Feels like cheating - openSSL updates

Contributed by Michael Felt

But maybe pragmatic is a better word.

What did I do? - that feels like cheating!

I am copying archive members from one library to another library so that the run-time linker finds the archive member it is looking for. The run-time linker search for a member is not exhaustive.

FYI - this article sat as draft for over year. I will write part two today - which covers what I did - better learned about packaging - between July 2015 when I wrote what is below - and today, 23 Sept 2016

AIXTOOLS and packaging Open-SourceOne of the goals for AIX 5L was to be supportive of the Linux run-time environment - aka Linux affinitive or compatible. LINUX is, or was, not POSIX compliant. The Linux community adds something or modifies something that does not always co-exist with POSIX or even previous Linux definitions. I am told that co-existing holds back innovation. And while this may be true it does create difficulties for others that want to adopt an innovation but still need a previous environment or API (or should I say ABI?).

AIXTOOLS: Goal #1

Besides a purpose - "Make Opensource packages available in installp format" one of my goals for packaging - as AIXTOOLS "brand" - is to co-exist with existing packages -
e.g., do not overwrite existing files. I have suffered packaging that does not attempt to achieve this "simple" goal. A common example comes from RPM packaging that overwrites a file that is also in an installp package. One of two things go wrong with some frequency.
A) Uninstalling only one of the two packages means the other is broken because the common file is removed while both packages depended on it.
B) The first (lets call it the original) package breaks because it is a different version of an API - especially when older, or because it has specific RTE (run-time environment) demands that are different (aka ABI differences).

"To co-exist" or "Not to co-exist"Packaging software, especially as a "guest", i.e. not the owner of the specification for the run-time environment, is a complex process. Default locations such as /usr/lib are occupied with libraries that many programs depend on - and how to make sure that libraries with the same name are used rather than the libraries in a legacy location.

A start of a 'solution' that IBM and/or BULL choose back during the AIX 5.0 pre-release testing (put the libraries in /opt/freeware/lib and a symbolic link in /usr/lib) still reserves a location in /usr/lib because the location (the) rtl (run-time linker) searches for the library is /usr/lib. Remove or replace the symbolic link is the same as removing the library in /opt/freeware/lib.

API and ABI - why should I care?

There are many ways to start this discussion. I hope I am choosing one that is easy to understand. OpenSSL is an API. LibreSSL is a new implementation of the OpenSSL API. LibreSSL is not binary compatible with OpenSSL. (Compare this with MariaDB that means to be a binary replacement for MySQL).

OpenSSL (wiki) is an open-source implementation of the SSL and TLS protocols. This project is available - in source form from www.openssl.org. Further, this implementaion is so popular that it can be seen as an API to SSL and TLS protocols. And then we can speak of different versions of OpenSSL (e.g., OpenSSL-0.9.8, OpenSSL-1.0.1).

A library, in binary form, represents a version of the API. On AIX different binary versions "co-exist" as different members in an archive. For example, the archive /usr/lib/libssl.a has members libssl.so.0.9.8 and libssl.so.1.0.0. Other *NIX versions do not dynamically link to a library member - instead they link to a shared object (.so) file - e.g. /usr/lib/libssl.so.0.9.8 or /usr/lib/libssl.so.1.0.0

Ideally, when a new binary version differs in a way that programs that dynamically linked to a previous version run functionally differently - the program(s) do(es) not run. The new binary version can be considered to not be ABI compatible with the previous version - even if (rather especially) when the program is still - source-code compatible to the API. (See http://stackoverflow.com/questions/3784389/difference-between-api-and-abi for one description/discussion of the difference between API/ABI). Having an archive (.a file) with multiple members, or multiple .so files in a certain directory makes it possible for multiple binary (aka ABI) versions to co-exist.