Monday 15 April 2019
Jamie.Shiers@cern.ch

Introduction

This document started off as a summary of software preservation and legacy issues at LEP and was subsequently expanded to address the possibility / appropriateness of applying FAIR[1] principles to software. It should be understood that memories of this era are now as dusty as the card decks that preceded it. (But not us?)

PART A – S/W PRESERVATION AND LEGACY ISSUES AT LEP

Foreword

An almost unique case of "data rescue" in High Energy Physics concerns that of the JADE experiment at DESY that took data from 1979 to 1986. This rescue operation involved not only the data itself but also the software, documentation and necessary "constants[2]" and other information required to be able to re-use and re-analyse the data. (More on this "Dan Brown"-like rescue operation is given below).

If there is one lesson in this story it is the need to take a "holistic approach" – data without the software is often useless, as is software without build and verification systems and / or necessary additional data (alignment, calibration, magnetic field maps etc.) These are typically stored separately and involve distinct services that evolve on independent timescales and with lifetimes typically much shorter than the period for which the corresponding "data" needs to be preserved.

This may be summarized in the saying *"there is no such thing as preserved data, only data that is being preserved[3]".*

Introduction

This document was prepared as input to the 2019 Workshop on Sustainable Software Sustainability and covers the source code management, build, test and distribution environment and policies for the CERN Program Library (CERNLIB). This software was (is) used by High Energy Physics (HEP) experiments

[1] See https://www.force11.org/group/fairgroup/fairprinciples.
[2] At the time of LEP, these were indeed referred to as "constants" although they were typically time-dependent. Later, they began to be referred to as "conditions".
[3] From "The Theory and Craft of Digital Preservation" by Trevor Owens, submitted to JHU Press.

worldwide and is a pre-requisite for using the data from the experiments at CERN's Large Electron Positron Collider (LEP).

Significant background material can be found at
https://indico.cern.ch/event/801649/ – until someone decides to delete it!

Construction of LEP started in 1985 with first beams on July 14th 1989. Data taking continued until the end of 2000 when it was shutdown and subsequently dismantled to make way for the Large Hadron Collider (LHC), housed in the same 27km circular tunnel.

When work on LEP started, computing was very much in the batch and mainframe era. During its lifetime it saw the transition to interactive computing, minis, micros, clusters, farms and even grid computing. Adapting to these very different environments was made possible by the considerable foresight and indeed genius of those who laid the foundations of CERNLIB[4].

Regrettably and perhaps shamefully much, if not all, of the software that preceded the LEP era has been lost – sometimes due to inaction and at others due to policy (described in more detail later).

Throughout much of LEP data taking, there was no such thing as "managed storage". Users were responsible for managing their own tapes, including book-keeping[5]. Managed storage only appeared rather late in the day as preparations for LHC Computing ramped up. It is therefore notable that analysis of LEP data continues to this day with the software of 3 of the 4 experiments continuing to be usable with today's hardware, operating systems and compilers. To claim that this was planned from the beginning would be more than an exaggeration. However, the move to such storage, together with fortuitous technological advances (massive increases in storage density that arrived "just in time" for LEP), have contributed to the fact that LEP data is still "alive" today.

Some examples of "data resurrection" exist that suggest that data and the necessary software can be "recovered", given sufficient motivation, residual knowledge and also a good helping of luck.

Whilst at the end of LEP it was believed that humankind would never build such a machine again, there are now detailed studies of a possible follow-on machine to be discussed *inter alia* at the Open Workshop in May 2019 on the update of the European Strategy for Particle Physics[6], although it may still before many years / decades before one sees the light of day.

---

[4] Some of this work never made it into CERNLIB *per se* but many of the ideas and much of the experience fortunately did.
[5] The author was responsible for a package that allowed data to be referenced by a Posix-style file name, rather than tape volume and file sequence number. This was adopted by 3 of the 4 LEP experiments.
[6] See https://cds.cern.ch/record/2653669/files/CERN-ACC-2019-0003.pdf.

Monday 15 April 2019
Jamie.Shiers@cern.ch

CERNLIB OVERVIEW (FROM "SHORT WRITEUPS")

The CERN Program Library is a large collection of general-purpose programs maintained and offered in both source and object code form on the CERN central computers. Most of these programs were developed at CERN and are therefore oriented towards the needs of a physics research laboratory. Nearly all, however, are of a general mathematical or data-handling nature, applicable to a wide range of problems.

The library is heavily used at CERN and it is distributed in binary or source form to several hundred laboratories and computer centres outside CERN.

The library contains about 2500 subroutines and complete programs which are grouped together by logical affiliation into little over 450 program packages. 80% of the programs are written in FORTRAN77 and the remainder in C and in assembly code, usually with a FORTRAN version also available.

A unique code is assigned to each package. This code consists of one letter and three or four digits, the letter indicating the category within our classification scheme. A package consists of one or more related subprograms with one package name and one or more user-callable entry names, all described briefly in a "Short write-up", and if necessary, an additional "Long write-up".

CERNLIB SOURCE CODE MANAGEMENT

One of the striking factors of the CERNLIB source was the adoption of an in-house "source code management system". Quotes are used as it was (is) quite different to what is conventionally referred to as an SCM today but nonetheless this is what it was. One aspect of this SCM (PATCHY[7]) was similar functionality to that offered by the C pre-processor and to a certain (large?) extent this subset of the full PATCHY functionality is what was used for the CERNLIB source. In other words, the source code contained platform specific code that was selected at pre-compile time with a single source file containing the code for all of the many supported platforms. Another aspect that was heavily used to manage the code of the various physics experiments was closer to that of a batch editor. Starting from a reference version, additions or corrections could be applied until such time as a new reference version was prepared[8]. Program sources were maintained in a compact binary format that was platform dependent, although a platform-independent version (called "ceta" format) also existed and programs

[7] Other similar systems existed, such as Control Data Corporation's "Update" program and the commercial "Historian" offering.

[8] A tool existed in the PATCHY suite to automate the production of a new reference version. However, the author witnessed and even participated in a "manual" update, where the previous version was punched onto cards that were then interpreted (that is, the meaning of the holes printed on them in text) and all edits applied by inserting new cards into the appropriate places (and/or deleting old ones). You couldn't make it up…

were typically distributed in this format. The move to text format (ASCII or EBCDIC depending on the platform, although happily translated by *ftp*) occurred only in the early 1990s.

CERNLIB BUILD PROCEDURES

Three versions of the CERNLIB short write-ups are known to exist (there may well be many others on dusty shelves around the world:

1. A printed version from May 1989;
2. A postscript (whence PDF) version from 1996;
3. A new(-er) PDF version produced from the same LaTeX source circa 2016.

Whilst every effort was made to capture additional meta-data and "collective knowledge" in the most recent version, some important background information was not carried over from the print version, including instructions on how to obtain and / or build the libraries, their internal hierarchy and even how to find related software managed by another CERN department. Collating and interpreting these different versions is hard even for those who lived through this period and is likely to be enigmatic if not highly error prone for anyone that did not. With hindsight, it is easy to suggest that we should have been more careful to document these things "for posterity" but at the time, there was simply no need. There was a fairly large team actively supporting CERNLIB and someone was sure to know the answer to any question and if not, one only had to "ask Harry" – a former program librarian. Preserving knowledge beyond the team that created it and maintained it is particularly challenging.

DATA PRESERVATION IN HEP

The 3 pillars of long-term data preservation (for re-use, re-analysis and sharing) in HEP are considered to be:

1. The data itself (aka "bit preservation" at the multi-hundred PB level);
2. The associated documentation (aka "knowledge" – i.e. much more than a few PDF files);
3. The software including the environment in which it runs.

An overview of the services that CERN currently offers can be found in our iPRES 2016 paper[9].

A recent HEP "Community White Paper[10]" concluded that *"bit preservation with an acceptably low bit-error rate can be considered a solved problems"* (using HEP-specific repositories).

[9] See https://cds.cern.ch/record/2195937/files/iPRES2016-CERN_July3.pdf.

Whilst this document focuses primarily on (the) software, the associated environment can also be a complex issue. Some examples are given below of data that has been lost and in some cases "found".

(ALMOST COMPLETELY) LOST

Prior to the LEP era, a significant number of experiments recorded at least part of their data on film (this is still true in some rare cases, such as certain "neutrino oscillation" experiments). *Some* of this film still exists[11] but the machines to measure it do not (nor are the "operators" of these machines still available. Also missing (if the above was not enough) is the software and some critical data such as the positions and angles of the cameras (used to record particle tracks on the film), the distortion matrices for these cameras and even the map of the magnetic field used to bend charged particles and hence allow their momentum to be determined. Some residual information remains, for example the magnetic field of the Big European Bubble Chamber was rated at 3.5 T[12], and substituting the field map with a constant value could *possibly* allow the data to be re-processed in some simple ways (e.g. one could trace the approximate trajectories of the particles through the detector).

Although the devices originally used to "measure" the films have most likely long since been dismantled, the film itself could be used for some basic "analysis". That is, by projecting it one could count the number of particle tracks – positive and negative – tentatively identify particles such as electrons (tight spirals) and protons (relatively straight but "heavily ionizing" and hence "thick and dark" with respect to the other tracks. Neutral particles decaying to positive and negative pairs could also be identified, as could other decays and secondary interactions. Indeed, such "analysis" was typically performed before the films were measured in detail (uninteresting frames were not measured) and was referred to as "scanning". It would also be possible to digitize any remaining film which would eventually allow it to be preserved along with all "born digital" data that CERN preserves today.

It has been suggested that an audit is made of any data – including film data – from CERN experiments, possibly using the DPC's "Endangered Species[13] criteria, together with the DPHEP preservation levels to indicate what data is or might be still available (it is not clear whether DPHEP level 4 includes film data):

Preservation Model	Use case

[10] See https://hepsoftwarefoundation.org/organization/cwp.html and https://arxiv.org/abs/1810.01191 in particular.
[11] There is even an incomplete catalogue of 28 such films at CERN dating back to the early 1960s, based on "reminiscences" and decrying the lack of documentation at the time of storage.
[12] See https://cds.cern.ch/record/850617/files/Tech-Note-M12.pdf.
[13] See https://www.dpconline.org/our-work/bit-list.

1. Provide additional documentation	Publication-related information search
2. Preserve the data in a simplified format	Outreach, simple training analyses
3. Preserve the analysis level software and data format	Full scientific analysis based on existing reconstruction
4. Preserve the reconstruction and simulation software and basic level data	Full potential of the experimental data

More details can be found in the DPHEP Blueprint of 2012: https://arxiv.org/pdf/1512.02019.

LOST AND FOUND – THE JADE EXPERIMENT AT DESY

Perhaps the most famous example of "data rescue[14]" in HEP is that of the JADE experiment at DESY. Strong scientific reasons motivated the "recovery" of the data and the resurrection of the software. The exercise included:

- Recovery of the data from the original archive tapes and copying them to more modern media;
    - *"Peter Bock, when cleaning out an old lab at the Physics Institute at Heidelberg University, found a few 9-track tapes containing original JADE Monte Carlo files which were very valuable for validating results of our first re-analyses in ~1997";*
- The data itself (binary "BOS" format) consisted of a mixture of 16 and 32 bit integers, together with floating point numbers in "big endian" byte order – about 1TB in total;
    - *"An old version of the original BOSlib 1979 version was found, on our request, at the Tokyo computer centre";*
- The software – originally written in obsolete dialects, including Mortran, Sheltran and FORTRAN but with (now) non-standard features, such as ENTRY statements and alternate returns, plus some assembler, was resurrected over almost a decade starting from 1995;
- But software and data was not enough – additional met-data was required, including calibrations, alignments etc.;
    - *"Jan Olsson, when cleaning up his office in ~1997, found an old ASCII-printout of the JADE luminosity file.*
    - *Unfortunately, it was printed on green recycling paper - not suitable for scanning and OCR-ing.*
    - *A secretary at Aachen re-typed it within 4 weeks. A checksum routine found (and recovered) only 4 typos";*

[14] Described in https://arxiv.org/pdf/1009.3763.pdf and elsewhere.

Monday 15 April 2019
Jamie.Shiers@cern.ch

FROM BATCH JOBS TO PSEUDO-MAKE TO MAKE AND CVS

From today's perspective, it is hard to imagine an era with little or no interactive computing. However, the first central interactive services at CERN were only installed in the run-up to LEP. This saw a change in the installation methodology from batch to "make-like" scripts. At the same time, advances in code management outside of HEP began to have an impact with proposals that we move away from HEP-specific tools to open sources / community-supported ones. It is not to say that this was without controversy, particularly as the long-term support for the FORTRAN-based libraries was already in question. However, the source code was converted from using PATCHY-style mark-up to C pre-processor directives, with the master repository being managed by CVS. This, in principle, allowed much more detailed tracking of code changes and bug fixes, as the previous policy had been to deliver new versions in their entirety at the time of a release (changes and fixes were documented in comment files but this in no way substituted for the multiple versions with time-stamping and/or tagging offered by systems such as CVS.

A non-official distribution of CERNLIB makes the following observations:

"Patchy … is an ancient set of programs that acted as a combination preprocessor and revision control system. They apparently developed more or less independently of diff, patch, rcs, and related ***Unix utilities****".*

Well, yes, they were developed independently but not only pre-date Unix but also were intended for a highly heterogeneous environment (different operating systems, word-length, byte order etc). Unix was a latecomer to the game, even if, in its Linux guise, it is now quasi-ubiquitous for batch (and interactive) processing.

Whereas, for LEP-era software, CVS (or in some cases still PATCHY) was the "end of the road", things have moved on since, e.g. for the LHC experiments (first to SVN and then later to git/github). Whilst these bring advantages, notably in the "findability" of the software, each migration has involved some "knowledge loss". It is not clear that service migration is commonly accepted as a threat to "long" term data preservation (where "long" can be quite short – just a few years). If you do not accept or acknowledge that you have a problem, you are somewhat unlikely to solve it!

SOURCE CODE "RETENTION" POLICY

Releases of CERNLIB were made periodically, the significant changes documented in the CERN Computer Newsletters (CNL). For a long time the releases were therefore identified by the corresponding newsletter, e.g. CNL 197. With perfect 20-20 retro-vision, it would have made sense to have archived (at least) the complete tree corresponding to every release[15]. This did not happen however. Thus, the history of modifications – other than what was recorded in the CNL articles and / or comments in the code – has been lost. Moreover "obsolete" routines and / or packages were subject to termination with extreme prejudice. The CNL in question cites 4 occurrences of:

Obsolete package deleted from source and binary.

On Unix systems, at least 3 versions were available: *old, pro and new*. These were symbolic links to e.g. 93a. On the central Linux systems at CERN today, only *pro* exists in the official /cern tree and this points to the release 2006a. The sources are available, both in pre-CVS PATCHY card-image format, FORTRAN files (with the .F suffix – meaning with C pre-processor directives). A few other versions and builds can be found elsewhere on the Internet but there was no systematic archiving. Re-processing old data with the version of the time (even if the operating system / hardware platform / compiler version combination could be found) was hence ruled out[16].

```
*
* $Id: fmexst.F,v 1.1.1.1 1996/03/07 15:18:01 mclareni Exp $
*
* $Log: fmexst.F,v $
* Revision 1.1.1.1  1996/03/07 15:18:01  mclareni
* Fatmen
*
*
#include "fatmen/pilot.h"
      SUBROUTINE FMEXST(GENEN,IRC)
*
*     Routine to check whether specified generic name
*     already exists on RZ file. Does not attempt to
*     read in data structure - just uses keys
*     N.B. the maximum number of copies of a file is limited
to MAXCOP
*
      CHARACTER*(*) GENEN
#include "fatmen/fmaxcop.inc"
      PARAMETER (LKEYFA=10)
      DIMENSION KEYSIN(LKEYFA),KEYSOU(LKEYFA,MAXCOP)
#include "fatmen/fatpara.inc"
      IRC    = 0
      NFOUND = 0
```

(etc.)

[15] In the early 1990s, the connection with the CNL was broken and releases referred to as e.g. 93a.

[16] These somewhat harsh limitations are addressed for LHC and current era experiments but this is beyond the scope of this paper.

Monday 15 April 2019
Jamie.Shiers@cern.ch

SUMMARY

The source code management of the CERN Program Library – particularly in terms of its inherent multi-platform support – was extremely advanced for its time. Unfortunately, we were far less disciplined when it came to preserving (sometimes legacy) software for posterity.

Is it too late to recover some of the older versions of the software? Does anyone care enough to do it? It is the author's opinion that this software is as much a part of CERN's legacy / memory as those things that are currently being preserved in this context (papers, minutes of meetings, photos and videos) and arguably more representative of at least part of its intellectual output.

PART B – LESSONS LEARNT AND FAIR FOR S/W

This section discusses how one might interpret past practices in the context of "FAIR for software", as defined in the report from the previous WOSSS. It also covers today's "best practices" in HEP that have been developed independently from the FAIR principles but owe a lot to experience from the past.

It is important to recollect that the software needed by a specific project will typically come from a variety of sources. In the case of a "typical" LEP experiment, this might have included:

- The CERN Program Library – CERNLIB;
- Vendor or 3rd party TCP/IP or other networking libraries;
- Vendor or 3rd party graphics libraries, e.g. those conforming to the PHIGS standard;
- Run-time libraries provided with the operating system;
- Experiment-specific code coupled with that from an "analysis group" and often a specific user.

A specific sub-program

As described above, each CERNLIB routine or complete package was given a code consisting of a letter and a 3-digit number. Coupled with the corresponding release, this identified a routine or package fairly uniquely, e.g. C312 (Bessel Functions J and Y of Orders Zero and One) and release 93a. The documentation lists when the subprogram was submitted (18/10/1967) as well as the most recent revision (15/03/1993). In this particular case, the source code (still available) contains no record of any modifications (nor indeed any comments at all), nor does the corresponding CNL list any. As an archive of each released version of CERNLIB does not exist, it is not possible to compare versions to identify the changes. Whereas the documentation is reasonably descriptive (to an expert) – even including a reference to a published paper (but before DOIs) –

the code itself is somewhat (i.e. very) opaque[17]. This example was to illustrate a case where the documentation is essential – whereas no-one books an EasyJet flight having first read their documentation, no-one could realistic use these routines without the manual (nor indeed a strong background in maths).

What does this example tell us?

- It is not obvious that completing the FAIR vs DULF (developer; user; librarian; funder) matrix makes sense at this level of granularity;
- Access to the documentation – and perhaps more importantly expert support and guidance – is quasi-essential for the potential user;
- In this respect the pre-WWW version of the CERNLIB manual was arguably better[18] as it included not only direct access to individual sub-programs but also a category-based index ("Equations and special functions" in this case);
- Access to the source code was relatively straight-forward (example extraction job in the PATCHY-era; direct access to the .F files now) but arguably of little use other than:
    - To potentially port to a different platform / language;
    - To investigate problems;
- In this latter respect it would be nice to know what modifications were made and why.

(There are numerous sub-programs in CERNLIB that were essentially run-time library level, e.g. zeroing or copying an array, where only minimal documentation, such as the calling sequence, was needed).

Another example of individual sub-programs is the set of random number generators. These are extensively used in Monte Carlo simulation programs (see later). From time to time, new algorithms were deployed, e.g. when older ones turned out to be insufficiently "random" and/or to have too short sequences. This caused significant grief when backward incompatible changes were made:

- Users had to adapt their programs to the new routines / calling sequences (or accept results that were considered at least sub-optimal);
- Once adapted, these programs could no longer be run against older versions of the libraries.

Such backward incompatible changes may be relatively rare but nevertheless do occur, throughout the stack, probably no less frequently than every 5 years.

[17] Source code has sometimes been described as "bad documentation". In this particular case, it is so opaque that I doubt that even an expert could figure out what it was intended to do!
[18] The write-ups are now stored in the CERN Document Server but CERNLIB support has long-since stopped. Can you support something that you don't understand? (People try).

As a minimum, such changes needed to be clearly motivated and documented, although the information typically gets buried over time.

A Complete Package

Moving up in complexity, we next consider a "complete" package. As is typically the case in CERNLIB, these packages built on sub-packages and individual routines and could only work in an environment where (essentially) the complete CERNLIB was installed. Furthermore, they often were steered by environment variables, configuration files and / or databases and were – at least during the early years of LEP – under continuous development. This meant that it was often very hard (read "impossible") to know precisely which version was being used, despite efforts to document changes in the source itself as well as in release notes. This caused considerable frustration with users reporting bugs that had been fixed "the night before" and / or documentation that was out of sync with the code.

This almost certainly represents a reality – at least in this environment it is not realistic to believe that even simple packages can be completely developed and debugged prior to the start of a multi-year / decade experimental program – and that is before one includes experiment-specific software, the influence of the computing platform (operating systems, compilers, grid / cloud / etc.).

Rather than continue to give (ever more complex) examples, it is probably time to admit that the past approach – however well intentioned and however constrained by the highly heterogeneous environment and rapid rate of change – could not have been massaged to fit with FAIR(-like) principles.

A very different approach has been adopted by the LHC (and many other) experiments in HEP and beyond. The following is a direct quote from an iPRES 2016 paper:

"The HEP community has a long tradition of sharing and developing common, open-source software stacks within international collaborations. The software systems required to operate on LHC data comprise physics simulation and reconstruction algorithms to determine physics processes from detector signals, data analysis frameworks to extract (new) scientific knowledge from data sets, and distributed systems for data access and compute job management. Altogether, HEP software stacks add up to tens of millions of lines of code, half a dozen different languages and tens to hundreds of modules with dependencies on each other. Several millions of lines of code are specific to an experiment and there are numerous dependencies on standard software, most notably the GNU/Linux operating system and language compilers and interpreters. The support life cycle of the 3rd party software components is much shorter than the envisaged preservation period of several decades. Operating system versions are supported for a maximum of 5-10 years, for instance, and most developers abandon their software releases much earlier.

Running experiments invest many person months in the porting and the validation of their software stacks, coordinated by a dedicated software librarian. For a decommissioned experiment, that is one that is no longer in its data-taking phase, such an amount of effort would be impractical. Hardware virtualization (such as KVM, Xen and VirtualBox) and container virtualization (such as Docker) provide a potential solution. Virtualization allows for execution of a frozen, historic software environment on contemporary hardware and operating systems. In a straightforward application of virtualization technology, a software environment is frozen in the form of a disk image, a large and opaque stream of bytes containing all the necessary software binaries. This approach tends to be clumsy and too rigid for HEP software. In order to be useful, even "frozen" environments need to stay open for minor modifications: patches to faulty algorithms, updates to physics models, updated tuning parameters for simulation algorithms, new configuration for data access software and so on. Software development communities have long solved similar problems by version control systems. Version control systems only store a track of changes to the source code and at the same time they can provide access to the state of a directory tree at any given point in the past.

In an attempt to get similar benefits for compiled and configured software stacks, since several years HEP experiments install all released software components in its final configuration on the versioning, open source, and distributed file system CernVM-FS. By selecting different versions in the history provided by the file system, experiments can access any software state ever released. Thus we can separate virtualization – in this case handled by CernVM – itself from the concerns of accessing software binaries. A minimal and stable virtual machine or container (~20MB) connects to the remote file system CernVM-FS that hosts the operating system and software binaries. By selecting different states of the versioned file system, experiments can go back and forth in time and create software environments compatible with Red Hat Enterprise Linux (RHEL) 4 to RHEL 7 (spanning 15+ years) with the very same virtual machine on the very same hardware.

Concretely, we have demonstrated that by resurrecting the software of the ALEPH experiment at LEP more than 15 years after the experiment was decommissioned. Contemporary virtual machines provide data access tools and middleware with support for the latest network protocols and security settings. Containers inside the virtual machines spawn historic operating system and application software environments. Data is provided from the container host to the historic applications through the very stable POSIX file system interface.

Among the currently active HEP experiments, many operate dedicated CernVM-FS services to satisfy their day-to-day needs for global software distribution. These services are operated in an "append-only" mode, so that software versions, once released to the production infrastructure, remain readily available for future use. Due to the file system's internal data de-duplication, this model proved to be sustainable even for the largest users. After more than five years of experience with LHC experiment software and more than hundred million registered files (software only!), the storage volume is still at only a few terabytes."

Monday 15 April 2019
Jamie.Shiers@cern.ch

Complex Software Environments and FAIR

Whilst the FAIR acronym has gained a lot of traction, it is not clear that it is directly applicable – or even useful (in the sense of representing the most important criteria) – for complex software stacks. Some important criteria include:

- Methodology – do the algorithms involved conform to agreed standards and / or relevant published papers? This applies not only to "simple" algorithms, such as the Bessel functions cited above, but also to more complex systems (e.g. access protocols) that may need to be supported to different subsystems. All to often the code is the documentation…
- Trustworthiness – under what conditions has the software been tested and how? (Some mathematical libraries guarantee that every line has been executed as part of their tests but not necessarily every code path).
- Environment – what environment(s) does the software run in?
- Dependencies – what additional "information" is needed to make the software useful? (See examples above).
- Etc.

Perhaps it is useful to consider three distinct cases:

1. Publications are the entry point: from here one needs to be able to navigate to the data not only "in" the publication (tables, graphs etc.) but also "behind" the paper – together with the software / algorithms used to process the data and produced the published results (reproducibility here can require significant resources / expertise);
2. Data are the entry point, e.g. "open data" releases. In this case the software and documentation required to perform defined actions is required, together with sufficient details of the data format for other actions to be performed;
3. Software itself is the entry point, as is / was the case with the CERN Program Library. This would also be valid for libraries such as LAPACK or the NAG mathematical libraries and other "generic" software.

Notwithstanding the LEP / CERNLIB experience, one may also comment as below, regarding current (LHC and other) software practices:

Findable	"Modern" tools such as github and Zenodo (may) play a role here (but did not exist in the LEP era)
Accessible	Open Source – if not an absolute requirement – certainly helps here
Interoperable / Reusable	Covers a wide range of issues, include licensing, programming language(s) and interoperability, software architecture (well defined and "open" APIs versus monolithic approaches), dependencies on other packages, compilers and operating systems, porting, build, integration and test issues.

Tentative Conclusions

In conclusion, this probably boils down to just two cases:

1. Specialised software, such as that written for the LEP, LHC and other HEP experiments. FAIR for software is arguably not (particularly?) useful here – the main entry point being either data or associated publications;
2. Generic software and tools, where FAIR for software would appear to be more appropriate.

There is no doubt that today's tools and practices offer advantages over those of previous eras but we should not fall into the trap of assuming that "everything is solved"[19].

These are marked as tentative, pending discussions at the workshop. Based on the outcome, this document will be updated and stored in in TDR, with a DOI, for a lot less than eternity.

As a final word, one could perhaps add that the holistic approach needs to be in both time and space. Time – addressing service changes and the necessary migration, adaption, re-validation and space – addressing software and its complete eco-system, including the data for which it is intended / associated.

[19] It was once claimed that, with the move (from mainframe, then client-server) to "Internet computing" that *"nothing would change for one thousand years".* But it did.