## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLT Developers Checklist</td>
<td>1</td>
</tr>
<tr>
<td>Shortlist</td>
<td>1</td>
</tr>
<tr>
<td>Make or CMT make?</td>
<td>2</td>
</tr>
<tr>
<td>Creating your development/testing sandbox</td>
<td>3</td>
</tr>
<tr>
<td>Writing a QMTest</td>
<td>3</td>
</tr>
<tr>
<td>Simplify the problem as much as possible</td>
<td>3</td>
</tr>
<tr>
<td>Easy ways to simplify things: use GaudiExcise</td>
<td>3</td>
</tr>
<tr>
<td>How can I check I didn't change anything else</td>
<td>3</td>
</tr>
<tr>
<td>Using Moore QMTests</td>
<td>4</td>
</tr>
</tbody>
</table>
HLT Developers Checklist

LHCb code development is fully described in SVNUsageGuidelines. You must be familiar with the guidelines laid out there, for which this page is just a summary.

Shortlist

In this section, the ideal development strategy is presented, summarizing what's in SVNUsageGuidelines, in a bit more ordered fashion.

<table>
<thead>
<tr>
<th>Description:</th>
<th>Video Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before you start editing:</td>
<td></td>
</tr>
<tr>
<td>1. Discuss your proposed development with the release manager.</td>
<td></td>
</tr>
<tr>
<td>2. Make a savannah task and assign to yourself, or take over an existing task for this development. You can be as verbose as you like on the tasks you are assigned to. Treat it like a log book, if you like.</td>
<td></td>
</tr>
<tr>
<td>3. Decide or discuss which version of Moore this development is for, most likely it is for the Head. HEAD modifications can normally be developed either on top of the latest released version or the HEAD which is being built each night in the nightlies. If you're working with the head release, take care that the previous nightlies did actually work, and if not, discuss with the release manager what to do.</td>
<td></td>
</tr>
<tr>
<td>4. Consider talking about your proposed development in the HLT operations meetings (on Friday afternoons)</td>
<td></td>
</tr>
<tr>
<td>Setup your local project structure and environment,</td>
<td></td>
</tr>
<tr>
<td>1. first by cleaning out everything which is already in ~/cmtuser/Moore HEAD (replace HEAD with your version, or the latest released version)</td>
<td></td>
</tr>
<tr>
<td>2. Setup the appropriate project. For the HEAD in the nightlies this is: SetupProject Moore HEAD --nightly lhcb-head --build-env #replace head with your version...</td>
<td></td>
</tr>
<tr>
<td>3. Checkout ONLY the code you want to modify: getpack Hlt/MyPackage head #replace with your package name. If you don't getpack the head here, you'll have problems later.</td>
<td></td>
</tr>
<tr>
<td>4. make #typed directly in the top directory</td>
<td></td>
</tr>
<tr>
<td>5. Setup a second time! SetupProject Moore HEAD --nightly lhcb-head i.e. without the Build-env. You must have typed SetupProject twice, once before and once after the make.</td>
<td></td>
</tr>
</tbody>
</table>

Think about TestDrivenDevelopment:
1. To fix any problem, you will anyway first need to check you can reproduce it, and then re-run this test to verify when you think you have fixed it. It's best to encapsulate this in the existing testing framework.

2. Ideally you will add a QM-Test, either in Moore or in your package which demonstrates the problem, so that you know when it is fixed and so that the nightly tests will clearly fail before it is fixed. Contact your release manager if you don't know how to do this.

3. Running your own qmtest can be done simply by going to the cmt directory of your project, and typing something like `cmt TestPackage mypackage.mynewtest`.

The editing process:

1. Once your local package structure is created, and the first build worked, you will then only need to type `SetupProject` once (without build-env). If you get pack something else, best to repeat the earlier steps.
2. Modify what you want to modify.
3. Compile your changes (`make` in the top-level directory, or see below)
4. Test your changes
5. Repeat 2-4.
6. Document continuously. In the savannah task, if you have one, and also always in the doc/release.notes file.
7. Commit often, but only code which compiles and runs. If you've added a test this is easy to verify. Read the Rules here!! Commit messages should really tell us what has changed, and whether it works. Don't use stupid trivial commit messages.

8. Communicate effectively with your release manager during this process. Easiest way to do this is through savannah and the tag collector.
9. Keep an eye on the nightlies (of your modifications take more than a day to complete).

Aftercare:

1. Update/close savannah tasks
2. Ensure your changesets appear in the the tag collector
3. Ensure your documentation is really complete and understandable.
4. Double-check that any failures in the nightlies are not due to your modifications. Probably you need to check with your release manager.
5. Present results in the Friday HLT operations meeting.

Make or CMT make?

- Typing `make` in the top level directory will ensure everything you have getpacked is up-to-date and built. It is slow, though.
If you've only modified one package, you can type `cmt make` in the cmt directory of that package instead, which is quicker, but can cause problems if other things have changed.

### Creating your development/testing sandbox

When you start your development, you're going to want to have some measurable outcome that really tells you if you have made the improvement you though you had made, and not messed anything else up in the process.

Creating this safe space where you know what and where you are changing things, and you know what is going on before and after your test is "sandboxing" and can be used to be really sure you are only changing what you thought.

### Writing a QMTest

If you are starting your development, you can write a QMTest which verifies the problem you thought was there really is there, in some measurable way that you can really see. Any generic Gaudi job can be wrapped rather straightforwardly into a test within whatever package you are using.

### Simplify the problem as much as possible

The first step in correcting a problem is isolating a problem, to identify the cause. This isolation step should result in some repeatable behaviour which you can clearly identify, and maybe solidify into a QMTest.

### Easy ways to simplify things: use GaudiExcise

GaudiExcise can cut out a single algorithm from a complicated sequence and make a sandbox for you to repeatedly run only the algorithm you are changing.

### How can I check I didn't change anything else:

Ideally the release manager and the integration tests in the nightlies will be checking this for you. However, even if the nightly tests do break, you initially probably have no way of knowing if it was you that really broke something. A lot of the time, the nightlies are broken for completely different reasons and failures are nothing to do with you.

Also, the nightly tests might not be useful for your exact use case, and might be too time-consuming to run yourself. You may be working on some branch, or in some other complicated mode.

We are working on a tool called GaudiRegressionTester to make this easy for you.... pretty much you should be running:

- **Collect the benchmark from before your change**

  ```bash
  SetupProject Moore --no-user-area #choose the version you want to test your change against, --no-
gaudiregressiontester.py $MOOREROOT/tests/qmtest/moore.qms/physics.qms.2012 #or some options file
  ```

  - This will store a local benchmark.xml (xmlsummary), benchmark.csv (timingtable), and benchmark.log (warnings/errors/fatal)
  - **Run the same thing with your changes**

    ```bash
    SetupProject Moore #choose the version you want to test your change against, --no-user-area means
    #getpack/make whatever you may need to pick up your changes
gaudiregressiontester.py $MOOREROOT/tests/qmtest/moore.qms/physics.qms.2012 #or some options file
    ```
**Using Moore QMTests**

There are several qmtests that run moore in Hlt/Moore/tests/

`cmt TestPackage` will run all the tests, as usual.

`cmt TestPackage moore.physics.deferral` will only run the tests suit for the split HLT

To manually run Moore with the different options available, look into Hlt/Moore/tests/options/. For example to run Hlt1 Only (split scenario) do:

```
 gaudirun.py Moore_Hlt1AndHlt2.py Moore_Hlt1Only.py | tee mylogfile.log
```

How can I check I didn't change anything else: GaudiRegressionTester