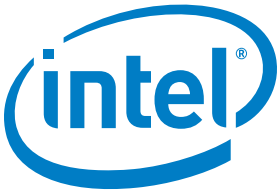

Intel® Xeon Phi™ Processor x200 Offload Over Fabric

User's Guide

June 2016

US

Revision: 1.0

World Wide Web: http://www.intel.com

Legal Disclaimer

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at http://www.intel.com/ or from the OEM or retailer.

No computer system can be absolutely secure.

Intel, Xeon, Xeon Phi and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Intel does not warrant or guarantee the performance or compatibility of third party commercial products. Reference in this site to any specific commercial product, process, or service, is for the information and convenience of the public, and does not constitute endorsement, or recommendation by Intel.

*Other names and brands may be claimed as the property of others.

Table of Contents

Table of Figures

List of Tables

Revision History

Date	Revision	Description
February 2016	1.0	Initial official version
February 2016	0.5	Draft revision for review.
June 2016	1.0	Public version

§

1 Introduction

This document pertains the Offload over Fabric technology, which allows users to offload workloads to numerous interconnected compute nodes.

This guide provides an overview of the OOF technology, shows how to install and configure it, and how to make use of its features.

Please note that the OOF technology was designed for systems containing Intel® Xeon Phi™ x200 processors.

1.1 Terminology

OFED	OpenFabrics Enterprise Distribution
OFI	OpenFabrics Interfaces
OOF	Offload over Fabric
COI	Coprocessor Offload Infrastructure

1.2 Notational Conventions

This document uses the following notational conventions.

yum install	Commands and their arguments in prose sections are *italicized*.
OFFLOAD_NODES is *node0*	Configuration parameter names are *italicized* when they appear in prose sections.
/tmp/intel-coi	Files and directories in prose sections are *italicized.*
`COURIER text`	Code and commands entered by the user. A backslash symbol: \ indicates that command is continued in the next line.
`Italic COURIER text`	Terminal output by the computer.
`"[host]$"`	Command entered on the offload host server with user or root privileges.
`"[host]#"`	Command entered on the offload host server with root privileges.
`"[target]$"`	Command entered on the offload target server with user or root privileges.
`"[target]#"`	Command entered on the offload target server with root privileges.

1.3 Reference Documents

Programming and Compiling for Intel® Many Integrated Cores Architecture	https://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture
Intel® Xeon Phi™ Processor Software User's Guide	Available with Intel® Xeon Phi™ Processor Software
The OpenFabrics Enterprise Distribution (OFED™)	https://www.openfabrics.org/index.php/openfabrics-software.html
COI documentation	*/usr/share/doc/intel-coi-<version>*

§

2 Offload over Fabric Overview

The Intel® Xeon Phi™ coprocessors x100 introduced the offload programming model, allowing users to offload workloads over PCIe. With the introduction of the Intel® Xeon Phi™ x200 processor, this programming model is implemented as Offload over Fabric (OOF) and enables offloading to compute nodes connected within a high-speed network. Communication with the networking layer in is realized by the Open Fabric Interface API (OFI). Refer to the *Programming and Compiling for Intel® Many Integrated Core Architecture* article for more information on the available programming models.

The Intel® C/C++ and Fortran Compilers support offloading directives in the source code. This feature allows the application developer to specify which parts of the program will be offloaded to the Intel® Xeon Phi™ processor-based nodes.

The compilers and Offload over Fabric programming model use the Intel® Coprocessor Offload Infrastructure (Intel® COI) to perform offloading. Advanced users may also use the Intel® COI API directly. Please refer to Section 5 for details.

The Offload over Fabric software is part of the Intel® Xeon Phi™ processor software package. Additional information on it can be found in the *Intel® Xeon Phi™ Processor Software User's Guide*.

Although Offload over Fabric can coexist alongside other programming models, it was especially designed to be used in MPI applications in HPC cluster environment.

Figure 1 shows an example OOF application architecture and its expected execution flow. The presented system consists of several Intel® Xeon® processor-based servers hosting an MPI application (offload host servers). Massively parallel sections of the application can be offloaded to the Intel® Xeon Phi™ processor-based servers (offload target servers) using compiler offloading. This model takes advantage of the heterogeneous nature of the cluster while maintaining clear distinction in the code between the MPI-based homogenous part and the offloaded code.

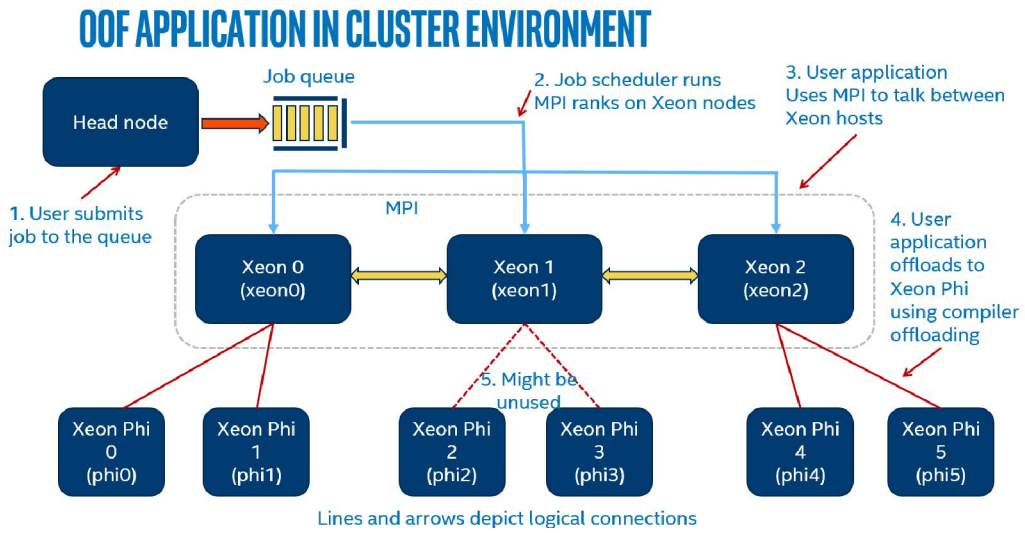

Figure 1 Example OOF application

§

3 Offload over Fabric Installation

Instructions in this section show how to install and uninstall the Offload over Fabric software.

Note: It is strongly recommended to read through this chapter before actually proceeding with installation to ensure that all required components and facilities are available. It is also strongly recommended that these installation steps be performed in the order they are presented.

3.1 Prerequisites

Offload over Fabric requires both Intel® Xeon® processor-based host systems and Intel® Xeon Phi™ processor-based target systems to be configured and able to communicate over fast fabric interconnection. The target system must contain at least one Intel® Xeon Phi™ processor.

3.1.1 Operating System

Offload over Fabric has been validated against specific versions of Red Hat* Enterprise Linux* (RHEL*) operating system. Table 1 and Table 2 list supported versions of the operating system for the host and target systems.

Table 1 Supported Host Operating Systems

Intel® Xeon Phi™ Processor Software Versions	Supported OS Versions	Kernel Version
1.3.3	Red Hat* Enterprise Linux* 64-bit 7.2	3.10.0-327.el7.x86_64

Table 2 Supported Target Operating Systems

Intel® Xeon Phi™ Processor Software Versions	Supported OS Versions	Kernel Version
1.3.3	Red Hat* Enterprise Linux* 64-bit 7.2	kernel-3.10.0-327.el7.xppsl_1.3.3

To obtain the version of the kernel running on the host, execute:

```
[host]$ uname –r
```

Note: Access to standard distribution packages and repositories is required to install some of the Intel® Xeon Phi™ processor software packages. Disabling any standard repository may lead to *failed dependencies* issues. To get more information please refer to the information provided in your Operating System documentation.

3.1.2 Root Access

Many of the tasks described in this document require administrative access privileges (i.e. root access). Verify that you have such privileges to the machines which you will configure.

The use of *sudo* to acquire root privileges should be done carefully because its use may cause subtle and undesirable side effects. *Sudo* might not retain the non-root environment of the caller. This could, for example, result in use of a different *PATH* environment variable than expected, ending up with execution of the wrong code.

When *su* is used to become root, the non-root environment is (mostly) retained. (*HOME, SHELL, USER, LOGNAME* are reset unless the *-m* switch is given. See the *su* man page for details).

3.2 OpenFabrics Enterprise Distribution (OFED)

OpenFabrics Interface (OFI) is part of the OpenFabrics Enterprise Distribution (OFED). This API was first included as a part of the OpenFabrics Alliance (OFA) OFED 3.18. It is now also a part of other OFED distributions or operating systems. OFI source code can be downloaded from Libfabric OpenFabrics website. A version obtained from the website, compiled, and installed on the system is referred to as OFA OFI or OFA libfabric in this document. It is assumed that OFI (libfabric) is installed on the host system and the target system. Use the OFI test suite to verify the connectivity between the two systems.

Offload over Fabric has been validated against specific versions of OFED software. Table 3 lists the supported versions of the fabric hardware, OFED, and OFI.

Table 3 Matrix of validated fabric hardware, OFED, and OFI versions

Fabric hardware	OFED	OFI
Mellanox* MT4099	MLNX_OFED-3.2-2.0.0	OFA libfabric-1.3.0
Intel® Omni-Path HFA 100 Series 1 Port PCIe 16x	Intel 10.1.0.0.110	OFA libfabric-1.3.0

3.3 Installation

Uninstall previous version of the Intel® Xeon Phi™ processor software prior to installing a new one. Refer to Section 3.3.3 for instructions.

3.3.1 Get the Intel® Xeon Phi™ processor software distribution

Offload over Fabric software is distributed with the Intel® Xeon Phi™ processor software. The software package releases for the host and target systems are available in separate tar files for each supported OS (*xppsl-<version>-offload-host-<os>.tar* and *xppsl-<version>-<os>.tar*).

After downloading, extract the Intel® Xeon Phi™ processor software offload host package on your Intel® Xeon® processor-based server (offload host):

```
[host]$ tar xvf xppsl-<version>-offload-host-<os>.tar
[host]$ cd xppsl-<version>-offload-host
```

The Intel® Xeon Phi™ processor software offload target package should be downloaded and extracted on your Intel® Xeon Phi™ processor-based server (offload target):

```
[target]$ tar xvf xppsl-<version>-<os>.tar
[target]$ cd xppsl-<version>
```

3.3.2 Offload over Fabric Installation

Perform the following steps on both systems:

- Red Hat* Enterprise Linux*:

  ```
  [host]$ cd rhel<os-version>/
  ```

  Install RPMs:

  ```
  [host]# yum install x86_64/*.rpm devel/x86_64/*.rpm
  ```

3.3.3 Offload over Fabric Uninstallation

1. To check for a previously installed version of the Offload over Fabric package execute:

   ```
   [host]$ rpm -qa | grep xppsl-coi
   ```

2. Packages that correlate with Offload over Fabric will be listed and must be uninstalled:

   - Red Hat* Enterprise Linux*:

   ```
   [host]# yum remove [package-name]
   ```

§

4 Offload over Fabric Configuration

4.1 System configuration for OOF

4.1.1 Host system configuration

No special system configuration is required on the offload host system. It is recommended, however, to increase the maximum number of opened file descriptors to enable more complex workloads:

```
[host]# ulimit -n 10024
```

4.1.2 Target system configuration

The OOF runtime uses a virtual file system features to perform tasks connected to the offload process and memory management. The runtime expects the */tmp/intel-coi* directory to be mounted, otherwise it will attempt to mount it during initialization and return an error if it is not possible. The *size* parameter of the mounted file system is treated by the offloading runtime as a limit of memory that can be allocated by offloading processes on a target device. Perform the following steps to mount the required file system:

```
[target]# mkdir -p /tmp/intel-coi
[target]# mount -t tmpfs -o size=32g tmpfs /tmp/intel-coi
[target]# chmod 1777 /tmp/intel-coi
```

Add the following line to the */etc/fstab* file to allow all users to mount the file system:

```
tmpfs /tmp/intel-coi tmpfs \
defaults,user,exec,size=32g,mode=1777 0 0
```

To allow the offloading processes to use huge pages, which may improve performance, mount another virtual file system on the target node:

```
[target]# mkdir -p /tmp/intel-coi/COI2MB
[target]# echo 4000 > /sys/kernel/mm/hugepages/ \
hugepages-2048kB/nr_overcommit_hugepages
[target]# mount none -t hugetlbfs /tmp/intel-coi/COI2MB
[target]# chmod 1777 /tmp/intel-coi/COI2MB
```

The following line can be added to the */etc/fstab* to allow all users to mount the *hugetlbfs*:

```
hugetlbfs /tmp/intel-coi/COI2MB hugetlbfs \
defaults,user,mode=1777 0 0
```

The OOF runtime will try to mount the *hugetlbfs* file system if it is not mounted and will use */tmp/intel-coi/COI2MB* as a mount point. It will return an error if the file system cannot be mounted.

It is recommended to increase the maximum number of file descriptors available for offloading processes:

```
[target]# ulimit -n 10024
```

4.2 Configuration of offloading application

Offload over Fabric uses *OFFLOAD_NODES* and *OFFLOAD_DEVICES* environment variables to configure nodes available for the offloading operation from the offload host. The value of *OFFLOAD_NODES* variable is a comma-separated list of nodes' names.

The *OFFLOAD_DEVICES* can be used to configure which nodes (listed in the *OFFLOAD_NODES* variable) will be available for offloading from a particular user process. This variable is a comma-separated list of integer values. Each value is an index of a node specified in the *OFFLOAD_NODES* variable. User can specify a maximum of 8 indices in the *OFFLOAD_DEVICES* variable. If *OFFLOAD_DEVICES* is not set, OOF runtime will try to use all the nodes from *OFFLOAD_NODES*, but it will return an error if the number of nodes is greater than 8.

Example

To establish a configuration shown in Figure 1 the user can choose one of two options.

Option 1 (shown in Figure 2) uses the same value of *OFFLOAD_NODES* variable for every MPI rank and specifies offload targets using *OFFLOAD_DEVICES* variables. Option 2 (shown in Figure 3) uses only *OFFLOAD_NODES* variables to achieve the same goal (*OFFLOAD_DEVICES* is not set). The *OFFLOAD_NODES* variable is set to a different value for each MPI rank.

Those options are equivalent, the user should choose the one that better suits the needs of a particular application.

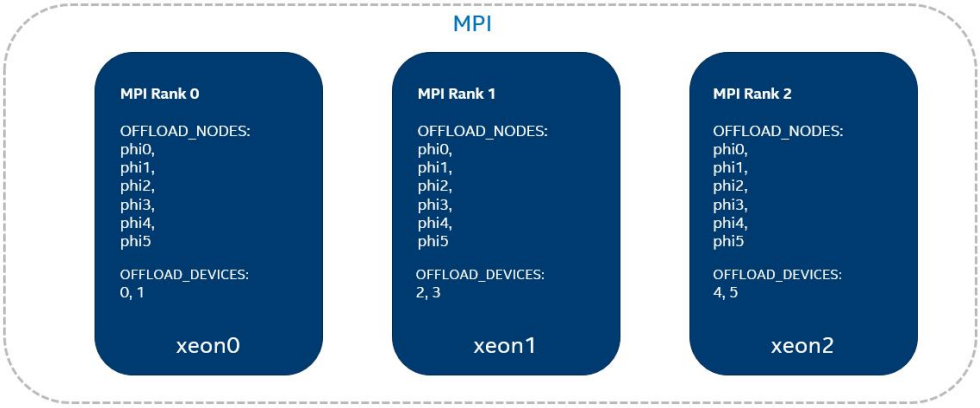

Figure 2 Example OOF configuration - Option 1

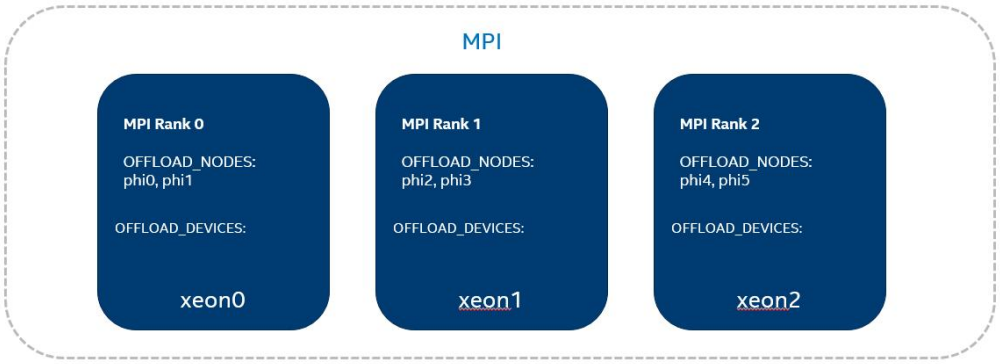

Figure 3 Example OOF configuration - Option 2

§

5 Coprocessor Offload Infrastructure

5.1 Overview

Intel® Coprocessor Offload Infrastructure (Intel® COI) is a module that provides the following functionalities:

- Enumeration of the offloading engine.
- Management of user processes and their dependencies (shared libraries).
- Memory management and data transfers between offload target and host.
- Execution of user (offloaded) code on the target host.
- Management of dependencies between functions executed on the target, buffers used by those functions, and implicit and explicit buffer data transfers.

COI plays a critical role in the offloading process, as it abstracts most of the details of interconnectivity between the host and target. It implements reach API that can be directly used to perform offloading by the user or by the compiler runtime.

5.2 Directories

By default, COI is installed in multiple locations on the offload host. These locations include:

/usr/share/doc/intel-coi-<version>	Documentation, including this document, the COI API Reference Manual, and the release notes.
/usr/include/intel-coi	Include files needed to build COI applications.
/usr/share/doc/intel-coi-<version>/tutorials	Simple code samples that can be helpful in learning how to write COI applications.
/usr/bin	COI tools to assist in development.
/usr/lib64	COI shared libraries needed to build COI applications. Four different versions of each of these libraries are provided with all combinations of host or device and debug or release.
/opt/mpss/x200/minsdk	Compatibility libraries used for building and compiling code that should be executed on target devices. Those libraries have no dependencies and user applications should use them for linking only. It is assumed that full versions of libraries are installed on target systems.

5.3 Building and Running Tutorials

The COI release comes with a number of simple code tutorials, including the following:

hello_world	Shows an application with no pipelines that uses the COI I/O proxy to print output on the source. This type of usage can be suitable for users who would like to run a remote application.
coi_simple	Shows the use of a single pipeline and run function.
buffers_with_pipeline_function	Shows simple buffer operations.
multiple_pipeline_implicit	Shows how to use implicit buffer dependencies to coordinate between multiple pipelines.
multiple_pipeline_explicit	Shows how to use explicit dependencies to coordinate between multiple pipelines.
user_event	Shows how to utilize user-created barriers for synchronization between sink and source.
buffer_references	Illustrates the use of buffer reference counting to implement out-of-order asynchronous operations.

Each tutorial directory contains the source code and makefiles needed to build binaries. The makefiles are configured to use *gcc*.

The tutorials can be built by copying a tutorial's entire directory into a user's directory, and issuing the *make* command:

```
[host]$ cp –r /usr/share/doc/ \
intel-coi-<version>/tutorials/hello_world .
[host]$ cd hello_world
[host]$ make
```

After building the tutorial, it can be executed by running the host executable:

```
[host]$ cd debug
[host]$ ./hello_world_source_host
1 engines available
Hello from the sink!
Press enter to kill the sink process
[host]$
```

5.4 Using *coitrace* to assist with debugging

The *coitrace* tool is included in the installation package. This trace utility functions similarly to Unix*-style tools like *strace* and shows all of the COI API invocations and input parameters. This can be helpful in identifying what COI commands are being executed for tracing and debugging. To see a complete list of options run:

```
[host]$ coitrace –h
```

To use it run:

```
[host]$ cointrace <application>
```

For example executing the *hello_world* through *coitrace* will produce the following output:

```
[host]$ coitrace ./hello_world_source_host

COIEngineGetCount [ThID:0x7f905bcf17c0]
        in_DeviceType = COI_DEVICE_MIC
        out_pNumEngines = 0x7fff5ae1f180 0x00000001 (hex) : 1
(dec)

1 engines available
COIEngineGetHandle [ThID:0x7f905bcf17c0]
        in_DeviceType = COI_DEVICE_MIC
        in_EngineIndex = 0x00000000 (hex) : 0 (dec)
        out_pEngineHandle = 0x7fff5ae1f1b0 0x7f905b8632c0

Got engine handle
COIProcessCreateFromMemory [ThID:0x7f905bcf17c0]
        in_Engine = 0x7f905b8632c0
        in_pBinaryName = hello_world_sink_mic
        in_pBinaryBuffer = 0x7f905bcf9000
        in_BinaryBufferLength = 0x00000000000021d4 (hex) :
8660 (dec)
        in_Argc = 0
        in_ppArgv = 0
        (bool) in_DupEnv = false
        in_ppAdditionalEnv = 0
        (bool) in_ProxyActive = true
        in_Reserved = (nil)
        in_BufferSpace = 0x0000000000000000 (hex) : 0 (dec)
        in_LibrarySearchPath = (nil)
        in_FileOfOrigin = hello_world_sink_mic
        in_FileOfOriginOffset = 0x0000000000000000 (hex) : 0
(dec)
        out_pProcess = 0x7fff5ae1f1a0 0x27af290

COIProcessCreateFromFile [ThID:0x7f905bcf17c0]
        in_Engine = 0x7f905b8632c0
        in_pBinaryName = hello_world_sink_mic
        in_Argc = 0
        in_ppArgv = 0
        (bool) in_DupEnv = false
        in_ppAdditionalEnv = 0
        (bool) in_ProxyActive = true
        in_Reserved = (nil)
        in_BufferSpace = 0x0000000000000000 (hex) : 0 (dec)
        in_LibrarySearchPath = (nil)
        out_pProcess = 0x7fff5ae1f1a0 0x27af290
```

```
Sink process created, press enter to destroy it.
Hello from the sink!

COIProcessDestroy [ThID:0x7f905bcf17c0]
        in_Process = 0x27af290
        in_WaitForMainTimeout = -1
        (bool) in_ForceDestroy = false
        out_pProcessReturn = 0x7fff5ae1f170
Sink process returned 0
Sink exit reason SHUTDOWN OK
```

5.5 *micnativeloadex* for remote execution

The *micnativeloadex* utility included in the package can be used to remotely execute native code from a host console. This tool works similarly to ssh, which can be used to start a remote process, but does not require logging in, will automatically transfer dependent libraries, and will redirect console IO back to the host console. Internally *micnativeloadex* uses COI so it follows the same library loading rules and requirements for the *SINK_LD_LIBRARY_PATH* environment variable. *COI_OFFLOAD_NODES* and *COI_OFFLOAD_DEVICES* variables should be used to choose target machine as described in Section 4.

5.6 Troubleshooting

This section presents several techniques that can be performed to fix or mitigate problems with the OOF. If for some reason following these steps does not resolve the occurring problems, or if some of these steps need to be done consistently, please file a defect or contact your Intel support representative.

5.6.1 COIEngineGetHandle Hangs

COIProcessCreate call may hang for a number of reason. Initially check whether the Linux* OS on the target system is active and accessible. SSH can be to verify this:

```
[host]$ ssh 172.31.1.1
```

The user should be able to log into the target system without using password. If this operation was successful, use the *ssh* session to validate whether the virtual file systems required for COI to run are mounted and accessible to the user (see Section 4.1).

The user can also verify the connectivity and configuration correctness using OFI tests, which can be downloaded from https://github.com/ofiwg/fabtests.

5.6.2 COI API Returns an Error Code

Sometimes providing an accurate error code does not clarify the source of a problem. For example, if *COIProcessCreateFromFile* returns *COI_MISSING_DEPENDENCY*, it indicates that a dynamic library needed by the executable could not be found in the host or target file systems. However, if the debug version of the COI library is used, more information can be learned by looking at the automatically generated log file.

This file is named <*executable*>*.coilog*, where <*executable*> is the name of the source executable. The log file is located in the directory the user was in when the application was launched.

§