the simulation, as well as the accurate numbers of the parameters displayed in the histograms.

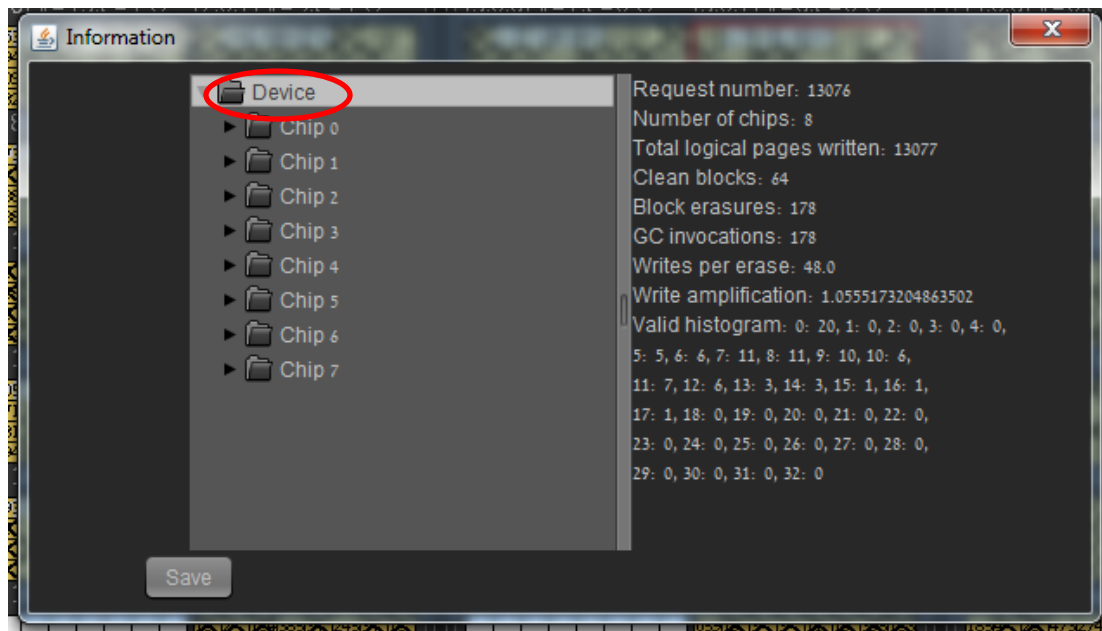
5.9.1. Open the information screen

Press the 'Information' button. This will pause the simulation and open the 'Information' window.



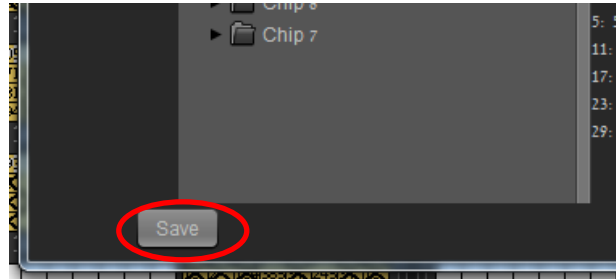
Note: this button is also enabled after the simulation has terminated normally.

Choose the entity you wish to view. You can view the information of the entire device, or a specific chip, plane, block, or page. The entity's state will be displayed on the right-hand side of the window.



5.9.2. Saving the simulation state

You can save the information displayed in the information window for future reference by clicking the 'Save' button.



The state of all entities (those displayed as well as those that are not) will be saved in an xml file in the location you choose.

5.10. Speedup with sampling rate

You can increase the simulation speed by allowing the display to “skip” simulation steps – the simulation will continue but the display will only show a sample of the simulated operations. A *sampling rate* of X means that the display will be updated every X operations (garbage collection is considered a single operation in simulation mode). You can modify the sampling rate at any time during the simulation.

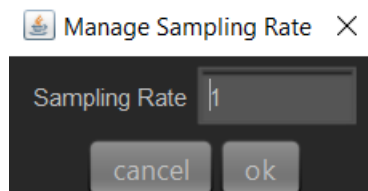
5.10.1. Open the sampling rate screen

Press the 'sampling' button. This will pause the simulation and open the 'manage sampling rate' window.



5.10.2. Set the sampling rate

Set the sampling rate to your preferred value and press the 'ok' button. The default value is taken from the configuration file. A sampling rate of 1 means that the display will be updated every 1 frame.



6. Running SSDPlayer in CLI mode

Although SSDPlayer is intended, primarily, for visualization purposes, its graphical interface and functionality can be bypassed for quick execution and experimentation. The command line interface (CLI) allows you to execute the simulation mode by specifying all the configuration and runtime parameters.

When executed in CLI mode, SSDPlayer will perform the simulation without displaying the SSD components, and without opening the display window at all. It will log the final state and statistics of the simulation in an information file similar to the one saved from the information screen (See section 5.9).

When the simulation completes successfully, a message will be displayed on the standard output.

6.1. Command line parameters

6.1.1. -C <config file name>

Specify the name of the configuration file (this can be the same file you use for running SSDPlayer in the default (GUI) mode).

6.1.2. -M <manager name>

Specify one of the names from the list of managers in Section 10. The name of the manager should be identical to its name in the configuration file.

6.1.3. -F <trace file name>

Specify the name of the input trace file. The input trace should be compatible with the chosen manager.

Note: you must specify either an input trace or a workload generator.

6.1.4. -G (use workload generator)

This option invokes one of the workload generators described in Section 3.2.1. The -G option is followed by required and optional parameters.

6.1.4.1. Required parameters

generator type	Either “-U” (uniform) or “-Z” (Zipf)
workload length	Number of generated write requests
seed	Random seed (used for reproducible results)
Exponent	The exponent for the Zipf distribution (required for the Zipf workload generator)

6.1.4.2. Optional parameters

Max write size	Maximum size of write request (in pages)
is write size uniform	Either “1” (uniform) or “0” (exponential) distribution of request sizes

Note: you must specify either an input trace or a workload generator.

6.1.5. -O <output file name>

Specify the name of the output (information) file. If a file with this name already exists, it will be overwritten.

6.1.6. -help

Displays the following help message with the allowed command line formats:

1. -C <config file name> -M <manager Name> -F <trace file name><trace file extension> -O <output file name>
2. -C <config file name> -M <manager Name> -G -U <workload length> <seed> -O <output file name>
3. -C <config file name> -M <manager Name> -G -U <workload length> <seed> <max write size> <is write size uniform> -O <output file name>
4. -C <config file name> -M <manager Name> -G -Z <workload length> <seed> <exponent> -O <output file name>
5. -C <config file name> -M <manager Name> -G -Z <workload length> <seed> <exponent> <max write size> <is write size uniform> -O <output file name>

6.2. Examples

The following examples correspond to the allowed formats listed above.

1. `java -jar SSDPlayer.jar -C resources/ssd_config.xml -M Greedy -F traces/Small_Uniform.trace -O target/out`
2. `java -jar SSDPlayer.jar -C resources/ssd_config.xml -M "RAID 5" -G -U 10000 0 -O target/generateUniform`
3. `java -jar SSDPlayer.jar -C resources/ssd_config.xml -M "RAID 5" -G -U 10000 0 1 T -O target/generateUniformResizableUniform`
4. `java -jar SSDPlayer.jar -C resources/ssd_config.xml -M "RAID 5" -G -Z 10000 0 0.5 -O target/generateZipf`
5. `java -jar SSDPlayer.jar -C resources/ssd_config.xml -M "RAID 5" -G -U 10000 0 0.5 1 T -O target/generateZipfResizableUniform`

6.3. Error messages

In CLI mode, error messages will be displayed in the standard output. This includes errors related to CLI parameters, configuration and input files, and runtime errors.

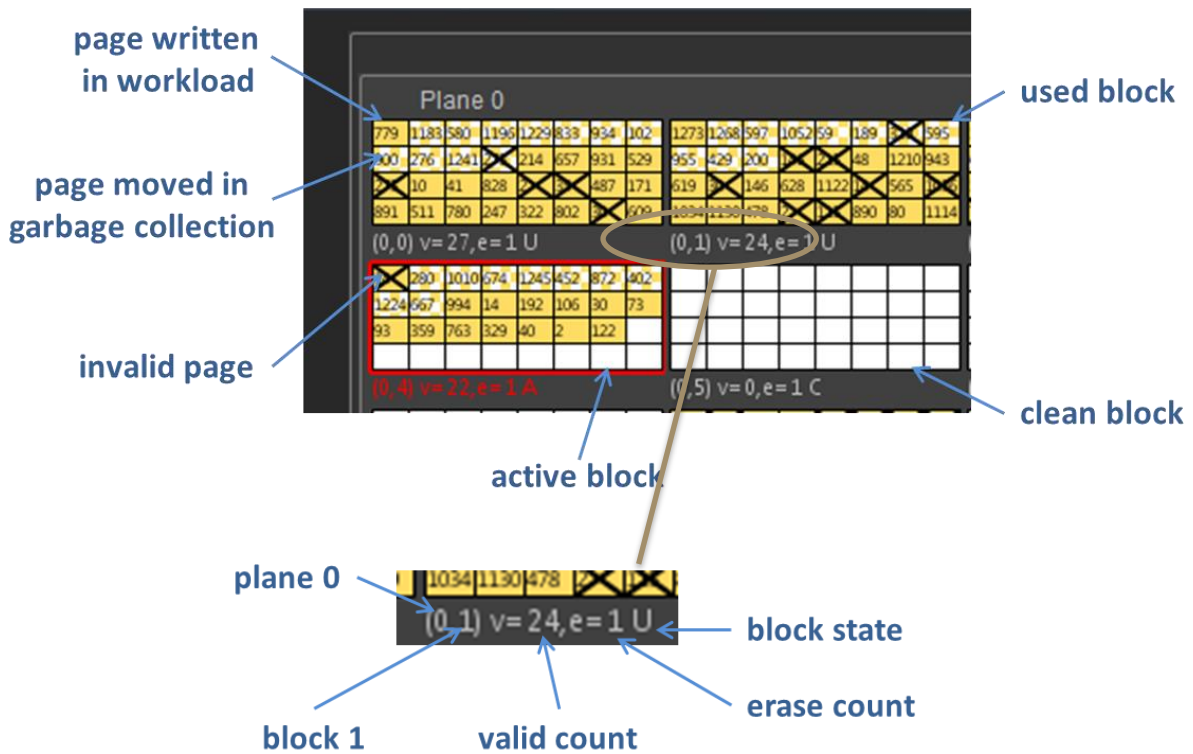
7. Understanding the SSDPlayer Display

7.1. Physical Device Display

On the main display, you will see pages change their color and texture according to the data movement on the device. The block counters will also be updated in each step.

The main display features are:

- A white page is clean.
- A page is filled with a plain pattern when it is written. The logical page number appears on the physical page (unless counters are turned off in the parameter file).
- A bold cross over a page shows that this copy is invalid, and the page has been written elsewhere.
- A checkered fill pattern shows that the logical page has been moved to this location by a garbage collection process.
- A bold red frame around a block shows that it is currently allocated as an active block in this plane. There may be more than one active block per plane.
- In visualization mode, a bold black frame shows that this block is the victim block of an ongoing garbage collection process.
- The valid count shows how many pages in this block contain valid data.
- The erase count shows how many times this block has been erased.



7.2.Histograms

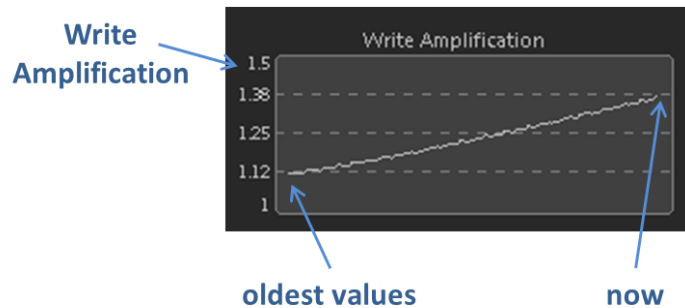
7.2.1. Write Amplification

This histogram shows the write amplification. The rightmost point in the graph is the current value, and the histogram shows a history of the last 1000 frames. There are 200 points in the graph, each one of them is the average value of the write amplification in 5 frames.

The write amplification is calculated as follows:

$$\text{write amplification} = \frac{\text{writes} + \text{moves}}{\text{writes}},$$

where *writes* and *moves* are counted since the beginning of the simulation.



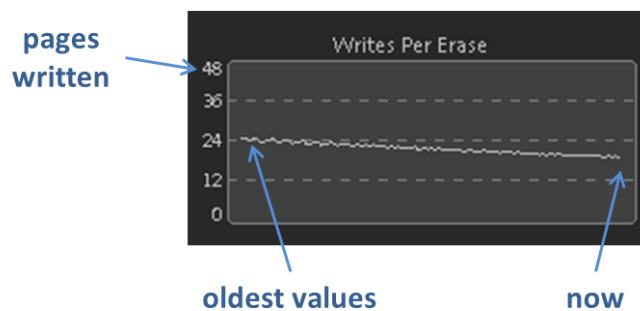
7.2.2. Writes Per Erase

This histogram shows how many pages were written on average for every erase operation. The rightmost point in the graph is the current value, and the histogram shows a history of the last 1000 frames. There are 200 points in the graph, each one of them is the average value of the writes per erase in 5 frames.

Writes per erase are calculated as follows:

$$\text{write amplification} = \frac{\text{writes} - \text{initial writes}}{\text{erases}},$$

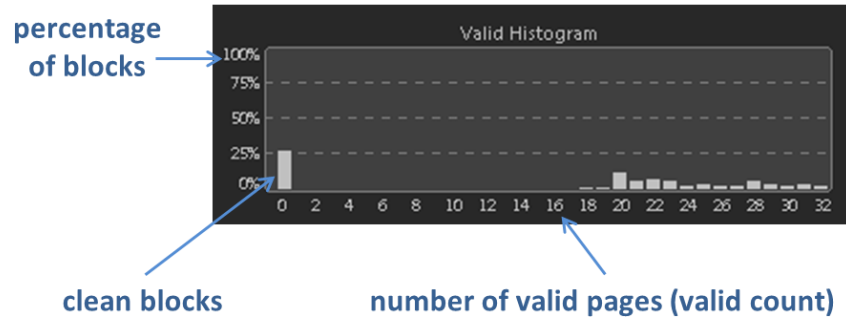
where *writes* and *erases* are counted since the beginning of the simulation. *Initial writes* is the number of pages written before the first garbage collection.



Note: this measure is useful when performing multiple writes (see Sections 0 and 8.2.5), in which case write amplification does not accurately reflect the utilization of the pages². With standard SSDs, such as when using the Greedy manager, these measures are equivalent: writes per erase = pages per block / write amplification.

7.2.3. Valid Histogram

This histogram shows the distribution of valid pages in the blocks. For each valid count, it shows the percentage of the device's blocks that have this valid count. The values are updated in each frame to represent the current state of the device.



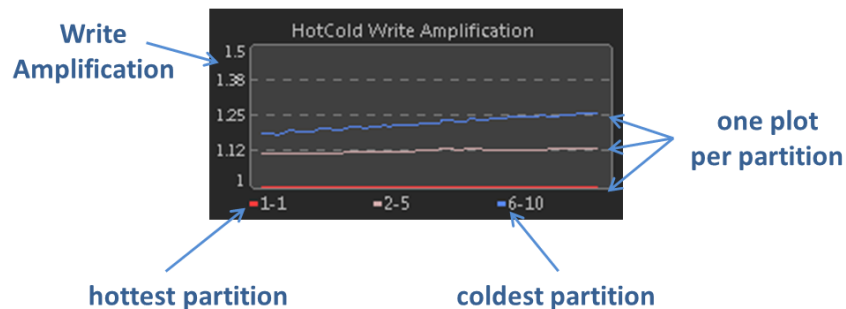
7.2.4. HotCold Write Amplification (HotCold Manager)

This histogram shows the write amplification within each partition – the color of the plot is the color of the hottest pages in the partition. The rightmost points in the graph are the current values, and the histogram shows a history of the last 1000 frames. There are 200 points in the graph, each one of them is the average value of the write amplification in 5 frames.

The write amplification within each partition p is calculated as follows:

$$write\ amplification(p) = \frac{writes(p) + moves(p)}{writes(p)},$$

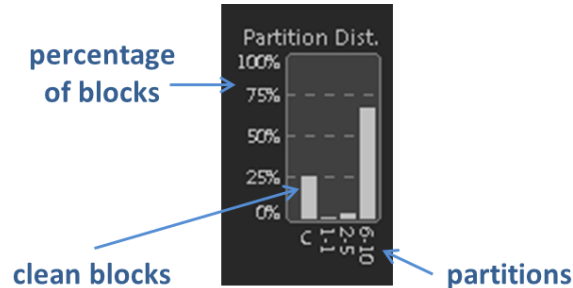
where $writes(p)$ is the number of pages written in partition p , and $moves(p)$ is the number of pages moved during garbage collection in partition p . $writes(p)$ and $moves(p)$ are counted since the beginning of the simulation.



² Eitan Yaakobi, Alexander Yucovich, Gal Maor, Gala Yadgar. When Do WOM Codes Improve the Erasure Factor in Flash Memories? In proceedings of IEEE International Symposium on Information Theory (ISIT '15), June 2015.

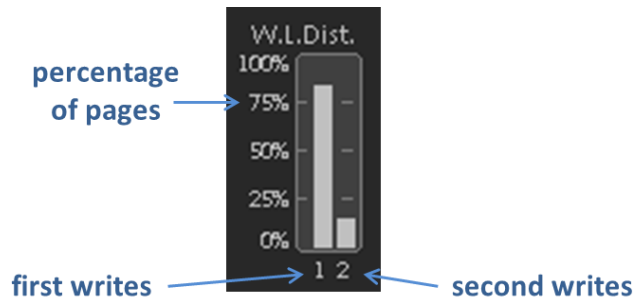
7.2.5. Partition Distribution (HotCold Manager)

This histogram shows the distribution of blocks into partitions. For each partition, it shows the percentage of the device's blocks that store pages for this partition. It also shows the percentage of blocks that are in the Clean state, and are not currently allocated to any partition. The values are updated in each frame to represent the current state of the device.



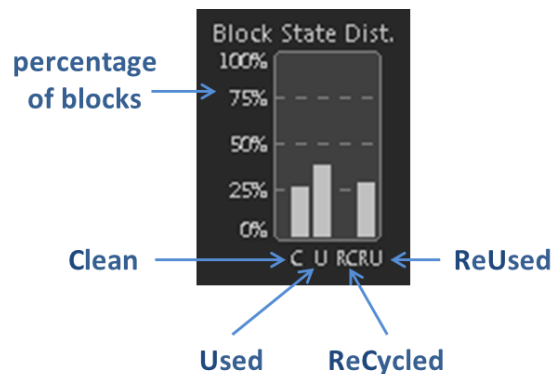
7.2.6. Write Level Distribution (Reusable Manager)

This histogram shows the distribution of written pages into write levels (first or second). For each write level, it shows the percentage of the device's valid logical pages that are currently written in this level. The values are updated in each frame to represent the current state of the device.



7.2.7. Block State Distribution (Reusable Manager)

This histogram shows the distribution of blocks into states. For each state, it shows the percentage of the device's blocks that are currently in this state. The values are updated in each frame to represent the current state of the device.



7.2.8. Valid 1 and Valid 2 Histograms (Reusable Manager)

These histograms are identical to the Valid Histogram in the Greedy and HotCold managers. They show the histogram of blocks with valid pages in each level:

- The Valid 1 Histogram shows the distribution of blocks according to valid pages in first write.
- The Valid 2 Histogram shows the distribution of blocks according to valid **logical** pages in second write.

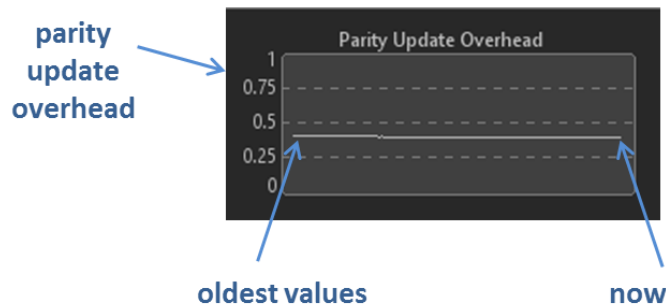
7.2.9. Parity Update Overhead Histogram (RAID Managers)

This histogram shows the parity overhead, which measures the extra writes generated by parity updates. The rightmost point in the graph is the current value, and the histogram shows a history of the last 1000 frames. There are 200 points in the graph, each one of them is the average value of the parity overhead in 5 frames.

The parity overhead is calculated as follows:

$$\text{parity update overhead} = \frac{\text{parity writes}}{\text{data writes} + \text{parity writes}},$$

where *data writes* and *parity writes* are counted since the beginning of the simulation.



7.3.Stripe Highlighting (RAID Managers)

You may specify stripes for highlighting in the Stripe Info window (Section 5.6), so that their pages will be highlighted on the main simulation view. If several stripes are specified, the pages of each stripe will be highlighted in with a different frame color.

In this example below, stripe 0 is highlighted in red. Logical page 4 and its parity are marked with a red page frame. The invalidated copy of the parity is also highlighted because `show_old_parity` was set in the configuration file (Section0).



7.4.Information Screen

The information screen displays the internal device state in text format, so that it can be inspected closely and saved for future reference and analysis.

Field	Level	Meaning
Request number	Device	Last write request that was handled
Number of chips	Device	Number of chips in the device
Number of planes	Chip	Number of planes in the chip
Number of blocks	Plane	Number of blocks in the plane
Number of pages	Block	Number of pages in the block
Total logical pages written	Device, chip, plane	Number of logical pages written in the viewed entity Note: in the RAID managers, this is the sum of data and parity pages
Total parity pages written	Device, chip, plane	RAID managers only: number of parity pages written in the viewed entity
Total data pages written	Device, chip, plane	RAID managers only: number of data pages written in the viewed entity
Clean blocks	Device, chip, plane	Current number of clean block in the viewed entity
Block erasures	Device, chip, plane	Number of block erasures performed so far in the viewed entity
GC invocations	Device, chip, plane	Number of times the garbage collection process was invoked so far in the viewed entity Note: this is equivalent to the number of erasures in all managers except the Reusable managers.
Recycled blocks	Device, chip, plane	Reusable managers only: current number of recycled blocks in the viewed entity
Writes per erase	Device	The latest value from the “writes per erase” histogram
Write amplification	Device	The latest value from the “write amplification” histogram
Hot cold write amplification	Device	HotCold manager only: The latest value from the “HotCold write amplification” histogram
Parity overhead	Device	RAID managers only: The latest value from the “parity update overhead” histogram

Field	Level	Meaning
Valid histogram	Device	Number of pages with each valid count (equivalent to the values displayed in the "valid histogram")
Valid1 histogram	Device	Reusable managers only: number of pages with each valid1 count (equivalent to the values displayed in the "valid1 histogram")
Valid2 histogram	Device	Reusable managers only: number of pages with each valid2 count (equivalent to the values displayed in the "valid2 histogram")
Status	Block	The state of the block (see list in Section 3.2.2.2.1)
Erase count	Block	Number of times this block was erased
Valid count	Block	Current number of valid pages in this block
Write level	Block, page	Reusable managers only: Block: highest write level of a page in this block Page: write level of this page
Partition	Block	HotCold manager only: the partition this block currently belongs to
Average page temperature	Block	HotCold manager only: average temperature of all pages in this block (valid and invalid)
Average page write level	Block	Reusable managers only: average write level of pages in this block (valid and invalid pages)
LP	Page	Logical page currently written on this physical page
Is clean	Page	True if this physical page is clean, False otherwise
Is valid	Page	True if the logical page written on this physical page is valid, False otherwise
Moved by GC	Page	True if the logical page written on this physical page was copied from another page during garbage collection, False otherwise
Stripe	Page	RAID managers only: the number of stripe the logical page written on this physical page belongs to
Parity number	Page	RAID managers only: the RAID functionality of the logical page written on this physical pages: 0 for data pages, 1 or 2 for parity pages

Temperature

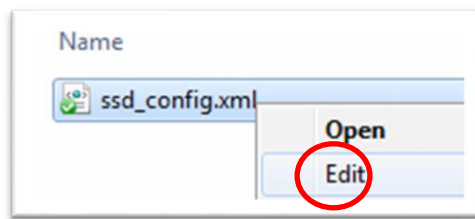
Page

HotCold managers only: temperature of the logical page written on this physical page.

8. Editing the Configuration File

The configuration file is an XML file that contains the parameters of the flash device and its manager. To modify these parameters, edit and save the configuration file **before** you start SSDPlayer. The configuration file is *ssd_config.xml*, located in the *resources* directory.

- Right click on the configuration file name and choose Edit.



- Select the parameter you wish to modify and type in the value you want.

```
</physical>
<visual>
  <show_counters>yes</show_counters>
  <speed>9000</speed><!-- Number of frames per minute -->
  <page_width>18</page_width>
  <page_height>13</page_height>
  <block_space>2</block_space><!-- spacing between the bl
  <pages_in_row>8</pages_in_row>
```

- Save the configuration file.

There are three categories of configuration parameters. The *physical* parameters define the properties of the flash device. The *visual* parameters define the way the device is displayed on the screen. The *management* parameters define the behavior of each of the managers.

8.1. Physical Parameters

Parameter	Values	Meaning
<i>overprovisioning</i>	integer	Size of overprovisioned space (percent of total capacity).
<i>gc_threshold</i>	integer	Garbage collection threshold (as percent of total capacity). * If this parameter is specified then <i>gc_threshold_blocks</i> <u>should not</u> be specified
<i>gc_threshold_blocks</i>	integer	Garbage collection threshold (absolute number of blocks per plane). * If this parameter is specified then <i>gc_threshold</i> <u>should not</u> be specified

<i>chips</i>	integer	Number of flash chips in the device. → Visualization mode currently supports only 1 chip.
<i>planes</i>	integer	Number of planes in each chip.
<i>blocks</i>	integer	Number of blocks in each plane.
<i>pages</i>	integer	Number of pages in each block.

The logical capacity of the device is computed as follows:

$$(chips \times planes \times blocks \times pages \times (1 - overprovisioning)) - (planes \times pages)$$

Note: one block in each plane is reduced from the logical capacity to accommodate the active block.

8.2. Visual Parameters

Parameter	Values	Meaning
<i>show_counters</i>	yes/no	Yes: Display logical page numbers, block numbers and block states. No: Display only page pattern and color. → Use this option to define smaller pages and fit larger chips on the SSDPlayer display.
<i>speed</i>	integer	Simulation speed (number of frames per second). → The maximum speed depends on the machine you use to run SSDPlayer. If <i>speed</i> is higher than this maximum, the simulation speed will be the maximum possible on your machine.
<i>view_sample</i>	Integer	The sampling rate of the display. For a sampling rate of <i>X</i> , the display will be updated once every <i>X</i> operations.
<i>page_width</i>	integer	The width of the rectangle representing one physical page (pixels).
<i>page_height</i>	integer	The height of the rectangle representing one physical page (pixels).
<i>block_space</i>	integer	The space between blocks in the same plane (pixels).
<i>pages_in_row</i>	integer	The number of pages in a row within a block. → The number of rows will be set automatically to $pages / pages_in_row$
<i>blocks_in_row</i>	integer	The number of blocks in a row within a plane → The number of rows will be set automatically to $blocks / blocks_in_row$
<i>planes_in_row</i>	integer	The number of planes in a row within a chip → The number of rows will be set automatically to $planes / planes_in_row$
<i>font_type</i>	font	The font used in the display. The supported fonts are system dependent. → Windows: see fonts in "Control Panel\Appearance and Personalization\Fonts" → Linux: the 'fc-list' command lists all supported fonts. → Mac: see fonts in "/Library/Fonts" and "/System/Library/Fonts"
<i>caption_font_size</i>	integer	The font size used for titles

<i>control_font_size</i>	integer	The font size used for counters, graph titles and messages
<i>page_font_size</i>	integer	The font size used for page numbers
<i>active_color</i>	color	The frame color used for active blocks
<i>outer_bg_color</i>	color	The background color of the entire display
<i>intermediate_bg_color</i>	color	The background color within chips
<i>inner_bg_color</i>	color	The background color within planes and histograms
<i>border_color</i>	color	The color of border between displayed objects
<i>page_text_color</i>	color	The color used for page numbers
<i>control_text_color</i>	color	The color used for counters, graph titles and messages
<i>highlight_color</i>	color	The color used for highlighting user selections in menus etc.

Manager Parameters

By editing the manager parameters, you can:

1. Control additional specific display characteristics like colors and patterns.
2. Fine tune the behavior of the manager.

There are several parameters that are common for all managers. Other parameters are defined specifically for certain managers.

8.2.1. Defining Colors

Colors are a key feature of SSDPlayer. They are part of the definition of every manager. The colors are defined in the parameter file in RGB format: R represents the amount of **Red**, G represents **Green**, and B represents **Blue**.

For example, by setting `<r="0" g="0" b="255">` you get blue.

Here is a list of the RGB colors used in the default configuration. You can modify the colors used for each manager in the specific manager parameters.

Color	Used in	Default Use	Red	Green	Blue
White	All managers	Clean pages	255	255	255
Dark Yellow	Greedy manager	Written pages	255	220	100
Light green	Reusable manager	First writes	50	255	255
Light blue	Reusable manager	Second writes	50	150	255
Dark red	HotCold manager	Hot pages	60	120	255
Dark blue	HotCold manager	Cold pages	255	60	60
Gray	HotCold manager	Lukewarm pages	200	200	200
Dark gray	RAID managers	Data pages	175	175	175
Dark blue	RAID managers	Parity pages	100	150	255
Dark green	RAID 6 manager	Second parity pages	50	150	150

8.2.2. Parameters In Common for All Managers

These basic parameters are defined and used in the same way in all the managers:

Parameter	Values	Meaning
<i>name</i>	string	The name of the manager as it will be displayed in the manager menu.
<i>clean_color</i>	color	The color of clean (erased) pages. The default is white.

8.2.3. Greedy Manager

Parameter	Values	Meaning
<i>written_color</i>	color	The color of written pages. The default is dark yellow.

8.2.4. HotCold Manager

Parameter	Values	Meaning
<i>cold</i>	color	The color of written pages in the coldest temperature. The default is dark blue.
<i>hot</i>	color	The color of written pages in the hottest temperature. The default is dark red.
<i>intermediate</i>	color	The color of written pages in the intermediate (neutral) temperature. The default is gray. → Modify this parameter only if you wish to define your own color scale.
<i>min_temperature</i>	integer	The temperature of the hottest pages in the trace → Small \equiv Hot
<i>max_temperature</i>	integer	The temperature of the coldest pages in the trace → Large \equiv Cold
<i>partition</i>	integer	The coldest pages that should be written in the partition. → This parameter should appear for each partition, from hottest to coldest. The temperatures should be within the range defined by <i>min_temperature</i> and <i>max_temperature</i> . → The last partition should be for <i>max_temperature</i> . → If you wish to disable hot/cold separation define only the last partition. → SSDPlayer assumes that the partitions are defined correctly and in order. If they are not, it will not start.

Reusable Manager

The difference between 1st and 2nd writes is explained in Section 10.3.

Parameter	Values	Meaning
<i>first_write</i>	color	The color of pages written in first write. The default is light green.
<i>second_write</i>	color	The color of pages written in second write. The default is light blue.

8.2.5. HotCold-Reusable Manager

Parameter	Values	Meaning
<i>first_write</i>	color	The color of pages written in first write. The default is light green.
<i>second_write</i>	color	The color of pages written in second write. The default is light blue.
<i>temp_limit</i>	integer	The coldest pages that can be written in second writes

8.2.6. Reusable visualization Manager

Parameter	Values	Meaning
<i>first_write</i>	color	The color of pages written in first write. The default is light green.
<i>second_write</i>	color	The color of pages written in second write. The default is light blue.

8.2.7. Parameters In Common for All RAID Managers

Parameter	Values	Meaning
<i>data_color</i>	color	The color of the data pages. The default is dark gray.
<i>parity_color</i>	color	The color of the parity pages. The default is dark blue. → This parameter should appear once for each parity. E.g., in RAID 6, two parity colors are specified, and the default color of the second parity page is dark green.
<i>show_old_parity</i>	yes/no	Yes: highlight invalidated parity as a part of their stripe until they are erased. No: highlight only valid parity pages
<i>show_old_data</i>	yes/no	Yes: highlight invalidated data as a part of their stripe until they are erased. No: highlight only valid data pages
<i>stripe_frame_color</i>	color	The base color for determining page frames for highlighted stripes. The page
<i>stripe_frame_step</i>	color	The step color for determining page frames for highlighted stripes.

frame color for highlighted stripes is determined as follows. For strip n , the value of $x \in \{R, G, B\}$ in the frame is based on the value of x in *stripe_frame_color* and *stripe_frame_step* and computed by:
 $(\text{stripe_frame_color}(x) + n \cdot \text{stripe_frame_step}(x)) \bmod 256$,

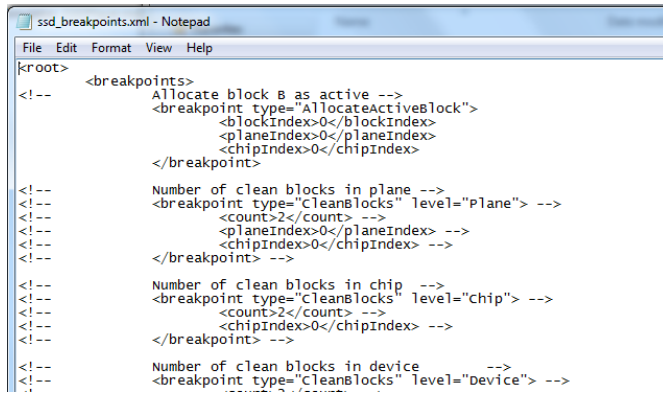
7.3.9 RAID Visualization Manager

Parameter	Values	Meaning
<i>stripe_size</i>	integer	The number of data pages in each stripe.

9. Editing the breakpoints configuration file

The `ssd_breakpoints.xml` file contains definitions of breakpoints that will be defined on startup. This option allows you to run several simulations with the same breakpoints, without having to manually define them in the 'Manage breakpoints' window. The file should be located in the 'resources' directory.

Define new breakpoints inside the `<breakpoints>` tag.



```

<!-- Allocate block B as active -->
<!-- <breakpoint type="AllocateActiveBlock" -->
<!--   <blockIndex>0</blockIndex>
<!--   <planeIndex>0</planeIndex>
<!--   <chipIndex>0</chipIndex>
<!-- </breakpoint>
<!--
<!-- Number of clean blocks in plane -->
<!-- <breakpoint type="CleanBlocks" level="Plane" --> -->
<!-- <count>2</count> -->
<!-- <planeIndex>0</planeIndex> -->
<!-- <chipIndex>0</chipIndex> -->
<!-- </breakpoint> -->
<!--
<!-- Number of clean blocks in chip -->
<!-- <breakpoint type="CleanBlocks" level="chip" --> -->
<!-- <count>2</count> -->
<!-- <chipIndex>0</chipIndex> -->
<!-- </breakpoint> -->
<!--
<!-- Number of clean blocks in device -->
<!-- <breakpoint type="CleanBlocks" level="device" --> -->
  
```

The type attribute determines the type of the requested breakpoint.

Some breakpoints may be defined for different levels, such as a specific chip or plane or the entire device. They are specified in the table below.

For your convenience, the default `ssd_breakpoints.xml` file contains commented definitions for all breakpoints as reference.

Breakpoint type	Available levels	Meaning
<i>AllocateActiveBlock</i>	N/A	Block B is allocated as active
<i>CleanBlocks</i>	Plane/Chip/Device	Number of clean blocks in plane/chip/device is C
<i>EraseBlock</i>	N/A	Erase block B
<i>EraseCountAnyBlock</i>	N/A	Any block reaches erase count C
<i>EraseCountBlock</i>	N/A	Block B reaches erase count C
<i>GCnthTime</i>	Plane/Chip/Device	Garbage collection is invoked for the i-th time in plane/chip/device
<i>HotColdPartitionHolds-PercentOfBlocks</i>	N/A	Partition P holds percent R of blocks
<i>PagesWritten</i>	Plane/Chip/Device	X logical pages are written in plane/chip/device
<i>ReusableBlockRecycled</i>	N/A	Reusable block B is recycled
<i>ReusableLevelBlocksPercent</i>	N/A	Reusable percent of blocks in write level
<i>WriteAmplification</i>	N/A	Write Amplification reaches W
<i>WriteLp</i>	N/A	Logical page L is written
<i>WritePp</i>	N/A	Physical page P is written
<i>WritesPerErase</i>	N/A	Writes per erase reach W
<i>HotColdWriteAmplification</i>	N/A	Partition P write amplification reaches W
<i>VictimBlockHasValidPages</i>	Plane/Chip/Device	Victim block has X valid pages
<i>ParityOverhead</i>	N/A	Parity overhead reaches W
<i>WriteInStripe</i>	N/A	Logical write in stripe P

10. Supported Managers

10.1. Greedy Manager

This is the simplest manager that includes only the basic features required by an FTL.

Page allocation. When a write request arrives for a page that is written for the first time, the page is written in the plane that has less valid blocks. When a write request arrives for a page that was written before, the old copy is invalidated and the page is written on the same plane.

Block allocation. In each plane, one block is the active block, and this is where new pages are written. When the active block is full, another clean block in the same plane is allocated as active and used for writing pages according to the requests in the workload.

Garbage collection. The *greedy* garbage collection algorithm erases the block with the minimum number of valid pages (*MinValid*), breaking ties according to the number of erasures: the “youngest” block, with the lowest number of block erasures, will be chosen. This prevents extreme cases of uneven wear, without performing active migration for wear-leveling. Garbage collection is activated whenever the number of clean blocks drops below the threshold. The threshold is computed by $gc_threshold \times blocks$.

Sample Trace. You can use the following trace from the traces directory of the SSDPlayer distribution with the default parameters in the configuration file:

- Uniform.trace
- Zipf.trace

10.2. HotCold Manager

This manager separates logical pages into partitions (within every plane) according to the frequency in which they are being written. We call this frequency *temperature*: frequently written pages are *hot* and rarely written pages are *cold*.

Special input. The HotCold manager requires a workload with temperature tags, so it must include the optional 6th field in each line. If this field is absent (for example, if you use the workload generator), all pages are assumed to have the hottest temperature. See Section 3.2.2.1.2 for more details on the trace format.

You should also edit the configuration file to tell the manager how you would like to separate the temperatures to be divided into partitions. See Section 3.2.2.1.2 for more details on the specific parameters of this manager.

Page allocation. When a write request arrives for a page that is written for the first time, the page is written in the plane that has less valid blocks. Within the plane, the page is written on the active block of the partition that this page’s temperature belongs to. When a write request arrives for a page that was written before, the old copy is invalidated and the page is written on the same plane, in the same partition.

Block allocation. In each partition, one block is the active block, and this is where new pages are written. When the active block in a partition is full, another clean block in the same plane is allocated to this partition.

Garbage collection. The HotCold manager uses the same greedy garbage collection algorithm as the Greedy manager. This means that the victim block for erasure is chosen regardless to which partition it currently belongs to. The consequence is that partition sizes are determined, implicitly, according to the number of writes with each temperature.

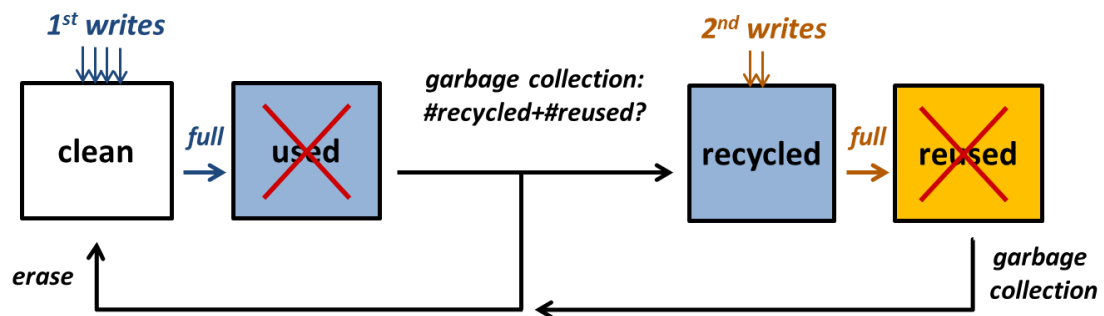
Sample Trace. You can use the following trace from the traces directory of the SSDPlayer distribution:

- Zipf_w_Temperatures.hotcold

You can use the default parameters in the configuration file, or you can add, remove and change partitions to see how this affects the behavior of the HotCold manager.

10.3. Reusable Manager

This manager is a simplified version of Reusable SSD³. Pages can be used for writing twice: the first write is the same as in the Greedy manager (and any standard SSD). The second write uses special codes for writing one logical page onto two physical pages whose content has been invalidated, but have not been erased. SSDPlayer does not perform any actual coding. It only simulates the amount of space required for each logical page, and its location.



Special input. No special input or parameters are required.

Page allocation. Logical pages are divided between planes like in the Greedy manager. If the active block in the plane is clean, then the logical page is written in first write. If the active block in the plane is recycled, then the logical page is written in second write on two physical pages in the active block. These pages must contain logical pages that have been previously invalidated.

Block allocation. Each plane has one active block. When the active block is full, if there are recycled blocks in the plane, one of them is allocated as active. If not, one of the clean blocks is allocated as active.

Garbage collection. The Reusable manager uses the greedy garbage collection algorithm with a slight modification. The victim block in the plane is the one with the lowest number of logical valid pages (regardless of whether they are written in first or second writes).

The victim block is recycled if the following three conditions hold:

- It has only been used for first writes.

³ Gala Yadgar, Eitan Yaakobi, Assaf Schuster. *Write Once, Get 50% Free: Saving SSD Erase Costs Using WOM Codes*. In proceedings of 13th USENIX Conference on File and Storage Technologies (FAST '15), February 2015.

- There are at least 2 clean blocks in this plane already.
- The number of recycled blocks in the plane is lower than the reserved value:
 $reserved = overprovisioning \times 2 \times blocks$

Otherwise, the victim block is erased.

Sample Trace. You can use the following trace from the traces directory of the SSDPlayer distribution with the default parameters in the configuration file:

- Uniform.trace
- Zipf.trace

10.4. HotCold-Reusable Manager

This manager adds hot and cold data separation to the Reusable manager. It writes in second write only hot logical pages. These pages are likely to be invalidated by the time the block has to be erased, so garbage collection will be more efficient.

Special input. The HotCold-Reusable manager requires a workload with temperature tags, like the one used by the HotCold manager. It must include the optional 6th field in each line. If this field is absent (for example, if you use the workload generator), all pages are assumed to have the highest temperature. See Section 3.2.2.1.2 for more details on the trace format.

You should also edit the configuration file to specify the temperature threshold for pages that will be written in second writes. Colder pages will be written in first writes. See Section 8.2.5 for more details on the specific parameters of this manager.

Page allocation. Logical pages are divided between planes like in the Greedy manager. If the page is hot and a recycled active block is allocated in this plane, then the logical page is written in second write on two physical pages in this active block. Otherwise, the logical page is written in first write on the clean active block.

Block allocation. Each plane has one or two active blocks. There is always a *clean active block* for first writes, and whenever possible, there is a *recycled active block* for second writes.

When the clean active block is full, another clean block in the same plane is allocated as active.

When the recycled active block is full, if there are recycled blocks in the plane, one of them is allocated as active. If not, the manager will search again for a recycled block in this plane the next time a page will be written.

Garbage collection. The HotCold-Reusable manager uses the same garbage collection algorithm used by the Reusable manager.

Sample Trace You can use the following trace from the traces directory of the SSDPlayer distribution:

- Zipf_w_Temperatures.hotcold

You can use the default parameters in the configuration file, or you can choose another temperature as the threshold for second writes to see how this affects the behavior of the HotCold-Reusable manager.

10.5. RAID Managers

These managers are based on the basic Greedy Manager, which was enhanced with RAID within the device's chips. They distinguish between data and parity pages, which are represented by different colors, and between the stripes pages belong to.

Parity pages are represented by pairs: <stripe,parity number> because there may be more than one parity page in each stripe. Thus, when a parity page is displayed on the main view, the number written on it represents the stripe it belongs to, and its color determines the parity number.

Note: while the logical page numbers are exclusive across the entire SSD, the parity page number is exclusive only within its stripe.

Page allocation. Write requests arrive for data pages only. Data pages are mapped to chips according to the chosen RAID level. The allocation is the same as in the Greedy manager. When a data page is written, the parity pages in its stripes are updated. This means the old parity pages are be invalidated, and the new parity pages are written on the same chip of the invalidated parity pages.

Block allocation. Same as in the Greedy Manager.

Garbage collection. Same as in the Greedy Manager.

Sample Traces. Stripes are determined for each page according to its logical page number. Thus, you can use the same traces used by the Greedy Manager from the traces directory of the SSDPlayer distribution, with the default parameters in the configuration file:

- Uniform.trace
- Zipf.trace

The RAID managers support write requests of several pages, according to the write size as specified in the write command. If more than one page is written in the same stripe, the stripe parities will be updated only once for each command.

The workload generator allows you to specify the maximum write size and the write size distribution (uniform/zipf) when initializing the simulation.

10.5.1. RAID 1 Manager

This manager implements RAID level 1 (mirroring): each chip is mirrored by another chip, i.e., the stripe size is one and there is one parity page in each stripe.

Note: RAID 1 requires an even number of chips.

8.5.2. RAID 5 Manager

This manager implements RAID level 5 (interleaved parity): the number of data pages in a stripe is computed by $number\ of\ chips - 1$. There is one parity in each stripe, and the chip holding the parity page in stripe s is computed by $number\ of\ chips - 1 - (s \% number\ of\ chips)$.

10.5.2. RAID 6 Manager

This manager implements RAID level 6 (erasure coding): the number of data pages in a stripe is computed by $number\ of\ chips - 2$. There are two parities in each stripe and the chip holding parity page i in stripe s is computed by $number\ of\ chips - 2 - (s \% number\ of\ chips) + i$.

10.6. Reusable visualization Manager

This manager processes the commands in the FTL command trace, and updates the device state and histograms. The display is similar to the one in the HotCold-Reusable manager: the same histograms are included, and the pages are colored according to their write level. In addition, during garbage collection the frame of the victim block is emphasized.

Note: this manager does not make any management decisions. It simply displays the decisions made by the manager of the system used to generate the trace.

Sample Trace. You can use the following traces from the traces directory of the SSDPlayer distribution with the default parameters in the configuration file:

- DiskSim_prn_0.log
- DiskSim_Zipf_w_Temp.log

10.7. RAID Visualization Manager

Similar to the Reusable visualization Manager, this manager processes the commands in the FTL command trace, and updates the device state and histograms. However, its semantics resemble those of the RAID Managers: it distinguishes between data and parity pages and includes the notion of stripes. The display is similar to the one in the RAID Managers: the same histograms are included, and the pages are colored according to their function as data or parity pages. Stripe highlighting is enabled and works in a similar manner.

Note: some FTL commands have a different syntax than in the Reusable visualization Manager (Section 3.2.2.2.2). The FTL command trace must be generated on a platform that implement striping:

The stripe size is determined in the configuration file, and the number of parity pages in each strip is determined by the number of *parity_color* parameters specified: instance *i* defines the color for parity *i*.

Note: while the logical page numbers are exclusive across the entire SSD, the parity page number is exclusive only within its stripe

Sample Trace. You can use the following traces from the traces directory of the SSDPlayer distribution with the default parameters in the configuration file:

- uni_1200_raid5.rlog
- uni_1800_raid6.rlog
- zipf_1200_raid5.rlog
- zipf_1800_raid6.rlog

A. FAQ

a. Q: I double click on SSDPlayer.jar but nothing happens. What's wrong?

A: First make sure you extracted the zipped distribution files, and are executing from a local directory on your machine. Then make sure you have Java Runtime Environment (JRE) installed on your machine.

b. Q: I run the HotCold manager with the Zipf workload, but all the pages are red. Why?

A: The workload generator does not generate temperature tags, so the HotCold manager assumes the default (hottest) temperature for all pages. You can use the Zipf_w_Temperatures.hotcold sample trace file instead.

c. Q: How did you create the online demos?

A: The demos on the [SSDPlayer home page](#) were recorded and edited with Camtasia®. We plan to add a recording option to SSDPlayer in the future.

d. Q: I have a cool idea how to improve SSDPlayer, can you do it?

A: SSDPlayer is an ongoing, open source project. Feel free to contact us to find out if someone is working on something similar. You can also go directly to the [development tree](#) and add features yourself!

e. Q: Where can I get the source code and Programmer's Guide?

A: You can find the full SSDPlayer distribution, source code, demos and guides on the [SSDPlayer home page](#).

f. Q: I found a bug, how do I report it?

A: You can use the contact information on the [SSDPlayer home page](#). We do our best to fix critical problems.

B. Copyright Notice

SSDPlayer Visualization Platform (Version 1.3.1)

Authors: Roman Shor, Gala Yadgar, Eitan Yaakobi, Assaf Schuster

Copyright (c) 2015, Technion – Israel Institute of Technology

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

C. Citation

If you use SSDPlayer for academic purposes, please cite the [MSST '17 paper](#) for reference:

Gala Yadgar, Roman Shor. *Experience from Two Years of Visualizing Flash with SSDPlayer*. In proceedings of 33rd International Conference on Massive Storage Systems and Technology (MSST 2017), May 2017, Santa Clara, CA.

```
@INPROCEEDINGS{SSDPlayer17,  
  author = {Yadgar, Gala and Shor, Roman},  
  title = {Experience from Two Years of Visualizing Flash with {SSDPlayer}},  
  booktitle = {33rd International Conference on Massive Storage Systems and Technology (MSST)},  
  year = {2017},  
}
```

The original [HotStorage '15 paper](#): Gala Yadgar, Roman Shor, Eitan Yaakobi, Assaf Schuster. *It's Not Where Your Data Is, It's How It Got There*. In proceedings of 7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '15), July 2015, Santa Clara, CA.

```
@INPROCEEDINGS{SSDPlayer15,  
  author = {Yadgar, Gala and Shor, Roman and Yaakobi, Eitan and Schuster, Assaf},  
  title = {It's Not Where Your Data is, It's How It Got There},  
  booktitle = {7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)},  
  year = {2015},  
}
```