Merging commits from trunk to fixes branch

The Camel 2.x branches are here:

- https://svn.apache.org/repos/asf/camel/branches/

What should be merged

The idea is to be able to apply bug fixes and backwards compatible improvements and new features to our Camel maintenance branches (for example 2.10.x and 2.9.x) while leaving **NON** backwards compatible changes on the trunk. The idea is to give our users the best experience possible on any supported branch as long we have 100% backward compatibility on patch versions.

What should be considered as NOT backwards compatible

- change in our public API
- change in the behavior of a component
- change in a default value of a component

Changes which SHOULD apply to maintenance branches

- Bug fix which is backwards compatible
- Smaller improvement which is backwards compatible
- Small new feature which is backwards compatible
- Third party dependency updates on micro/patch versions (3rd digit)

Changes which MUST NOT be applied to maintenance branches

- Bug fix which is **NOT** backwards compatible
- Improvement which is **NOT** backwards compatible
- New feature which is **NOT** backwards compatible
- Non trivial refactoring

Changes which MAY be applied to maintenance branches

For all the changes in this category we have to be especially carefully to not break backwards compatibly. Again, the goal is to be 100% backward compatibility on patch versions. Take extra time to review and test your change. Even better, send a [HEADS UP] on the dev@ mailing list and ask for assistance/review.

- Non trivial improvement which is backwards compatible
- Non trivial new feature which is backwards compatible
- Third party dependency updates on major or minor versions (1st and 2nd digit)
- Trivial refactoring

Who should do the merge

It is preferred that the committer who applied the change to trunk also merge it back to the maintenance branches. He knows best whether this fix should go into the maintenance branche(s) or not and he can also make sure the WIKI pages are up to date. However, other people may also merge fixes back if they require it there. In that case, those people should pay extra attention to make sure the changes meet the above criteria.

How to merge

Using svnmerge.py script

I've set up svnmerge.py to track commits from the trunk to the 2.x branches.

Example workflow:

1. You just committed a fix to the trunk in revision 123456 and think that it would be back ported to Camel 2.8.x users

2. Check out the branch

```
svn co https://svn.apache.org/repos/asf/camel/branches/camel-2.8.x camel-2.8.x
```

3. In camel-2.8.x directory, you can get a list of commits available from the trunk

```
svnmerge.py avail
```

4. Merge your commit by running

```
svnmerge.py merge -r 123456
```

5. Resolve any conflicts in the merge

6. Commit it by running

```
svn ci -F svnmerge-commit-message.txt
```

7. If you have a JIRA associated with this fix, make sure it says fix for 2.8.x.

Trouble with svnmerge.py

If you have trouble with the svnmerge.py file such as Claus Ibsen had, then he attached an older svnmerge.py file, to this wiki page that works.

Using DoMerges tool

If you look in:

- http://svn.apache.org/repos/asf/cxf/trunk/bin/

there is a `DoMerges.java` file in there that you can compile and run from a fixes branch checkout.
It pretty much walks you through the entire process of backporting fixes.

It lists all the outstanding commits that haven't been reviewed, allows you to merge commits individually, block commits, show the diffs, etc.
For the most part, it's quite easy to walk through a bunch of commits and merge things back with it. Takes very little time.

To run the file do, from the directory with the branch.

```
java DoMerges
```

You need `svnmerge.py` to be runnable from the command line.

There is a compiled .class of the DoMerges attached to this wiki page you can download. However its easy to compile
the source file, as it has no other dependencies so its all plain

```
javac DoMerges.java
```

Using git

If you already use git-svn, you could consider using the great git merge capabilities.

Create a local branch from the remote tracking branch (e.g. camel-2.8.x)

```
git checkout -b camel-2.8.x remotes/camel-2.8.x
```

or switch into the existing branch

```
git checkout camel-2.8.x
```

To merge one revision (e.g. 1176050) into this branch, run

```
git cherry-pick 1176050
```

This will merge and commit the changes into your local git repository.

Combining multiple revisions

If you have to combine multiple revisions into one commit, run

```
git cherry-pick -n <revision>
```

for each revision (the -n options prevents the commit after the merge). Afterwards you have to run

```
git commit -a -m "<commit message>"
```

to commit all the changes with one commit into your local git repository.

Run

```
git svn dcommit
```

to push your local changes into the Apache SVN repository.

If you have a JIRA associated with this fix, make sure it says fix for 2.8.x.

Git Tooling

There is a number of Git Graphical Tools which can be used as well for backporting fixes. For example GitX or GitTower for Mac users.

Closing github PRs

The PRs at github is only automatic closed if the commit log has words like closes, fixes etc. And therefore we often ask the author of the PR to close the PR after it has been merged, or rejected. However the author may not see the notification or he/she does not react. So the Camel team can force close the PRs using an empty git message:

```
git commit --allow-empty -m "This closes #xxxx"
```

You can then include multiple PRs etc. This closes #123. This closes #456.