# Developing/Modifying SDKs

**Note**: all instructions on this page assume you are at a SLAC system with access to Lab 1.

## Developing the SDK

Each SDK has a development branch, which is only modifiable when checking out from a development SDK. The user SDKs (q.v.) aren't modifiable by users.

### Fetch Development branch of SDK

To fetch the development branch of an SDK, the script **fetchDevSdk** is provided in `/afs/slac/g/cci/bin/sdktools/fetchDevSdk`. Here is the usage statement when invoked without arguments:

```
Usage: fetchDevSdk <architecture> [<architecture> ...]
       fetchDevSdk all
Fetches one or more SDKs from the development tree.
Valid architectures: arm-linux-rceCA9 arm-rtems-rceCA9 i86-linux-32 i86-linux-64
```

Your SDKs will end up in your current directory with the architecture name. Once you wish to start developing, you should create a new branch and then merge back to development. For branching and merging info see the Git - Book.

### Modifying the SDK

Once you have a development SDK, you can start modifying files.  If you're updating libraries and include files from the workspace, see the next section.

Create your own branch with `git branch`.  Change the scripts/examples/host tools as you wish. `git status` will tell you what you have modified, then use `git add`, `git commit`, and finally `git merge` and `git push` to propagate your changes back to the origin development repositories.

### Updating the SDK from the workspace

If you are updating the SDKs with code compiled in a workspace (the core development environment, not the SDK environment), a python script in the workspace provides hooks for copying the relevant files to the SDK:

```
$ workspace/make/tools/updateSDK.py --help
Usage: updateSDK.py [options]

Options:
  -h, --help            show this help message and exit
  --sdk=SDK             Location of SDK git directory
  --work=WORK           Location of workspace (for RTEMS template code)
  --build=BUILD         Location of build directory
  --arch=ARCH           Target Architecture (e.g. arm-rtems-rceCA9-opt)
  --exclude=EXCLUDE     File of items to exclude
  --rtemsInclude=RTEMSINCLUDE
                        Location of RTEMS Include files
  --fpgaSrc=FPGASRC     Location of FPGA bit files
```

Options, with a bit more explanation:

- --sdk The location of the SDK to update **REQUIRED**
- --arch The SDK architecture **REQUIRED**
- --work The location of the workspace (default=./)
- --build The location of the workspace build directory (default=work/build)
- --exclude A file saying what **not** to put into the SDK. (Defaults are in work/make/sdk/<arch>)
- --rtemsInclude Only used if RTEMS include files are being updated
- --fpgaSrc Use when updating 'fpga.bit' files

### Regression test the prospective tag

- Test install onto an RCE from the development SDK using the instructions at SDK Regression.
- Create an SDK card from the SDK (also following the instructions referenced above).

## Releasing

### Snapshot and Tag the core code

Typically in SLAC DAT's development model, the core code is being tagged from the current trunk, not from a set of branches. A script is available in the core system to do a snapshot of the core trunk, collecting the workspace and the other projects together into a single chunk that's easy to deal with:

```
$ workspace/make/tools/snapshot.sh "commit message"
```

After running, this script prints out the location of your new snapshot in the DAT SVN repository. The location is relative to $SVNROOT, and is of the format "core_tags/snapshot/YYYYMMDD_HHmm". From here, you can do the "final" tag by copying the snapshot (using svn cp) to core_tags/prod/VX.Y.Z, like in this example:

```
$ svn cp $SVNROOT/core_tags/snapshot/20140813_1225 $SVNROOT/core_tags/prod/V0.4.0 -m "Official tag for V0.4.0"
```

### Build the tagged workspace in AFS

We build the workspace in AFS `/afs/slac/g/cci/sdimages/workspaces` and do the final population of the SDKs from this source. A script is provided to build exactly what's needed, and no more.

1. `cd` to `/afs/slac/g/cci/sdimages/workspaces`
2. `svn co -q $SVNROOT/core_tags/prod/VX.Y.Z`
3. `cd VX.Y.Z`
4. ../makerelease.sh

### Prepare the DEV SDKs for release

One last time, run `updateSDK.py` from the workspace onto each of the development SDKs. Review the changes using `git status` and `git diff`.

Create the tag by updating the TAG file in the root of each SDK. After updating TAG, update the RELEASE_NOTES for each SDK to reflect the change set. Once you're confident that you have everything, add and commit the new/changed files with `git add` and `git commit`. Now, push the changes to the remote repository with `git push` (run from each git repository you've modified).

### Tag the SDK

`cd` to the development master repositories in `/afs/slac/www/projects/CTK/SDK/dev/<arch>`. For each SDK repository subdir (include, lib, tgt,...):

```
/afs/slac/www/projects/CTK/SDK/dev/sdk_scripts/tag_repo.sh <os> <tag>
```

where `os` is either linux or rtems, and `tag` is the VX.Y.Z tag you chose above.

### Propogate the tag to the user SDK

The final step is to make these changes visible to the user outside SLAC.

```
cd /afs/slac/www/projects/CTK/SDK/
dev/sdk_scripts/pull-user-from-dev
```

### Check out the user SDKs into AFS

A copy of the user SDK is kept on AFS in order to build SD cards.

```
cd /afs/slac/g/cci/sdimages/sdk
mkdir VX.Y.Z
cd VX.Y.Z
../fetchAllUser VX.Y.Z
```

### Populate the SDK root structure

The directory `/afs/slac/g/cci/sdimages` contains the top level structures needed for creating the SD cards used by the DPM/DTMs. This area contains several things, such as the linux kernel and the ramdisk image, which are only used while creating SD cards from scratch. Each of these structures are versioned, since the contents of an SD card is highly release dependent.

A good rule of thumb is to copy the previous version (using `/bin/cp -d` to prevent link dereferencing) and update the SVN revision number in version.txt. Retrieve the SVN revision with `svn info` in the tagged workspace. After updating version.txt, update the workspace and sdk soft links.

Finally, after this is all done, update the `current` links in `sdimages`, `sdimages/workspaces` and `sdimages/sdk`.

### Announce the tag

Finally, the RELEASE_NOTES should be propogated to the SDK Releases page, and an announcement should be made to the current user list.

# Appendix

### SDK image root structure

This documents the directory structure in `/afs/slac/g/cci/sdimages`. TBD