Mosa Project Documentation

Feb 13, 2020

Contents

1 Introduction **1**
- 1.1 Current Status . . . 1
- 1.2 Getting Started . . . 2
- 1.3 Join the Discussion . . . 3
- 1.4 License . . . 3

2 Frequently Asked Questions (FAQs) **5**
- 2.1 What does MOSA stand for? . . . 5
- 2.2 Who can join? . . . 5
- 2.3 What kind of .NET runtime will be available? . . . 5
- 2.4 How is the Cosmos project different than MOSA? . . . 5
- 2.5 Are Cosmos and MOSA working together? . . . 6

3 MOSA Tools **7**

4 Mosa Compiler Tool **9**

5 MOSA Boot Image Tool **11**

6 Flash Drive Installation **13**

7 Tests **15**

8 Get Involved **17**
- 8.1 Contribution process . . . 17
- 8.2 Jobs . . . 17
- 8.3 Rules . . . 17

9 Authors **19**

10 License **21**

11 About this documentation **23**
- 11.1 RestructuredText with Sphinx directives . . . 23
- 11.2 Filenames . . . 23
- 11.3 Whitespaces . . . 23
- 11.4 Line length . . . 24
- 11.5 Headings . . . 24

11.6 Code blocks and text boxes 25
11.7 Links and references 25
11.8 Tables 26
11.9 Troubleshooting 26
11.10 References 27

CHAPTER 1

Introduction

MOSA is an open source software project that natively executes .NET applications within a virtual hypervisor or on bare metal hardware!

The MOSA project consists of:

- Compiler - a high quality, multithreaded, cross-platform, optimizing .NET compiler
- Kernel - a small, micro-kernel operating system
- Device Drivers Framework - a modular, device drivers framework and device drivers
- Debugger - QEMU-based debugger

Read our Frequently Asked Questions for more information about this project.

1.1 Current Status

The target platforms are:

- Intel X86/32-bit (stable)
- Intel X64 (in development)
- ARM v6 (in early development)

The MOSA compiler supports nearly all and object oriented non-object oriented code, including:

- Generic Code (example: List<T>)
- Delegates (static and non-static) and with optional parameters
- Exception Handling (try, finally, and catch code blocks)

The MOSA compiler seeks to provide high quality code generation using the following optimizations:

- Constant Folding and Propagation
- Strength Reduction optimization

- Dead Code Elimination
- Single Static Assignment (SSA)
- Global Value Numbering / Common Subexpession Elimination
- Sparse Conditional Constant Propagation
- Inlined Expansion
- Loop-Invariant Code Motion
- Block Reordering
- Greedy Register Allocation

1.2 Getting Started

1.2.1 Download

The MOSA project is available as a zip download or via git:

```
git clone https://github.com/mosa/MOSA-Project.git
```

1.2.2 Prerequisites

You will also need the following prerequisites:

Windows

Install any Visual Studio version 2018 or newer. All editions are supported including the fully-featured free Community Edition.

Note: The MOSA source code repository includes Qemu virtual emulator for Windows.

The CodeMaid Visual Studio Extension is strongly recommended for MOSA contributors.

Linux

Install Mono and Qemu.

The minimum supported version of Mono is 5.16.

If using the APT package manager you can use the following command to quickly set up QEMU and Mono

```
sudo apt-get -y install mono-devel qemu
```

Mac

Install Mono and Qemu.

1.2.3 Running

Windows

Double click on the "Compile.bat" script in the root directory to compile all the tools, sample kernels, and demos.

Next double click on the "Launcher.bat" script, which will bring up the MOSA Launcher tool (screenshot below) that can:

- Compile the operating system
- Create a virtual disk image, with the compiled binary and boot loader
- Launch a virtual machine instance (QEMU by default)

By default, the CoolWorld operating system demo is pre-selected. Click the "Compile and Run" button to compile and launch the demo.

1.3 Join the Discussion

Join us on Gitter chat. This is the most interactive way to connect to MOSA's development team.

1.4 License

MOSA is licensed under the New BSD License.

CHAPTER 2

Frequently Asked Questions (FAQs)

These are questions we're frequently answering in our IRC channel, the mailing list or which come up during conversations.

2.1 What does MOSA stand for?

Managed Operation System Alliance. It was an alliance with the SharpOS project to create a .NET based operating system. As operating systems share a lot of common groundwork MOSA aims to standardize interfaces in order to foster portability and to provide both projects with basic implementations of these interfaces.

2.2 Who can join?

Anyone who is interested in operating system or .NET development can join the alliance. You have to keep in mind our [[New BSD LicenselLicense]] for your contributions though.

2.3 What kind of .NET runtime will be available?

We are developing an entirely new Runtime as part of the MOSA effort. This runtime is developed along the CIL specifications published by ECMA. We are building our own runtime with pluggable algorithms in order to be very flexible and usable for research.

2.4 How is the Cosmos project different than MOSA?

Cosmos is designed to be an operating system toolkit plugin for Visual Studio. The Cosmos toolkit, once installed, integrates with Visual Studio in two significant ways. First, the toolkit introduces a new Cosmos project type that can launch and control the build process. Second, the toolkit integrates with Visual Studio's debugger and provides break

points and watches. The toolkit requires Microsoft's implementation of the .NET framework to compile a Cosmos operating system.

In comparison, MOSA has no dependencies on any Microsoft's applications including Visual Studio, the .NET framework or Windows operation system. MOSA can run on Windows, Linux or the Apple's OSX operating systems.

Another important difference is Cosmos compiles to Assembly ASM and uses Netwide Assembler, NASM, to finally compile to binary. MOSA compiles directly to binary and has its own linker implementation.

2.5 Are Cosmos and MOSA working together?

No; Cosmos and MOSA are independent projects.

CHAPTER 3

MOSA Tools

The MOSA project is developing a set of tools along with its core operating system components. This page provides quick links to descriptions of all MOSA tools.

- *MOSA Compiler*
- MOSA Explorer
- MOSA Debugger
- *MOSA Boot Image Tool*

CHAPTER 4

Mosa Compiler Tool

The Mosa Compiler can also invoked via Command Line:

```
Mosa.Tool.Compiler.exe -o Mosa.HelloWorld.x86.bin -a x86 --mboot v1 --x86-irq-methods␣
↪--base-address 0x00500000 mscorlib.dll Mosa.Plug.Korlib.dll Mosa.Plug.Korlib.x86.
↪dll Mosa.HelloWorld.x86.exe
```

Sample for launcher.json, when using Visual Studio Code:

```
"request": "launch",
"program": "${workspaceRoot}/../bin/Mosa.Tool.Compiler.exe",
"args": ["-o", "Mosa.HelloWorld.x86.bin", "-a", "x86", "--mboot", "v1", "--x86-irq-
↪methods", "--base-address", "0x00500000", "mscorlib.dll", "Mosa.Plug.Korlib.dll",
↪"Mosa.Plug.Korlib.x86.dll", "Mosa.HelloWorld.x86.exe"],
```

CHAPTER 5

MOSA Boot Image Tool

The tool several command line options. Sample:

```
Mosa.Tool.CreateBootImage.exe -o bin/Mosa.HelloWorld.x86.img --mbr Tools/syslinux/3.
↪72/mbr.bin --boot Tools/syslinux/3.72/ldlinux.bin --syslinux --volume-label␣
↪MOSABOOT --blocks 25000 --filesystem fat16 --format img Tools/syslinux/3.72/ldlinux.
↪sys Tools/syslinux/3.72/mboot.c32 Demos/unix/syslinux.cfg bin/Mosa.HelloWorld.x86.
↪bin,main.exe
```

The following options are supported:

Table 1: Arguments

Option	Arguments	Description
–volume	Volume Name	Set the volume name for the first partition
–blocks	# of Blocks	Set the number of 512-byte blocks
–filesystem	fat12/fat16/fat32	File System type
–format	img/vhd/vdi/vmdk/img	Disk Image Format
–syslinux		Patch disk image for syslinux
–mbr	Filename	Use file for Master Boot Record
–boot	Filename	Use file for Boot Record
	Filename[,Destination]	Include file in file system. Optional Destination will rename the file

The tool can create disk images for the following emulators:

Table 2: File formats

Emulator	File format
Virtual PC 2004/2007	.VHD
Virtual Server	.VHD
VMware	.VHD
VirtualBox	.VDI
QEMU	.IMG
Raw Image	.IMG

CHAPTER 6

Flash Drive Installation

While most of the development and testing of MOSA is done using virtualization software, MOSA does indeed boot on real hardware too.

Below are the steps for deploying a MOSA disk image to a USB flash drive:

> **Warning:** These instructions may vary slightly depending on your installation.

1. Create a MOSA disk image using the MOSA Launcher Tool.
2. Download the dd utility for Windows.
3. Copy the `dd.exe` executable to the build directory (usually a sub-folder under temp):

```
%TEMP%\MOSA
```

4. Open a command prompt window and change directory to the build directory.

```
cd %TEMP%\MOSA
```

5. Connect the USB key you wish to ERASE and install the MOSA image onto.

> **Danger:** Data on the USB flash drive will be lost!

6. Determine the device path for the USB flash drive.

Get a list all the block devices on your system by typing the command below. Find the one for the USB flash drive you just connected. Be careful, if you select or mistype the wrong device, you can corrupt your hard drive or other storage devices. Unless you understand these steps completely, do not proceed.

```
dd -list
```

7. Type the following and substitute the of= parameter with the device path found in the previous step.

```
dd of=\\?\Device\HarddiskX\PartitionX if=bootimage.img bs=512 -progress
```

8. Wait until all the blocks are written to the USB key before disconnecting it.
9. Now boot a PC or laptop with the USB flash drive connected!

The MOSA Compiler Framework is based on the idea of a pipeline. Every method is compiled by a method compiler, that specifies the pipeline to use for compilation. Such a pipeline may consist of any number of compilation stages. These stages can be grouped into various kinds:

- Compiler Frontends - Create an instruction stream from a source specific representation, such as CIL or Java byte code
- Transformation Stages - Transform the instruction stream between various representations
- Optimization Stages - Various stages intended to optimize the code to execute faster
- Register Allocation - Allocate architecture specific registers to operands used in the instruction stream
- Compiler Backends - Generate code from the intermediate and architecture specific representations

The compiler framework provides predefined pieces of this pipeline. Some parts, especially the code generation, are provided by the architecture specific stages, such as for the x86 platform.

Intermediate representations

The compiler framework uses a linear intermediate representation to transform the source program text into machine code. There are several levels of intermediate representations before code generation. These are:

- CIL - Common Intermediate Language
- IR - High-Level Intermediate Representation
- MIR - Machine specific Intermediate Representation

During compilation of an CIL method the instructions are represented by CIL instruction classes and moving forward, the linear instruction stream is modified to use instructions from the intermediate representation. In some cases an instruction from the intermediate representation can not be emitted directly to machine code and it is replaced by a sequence of machine specific instruction objects. The machine specific instruction classes are provided by the appropriate platform. There are other uses for machine specific instruction classes, but the main use is effective code generation.

CHAPTER 7

Tests

Execute the script `RunAllUnitTestsWithPause.bat` in the `/Tests` subdirectory.

The unit tests will take a few minutes to execute on modern PC. The results of all tests will be automatically displayed on the screen. The last line shows the total number of tests and failed tests, and the total time. Similar to the following:

```
Total Elapsed: 95.3 secs

Unit Test Results:
   Passed:   68164
   Skipped:  4
   Failures: 0
   Total:    68168

All unit tests passed successfully!
```

CHAPTER 8

Get Involved

We need your help!

8.1 Contribution process

If you are interested in supporting this project, join the project itself. Start downloading the code, look at the documentation provided and join our IRC channel (#mosa on Freenode).

8.2 Jobs

We need help to achieve our goals. You can either help us with documentation, community, by fixing bugs in our code base, fixing reported issues or taking on one of the open jobs. If you want to contribute, please make yourself heard on our IRC channel.

8.3 Rules

We are open to contributions from all areas. There are some exceptions though, which mean that we can not accept your contributions if you:

- have inspected proprietary code with Reflector, ildasm or similar tools and you plan on providing MOSA an implementation of that.
- have access to proprietary code related to operating system development, managed code or runtime implementations for managed environments.

Please be careful with any contribution you make regarding to patents, other open source licenses and any potential restrictions. We can only accept contributions compatiable with the New BSD License. MIT license is a compatible license, while GNU licenses are incomparable. Read more about our *License* policy.

CHAPTER 9

Authors

The MOSA project is a team effort and we want to recognize everyone's contribution. We list all the contributors to MOSA in our source code repository.

View the Credits.txt file.

If you have contributed to MOSA and your name is not on the list, please add your name and submit a pull request.

CHAPTER 10

License

MOSA is licensed under the New BSD License:

```
Copyright (c) 2008, MOSA-Project
All rights reserved.

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

* Redistributions of source code must retain the
  above copyright notice, this list of conditions
  and the following disclaimer.

* 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.

* Neither the name of Managed Operating System Alliance (MOSA)
  nor the names of its contributors may be used to endorse
  or promote products derived from this software without
  specific prior written permission.

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 OWNER 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
```

(continues on next page)

(continued from previous page)

```
THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
```

We will accept contributions submitted under the New BSD License or the compatible MIT/X11 License. Note: GNU licenses are not compatible with the BSD license.

CHAPTER 11

About this documentation

This document focuses on style-guide and a short reference. It is a kind of coding standards applied to documentation files. It is not about documentation content.

Table 1: Links to Documentation

Html, Online	http://docs.mosa-project.org/en/latest
PDF	http://readthedocs.org/projects/mosa/downloads/pdf/latest
Html, as Zip	http://readthedocs.org/projects/mosa/downloads/htmlzip/latest

11.1 RestructuredText with Sphinx directives

This documentation uses Python-sphinx[1], which itself uses reStructuredText[2] syntax.

11.2 Filenames

Use only lowercase alphanumeric characters and − (minus) symbol.

Suffix filenames with the `.rst` extension, so GitHub can render them.

11.3 Whitespaces

11.3.1 Indentation

Indent with 2 spaces.

Except:

[1] http://sphinx.pocoo.org/
[2] http://docutils.sourceforge.net/rst.html

- `toctree` directive requires a 3 spaces indentation.

11.3.2 Blank lines

Two blank lines before overlined sections, i.e. before H1 and H2. One blank line before other sections. See *Headings* for an example.

One blank line to separate directives.

```
Some text before.

.. note::

  Some note.
```

Exception: directives can be written without blank lines if they are only one line long.

```
.. note:: A short note.
```

11.4 Line length

Technically, there's no limitation. But if possible, limit all lines to a maximum of 120 characters.

11.5 Headings

Use the following symbols to create headings:

1. # with overline
2. * with overline
3. =
4. -
5. ^
6. "

As an example:

```
##################
H1: document title
##################

Introduction text.


*********
Sample H2
*********

Sample content.
```

(continues on next page)

(continued from previous page)

```
**********
Another H2
**********

Sample H3
=========

Sample H4
---------

Sample H5
^^^^^^^^^

Sample H6
"""""""""

And some text.
```

If you need more than heading level 4 (i.e. H5 or H6), then you should consider creating a new document.

There should be only one H1 in a document.

Note: See also Sphinx's documentation about sections[3].

11.6 Code blocks and text boxes

Use the `code-block` directive **and** specify the programming language. As an example:

```
.. code-block:: python

  import this
```

Text boxes:

```
.. note::

   Note (blue box). possible values: attention, caution, danger, error, hint,␣
↪important, note, tip, warning, admonition.
   Every type has its own color.
```

will loook like:

Note: Note (blue box). possible values: attention, caution, danger, error, hint, important, note, tip, warning, admonition. Every type has its own color.

11.7 Links and references

Use links and references footnotes with the `target-notes` directive. As an example:

[3] http://sphinx.pocoo.org/rest.html#sections

```
#############
Some document
#############

Link without Reference: `Example <http://www.example.com>`__

Some text which includes links to `Example website`_ and many other links.

`Example website`_ can be referenced multiple times.

(... document content...)

And at the end of the document...

**********
References
**********

.. target-notes::

.. _`Example website`: http://www.example.com/
```

11.8 Tables

Table as CSV

```
.. csv-table:: Title of CSV table
  :header: "Column 1", "Column 2", "Column 3"

  "Sampel Row 1", Cell, Cell
  "Sampel Row 2", Cell, "Cell with multiple Words"
```

You can skip quotes, of cell content conains only a single word

Table as flat list

```
.. list-table:: Title of table as flat list
  :header-rows: 1

  * - Column 1
    - Column 2
    - Column 3
  * - Row 1
    - Cell
    - Cell
  * - Row 2
    - Cell
    - Cell
```

`:header-rows:` defines the number of header rows. Skip this line, if you do not need a header.

11.9 Troubleshooting

Why is my document not linked within the table of contents?

- put the filename into index.rst. Omit the `.rst` extension.
- The document requires at least one section. Section names are the label that are used for the table of content.

After commiting, the documentation will not update

- The build process of the documentation takes round about 1-2 minutes.
- You can check the status here: Builds
- Check the status if the build fails

Documentation is updating, but some content is missing or malformed

- There might be some parsing errors or warnings. Go to Builds and click on `Raw view`. Check the build output for warnings and fix them.

11.10 References

- https://sphinx-rtd-theme.readthedocs.io/en/latest/demo/demo.html
- http://www.ericholscher.com/blog/2016/jul/1/sphinx-and-rtd-for-writers/