Open|SpeedShop (O|SS)
Build and Installation Guide
Version 2.4.0
September 30, 2018

Introduction

This document supports the Open|SpeedShop (O|SS) 2.4 release and subsequent updates.

A new and preferred method of building and installing O|SS on laptops, desktops, and clusters using the spack build/package manager is now available. Please see the Spack Build and Install guider for more information. There is also a section (How to use SPACK to build O|SS) on building with Spack below. However, if that method does not work for your needs, this document describes the native install methods that have been used to build O|SS for some time now.

O|SS release 2.4 comes with an install script (install-tool) that will build all the supporting components (also referred to as the krellroot) and O|SS itself, in one step. Or the builder can invoke the script once to build the supporting component set (krellroot) and another time to build O|SS.

The install script relies on arguments to the script instead of environment variables like the previous install script (install.sh). With install-tool, the environment variables are set internally to the script so that the builder is not required to keep track of the numerous build environment variables that O|SS uses. In addition, more build environment variables are set by default, requiring less builder-required action.

The new install script also supports building the Component Based Tool Framework (CBTF) for the new O|SS CBTF instrumentor version that is described below.

What is CBTF and how does it relate to O|SS?

CBTF stands for Component Based Tool Framework and is a key scalability feature for the new version of O|SS that uses CBTF as its instrumentation methodology. The O|SS team has completed the base functionality for the process of changing the mechanisms that gather the performance data and transfer the performance data from the application, where it was gathered, to the O|SS client where it is stored into an SQLite database file. In order for O|SS to operate efficiently at high rank and thread counts it could not continue to write raw data files to disk and then read them up at the end of the application execution. This is a bottleneck, which is too time-consuming at high processor counts. So, CBTF version of O|SS using the new CBTF gathering and transfer mechanisms which transmit raw performance data to the client tool without writing files. This will be significantly faster than the offline version. CBTF can also be used independently from O|SS to rapidly prototype tools. Please see the CBTF wiki at: https://github.com/OpenSpeedShop/cbtf/wiki for more information.

A new O|SS build method: SPACK

General information about the spack build/package manager can be found at: https://github.com/spack/spack/wiki. Spack is a flexible package manager for HPC. It is intended to let you build for many combinations of compiler, architectures, dependency libraries, and build configurations, all with a friendly, intuitive user interface. Spack draws some ideas from Homebrew and Nix, with its own pure Python package format, extensive internal support for managing software dependency graphs, and powerful command line syntax.
The O|SS team has been working on creating spack package files for building O|SS under the spack build system. The O|SS build under spack is generally working for builds on clusters and pc/laptop type systems. Cray support is not ready yet.

Once spack is downloaded and set up, to build O|SS can be as easy as this command:

```
spack install openspeedshop
```

See the spack section "How to use SPACK to build O|SS" below for more specific information.

Some Initial Notes for the install-tool build method

- NOTE: with version 2.3.1(and beyond) the O|SS build will build a cmake version that works to build our software (version 3.2.2). This cmake will be used whenever the install-tool is invoked. In addition to some of the open source components using the "cmake" build tools to build their components, the O|SS team is now using cmake as the build tool for O|SS, and the CBTF components that the team supplies. So, cmake is now a build pre-requisite package.
- NOTE: If you have install-tool build scripts, for version 2.2 and 2.3, please replace "**--build-cbtf**" with "**--build-cbtf-all**", as we are now using cmake as our underlying build tool instead of GNU autotools. With cmake we are building cbtf, cbtf-krell, cbtf-argonavis, and cbtf-lanl separately. When the GNU autotools were used we built all of the CBTF components within one build. Now the CBTF components are individually built (four separate builds) when "--build-cbtf-all" is used.
- **Caution: boost-1.60 to 1.62 versions cause compile errors when building the cbtf components and the new cuda GUI. These are known boost problems. Please use a different version of boost (1.66 and above) when building O|SS.**
- **Caution: When installing a new version, please install into an empty/clean directory,** as we do not clean the install directories out upon install-tool invocation. You have end up with multiple copies of the same component, which can lead to execution issues.
- We refer to root or krell-root packages or building the krell-root. What does this mean? These are the external packages that O|SS and/or CBTF uses as

part of the tool. Sometimes one or several of these external packages are not available on the platform that O|SS and/or CBTF is being installed on, so we have provided the ability to build and install them. The root/external packages are provided as a convenience. You could install these packages without the use of the krell-root on the install system, if desired.

- The CBTF version of O|SS is now the default and official version of O|SS, as of the 2.3 release. The offline version is now legacy, but the offline capabilities have been built into the CBTF version and is usable on most platforms (not Blue Gene systems).
- You can install O|SS in non-standard locations (root permission is not needed)
- Versions of O|SS 2.3 and beyond can be installed using the information in this document.
- **NOTE: O|SS is normally built with the GNU compilers, but now building with the Intel compiler is supported. Options to the install-tool have been created to allow building with either the gnu or intel compilers. Using the gnu compiler is the default.**
    - --build-compiler gnu
    - --build-compiler intel
- O|SS cannot be built with PGI, Cray, or other compilers. Even though O|SS only builds with GNU or Intel compilers, O|SS supports performance analysis of applications built with a wide variety of compilers, including GNU, Intel, PGI, Pathscale, and Cray.
- There are a number of development environment packages that are required if you are starting from a clean install on your desktop or laptop. See the Prerequisite Packages section for the list of packages needed on non-development systems. Most laboratory systems normally have all the required packages installed, because most tools need these packages to build and execute successfully.
- There is now an option to build the krell root components that might be missing on the system or needed separately from O|SS. The builder can build the krell root then use that installation to build CBTF and/or O|SS. Or, you can choose to build all components into the same install directory.
- The install-tool script supports building both O|SS modes of execution:
    - cbtf – Performance information is transported from the application across a TCP/IP network as it is collected and stored in the O|SS database on the fly.
    - NOW LEGACY: offline – Write raw performance information to shared file system disk files while the application is executing. Then convert the raw data files to an O|SS database file when the application completes execution.
- install-tool script options for building O|SS are listed and described below:
    - To build the new CBTF version use three invocations of the install-tool script individually, in the order listed.
        - “--build-krell-root”
        - “--build-cbtf-all”

        - "--build-oss"
    - NOW LEGACY: To build the offline version use two invocations of the install-tool script individually, in the order listed.
        - "--build-krell-root"
        - "--build-offline"

- The examples below are for illustration and will need adjustment based on the software installations on your particular system.
- On systems with heterogeneous front-end and compute node processor sets, multiple install-tool invocations are required. Examples of these types of systems are the BG/Q, some Cray systems, or on any other systems where targeted builds are needed. We suggest a separate set of builds, one set for the front-end viewer tool build and another set of builds for the compute node runtimes and performance data collectors build.
    - For building the collectors and runtimes that execute on the compute node:
        - "--build-krell-root" using --target-arch=<platform>
        - "--build-cbtf-all" using --target-arch=<platform>
    - NOW LEGACY: For building the client tool that runs on the front-end node:
        - "--build-krell-root"
        - "--build-offline"

    - Note: It appears that some systems with heterogeneous front-end versus compute node processor types default to the compute node compilers. Most of our instructions assume the default compilers are front-end compilers. Putting "CXX=g++" and "CC=gcc" in front of the install-tool command when building the front-end version of O|SS may get around this issue on platforms where the defaults are not what we expected (defaults to compute node compiler version).
- **Note: In the openspeedshop-release-2.4 subdirectory "startup_files" there are several install-tool build scripts that have been used to build O|SS on several platforms. These are available to use as a guide for installs on other new platforms.**

Prerequisite Packages

There are some prerequisite packages that are necessary for building and running the O|SS performance tool. Most will be present on a system that is used for debugging and performance analysis. This is true of most national laboratory clusters, Cray, and Blue Gene platforms. However, if you are installing on machines without the necessary tool supporting packages installed then this section might be helpful.

Ubuntu/Debian:

It is necessary to apply these packages to the base system installation, if they are not already installed, in order to successfully build O|SS and the external tool specific packages required by O|SS. Example required packages based on Ubuntu 16.04 install.

Packages to aid in development:

```
sudo apt-get install git cmake cvs
```

Packages to aid in downloading and building:

```
sudo apt-get install flex bison libxml2-dev python-dev g++ make patch
environment-modules libz-dev binutils-dev libdwarf-dev libelf-dev
```

Packages that install-tool will build if you don't have them installed: (but makes build shorter if these are installed)

```
sudo apt-get install automake autoconf libtool m4 libltdl-dev(--build-
autotools)
sudo apt-get install binutils binutils-dev (--build-binutils)
sudo apt-get install libelf1 elfutils libelf-dev (--build-libelf)
sudo apt-get install sqlite3 (--build-sqlite)
```

If the package: qt3-apps-dev is available use it, if not, you may need these (libx11-dev libxext-dev) instead.

RedHat/Fedora:

It is necessary to apply a list of supporting packages to the base system installation, if they are not already installed, in order to successfully build O|SS and the external tool specific packages required by O|SS. This list is somewhat variable depending on what was installed during the initial installation and prior to attempting to install O|SS. Previous experiences have resulted in this list of candidate packages:

```
    yum install -y rpm-build \
        gcc gcc-c++ \
        openmpi \
        patch \
        autoconf automake \
        elfutils-libelf elfutils-libelf-devel \
        libxml2 libxml2-devel \
        binutils binutils-devel \
        python python-devel \
        flex bison bison-devel bison-runtime \
        libtool libtool-ltdl libtool-ltdl-devel cmake git
```

SLES/SUSE:

It is necessary to apply these packages to the base system installation, if they are not already installed, in order to successfully build O|SS and the external tool specific packages required by O|SS.

Packages to aid in development:

    Modules git

Packages to aid in downloading and building:

    flex bison rpm libxml2 libxml2-dev python-dev g++ make patch cmake

Packages that install-tool will build if you don't have them installed: (but makes the build shorter if these are installed)

    qt3 qt3-devel

If your system was installed with any development package group, then the need for extra package installs may be reduced significantly.

Using the install-tool that comes with the O|SS release will install many of the key open source tool related components, but the above mentioned system components are required to be installed by the system administrator.

Building from the Release Tarballs (located on https://www.openspeedshop.org/downloads)

The release top-level directory contains the install script (install-tool) that is intended to make the build and installation of the external tool support components (known as the krell root) that are needed to support O|SS and O|SS itself easier.

Because O|SS depends on components that are highly dependent on operating system interfaces and libraries, it is difficult to produce executable rpms for every installation platform. This is the reason why we offer the source build installation instead of rpms.

Installation Information

Please gunzip and untar the openspeedshop-release-2.4.0 tar.gz file and change directory into the openspeedshop-release-2.4 directory.

For example:

```
tar -xzvf openspeedshop-release-2.4.0.tar.gz
cd openspeedshop-release-2.4
```

Inside the top-level release directory is the key script, install-tool that can be used in building the O|SS performance tool.

The script, install-tool has a help option:

“install-tool –-help“

That can provide information about the possible options a builder of O|SS could use.

The typical build for O|SS is done with an install line something like this:

```
./install-tool --build-krell-root
               --krell-root-prefix /opt/krellroot_v2.4.0
               --with-openmpi /opt/openmpi-1.8.2

./install-tool   --build-cbtf-all
               --cbtf-prefix /opt/cbtf_v2.4.0
               --krell-root-prefix /opt/krellroot_v2.4.0
               --with-openmpi /opt/openmpi-1.8.2


./install-tool   --build-oss
               --openss-prefix /opt/osscbtf_v2.4.0
               --cbtf-prefix /opt/cbtf_v2.4.0
               --krell-root-prefix /opt/krellroot_v2.4.0
               --with-openmpi /opt/openmpi-1.8.2
```

The first install line above builds all the external (krell root) packages that O|SS needs and installs them in /opt/krellroot_v2.4.0. The second install line above uses those external packages to build the Component Based Tool Framework (CBTF). It also uses the openmpi MPI implementation to build the O|SS MPI experiment collectors. The third install line uses both the external packages built by the krell root install line and the cbtf package built by the cbtf-all install line to build the O|SS client tools.

Hint:
To get the paths for the "--with-mpt" or other mpi or other “--with” option clauses is to:
1) module load mpi-sgi/mpt.2.12r26
2) echo $LD_LIBRARY_PATH
3) Look for the MPT path (up to the "/lib" and use that in the "--with-mpt" clause in the install-tool command.

```
Example:  echo $LD_LIBRARY_PATH
/nasa/sgi/mpt/2.12r26/lib:/home4/jgalarow/python_root/lib

Use this --with-mpt clause: "--with-mpt /nasa/sgi/mpt/2.12r26"
```

However, on some platforms the typical build install line is not adequate. Platforms like the Cray and Blue Gene requires more options to the install-tool script. If you are familiar with the previous install.sh script method of building O|SS, in that build scenario extra environment variables needed to be set. With install-tool, we need additional arguments to the script to specify things like the target platform, so there is a --target-arch <target> option available in the install-tool script. For example, to build on a Cray system, this is an example build line:

```
./install-tool   --build-krell-root –target-arch cray
                  --krell-root-prefix /tmp/work/<userid>/krellroot_v2.4.0
                  --with-boost <boost install directory>
                  --with-mpich2 /opt/cray/mpt/default/gni/mpich2-gnu/47

./install-tool   --build-cbtf-all –target-arch cray
                  --cbtf-prefix /tmp/work/<userid>/cbtf_v2.4.0
                  --krell-root-prefix /tmp/work/<userid>/krellroot_v2.4.0
                  --with-mpich2 /opt/cray/mpt/default/gni/mpich2-gnu/47


./install-tool   --build-oss –target-arch cray
                  --openss-prefix /tmp/work/<userid>/osscbtf_v2.4.0
                  --krell-root-prefix /tmp/work/<userid>/krellroot_v2.4.0
                  --with-boost <boost install directory>
                  --with-mpich /opt/cray/mpt/default/gni/mpich2-gnu/47
```

One can specify their own versions of the components needed by O|SS to build properly or you can rely on the defaults that are installed on the system and used automatically by the build script. If the default system installed component is determined to be not adequate for the O|SS build (missing development headers) then the install script will build a version of that component into the externals/root directory. If the builder has a specific version of a component, such as, PAPI, they may use the "--with-papi" option to specify the path to that installation and O|SS will be built using that installation of PAPI.

One can also build individual components and specify the location to place the component. Here is the "install-tool --help" output:
$

```
$ ./install-tool --help
usage: ./install-tool [option]

--help, -h
  This help text.

Introduction:
  This install script can be used to build the krell externals package (--build-krell-root),
  the CBTF components and libraries (--build-cbtf-all), and/or the default version of
OpenSpeedShop
  which now contains both the offline mode of operation and the cbtf mode of operation
with
  the: (--build-oss) install-tool option.

  Typical usages examples are followed by the option descriptions. More examples and
explanations
  can be found in the Build and Install Guides for CBTF and Open|SpeedShop.

Typical usage example for cluster/PC build:
  # Build only the krell-root
  ./install-tool --build-krell-root
                --krell-root-prefix /opt/krellroot_v2.4.0
                --with-openmpi /opt/openmpi-1.8.2

  # Build cbtf components using the krell-root
  ./install-tool --build-cbtf-all
                --cbtf-prefix /opt/cbtf_v2.4.0
                --krell-root-prefix /opt/krellroot_v2.4.0
                --with-openmpi /opt/openmpi-1.8.2
                --with-cupti /usr/local/cuda-8.0/extras/CUPTI
                --with-cuda /usr/local/cuda-8.0

  # Build only OSS using the cbtf components and the krell-root
  ./install-tool --build-oss
                --cbtf-prefix /opt/cbtf_v2.4.0
                --krell-root-prefix /opt/krellroot_v2.4.0
                --openss-prefix /opt/osscbtf_v2.4.0
                --with-openmpi /opt/openmpi-1.8.2
                --with-cupti /usr/local/cuda-8.0/extras/CUPTI
                --with-cuda /usr/local/cuda-8.0

Typical in one command usage example for cluster/PC build:

  # Build the krell-root, the cbtf components, and OSS using cbtf instrumentor
  ./install-tool --build-all
                --cbtf-prefix /opt/cbtf_v2.4.0
                --krell-root-prefix /opt/krellroot_v2.4.0
                --openss-prefix /opt/osscbtf_v2.4.0
                --with-openmpi /opt/openmpi-1.8.2

Typical example for building the new Qt4/Qt5 GUI tool/interface:

  # Build graphviz for support of building the new cbtf-argonavis-gui
  ./install-tool --build-graphviz
                --krell-root-prefix /opt/graphviz-2.40.1

  # Build QtGraph for support of building the new cbtf-argonavis-gui, uses graphviz
```

```
 ./install-tool --build-QtGraph
              --krell-root-prefix /opt/QtGraph-1.0.0
              --with-graphviz /opt/graphviz-2.40.1
              --with-qt /usr/lib64/qt5

 # Build new cbtf-argonavis-gui, uses graphviz and QtGraph (installs into
/opt/osscbtf_v2.4.0)
 ./install-tool --build-cbtfargonavisgui
              --with-openss /opt/osscbtf_v2.4.0
              --with-cbtf /opt/cbtf_v2.4.0
              --krell-root-prefix /opt/krellroot_v2.4.0
              --with-graphviz /opt/graphviz-2.40.1
              --with-QtGraph /opt/QtGraph-1.0.0
              --with-boost /opt/boost-1.53.0
              --with-qt /usr/lib64/qt5

Option Descriptions:

--krell-root-prefix <directory>
 Where <directory> is the location to install the components
 needed for building CBTF and OpenSpeedShop
 and it's supporting tools and libraries. The default
 is /opt/KRELLROOT. It is not recommended to use /usr.
 NOTE: This option will override any setting for
 the environment variable KRELL_ROOT_PREFIX.

 NOTE: This prefix should be used as the install path
      when building a specific component (for example:
      install-tool --build-libelf --krell-root-prefix /opt/libelf-0.8.13

--cbtf-prefix <directory>
 Where <directory> is the location to install CBTF
 and it's supporting tools and libraries. The default
 is /opt/CBTF. It is not recommended to use /usr.
 NOTE: This option will override any setting for
 the environment variable CBTF_PREFIX.

--openss-prefix <directory>
 Where <directory> is the location to install OpenSpeedShop
 and it's supporting tools and libraries. The default
 is /opt/OSS. It is not recommended to use /usr.
 NOTE: This option will override any setting for
 the environment variable OPENSS_PREFIX.

--build-all
 Build the krell-root, cbtf components, and openspeedshop
 The cbtf-prefix, krell-root-prefix, and the openss-prefix must be specified

--build-cbtf-all
 Build only the cbtf components using the existing krell-root
```

```
  The krell-root-prefix and cbtf-install-prefix must be specified

--build-krell-root
  Build only the krell-root. The krell-root-prefix must be specified

--build-onlyosscbtf )
  Build only the OpenSpeedShop component for the cbtf instrumentor
  using the cbtf and krell-root components.

  Specify a target architecture
--target-arch <target>
  Where acceptable target values are: cray-xk, cray-xe, bgp, bgq, bgqfe

  Specify the compiler to be used for the build, gnu is default
--build-compiler {gnu, intel}

  Use these MPI installations when building
--with-mpich <directory>
--with-mpich2  <directory>
--with-mpich2-driver  <driver name>
--with-mvapich  <directory>
--with-mvapich2  <directory>
--with-mvapich2-driver  <driver name>
--with-openmpi  <directory>
--with-mpt  <directory>
  Where <directory> is the installed path to the mpi implementation
  to use for MPI support.

  Build only the component specified by the build clause.
  The krell-root-prefix must be specified and components will
  be installed into that krell-root-prefix specified directory path.
--build-autotools
--build-bison
--build-boost
--build-xercesc
--build-binutils
--build-flex
--build-dyninst
--build-elfutils
--build-GOTCHA
--build-libelf
--build-libdwarf
--build-libmonitor
--build-libunwind
--build-llvm-openmp
--build-mrnet
--build-ompt
--build-papi
--build-python
--build-sqlite
```

```
--build-symtabapi
--build-cmake
--build-ptgf
--build-ptgfossgui
--build-qcustomplot
--build-graphviz
--build-QtGraph
--build-qt3
--build-vampirtrace
  Build the experimental CUDA focused GUI Qt4/Qt5
--build-cbtfargonavisgui

  Enable certain configuration options
--enable-bfd-symbol-resolution|--enable-bfd
--enable-debug

  Force these components to be built and installed into the krellroot
  or OSS install directories.
--force-binutils-build
--force-boost-build
--force-cmake-build
--force-dyninst-build
--force-libelf-build
--force-libdwarf-build
--force-libunwind-build
--force-papi-build
--force-qt3-build
--force-sqlite-build
--force-xercesc-build
--force-ompt-build
  Build all of the above by force
--force-all

--skip-binutils-build
--skip-boost-build
--skip-cmake-build
--skip-dyninst-build
--skip-libdwarf-build
--skip-libelf-build
--skip-libunwind-build
--skip-mrnet-build
--skip-ompt-build
--skip-papi-build
--skip-qt3-build
--skip-sqlite-build
--skip-symtabapi-build
--skip-vampirtrace-build
--skip-xercesc-build
```

```
 Use these non-root or alternative components when building
--with-binutils <directory>
--with-boost <directory>
--with-dyninst <directory>
--with-expat <directory>
--with-graphviz <directory>
--with-libelf <directory>
--with-libdwarf <directory>
--with-libmonitor <directory>
--with-libunwind <directory>
--with-mrnet <directory>
--with-papi <directory>
--with-python <directory>
--with-python-vers <version number>
--with-qt <directory>
--with-qt3 <directory>
--with-sqlite <directory>
--with-symtabapi <directory>
--with-xercesc <directory>
--with-otf <directory>
--with-QtGraph <directory>
--with-vt <directory>
--with-zlib <directory>
 Where <directory> is the installed path to the alternative component.

 Use these non-root or alternative compute node components when building a cbtf-krell fe
 that points to targeted runtimes (cray, mic platforms)
--with-cn-cbtf <directory>
--with-cn-cbtf-krell <directory>
--with-cn-binutils <directory>
--with-cn-dyninst <directory>
--with-cn-libelf <directory>
--with-cn-libdwarf <directory>
--with-cn-libmonitor <directory>
--with-cn-libunwind <directory>
--with-cn-mrnet <directory>
--with-cn-symtabapi <directory>
--with-cn-xercesc <directory>
--with-cn-papi <directory>
--with-cn-boost <directory>
 Where <directory> is the installed path to the alternative component.

--with-tls < explicit | implicit >
 where the default is implicit
```

Optional install-tool arguments are needed for configuring O|SS for MPI experiments.   If the installation of O|SS is intended to support running MPI specific O|SS experiments, then one or more MPI implementation arguments must be specified.   The O|SS install-tool script will build MPI experiment collectors for one or more of these MPI implementations by specifying one or more of the "--with-<mpi implementation> arguments:

```
For openMPI:
 --with-openmpi <openmpi install path>
For mpich:
--with-mpich <mpich install path>
For mpich2:
--with-mpich2 <mpich2 install path>
For SGI mpt[1]:
--with-mpt <mpt install path>
For mvapich
--with-mvapich <mvapich install path>
For mvapich2:
--with-mvapich2 <mvapich2 install path>
```

If none of the above arguments are not specified, every non-MPI specific O|SS experiments will be built and execute properly, but the mpi, mpit, mpiotf experiments will not be built. The MPI implementation arguments are not necessary to run pcsamp, usertime, hwc, io, or fpe and variants of those experiments, even on MPI applications. They are only needed for mpi, mpip, and mpit experiment creation because the tool needs to know the MPI data structure definitions to process MPI performance data.

The install-tool script and the O|SS configuration code will operate on those MPI arguments and will configure O|SS to recognize these MPI implementations. When the MPI specific argument collectors are built and installed, users will have the ability to create MPI experiments (mpi, mpip, mpit) and gather performance data for MPI applications built with those specific MPI implementations.

Building the new Qt4/Qt5 based O|SS GUI

There is a new graphical user interface being developed whose initial focus was on the cuda experiment. However, support for all experiments is being added and covers all of the O|SS experiments now. To build this somewhat experimental, but useful tool use the following install-tool command line format:

```
./install-tool --build-cbtfargonavisgui
               --with-openss <path to the O|SS client installation>
               --with-cbtf <path to the CBTF installation>
               --krell-root-prefix <path to the krell root externals installation>
               --with-graphviz <path to graphviz installation directory>
               --with-QtGraph <path to QtGraph installation directory>
               --with-boost <path to boost installation directory>
```

[1] MPT is transitioning to MPI-3 support but it is not completed. Please use MPT-2.08 and above when building the MPI collectors. We have added special code to support their implementation of mpi.h MPI Function specifications with our wrappers.

--with-qt <path to qt4 or qt5 installation directory>

For example:

```
./install-tool --build-cbtfargonavisgui
                        --with-openss /opt/OSS/osscbtf_v2.4.0
                        --with-cbtf /opt/OSS/cbtf_v2.4.0
                        --krell-root-prefix /opt/OSS/krellroot_v2.4.0
                        --with-graphviz /opt/OSS/graphviz-2.40.1
                        --with-QtGraph /opt/OSS/QtGraph-1.0.0
                        --with-boost /opt/boost-1.53_0
                        --with-qt /usr/lib64/qt4
```

See the O|SS reference/users guide for information on how to use the new graphical user interface. The new GUI command is: openss-gui, while the existing GUI's command is: openss.

If graphviz is not on installed on the system, this install-tool command can be used to build and install graphviz. For example:

```
./install-tool --build-graphviz --krell-root-prefix /opt/OSS/graphviz-2.40.1
```

QtGraph is a component of the new Qt4/Qt5 GUI and also needs to be installed manually at this time. The install-tool line to install QtGraph is as follows:

```
./install-tool --build-QtGraph --krell-root-prefix /opt/OSS/QtGraph-1.0.0 --with-graphviz
/opt/OSS/graphviz-2.40.1 --with-qt /usr/lib64/qt4
```

Module Files, Dotkits, and softenv files

On most systems, a module file, dotkit file, or softenv file must be created after O|SS is built and that file is then activated/loaded prior to running O|SS. There are examples of each in the Runtime Environment Examples section below.

Install tool example commands from various systems

These examples show optional ways of building the default version of O|SS. The krell root components, the cbtf components, and the O|SS client components are installed into separate install directories allowing the ability to update each individually. One can use the "--build-all" option allows all the pieces to be built with one command where all the install locations are specified. The "--krell-root-prefix", "--cbtf-prefix", and "--openss-prefix" options must be specified, so that the build script knows where to install the separate components. Or you can pass the same install prefix for all of the three prefix options.

Generic Laptop or Desktop Platform Installation Examples

Please load the cmake module file or have cmake installed, as it is required for building some root components as well as O|SS and the CBTF components (named cbtf, cbtf-krell, cbtf-argonavis, and cbtf-lanl).

Build only the krell-root

```
./install-tool --build-krell-root
        --krell-root-prefix /opt/krellroot_v2.4.0
        --with-openmpi /opt/openmpi-1.8.2
```

Build cbtf components using the krell-root

```
./install-tool --build-cbtf-all
        --cbtf-prefix /opt/cbtf_only_v2.4.0
        --krell-root-prefix /opt/krellroot_v2.4.0
        --with-openmpi /opt/openmpi-1.8.2
        --with-cupti /usr/local/cuda-6.5/extras/CUPTI
        --with-cuda /usr/local/cuda-6.5
```

Build only OSS using the cbtf components and the krell-root

```
./install-tool --build-oss
        --cbtf-prefix /opt/cbtf_only_v2.4.0
        --krell-root-prefix /opt/krellroot_v2.4.0
        --openss-prefix /opt/osscbtf_v2.4.0
        --with-openmpi /opt/openmpi-1.8.2
        --with-cupti /usr/local/cuda-6.5/extras/CUPTI
        --with-cuda /usr/local/cuda-6.5
```

Build everything: the krell-root, the cbtf components, and OSS using cbtf instrumentor

```
./install-tool --build-all
        --cbtf-prefix /opt/cbtf_only_v2.4.0
        --krell-root-prefix /opt/krellroot_v2.4.0
```

```
--openss-prefix /opt/osscbtf_v2.4.0
--with-openmpi /opt/openmpi-1.8.2
```

Cray Platform Install Examples

Depending on what the default programming environment setting is, you must swap the PrgEnv-cray or PrgEnv-pgi environment module to PrgEnv-gnu. If you are building for CUDA, then you will have to load the cudatoolkit module file. The cmake tool is also required by the cbtf build and is not loaded by default on many Cray systems. A possible module load/swap scenario prior to invoking the install-tool scripts is as follows:

```
module swap PrgEnv-pgi PrgEnv-gnu
module load cudatoolkit
module load cmake
```

Instructions for building O|SS CBTF based versions on Cray platforms

We have moved to a multi-step process for building on platforms where the front-end node and the compute nodes are running different processor sets. By building the compute node O|SS and components with the compilers suggested for running on the compute nodes and by building the front-end node O|SS and components we are able to optimally support the Cray platform.

The high level view is to first build for the compute nodes by building the components needed by O|SS and CBTF (krellroot), then build the CBTF components, and finally build O|SS. After the component node versions are built, then we do a similar set of builds for the front-end components and O|SS and CBTF clients.

Building for the compute nodes

This must be done first because we pass the O|SS and CBTF compute node installation directories as arguments to the front-end install-tool build commands.

Setup up the build environment for building for the compute nodes

Note: that this is an example set of module settings. Newer or older versions of Cray software may require alternative module files to be loaded or unloaded. This is an example from the DOD Cray platform Gordon. It follows closely with most Cray type builds, although there are differences on the versions that are used on a particular system.

Also, note: there is a new interface that appears to be replacing the ALPS package and aprun as the default way to launch applications on the Cray. The CTI interface allows for launching application via native SLURM srun. On systems where the CTI interface is being use, please use the install-tool --use-cti install-tool option and do not pass the –-with-alps option.

Another note: Some Cray systems do not install expat-static package which. causes the MRNet package build to fail. In order to resolve this issue, we added the expat tarball to the external packages in the SOURCES directory. To build this into the krellroot, use: install-tool --build-expat --krell-root-prefix <install location>. Then add --with-expat to the install-tool --build-krell-root command line to resolve this issue.

```
module unload PrgEnv-pgi PrgEnv-cray PrgEnv-intel
module load PrgEnv-gnu
module unload gcc; module load gcc/4.9.3
export SYSROOT_DIR=/opt/cray/xc-sysroot/default
module load cmake-3.2.2

BASE_IDIR=/p/home/app/PET/pkgs/openss
TOOL_VERS="_v2.4.0.beta1"
KROOT_IDIR=${BASE_IDIR}/krellroot${TOOL_VERS}
CBTF_IDIR=${BASE_IDIR}/cbtf${TOOL_VERS}
OSSCBTF_IDIR=${BASE_IDIR}/osscbtf${TOOL_VERS}
OSSOFF_IDIR=${BASE_IDIR}/ossoff${TOOL_VERS}
PAPI_IDIR=/opt/cray/papi/default
MPICH_IDIR=/opt/cray/mpt/7.3.2/gni/mpich-gnu/5.1
ALPS_IDIR=/opt/cray/alps/default

export cc=gcc
export CC=gcc
export CXX=g++
```

Build the compute node versions of cbtf, cbtf-krell, cbtf-argonavis, cbtf-lanl

```
# Build root components runtime only
./install-tool --runtime-only
    --target-arch cray --target-shared
    --build-krell-root --krell-root-prefix ${KROOT_IDIR}/compute
    --with-mpich ${MPICH_IDIR}
    --with-papi ${PAPI_IDIR}
    --with-alps ${ALPS_IDIR}

./install-tool --build-cbtf-all --runtime-only
    --target-arch cray --target-shared
     --cbtf-prefix ${CBTF_IDIR}/compute
     --krell-root-prefix ${KROOT_IDIR}/compute
     --with-mpich ${MPICH_IDIR}
     --with-papi ${PAPI_IDIR}
```

Build the compute node version of krellroot

```
./install-tool --build-krell-root
        --runtime-only
        --target-arch cray
        --target-shared
        --krell-root-prefix /home/jgalaro/krellroot_stable/compute
        --with-papi /opt/cray/papi/5.3.1
```

```
./install-tool --build-cbtf-all
        --runtime-only
        --target-arch cray
        --target-shared
        --cbtf-prefix /home/jgalaro/cbtf_only_stable/compute
        --krell-root-prefix /home/jgalaro/krellroot_stable/compute
        --with-mpich2 /opt/cray/mpt/7.0.1/gni/mpich2-gnu/48
        --with-papi /opt/cray/papi/5.3.1
        --with-dyninst /home/jgalaro/compute/dyninst-9.0.0_gcc
```

Build the compute node versions of O|SS for the CBTF version

Note: There is nothing to build, as this version uses the CBTF compute node components for gathering the data. Those were built in the --build-cbtf-all install-tool build command.

NOTE:

Because the default GNU compilers are too old on some Cray platforms and if we build with the default compilers some of the necessary software components will fail to build. If we can build with the default GNU compilers, then this isn't an issue. No copy or LD_LIBRARY_PATH setup needed.

If a newer compiler was loaded via a module file in order to build the tool, then after building everything you have to copy the libstdc++.so.6 from the compiler directory (because it is not available on the compute nodes) into the compute node install directory (krellroot/lib64, for example) or setup LD_LIBRARY_PATH to point to that location.

Here is an example of how it could be done.

```
> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.7    2) eswrap/1.0.8      3) cray-mpich/7.0.1   4) cmake-3.2.2
5) gcc/4.8.2

> which gcc
/opt/gcc/4.8.2/bin/gcc

> lsr /opt/gcc/4.8.2//snos/lib64/libstdc++.so.6
0 lrwxrwxrwx 1 root root 19 Aug 8 2014 /opt/gcc/4.8.2//snos/lib64/libstdc++.so.6
-> libstdc++.so.6.0.18
```

Building for the front-end or login nodes

Setup up the build environment for building for the front-end or login nodes

- module unload PrgEnv-pgi PrgEnv-gnu PrgEnv-cray PrgEnv-intel
- module unload craype-network-gemini craype-mc8
- module load cmake-3.2.2
- module load gcc
- export CXX=g++
- export cc=gcc

Build the front-end node version of krellroot

```
./install-tool --build-krell-root
        --krell-root-prefix /home/jgalaro/krellroot_stable
        --with-papi /opt/cray/papi/5.3.1
        --with-alps /opt/cray/xe-sysroot/default/usr
        --with-mpich2 /opt/cray/mpt/7.0.1/gni/mpich2-gnu/48
```

Build the front-end node versions of cbtf, cbtf-krell, cbtf-argonavis, cbtf-lanl

Note: The --with-cn-<component> lines have compute node installations as their arguments. We are pointing the front-end build to the compute node runtime libraries in order to run the properly built compute node versions of the runtime libraries.

```
./install-tool --build-cbtf-all
        --runtime-target-arch cray
        --cbtf-prefix /home/jgalaro/cbtf_only_stable
        --krell-root-prefix /home/jgalaro/krellroot_stable
        --with-mpich2 /opt/cray/mpt/7.0.1/gni/mpich2-gnu/48
        --with-cn-boost /home/jgalaro/krellroot_stable/compute
        --with-cn-mrnet /home/jgalaro/krellroot_stable/compute
        --with-cn-xercesc /home/jgalaro/krellroot_stable/compute
        --with-cn-libmonitor /home/jgalaro/krellroot_stable/compute
        --with-cn-libunwind /home/jgalaro/krellroot_stable/compute
        --with-cn-dyninst /home/jgalaro/compute/dyninst-9.0.0_gcc
        --with-cn-papi /opt/cray/papi/5.3.1
        --with-cn-cbtf-krell /home/jgalaro/cbtf_only_stable/compute
        --with-cn-cbtf /home/jgalaro/cbtf_only_stable/compute
        --with-binutils /home/jgalaro/krellroot_stable
        --with-boost /home/jgalaro/krellroot_stable
        --with-mrnet /home/jgalaro/krellroot_stable
        --with-xercesc /home/jgalaro/krellroot_stable
        --with-libmonitor /home/jgalaro/krellroot_stable
        --with-libunwind /home/jgalaro/krellroot_stable
```

```
          --with-dyninst /home/jgalaro/krellroot_stable
          --with-papi /opt/cray/papi/5.3.1
```

Build the front-end node version of O|SS for the CBTF version

```
./install-tool --target-arch cray
          --build-oss
          --openss-prefix /home/jgalaro/osscbtf_stable
          --with-cn-cbtf-krell /home/jgalaro/cbtf_only_stable/compute
          --krell-root-prefix /home/jgalaro/krellroot_stable
          --with-mpich2 /opt/cray/mpt/7.0.1/gni/mpich2-gnu/48
          --with-papi /opt/cray/papi/5.3.1
          --with-boost /home/jgalaro/krellroot_stable
          --with-mrnet /home/jgalaro/krellroot_stable
          --with-xercesc /home/jgalaro/krellroot_stable
          --with-libmonitor /home/jgalaro/krellroot_stable
          --with-libunwind /home/jgalaro/krellroot_stable
          --with-dyninst /home/jgalaro/krellroot_stable
          --with-libelf /home/jgalaro/krellroot_stable
          --with-libdwarf /home/jgalaro/krellroot_stable
          --with-binutils /home/jgalaro/krellroot_stable
          --cbtf-prefix /home/jgalaro/cbtf_only_stable
```

Blue Gene Systems (only offline supported) Install Examples

For the Blue Gene systems, it is still a two-step process to build the offline version of O|SS. First build the compute node collectors and runtimes that run on the compute nodes. Note that the compute note installation paths have a "bgq" sub-directory appended to the viewer installation path by convention, not necessity. The location must be different than that of the viewer installation as not to clobber the viewer installation. The builder is responsible for keeping the two installations separate. Next build the O|SS viewer that runs on the front-end node, giving this build the –with-runtime-dir that points to the O|SS compute node install directory. That way the front-end knows where the compute node collectors and runtime libraries are located.

For the Blue Gene builds, you must build offline for the compute node first and then build offline for the front-end node. You may build offline using the krellroot concept or not. Using krellroot allows you to just build the new openspeedshop-2.4 updates when they arrive without building all the krellroot components over again.

Compute node – O|SS collectors and runtimes

Build O|SS collectors and runtimes without using krell root components

```
./install-tool --build-offline
          --runtime-only --target-shared --target-arch bgq
```

```
            --openss-prefix /usr/global/tools/openspeedshop/oss-dev/bgq/ossoff_v2.4.0/compute
            --with-mpich2 /bgsys/drivers/ppcfloor/comm/gcc
```

Build krell root components for compute node

```
./install-tool –-runtime-only –-target-shared --target-arch bgq --build-krell-root
            --krell-root-prefix /usr/global/tools/openspeedshop/ossdev/bgq/krellroot_v2.4.0/compute
            --with-mpich2 /bgsys/drivers/ppcfloor/comm/gcc
```

Build O|SS collectors and runtimes using krell root components

```
./install-tool --build-offline
            --runtime-only --target-shared --target-arch bgq
            --openss-prefix /usr/global/tools/openspeedshop/oss-dev/bgq/ossoff_v2.4.0/compute
            --krell-root-prefix /usr/global/tools/openspeedshop/ossdev/bgq/krellroot_v2.4.0/compute
             --with-mpich2 /bgsys/drivers/ppcfloor/comm/gcc
```

Front end node – O|SS viewer

Build O|SS Viewer not using krell root components

```
./install-tool --build-offline
            --openss-prefix /usr/global/tools/openspeedshop/oss-dev/bgq/ossoff_v2.4.0
            --with-mpich2 /bgsys/drivers/ppcfloor/comm/gcc
            --with-runtime-dir /usr/global/tools/openspeedshop/oss-dev/bgq/ossoff_v2.4.0/compute
```

Build krell root components

```
./install-tool --build-krell-root
            --krell-root-prefix /usr/global/tools/openspeedshop/oss-dev/bgq/krellroot_v2.4.0
            --with-mpich2 /bgsys/drivers/ppcfloor/comm/gcc
```

Build O|SS viewer using krell root components

```
./install-tool --build-offline
            --openss-prefix /usr/global/tools/openspeedshop/oss-dev/bgq/ossoff_v2.4.0
            --krell-root-prefix /usr/global/tools/openspeedshop/oss-dev/bgq/krellroot_v2.4.0
            --with-mpich2 /bgsys/drivers/ppcfloor/comm/gcc
            –-with-runtime-dir /usr/global/tools/openspeedshop/oss-dev/bgq/ossoff_v2.4.0/compute
```

Intel MIC (KNL) Platform Install Examples

The builds of O|SS, CBTF and supporting components are done on the Intel MIC KNL login node.   Use the generic cluster build examples above.

Intel MIC (KNC) Co-Processor Platform Install Examples

The builds of O|SS, CBTF and supporting components are done on the Intel MIC co-process login node (with Intel MIC specific software: compilers, libraries). This example build scenario is based on experiences from building NASA's maia system and the NERSC test bed MIC system named babbage.

Instructions for building O|SS CBTF based versions on Intel MIC platforms

We have moved to a multi-step process for building on platforms where the front-end node and the compute nodes are running different processor sets. By building the compute node O|SS and components with the compilers suggested for running on the compute nodes and by building the front-end node O|SS and components we are able to optimally support the Intel MIC platform.

The high level view is to first build for the compute nodes by building the components needed by O|SS and CBTF (krellroot), then build the CBTF components, and finally build O|SS. After the component node versions are built, then we do a similar set of builds for the front-end components and O|SS and CBTF clients.

Building for the compute nodes

This must be done first because we pass the O|SS and CBTF compute node installation directories as arguments to the front-end install-tool build commands.

Setup up the build environment for building for the compute nodes

Build the compute node version of krellroot

```
./install-tool --build-krell-root
        --runtime-only
        --target-arch mic
        --krell-root-prefix /home/jgalaro/krellroot_stable/compute
```

Build the compute node versions of cbtf, cbtf-krell, cbtf-argonavis, cbtf-lanl

```
./install-tool --build-cbtf-all
        --runtime-only
        --target-arch mic
        --cbtf-prefix /home/jgalaro/cbtf_only_stable/compute
        --krell-root-prefix /home/jgalaro/krellroot_stable/compute
```

Build the compute node versions of O|SS for the CBTF version

Note: There is nothing to build, as this version uses the CBTF compute node components for gathering the data. Those were built in the --build-cbtf-all install-tool build command.

Building for the front-end or login nodes

Build the front-end node version of krellroot

```
./install-tool --build-krell-root
```

```
        --krell-root-prefix /home/jgalaro/krellroot_stable
```

Build the front-end node versions of cbtf, cbtf-krell, cbtf-argonavis, cbtf-lanl

Note: The --with-cn-<component> lines have compute node installations as their arguments. We are pointing the front-end build to the compute node runtime libraries in order to run the properly built compute node versions of the runtime libraries.

```
./install-tool --build-cbtf-all
        --runtime-target-arch mic
        --cbtf-prefix /home/jgalaro/cbtf_only_stable
        --krell-root-prefix /home/jgalaro/krellroot_stable
        --with-cn-boost /home/jgalaro/krellroot_stable/compute
        --with-cn-mrnet /home/jgalaro/krellroot_stable/compute
        --with-cn-xercesc /home/jgalaro/krellroot_stable/compute
        --with-cn-libmonitor /home/jgalaro/krellroot_stable/compute
        --with-cn-libunwind /home/jgalaro/krellroot_stable/compute
        --with-cn-dyninst home/jgalaro/krellroot_stable/compute
        --with-cn-cbtf-krell /home/jgalaro/cbtf_only_stable/compute
        --with-cn-cbtf /home/jgalaro/cbtf_only_stable/compute
        --with-binutils /home/jgalaro/krellroot_stable
        --with-boost /home/jgalaro/krellroot_stable
        --with-mrnet /home/jgalaro/krellroot_stable
        --with-xercesc /home/jgalaro/krellroot_stable
        --with-libmonitor /home/jgalaro/krellroot_stable
        --with-libunwind /home/jgalaro/krellroot_stable
        --with-dyninst /home/jgalaro/krellroot_stable
```

Build the front-end node version of O|SS for the CBTF version

```
./install-tool --target-arch cray
        --build-oss
        --openss-prefix /home/jgalaro/osscbtf_stable
        --with-cn-cbtf-krell /home/jgalaro/cbtf_only_stable/compute
        --krell-root-prefix /home/jgalaro/krellroot_stable
        --with-mpich2 /opt/cray/mpt/7.0.1/gni/mpich2-gnu/48
        --with-papi /opt/cray/papi/5.3.1
        --with-boost /home/jgalaro/krellroot_stable
        --with-mrnet /home/jgalaro/krellroot_stable
        --with-xercesc /home/jgalaro/krellroot_stable
        --with-libmonitor /home/jgalaro/krellroot_stable
        --with-libunwind /home/jgalaro/krellroot_stable
        --with-dyninst /home/jgalaro/krellroot_stable
        --with-libelf /home/jgalaro/krellroot_stable
        --with-libdwarf /home/jgalaro/krellroot_stable
        --with-binutils /home/jgalaro/krellroot_stable
        --cbtf-prefix /home/jgalaro/cbtf_only_stable
```

ARM Platform Install Examples

Instructions for O|SS CBTF based versions on ARM platforms

Basically building for the ARM is similar to building for a generic cluster, except that we need to know that we are building for the ARM in order to add some special compiler options *(-funwind-tables -fasynchronous-unwind-tables)* for unwinding call paths. So, we ask for builders to specify the **"--target-arch arm"** phrase to the "install-tool" build commands.

Building for the ARM platform

Setup up the build environment for building for the front-end or login nodes

- module load cmake

Build the krellroot - components needed to support building cbtf and O|SS

```
./install-tool --build-krell-root
        --target-arch arm
        --krell-root-prefix /home/jgalaro/krellroot_stable
        --with-openmpi /home/projects/arm64/openmpi/1.8.2/gnu/4.9.1
        --with-qt3 /home/jgalaro/qt3
```

Build cbtf, cbtf-krell, cbtf-argonavis, cbtf-lanl

```
./install-tool --build-cbtf-all
        --target-arch arm
        --cbtf-prefix /home/jgalaro/cbtf_only_stable
        --krell-root-prefix /home/jgalaro/krellroot_stable
        --with-openmpi /home/projects/arm64/openmpi/1.8.2/gnu/4.9.1
```

Build O|SS for the CBTF version

```
./install-tool --build-oss
        --target-arch arm
        --cbtf-prefix /home/jgalaro/cbtf_only_stable
        --krell-root-prefix /home/jgalaro/krellroot_stable
        --openss-prefix /home/jgalaro/osscbtf_stable
        --with-openmpi /home/projects/arm64/openmpi/1.8.2/gnu/4.9.1
```

To Run O|SS

Please refer to the Quick Start Guide from the O|SS documentation web-site page:
https://www.openspeedshop.org/wp/documentation/
for a short introduction to how to use O|SS.

For access the User's Reference Guide from the O|SS documentation web-site page:

https://www.openspeedshop.org/wp/documentation/

These are the best sources for information on how to run O|SS.

Environment Setup

OPENSS_PLUGIN_PATH

This environment variable specifies where the *O|SS* main program will look for experiment plugins. This is in addition to the normal search path, which is "<installdir>/lib{64}/openspeedshop" Prior to O|SS initialization set this variable to the path to your non-default plugins, (e.g. "setenv OPENSS_PLUGIN_PATH /g2/install/lib/openspeedshop)

LD_LIBRARY_PATH

This environment variable points to the directories where O|SS component libraries and O|SS libraries are installed.
Set this environment variable to <installdir>/lib{64} (e.g. "setenv LD_LIBRARY_PATH <installdir>/lib{64}:$LD_LIBRARY_PATH)

DYNINSTAPI_RT_LIB

This environment variable points to the directory where the Dyninst (dynamic runtime library) component runtime library is installed. This library is used in O|SS to detect loops and create per-loop statistic performance information available in the O|SS views.
Set this environment variable to
```
        <installdir for dyninst>/lib{64}/libdyninstAPI_RT.so
```
For example:
```
   setenv DYNINSTAPI_RT_LIB <installdir for dyninst>/lib{64/libdyninstAPI_RT.so
```

OPENSS_MPI_IMPLEMENTATION (for offline mode of operation)

This environment variable specifies the MPI implementation being used by the MPI application whose performance is being analyzed. Should be set to one of the currently supported MPI implementations:

- mpich
- mpich2
- mpt
- mvapich
- mvapich2
- openmpi

O|SS can auto-detect most of the MPI implementations that an
MPI application is using. So, this variable will only be used to override the auto-detection code, if need be.

PATH

This environment variable specifies the path to the O|SS component executables and to the O|SS executables. Set this to: <installdir>/bin (e.g. setenv PATH <installdir>/bin:$PATH).

OPENSS_RAWDATA_DIR (offline mode of operation only)

This environment variable specifies the path to where the O|SS offline mode of operation will store the rawdata files representing the performance data that was gathered from the user application. This is needed on clusters where the default rawdata file directory, /tmp, is not shared across the nodes. If this is the case on the cluster you are running the offline mode of operation (for example: osspcsamp –offline ...), then OPENSS_RAWDATA_DIR must be set to a shared file system directory. Otherwise, /tmp is used. Set this to:
OPENSS_RAWDATA_DIR <path_to_subdir_for_rawdata_file_writing>
(e.g. setenv OPENSS_RAWDATA_DIR /p/lscratchc/jeg).

OPENSS_DB_DIR

This environment variable specifies the path to where the offline version of openss will build the O|SS database file. If you are on a file system that does not have file locking turned on, the sqlite component will not be able to create the database file. You can use this environment variable to specify a path to a file system that does have locking turned to be used for the database file creation.

Runtime Environment Examples

The following runtime environment examples can be used as examples for creating module, dotkit, or softenv files depending on your systems execution environment.

Generic Cluster Module File Example

If your system supports the environment module file runtime environment setup, then these examples will provide some guidance for your O|SS runtime environment file creation.

O|SS module file example

```
#%Module1.0#####################################################################
##
##
## openss modulefile
##
proc ModulesHelp { } {
	global version openss

	puts stderr "\topenss - loads the OpenSpeedShop software & application environment"
	puts stderr "\n\tThis adds $oss/* to several of the"
	puts stderr "\tenvironment variables."
	puts stderr "\n\tVersion $version\n"
}

#  NOTE ------------------------------------------------------------
#  The paths may need adjustment for different library naming schemes
#  NOTE ------------------------------------------------------------
#

module-whatis   "Loads the OpenSpeedShop runtime environment."

# for Tcl script use only
set     version        2.4.0.latest

# Set up variables to reference later for the krell root, cbtf, and OpenSpeedShop proper
set    base       /home/jeg/openss/power
set     root          ${base}/krellroot_v2.4.0.latest
set     cbtf          ${base}/cbtf_v2.4.0.latest
set     cbtfk         ${base}/cbtf_v2.4.0.latest
set     oss           ${base}/osscbtf_v2.4.0.latest
set     qtgraph       ${base}/QtGraph-1.0.0
set     graphviz      ${base}/graphviz-2.41.0

#  XPLAT_RSH is needed for MRNet which is now needed for use in CBTF
setenv       XPLAT_RSH    ssh

#  For the mpi experiments only - specify the MPI implementation of your
#  application that will be run with OpenSpeedShop.   These are the
#  mpi, mpit, and mpip experiments.  All other experiment types will
#  ignore this setting.  It is only needed for mpi, mpit, and mpip.
setenv        CBTF_MPI_IMPLEMENTATION    openmpi
setenv        OPENSS_MPI_IMPLEMENTATION    openmpi

#  This is needed if you use the --offline argument following the
#  convenience scripts, for example:   osspcsamp --offline "mpirun -np 4 ./nbody"
#  This is the offline mode of operation which is now built into the
#  CBTF based version of OpenSpeedShop
setenv        OPENSS_RAWDATA_DIR    .

# Only need these CBTF specific variables for situations where the environment is not passed
setenv        MRNET_COMM_PATH $cbtfk/sbin/cbtf_mrnet_commnode
setenv        CBTF_MRNET_BACKEND_PATH $cbtfk/sbin/cbtf_libcbtf_mrnet_backend
```

```
# Set up the paths for the OSS/CBTF version of OpenSpeedShop
prepend-path    PATH          $root/bin
prepend-path    PATH          $cbtf/bin
prepend-path    PATH          $cbtfk/sbin
prepend-path    PATH          $cbtfk/bin
prepend-path    PATH          $oss/bin
prepend-path    MANPATH           $oss/share/man

# Set up the dyninst runtime library path for the OSS/CBTF version of OpenSpeedShop
# This is required for finding loops and gathering symbol table information.
setenv DYNINSTAPI_RT_LIB $root/lib/libdyninstAPI_RT.so

# Set up the library paths for the OSS/CBTF version of OpenSpeedShop
prepend-path LD_LIBRARY_PATH $root/lib64
prepend-path LD_LIBRARY_PATH $root/lib
prepend-path LD_LIBRARY_PATH $cbtf/lib64
prepend-path LD_LIBRARY_PATH $cbtfk/lib64
prepend-path LD_LIBRARY_PATH $oss/lib64
prepend-path LD_LIBRARY_PATH $qtgraph/lib/5.6.1
prepend-path LD_LIBRARY_PATH $graphviz/lib
prepend-path LD_LIBRARY_PATH /usr/local/cuda-8.0/extras/CUPTI/lib64

# Set up the python path so that the python scripting API can find
# the openss python module files.
setenv PYTHONPATH $oss/lib64/openspeedshop
```

Generic Cluster Dotkit File Example

This is an example of a dotkit file used for a 64-bit cluster platform installation and is not generalized to support different platforms other than the 64-bit cluster it was written for. Use the “use <filename of dotkit file>” command to activate the O|SS runtime environment. Note: do not include the “.dk” portion of the filename when using the “use” command.

O|SS dotkit example

```
#c performance/profile
#d O|SS (Version 2.4.0)
dk_setenv OPENSS /usr/global/tools/openspeedshop/oss-dev/OSS_2.4.0
dk_setenv KROOT /usr/global/tools/openspeedshop/oss-dev/krellroot_2.4.0
dk_setenv CBTF /usr/global/tools/openspeedshop/oss-dev/cbtf_2.4.0
#  XPLAT_RSH is needed for MRNet which is now needed for use in CBTF
dk_setenv  XPLAT_RSH    ssh
# If cuda present, then we need hooks to the CUPTI interface
dk_setenv CUDA_PATH /usr/global/tools/cuda-8.0
# If the new Qt4/Qt5 GUI was built, we the paths to graphviz and QtGraph
dk_setenv QTGRAPH /usr/ global/tools/openspeedshop/oss-dev/QtGraph-1.0.0
dk_setenv GRAPHVIZ /usr/ global/tools/openspeedshop/oss-dev/graphviz-2.40.1

#  For the mpi experiments only - specify the MPI implementation of your
#  application that will be run with OpenSpeedShop.   These are the
#  mpi, mpit, and mpip experiments.  All other experiment types will
#  ignore this setting.  It is only needed for mpi, mpit, and mpip.
dk_setenv  OPENSS_MPI_IMPLEMENTATION mvapich2

dk_setenv OPENSS_PLUGIN_PATH $OPENSS/lib64/openspeedshop
dk_setenv OPENSS_DOC $OPENSS/share/doc/packages/OpenSpeedShop/
```

```
# Find Dyninst for generation of per-loop statistics
dk_setenv DYNINSTAPI_RT_LIB       $KROOT/lib64/libdyninstAPI_RT.so

dk_alter PATH          $KROOT/bin
dk_alter PATH          $OPENSS/bin

dk_alter PATH          $CBTF/bin
dk_alter PATH          $CBTF/sbin
dk_alter LD_LIBRARY_PATH $CBTF/lib64
dk_alter LD_LIBRARY_PATH $KROOT/lib64
dk_alter LD_LIBRARY_PATH $KROOT/lib

dk_alter LD_LIBRARY_PATH $OPENSS/lib64
dk_alter LD_LIBRARY_PATH $QTGRAPH/lib/5.6.1
dk_alter LD_LIBRARY_PATH $GRAPHVIZ/lib
dk_alter LD_LIBRARY_PATH $CUDA_PATH/extras/CUPTI/lib64
```

Blue Gene/Q Softenv File Example

Offline version (no krell root usage)

This is an example of a softenv file used for a Blue Gene/Q installation. Use the "resoft <filename of softenv file>" command to activate the O|SS runtime environment. Note that the installation paths in this file follow the convention where compute node version of O|SS is installed as a "bgq" subdirectory of the installation path for the viewer version. For illustration:

- Viewer version installed in: /home/projects/oss/oss
- Compute node version installed in: /home/projects/oss/oss/bgq

```
# The O|SS .soft file.
# Remember to type "resoft" after working on this file.

OSS = /home/projects/oss/oss
TARCH = bgq

# Set up OSS environment variables

# Find the executable portions of O|SS (order is important here)
PATH += $OSS/$TARCH/bin
PATH += $OSS/bin

# Find the libraries for O|SS (order is important here)
LD_LIBRARY_PATH += $OSS/$TARCH/lib64
LD_LIBRARY_PATH += $OSS/lib64

# Find the runtime collectors
OPENSS_PLUGIN_PATH = $OSS/$TARCH/lib64/openspeedshop

# Find Dyninst for generation of per-loop statistics
DYNINSTAPI_RT_LIB $OSS/lib64/libdyninstAPI_RT.so

# Tell the tool what the application MPI implementation is
# Needed if supporting multiple implementations and running the "mpi", "mpit", or "mpiotf" experiments
OPENSS_MPI_IMPLEMENTATION = mpich2
```

```
# Paths to documentation and man pages
OPENSS_DOC_DIR = $OSS/share/doc/packages/OpenSpeedShop
MANPATH = $OSS/share/man

# Use the basic environment.
@default
```

Offline version (with krell root usage)

```
# The O|SS .soft file.
# Remember to type "resoft" after working on this file.

OSS = /home/projects/oss/oss
KROOT = /home/projects/krellroot
TARCH = bgq

# Set up OSS environment variables

# Find the executable portions of O|SS (order is important here)
PATH += $KROOT/$TARCH/bin
PATH += $KROOT/bin
PATH += $OSS/$TARCH/bin
PATH += $OSS/bin

# Find the libraries for O|SS (order is important here)
LD_LIBRARY_PATH += $KROOT/$TARCH/lib64
LD_LIBRARY_PATH += $KROOT/lib64
LD_LIBRARY_PATH += $KROOT/lib
LD_LIBRARY_PATH += $OSS/$TARCH/lib64
LD_LIBRARY_PATH += $OSS/lib64

# Find the runtime collectors
OPENSS_PLUGIN_PATH = $OSS/$TARCH/lib64/openspeedshop

# Find Dyninst for generation of per-loop statistics
DYNINSTAPI_RT_LIB  $KROOT/lib64/libdyninstAPI_RT.so

# Tell the tool what the application MPI implementation is
# Needed if supporting multiple implementations and running the "mpi", "mpit", or "mpiotf" experiments
OPENSS_MPI_IMPLEMENTATION = mpich2

# Paths to documentation and man pages
OPENSS_DOC_DIR = $OSS/share/doc/packages/OpenSpeedShop
MANPATH = $OSS/share/man

# Use the basic environment.
@default
```

Cray Platform Module File Example

If your system supports the environment module file runtime environment setup, then these examples will provide some guidance for your O|SS runtime environment file creation.   Note that the compiler library path was prepended to the LD_LIBRARY_PATH below. That is necessary if building with a compiler that is installed in a non-standard location (a module file was loaded). O|SS needs to find the libstdc++ and libgomp libraries that the tool was built with when the tool executes.

Cray platform CBTF version module file example

```
#%Module1.0#####################################################################
##
##
## oss cbtf 2.4.0 modulefile
##
proc ModulesHelp { } {
	global version openspeedshop-cbtf

	puts stderr "\topenspeedshop-cbtf - Loads the OpenSpeedShop software target back-end (be) and front-end
(fe) execution environment for Cray"
	puts stderr "\n\tVersion $version\n"
}

module-whatis   "Loads the OpenSpeedShop target back-end node (be) and front-end (fe) execution
environment."


# for Tcl script use only
set     version         2.4.0
set     root_prefix /p/home/galarowi/openss/krellroot_v2.4.0
set    cbtf_prefix /p/home/galarowi/openss/cbtf_v2.4.0
set    oss_prefix /p/home/galarowi/openss/osscbtf_v2.4.0
# Path to the qt3 toolkit
set    qt /p/home/galarowi/openss/krellroot_v2.4.0/qt3
# Path to the libraries needed for the qt4/qt5 toolkit needed for the new Qt4/Qt5 based gui
set    graphviz    /p/home/galarowi/openss/graphviz-2.40.1
set    qtgraph       /p/home/galarowi/openss/QtGraph-1.0.0
#set    papi       /opt/cray/papi/5.4.3.1

setenv  OPENSS_DOC_DIR $oss_prefix/share/doc/packages/OpenSpeedShop

#  This is needed if you use the --offline argument following the
#  convenience scripts, for example:   osspcsamp --offline "mpirun -np 4 ./nbody"
#  This is the offline mode of operation which is now built into the
#  CBTF based version of OpenSpeedShop
setenv  OPENSS_RAWDATA_DIR        .

#  For the mpi experiments only - specify the MPI implementation of your
#  application that will be run with OpenSpeedShop.   These are the
#  mpi, mpit, and mpip experiments.  All other experiment types will
#  ignore this setting.  It is only needed for mpi, mpit, and mpip.
setenv  CBTF_MPI_IMPLEMENTATION    mpich
setenv  OPENSS_MPI_IMPLEMENTATION  mpich

#  XPLAT_RSH is needed for MRNet which is now needed for use in CBTF
setenv  XPLAT_RSH               ssh

# Only need these CBTF specific variables for situations where the environment is not passed
setenv MRNET_COMM_PATH $cbtf_prefix/sbin/cbtf_mrnet_commnode
setenv CBTF_MRNET_BACKEND_PATH $cbtf_prefix/sbin/cbtf_libcbtf_mrnet_backend

# oss_prefix_target/bin must come first to
# find the osslink in the target directory

#prepend-path    PATH                $papi/bin
```

```
prepend-path   PATH              $root_prefix/bin
prepend-path   PATH              $oss_prefix/bin
prepend-path   PATH              $cbtf_prefix/bin
prepend-path   PATH              $cbtf_prefix/sbin
prepend-path   MANPATH              $oss_prefix/share/man

eval set  [ array get env HOME ]
set     ownmoddir      $HOME/privatemodules

# Set up the dyninst runtime library path for the OSS/CBTF version of OpenSpeedShop
# This is required for finding loops and gathering symbol table information.
setenv DYNINSTAPI_RT_LIB $root_prefix/lib64/libdyninstAPI_RT.so

# Might need this if you use the system installed papi, here as a hint
#prepend-path LD_LIBRARY_PATH $papi/lib64

# Setup the library paths for the runtime environment
prepend-path LD_LIBRARY_PATH $root_prefix/lib
prepend-path LD_LIBRARY_PATH $root_prefix/lib64
prepend-path LD_LIBRARY_PATH $cbtf_prefix/lib64
prepend-path LD_LIBRARY_PATH $oss_prefix/lib64

# Setup the library paths for the qt3 runtime environment
prepend-path LD_LIBRARY_PATH $qt/lib64
# Setup the library paths for the libraries needed for the qt4/qt5 new GUI runtime environment
prepend-path LD_LIBRARY_PATH $graphviz/lib
prepend-path LD_LIBRARY_PATH $qtgraph/lib64/4.8.6
```

Intel MIC Platform Module File Examples

If your system supports the environment module file runtime environment setup, then these examples will provide some guidance for your O|SS runtime environment file creation.

Intel MIC (KNL) platform CBTF version module file example

Use the generic *Generic Cluster Module File Example* shown in the example above for KNL installations.

Intel MIC (KNC co-processor) platform CBTF version module file example

```
#%Module1.0#####################################################################
##
##
## osscbtf modulefile
##
proc ModulesHelp { } {
      global version osscbtf

      puts stderr "\topenss - loads the O|SS software & application environment"
      puts stderr "\n\tThis adds $oss/* to several of the"
      puts stderr "\tenvironment variables."
      puts stderr "\n\tVersion $version\n"
}

module-whatis   "Loads the O|SS CBTF instrumentor based runtime environment."
```

```
# for Tcl script use only
set    version       2.4.0
set    root          /global/u2/j/jgalaro/cori/krellroot_stable
set    dynroot       /global/u2/j/jgalaro/cori/krellroot_stable
set    boostroot     /global/u2/j/jgalaro/cori/krellroot_stable
set    mrnetroot     /global/u2/j/jgalaro/cori/krellroot_stable
set    cbtf /global/u2/j/jgalaro/cori/jgalaro/cbtf_only_stable
set    cbtfk /global/u2/j/jgalaro/cori/jgalaro/cbtf_only_stable
set    oss /global/u2/j/jgalaro/cori/jgalaro/osscbtf_stable
set    stdcpp        /opt/gcc/5.1.0/snos/

setenv        CBTF_MPI_IMPLEMENTATION    mpich2
setenv        XPLAT_RSH      ssh

# only need these for situations where the environment is not passed
setenv        MRNET_COMM_PATH $cbtfk/sbin/cbtf_mrnet_commnode
setenv        CBTF_MRNET_BACKEND_PATH $cbtfk/sbin/cbtf_libcbtf_mrnet_backend

prepend-path   PATH         $cbtfk/sbin
prepend-path   PATH         $cbtfk/bin
prepend-path   PATH         $oss/bin
prepend-path   MANPATH           $oss/share/man

eval set [ array get env HOME]
set    ownmoddir      $HOME/privatemodules

setenv DYNINSTAPI_RT_LIB $dynroot/lib64/libdyninstAPI_RT.so

prepend-path LD_LIBRARY_PATH $root/lib64
prepend-path LD_LIBRARY_PATH $root/lib
prepend-path LD_LIBRARY_PATH $mrnetroot/lib64
prepend-path LD_LIBRARY_PATH $dynroot/lib64
prepend-path LD_LIBRARY_PATH $oss/lib64
prepend-path LD_LIBRARY_PATH $cbtf/lib64
prepend-path LD_LIBRARY_PATH $cbtfk/lib64

# Because the module for the gcc was used to build OSS we need to make the libstdc++.so.6
# available when we are running OSS
prepend-path LD_LIBRARY_PATH $stdcpp/lib64
```

ARM platform CBTF version module file example

Use the generic *Generic Cluster Module File Example* shown in the example above for ARM installations.

Power 8 platform CBTF version module file example

Use the generic *Generic Cluster Module File Example* shown in the example above for Power 8 installations.

How to use SPACK to build O|SS

Spack Introduction

Spack is a multi-platform package manager that builds and installs multiple versions and configurations of software. It works on Linux, macOS, and many supercomputers. Spack is non-destructive: installing a new version of a package does not break existing installations, so many configurations of the same package can coexist. Most importantly, Spack is *simple*. It offers a simple *spec* syntax so that users can specify versions and configuration options concisely. Spack is also simple for package authors: package files are written in pure Python, and specs allow package authors to maintain a single file for many different builds of the same package. If you're new to spack and want to start using it, see Getting Started, or refer to the full manual below.

Open|SpeedShop (O|SS) is an open source multi-platform performance tool enabling performance analysis of HPC applications running on both single node and large scale Intel, AMD, ARM, Intel Phi, Power PC, Power 8, GPU processor based systems, including Cray and IBM Blue Gene platforms.

O|SS can be built with spack by downloading the spack package source, as described below. It is possible to build O|SS with one spack install command. Spack will download, on the fly from the package websites, all the dependent packages that O|SS needs, build and install them. After the dependent packages are built, spack will download, build, and install O|SS. Spack creates module file for all the packages that it builds. The instructions below identify where to find the module file and how to load it.

Spack allows for enabling and disabling O|SS optional build arguments/parameters via a spack feature named variants. For O|SS, the main variants are MPI implementation identifiers which are used to build the MPI collectors for those MPI implementations. The spack build command:

```
spack install openspeedshop +openmpi +mvapich2
```

will build O|SS with all the collectors, including MPI collectors that will work on OpenMPI and Mvapich2 based applications. Without those variants, all the non-mpi specific experiments will be built, but not the MPI collectors (no ossmpi, ossmpip, and ossmpit support).

As an optimization to reduce the number of packages that need to be built by the spack install command, spack has a mechanism to specify package installation locations for packages already built and installed on the system. Telling spack where to find these already installed packages, like Qt and/or MPI installations, can save a lot of build time and also allow usage of standard package installation that non-spack built executables and tools are using. The spack mechanism for this feature is the packages.yaml file that you create in your $HOME/.spack/linux directory. This file contains a list of packages that you don't want spack to build. The packages.yaml file is a list of package names and the directories where the package is already installed. See the example in the quick start guide section. This can save a lot of build time for packages like Qt, boost, python, mpi, etc...

Support for building O|SS on heterogeneous platforms is now supported as of February 2018 in the development branch of spack. Support for building on laptops, desktops, and homogeneous clusters (including Cray platforms) is working and available through the default and development version of spack.

There is now a way to have access to the latest development version of O|SS via the develop version specification when building O|SS with spack. To access the development version (master git tree), use:

```
spack install openspeedshop@develop [variants]
```

Using the @develop clause on the openspeedshop package build will also build newer versions of all the components that O|SS uses. Especially important are the cbtf packages, as they are now the core of performance data collection in O|SS.

Without the "@develop" on the openspeedshop By default, spack installs the newest/latest numbered version of a package. O|SS currently has numbered versions: 2.3.1.3, 2.3.1.4, and 2.3.1.5. By default, the 2.3.1.5 version will be built when this command is used:

```
spack install openspeedshop [variants]
```

To build another, specific version, use the version number:

```
spack install openspeedshop@2.3.1.4 [variants]
```

Quick start guides to get started building O|SS with Spack

The following is quick start guide for getting started on building O|SS with spack showing a list for commands outlining how to download spack and create an example build of O|SS.

```
$ mkdir openss_spack
$ cd openss_spack/

# Download the source from the Spack repository
$ git clone https://github.com/spack/spack.git

# Change directories into the spack repository top-level directory
$ cd spack

# Setup the spack root and path to the spack executable
$ export SPACK_ROOT=$HOME/openss_spack/spack
$ export PATH=$SPACK_ROOT/bin:$PATH

# optional aliases
        # This is an alias for getting to the package file directories
        $ alias cdpack="cd $SPACK_ROOT/var/spack/repos/builtin/packages"

        # This is an alias to get back to the top-level directory
        $ alias cdspack="cd $SPACK_ROOT"

        # Optionally, this is where the module files will be.
        # Note linux-fedora26-x86_64 was the name used on this particular platform.
        # The name will be different for other platforms
        $ alias cdmod="cd $SPACK_ROOT/share/spack/modules/linux-fedora26-x86_64 "

# Check for non-spack installation specifications in the packages.yaml file
# spack won't build packages that you specify in the packages.yaml file. spack
# will use the installation path for that package based on what path you point spack to.
$ cat ~/.spack/packages.yaml

# Locate compiler versions if you haven't already done that.   You may need to load or
# unload module files so that spack can find the compilers you want to use, if they are not
# in default install locations.
$ spack compiler find

# This file is created by the "spack compiler find" command and
# will contain the list of available compilers
$ cat $HOME/.spack/linux/compilers.yaml
```

Building cbtf, cbtf-krell, and openspeedshop using spack

```
# The + phrases are variants in spack. The +openmpi and +mvapich2
# below tell spack to build the openspeedshop mpi collectors for openmpi and mvapich2
# There are +mpt +mpich +mvapich variants as well.
$ spack install openspeedshop +openmpi +mvapich2
```

```
 # If you want a specific compiler version use the %gcc@ syntax or %intel@, etc
$ spack install openspeedshop%gcc@4.8.3 +openmpi +mvapich2

# To build a version of openspeedshop that supports the cuda experiment, use +cuda
$ spack install openspeedshop +openmpi +mvapich2 +cuda
```

Building the cbtf-argonavis-gui (new Qt4/Qt5 GUI) using spack

To build the new O|SS Qt4/Qt5 based graphical user interface use the spack install command:

```
$ spack install cbtf-argonavis-gui
```

Example usage following quick start guide commands

The following section contains the output of the quick start spack command execution with outputs showing the outputs of the example build of O|SS with spack.

```
$ cd
$ mkdir openss_spack
$ cd openss_spack /
$ git clone https://github.com/spack/spack.git
Cloning into 'spack'...
remote: Counting objects: 74393, done.
remote: Compressing objects: 100% (90/90), done.
remote: Total 74393 (delta 43), reused 46 (delta 9), pack-reused 74282
Receiving objects: 100% (74393/74393), 26.35 MiB | 4.67 MiB/s, done.
Resolving deltas: 100% (35388/35388), done.
$ ls -lastr
 total 48
40 drwxrwxrwx. 455 <username> <username> 36864 Jul 27 16:45 ..
4 drwxrwxr-x. 3 <username> <username> 4096 Jul 27 16:45 .
4 drwxrwxr-x. 8 <username> <username> 4096 Jul 27 16:45 spack

$ cd spack

$ ls -lastr
total 104
4 drwxrwxr-x. 3 <username> <username> 4096 Jul 27 16:45 ..
4 -rw-rw-r--. 1 <username> <username> 500 Jul 27 16:45 .codecov.yml
4 -rw-rw-r--. 1 <username> <username> 3695 Jul 27 16:45 .travis.yml

 4 -rw-rw-r--. 1 <username> <username> 1592 Jul 27 16:45 NOTICE
8 -rw-rw-r--. 1 <username> <username> 5896 Jul 27 16:45 .mailmap
28 -rw-rw-r--. 1 <username> <username> 26426 Jul 27 16:45 LICENSE
4 -rw-rw-r--. 1 <username> <username> 294 Jul 27 16:45 .gitignore
4 -rw-rw-r--. 1 <username> <username> 827 Jul 27 16:45 .flake8
4 -rw-rw-r--. 1 <username> <username> 746 Jul 27 16:45 .coveragerc
8 -rw-rw-r--. 1 <username> <username> 4512 Jul 27 16:45 README.md
4 drwxrwxr-x. 2 <username> <username> 4096 Jul 27 16:45 bin
4 drwxrwxr-x. 3 <username> <username> 4096 Jul 27 16:45 etc
4 drwxrwxr-x. 3 <username> <username> 4096 Jul 27 16:45 lib
4 drwxrwxr-x. 3 <username> <username> 4096 Jul 27 16:45 share
4 -rw-rw-r--. 1 <username> <username> 114 Jul 27 16:45 pytest.ini
4 drwxrwxr-x. 3 <username> <username> 4096 Jul 27 16:45 var
4 drwxrwxr-x. 8 <username> <username> 4096 Jul 27 16:45 .
```

```
4 drwxrwxr-x. 8 <username> <username> 4096 Jul 27 16:45 .git

$ **pwd**
 /home/<username>/openss_spack/spack

# Set up the spack root and path that spack needs to execute
$ export SPACK_ROOT=/home/<username>/openss_spack/spack
$ export PATH=$SPACK_ROOT/bin:$PATH

# These are just convenience aliases that I use
$ alias cdpack="cd $SPACK_ROOT/var/spack/repos/builtin/packages"
$ alias cdmod="cd $SPACK_ROOT/share/spack/modules/"
$ alias cdspack="cd $SPACK_ROOT"

$ cat ~/.spack/packages.yaml
packages:
   mvapich2:
        buildable: False
         paths:
             mvapich2@2.2%gcc@4.8.3 arch=linux-fedora26-x86_64: /opt/mvapich2-1.2p1
   openmpi:
         buildable: False
         paths:
             openmpi@1.8.2%gcc@4.8.3 arch=linux-fedora26-x86_64: /opt/openmpi-1.8.2
   cuda:
        buildable: False
        paths:
            cuda@6.0%gcc@4.8.3 arch=linux-fedora26-x86_64: /usr/local/cuda-6.0
  glib:
        paths:
            glib@1.2.10-39%gcc@4.8.3 arch=linux-fedora26-x86_64: /usr
  qt:  +krellpatch
        paths:
            qt@3.3.8b%gcc@4.8.3 arch=linux-fedora26-x86_64: /usr/lib64/qt-3.3


$ spack compiler find
 ==> Found no new compilers
==> Compilers are defined in the following files:
/home/<username>/.spack/linux/compilers.yaml

$ cat /home/<username>/.spack/linux/compilers.yaml
compilers:
- compiler:
      environment: {}
      extra_rpaths: []
      flags: {}
      modules: []
      operating_system: fedora26
      paths:
         cc: /usr/bin/gcc
         cxx: /usr/bin/g++
         f77: /usr/bin/gfortran
         fc: /usr/bin/gfortran
      spec: gcc@4.8.3
      target: x86_64
```

# You only need the %gcc@4.8.3 if you want to build with different compiler versions
# in the future and want to keep the builds separate.
$ **spack install openspeedshop%gcc@4.8.3 +openmpi +mvapich2**

```
 ==> Installing libsigsegv
==> Fetching https://ftp.gnu.org/gnu/libsigsegv/libsigsegv-2.11.tar.gz
######################################################################## 100.0%
==> Staging archive: /home/<username>/openss_spack/spack/var/spack/stage/libsigsegv-2.11-
zcczh3kvlgq6mhlb7oogx6yjoxlsiuow/libsigsegv-2.11.tar.gz
==> Created stage in /home/<username>/openss_spack/spack/var/spack/stage/libsigsegv-2.11-
zcczh3kvlgq6mhlb7oogx6yjoxlsiuow
==> Building libsigsegv [AutotoolsPackage]
==> Executing phase : 'autoreconf'
==> Executing phase : 'configure'
...
...
```

# Once the build is finished the module file will be located in this location:
# Note: **linux-fedora26-x86_64** will be a different name in your builds == platform type dependent.
$ $SPACK_ROOT/share/spack/modules/linux-fedora26-x86_64

# To use O|SS and the underlying cbtf components
$ module use $SPACK_ROOT/share/spack/modules/linux-fedora26-x86_64
$ module load openspeedshop...(long list of characters)

Getting the latest spack sources, i.e. the development branch and updating to the latest source.

If there are fixes that have not made it into the master version and you need those fixes, this is a way to get the latest sources.

# Download the source from the Spack repository
$ git clone https://github.com/spack/spack.git

# Change directories into the spack base directory
$ cd spack

# Change git to point to the development sources
$ git checkout -t origin/develop

# Add the repository as the remote upstream value
$ git remote add upstream https://github.com/spack/spack.git

# Fetch the latest sources from the repository
$ git pull upstream develop

# Now you have the latest sources to use for the build or development work
# Note: When you update there is a good possibility that the versions of many packages will change. If you are using a spack mirror (source for packages on platforms with restricted internet access) you will need to update the spack mirror so that you have the new source code corresponding to the new versions being used in spack due to the update.

Details about O|SS specific spack options

What components are supported

Builds for the spack packages listed below have been developed and are supported by the Krell Institute.

- openspeedshop
- cbtf
- cbtf-krell
- cbtf-argonavis
- cbtf-lanl
- cbtf-argonavis-gui
- qtgraph (support for the cbtf-argonavis-gui)

Component Variants Explained

Each of the Krell Institute supported packages allows a number of spack variants. A variant is a build option that the user can specify to control build options for the component. For example, if the user wants cbtf-krell to build openmpi MPI data collectors then they would specify the spack variant +openmpi on the spack install command line for building cbtf-krell, like this: spack install cbtf-krell +openmpi

Internal to the cbtf-krell package file, this looks like this, where the variant is causing a cmake variable to be set:

```
if spec.satisfies('+openmpi'):
        MPIOptions.append('-DOPENMPI_DIR=%s' % spec['openmpi'].prefix)
```

Without the openmpi variant, the spack build for cbtf-krell would not build MPI experiment collectors for the openmpi MPI implementation.

The following tables list the component, variant name and explanation of the purpose for the variant. cbtf-argonavis-gui and qtgraph have no variants at this time.

Component	Variant	Explanation / Purpose
openspeedshop	cbtf	Build with cbtf instrumentor enabled
	crayfe	Build only the front-end tool using the runtime libraries from the compute node build. Used for heterogeneous builds.
	cti	Build MRNet with the CTI startup option.
	cuda	Build with cuda packages included.
	gui	Build or not build a GUI of choice, options: none, qt3, qt4
	runtime	Build only the runtime libraries and collectors. Used for heterogeneous builds.
	offline	Not used – old offline build option
	mpich	Build the mpich MPI implementation collectors, passed to cbtf-krell for actual build
	mpt	Build the mpt MPI implementation collectors, passed to cbtf-krell for actual build
	mvapich	Build the mvapich MPI implementation collectors, passed to cbtf-krell for actual build
	mvapich2	Build the mvapich2 MPI implementation collectors, passed to cbtf-krell for actual build
	openmpi	Build the openmpi MPI implementation collectors, passed to cbtf-krell for actual build

Component	Variant	Explanation / Purpose
cbtf	runtime	Build only the runtime libraries and collectors. Used for heterogeneous builds.

Component	Variant	Explanation / Purpose
cbtf-krell	crayfe	Build only the front-end tool using the runtime libraries from the compute node build. Used for heterogeneous builds.
	cti	Build MRNet with the CTI startup option.
	runtime	Build only the runtime libraries and collectors. Used for heterogeneous builds.
	mpich	Build the mpich MPI implementation collectors
	mpt	Build the mpt MPI implementation collectors
	mvapich	Build the mvapich MPI implementation collectors
	mvapich2	Build the mvapich2 MPI implementation collectors
	openmpi	Build the openmpi MPI implementation collectors

Component	Variant	Explanation / Purpose
cbtf-argonavis	runtime	Not implemented: Build only the runtime libraries and collectors. Used for heterogeneous builds.

Component	Variant	Explanation / Purpose
cbtf-lanl	runtime	Not implemented: Build only the runtime libraries and collectors. Used for heterogeneous builds.

Module file creation is automatic

Spack will create module files for each component, but they will be very basic, mainly setting the PATH variable to the install bin directory for the package being built.   However, spack allow the authors of packages to add to the module file by supplying spack functions that will set environment variables and append to library paths or path variables.   So far, only the openspeedshop module file has been enhanced by the Krell Institute.

The module file directory can be found at this location: <spack directory>/share/spack/modules/<machine type>.   All the spack created modules will be located at this location.   For O|SS, all one should have to do is load the openspeedshop-<long name> module file in order to run experiments and view performance databases.

Restricted network access solution

Spack has a feature to handle cases when your network prevents accessing source on the platform you are building on.   Many laboratory systems have security measures in place that prevent the default workflow of spack from working. Spack is unable to download packages from across the web due to the security checks. For this situation, spack provides a mirror capability, where a directory of the package source files can be created based on a spack mirror create command. For more spack documentation see this url:http://spack.readthedocs.io/en/latest/mirrors.html?highlight=mirror

See below for an example of an O|SS mirror creation on a platform that allows downloading package sources and activation on the platform that doesn't.

Note that for the build on the restricted access platform can succeed, make sure that the restricted platform version of spack matches with the spack being used to create the mirror. If not, some package source versions might not be available in the mirror, causing the build to fail.

# Create mirror which contains all possible variants you might run

# Do this on a machine that has full access to the internet and the files needed
# Note: the gcc version information is an example only, your gcc version may be different or not required
$ spack mirror create -d /home/<username>/spack-mirror -D openspeedshop+openmpi+mvapich2+mpich%gcc@4.8

# Because the new O|SS gui is not built in the base openspeedshop package,
# a separate mirror command is necessary if building the Qt4/Qt5 GUI is desired
$ spack mirror create -d /home/<username>/spack-mirror –D cbtf-argonavis-gui

# Tar and gzip this directory for move to build platform
$ tar -cvf spack_mirror.tar /home/<username>/spack-mirror-2018-04-05
$ gzip spack_mirror-2018-04-05.tar

# Un-zip and tar on secure/build platform
$ tar -zxvf spack_mirror-2018-04-05.tar.gz

# Set up spack to use this mirror directory
# After executing this command, the spack build will read files from the mirror directory.
$ spack mirror add linux-spack-mirror /home/user/spack-mirror-2018-04-05

# Note: For heterogeneous builds the compute node build needs its own mirror and be sure to change the module file for the compute node build and mirror. The linux front-end mirror is not visible when building for the compute node. This is a spack-ism, not related to O|SS.
$ spack mirror add compute-spack-mirror /home/user/spack-mirror-2018-04-05

# To remove a spack mirror from being used to find the package file source use this command:
$ spack mirror remove linux-spack-mirror

Example Cray Spack Build: Homogeneous processor platform

Building O|SS on a homogeneous processor platform is very similar to building on a cluster with the exception of handling the programming environment module files. Basically, one needs to unload whatever PrgEnv-{intel, gnu, cray, pgi} environment module is loaded and then begin the spack build process. The example below will use a spack mirror in order to show that aspect of the build process. This is an optional step and is not needed on systems with full access to the internet for accessing source files needed in the build. This example will be a complete list of the steps required to build on a homogenous Cray system.

# Create a sub-directory to down load spack into
$ **cd <mypath>**
$ **mkdir openss_spack**
$ **cd openss_spack /**

# Download the source from the Spack repository
$ git clone https://github.com/spack/spack.git

# Change directories into the spack base directory
$ cd spack

# Change git to point to the development sources
$ git checkout -t origin/develop

```
# Add the repository as the remote upstream value
$ git remote add upstream https://github.com/spack/spack.git

# Fetch the latest sources from the repository
$ git pull upstream develop

# Set up the spack root and path that spack needs to execute
$ export SPACK_ROOT=<mypath>/openss_spack/spack
$ export PATH=$SPACK_ROOT/bin:$PATH

# These are just convenience aliases that can be used to navigate the spack directories
$ alias cdpack="cd $SPACK_ROOT/var/spack/repos/builtin/packages"
$ alias cdmod="cd $SPACK_ROOT/share/spack/modules/"
$ alias cdspack="cd $SPACK_ROOT"

# Next: examine and if needed set up paths to the packages you may not want spack to build
# Using existing installations of common packages can be accomplished in spack by using the
packages.yaml file located in $HOME/.spack directory.   In this example, only the existing version of
mpich is being used in place of spack building mpich.

$ cat ~/.spack/packages.yaml
openss_spack/spack> cat ~/.spack/packages.yaml
packages:
  mpich:
    buildable: False
    paths:
      mpich@7.6.2%gcc@4.8 arch=cray-CNL-haswell: /opt/cray/pe/mpt/7.7.0/gni/mpich-gnu/5.1

$ module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                             13) job/2.2.2-6.0.5.0_8.47__g3c644b5.ari
  2) eproxy/2.0.22-6.0.5.0_2.1__g1ebe45c.ari      14) dvs/2.7_2.2.53-6.0.5.2_8.11__ge13250e
  3) intel/17.0.4                                 15) alps/6.5.28-6.0.5.0_18.6__g13a91b6.ari
  4) craype-network-aries                         16) rca/2.2.16-6.0.5.0_15.34__g5e09e6d.ari
  5) craype/2.5.13                                17) atp/2.1.1
  6) cray-libsci/17.12.1                          18) perftools-base/7.0.0
  7) udreg/2.3.2-6.0.5.0_13.12__ga14955a.ari      19) PrgEnv-intel/6.0.4
  8) ugni/6.0.14-6.0.5.0_16.9__g19583bb.ari       20) cray-mpich/7.7.0
  9) pmi/5.0.13                                   21) craype-haswell
 10) dmapp/7.1.1-6.0.5.0_49.8__g1125556.ari       22) slurm/17.02.10-SSE.0
 11) gni-headers/5.0.12-6.0.5.0_2.15__g2ef1ebc.ari 23) craype-hugepages2M
 12) xpmem/2.2.4-6.0.5.1_8.17__g35d5e73.ari

# Note: These are the compute node compilers and not what we want, although the build might be
successful.   These were found because the PrgEnv-intel module was set.  This indicates we are building
code for the compute nodes.   We want the login/front-end node compilers.
 $ spack compiler find
==> Added 25 new compilers to $HOME/.spack/cray/compilers7.1.0  gcc@6.1.0  gcc@4.9.3  cce@8.6.1
cce@8.5.7  cce@8.5.3  cce@8.4.6
   intel@17.0.1  intel@15.0.5  gcc@6.3.0  gcc@5.3.0  cce@8.6.5  cce@8.6.0  cce@8.5.6  cce@8.5.2
   intel@16.0.3  gcc@7.2.0    gcc@6.2.0  gcc@5.2.0  cce@8.6.2  cce@8.5.8  cce@8.5.5  cce@8.5.0
==> Compilers are defined in the following files:
   $HOME/.spack/cray/compilers.yaml

# Unload the compute node build environment module
$ module unload PrgEnv-intel/6.0.4
```

```
# Now ask spack to find front-end node compilers that we can use to build
$ spack compiler find
==> Added 1 new compiler to /users/<username>/.spack/linux/compilers.yaml
    gcc@4.8
==> Compilers are defined in the following files:
    /users/<username>/.spack/linux/compilers.yaml

# This is what the compilers.yaml file looks like
# You may have to edit the f77 and fc lines, this is a spack bug
# They should point to the GNU Fortran compiler
$ cat /users/<username>/.spack/linux/compilers.yaml
compilers:
- compiler:
    environment: {}
    extra_rpaths: []
    flags: {}
    modules: []
    operating_system: sles12
    paths:
      cc: /usr/bin/gcc-4.8
      cxx: /usr/bin/g++-4.8
      f77: null
      fc: null
    spec: gcc@4.8
    target: x86_64


# Note: If the system where the build is taking place has restricted internet connection
# a spack mirror may be necessary.   See the Restricted network access solution section
# in this document for how to set the mirror up prior to continuing this build scenario.
# If needed the following command will direct spack to find the package source
# files in the mirror directory.
# After executing this command, the spack build will read files from the mirror directory.
# $ spack mirror add linux-spack-mirror /home/user/spack-mirror-2018-04-05

# Now we are ready to build O|SS using spack on this Cray system
# This spack command causes a dependency based build of all the packages that
# O|SS depends on to be built.  This then causes all the dependencies of
# those packages to be built and so on.
$ spack install openspeedshop +cbtf+mpich %gcc@4.8 ^elfutils@0.170

# Once the build is completed, one can visit the module file directory to see
# the name of the O|SS module file.  That will be what you
# module load to get access to the O|SS performance tool.
# The actual location of the modules is dependent on the machine type name
# Below, we change directories to the upper level directory for the module files location
# Do an "ls" command to see the name of the sub-directory for the actual location of the module files
$ cd $SPACK_ROOT/share/spack/modules
$ ls
linux-sles12-x86_64

# On this platform it is: linux-sles12-x86_64
# To use the module file for O|SS, first do a module use
$ module use SPACK_ROOT/share/spack/modules/linux-sles12-x86_64
```

```
# Then load the openspeedshop-<long name for the module>
# In this particular spack build, the O|SS module name is:
# openspeedshop-2.3.1-gcc-4.8-t34et3a
```
$ module load openspeedshop-2.3.1-gcc-4.8-t34et3a

```
# Now one can run O|SS experiments, such as osspcsamp, ossusertime, etc.
# The module file load, loads all the necessary runtime and data
# viewing bin and library paths and sets up the necessary environment variables.
```

Convenience cut and paste commands for the above scenario:

```
mkdir openss_spack
cd openss_spack
git clone https://github.com/spack/spack.git
git checkout -t origin/develop
cd spack
git checkout -t origin/develop
git remote add upstream https://github.com/spack/spack.git
git pull upstream develop
setenv SPACK_ROOT `pwd`
setenv PATH "${SPACK_ROOT}/bin:$PATH"
alias cdpack "cd $SPACK_ROOT/var/spack/repos/builtin/packages"
alias cdmodc "cd $SPACK_ROOT/share/spack/modules/cray-CNL-haswell"
alias cdmodl "cd $SPACK_ROOT/share/spack/modules/linux-sles12-x86_64"
alias cdmod "cd $SPACK_ROOT/share/spack/modules/linux-sles12-x86_64"
alias cdspack "cd $SPACK_ROOT
cat ~/.spack/packages.yaml
packages:
  all:
    providers:
      pkgconfig: [pkg-config]
  elfutils:
    versions: [0.170]
  mpich:
    buildable: False
    paths:
      mpich@7.6.2%gcc@4.8 arch=cray-CNL-haswell: /opt/cray/pe/mpt/7.7.0/gni/mpich-gnu/5.1
module list
spack compiler find
module unload PrgEnv-intel/6.0.4
spack compiler find
cat /users/<username>/.spack/linux/compilers.yaml
spack find
# Optional mirror steps
#  On machine with internet access
spack mirror create -d /home/<username>/spack-mirror –D
openspeedshop+openmpi+mvapich2+mpich%gcc@4.8
spack mirror create -d /home/<username>/spack-mirror –D cbtf-argonavis-gui
tar -cvf spack_mirror.tar /home/<username>/spack-mirror-2018-04-05
gzip spack_mirror-2018-04-05.tar
# scp spack_mirror-2018-04-05.tar.gz to build machine
#  On build machine without or limited internet access
tar -zxvf spack_mirror-2018-04-05.tar.gz
spack mirror add linux-spack-mirror /home/<username>/spack-mirror-2018-04-05

spack install openspeedshop +cbtf+mpich %gcc@4.8 ^elfutils@0.170
cd $SPACK_ROOT/share/spack/modules/linux-sles12-x86_64
module use SPACK_ROOT/share/spack/modules/ linux-sles12-x86_64
module load openspeedshop-2.3.1-gcc-4.8-t34et3a
```

Spack Based Heterogeneous O|SS Builds

With the current feature set of spack, in order to build O|SS to run a heterogeneous platform, components that run on the front-end processor must be built with a separate spack invocation than that of the components that run on the compute nodes.   Therefore, building requires two spack commands, one aimed at building the front-end components and one aimed at building the compute node components.   The scenario also involves changing modules in order for spack to find the proper compilers.   So, for example if a Cray or other platform truly has processors which can't share common compiled code across the compute and front-end process, then one needs to use this scenario build O|SS.

You need to do the steps outlined in the quick start section above before falling into Step 1 and Step 2 detailed below.

First, the compute node components of O|SS must be built using these commands:

Step 1:
# Use the compute node GNU compilers
$ module load PrgEnv-gnu

# Find compilers to use in the build, use the default on if possible
$ spack compiler find

# Install cbtf-krell contains the runtime data collection code that runs on the compute nodes
# Note: the %gcc… indicates that this version of the compiler be used in the build process
$ spack install cbtf-krell %gcc@4.8 +runtime+mpich os=CNL

Step 2:

# Install the O|SS client tool which will pick up the compute node components in its build
$ spack install openspeedshop +cbtf+mpich %gcc@4.8

Post build steps:

# Load the O|SS module file to setup variables and load the runtime components
$ module use <spack module file locaton>
For example: module use ${SPACK_ROOT}/share/spack/modules/linux-sles12-x86_64

$ module load openspeedshop-<module file name, which is build dependent>
# For example:
$ module load openspeedshop-2.3-gcc-7.2.0-fxhymxn