LHC Grid Computing Project

REQUIREMENTS ON SOFTWARE INSTALLATION

Document identifier:	**LCG-GAG-SOFINST**
Date:	**23-Nov-04**
Authors:	**D.Adams (BNL) D.Barberis (CERN/Genoa), L.Bauerdick (FNAL), I.Bird (CERN), N.Brook (Bristol), S.Burke (RAL), P.Buncic (CERN), F.Carminati (CERN), P.Cerello (INFN), F.Donno (CERN/INFN), D.Foster (CERN), F.Harris (CERN/Oxford), S.Lacaprara (INFN), L.Perini (INFN), A.Pfeiffer (CERN), R.Pordes (FNAL), A.Sciabà (CERN/INFN), O.Smirnova (CERN/Lund), J.Templon (NIKHEF), A.Tsaregorodtsev (IN2P3)**
Editors:	**F.Carminati (CERN), J.Templon (NIKHEF)**
Document status:	**Version v0.8 – Draft**

Abstract: This document contains common requirements for software installation.

Table of contents

1 Introduction

Experiment software installation in a Grid environment is more complicated than on a local system, due to several reasons:

1. Large amount of systems at which the software has to be promptly maintained (installed, upgraded, removed),
2. Different ownerships and hence configurations and policies at the systems,
3. Remoteness of most systems,
4. Hardware and software heterogeneity of the systems.

This note contains the common view from the four experiments on the requirements on software installation. Although the experiments have already provided these requirements, this note is an updated version of them and supersedes previous common statements. The experiment experience during the 2004 Data Challenges have provided an important input to these requirements.

2 General statement of the problem

The common situation regarding the software packages used by the LHC experiments is as follows:

- Each experiment develops a set of specific software packages, necessary to process the data;
- The experiment-specific software develops rapidly, thus being released frequently, in many versions;
- It is often necessary to keep several versions of the same software installed at the same site – contrary to the widely known concept of "software upgrade";

- Installation of each software package needs to be validated in the way it suits the respective experiment;
- Users may require installation of a personally tailored software package, which is especially true for the distributed analysis situation.

The general high-level requirement for software installation can be formulated in the following way:

The software installation system should ensure that an experiment is not hindered using an otherwise available resource just because the requested version of published software is not present there.

While this requirement captures the very essence of what we want, it may turn out to be too general to be really useful. It is therefore worthwhile to go into some details of how this objective can be achieved.

The jobs of the experiments need some software proper to the experiment (hereon ***software***) to be present on the computing system where they run. This consists of programs in various forms (source, object files, archive or shared libraries, executables etc.), usually small configuration files and other ancillary files. This software is subdivided in one or more packages. Packages can depend on each other and on external software, such as common software for the four experiments (e.g. ROOT, POOL, GEANT4, FLUKA) or system components. Each package is released in subsequent versions, identified by a tag.

No assumption is made on the structure of these tags, which are typically alphanumeric strings, even if it is quite possible that they are standardised in some way. Usually at least part of the tag is numeric so that dependencies between packages can be expressed by arithmetic inequalities[1].

A very widespread, even if not universal, convention is to indicate tags as <major version>.<minor version>-<revision>. This is, in some sense, an implementation issue. However, provided there are agreed conventions and proper discipline in incrementing major, minor, and patch numbers, this syntax can embed quite useful information about the behaviour and compatibility of the software between old versions and new versions, depending on which element is increased. In this case, we could express requirements such as a user being able to say "I want to use the most recent patch level of v15.12 release of package SolveIt." Of course it must be also be possible to query the exact version number that used for provenance. We feel that some more discussion is necessary between experiments to express some common position on the tag syntax and semantics.

Tags can identify software other than experiment-specific software as well. This is necessary for defining dependencies on support subsystems such as

[1] We note that in the current JDL it is not possible to do such inequality arithmetic, so this is an implicit requirement for future JDL development. We also note that sometime the version is expressed by a combination of numbers and other characters and it is hence impossible to establish inequalities.

scripting languages or system libraries. For experiment-specific software, these tags will refer to actual physical packages published by the relevant VO; for system-type software (for example the system-provided version of g++) the tags are more likely to refer to capabilities or the measured state of the system, since VOs will not be able to cause upgrades of this software using their own packages.[2] We realize that there are unresolved issues here, for example how does such a capability tag get removed or revoked (or does it even?) if a system is upgraded?

In the following the generic term software is synonymous of one or more packages, unless otherwise specified.

In the following we will mention several times the concept of "***validation procedure***". A validation procedure is an executable (script or program) or a set of such, that is run after that the software is installed and makes use of the fresh software installation as extensively as necessary, producing an assessment of whether the expected software behaviour is achieved or not. If the assessment is positive, the installation is considered validated, and therefore successful. If the result is negative, then installation is considered as failed and problem tracking begins. The validation procedure is entirely defined by the experiment, and we make no assumption on its quality or thoroughness in testing the software installed. In general the validation procedure is expected not to take too many resources, but there may be cases where this assumption is violated.

This document specifically deals with software that has a certain level of commonality between users belonging to a given group, such as an experiment or a subset of it. In this sense we do not specifically address software that is needed by a single user in her work. It is clear however that such support will be needed in the future, and hence it is reasonable to keep this application in mind when designing the system being discussed here.

3 Procedures

In each collaboration there exists a role, which can be assumed by one or more persons, that of the ***Experiment Software Manager*** (ESM). Subgroups within a VO may have their own ESM, with privileges limited to the software that is common to the members of the subgroup. On the other hand, there are many packages that will be used by multiple VO's and there should also be mechanisms to publish tags at a level above the VO. This reduces the burden on the ESM and enables sites to have a multi-VO cache of common software packages. In the following we present typical scenarios of software management by the ESM, but these can be easily translated to a different level of generality.

Software Installation

[2] Note that an experiment could always decide to declare g++ as "experiment software" and install their private version as a VO-dependent package.

An experiment has developed a new version of a package and wants to make it available for installation on the computing resources of the Grid so that it is available to the users.

1. ESM assigns[3] a tag to (tags) a new version of the software Grid-wise. This includes declaration of:
    a. All the files necessary for the collaboration jobs to use this software,
    b. The tag under which this software will be recognised,
    c. The possible dependencies of this software on other packages, expressed via the corresponding tags and
    d. A validation procedure[4].
2. ESM publishes this information so that it is known by the Grid;
3. The appropriate Grid service makes sure that the software is present and validated on each site where the Workload Management System (WMS) sends a job to be executed.

Software revocation

The general scenario for this use case is the following. An experiment has determined that a given version of the software produces wrong results or is badly outdated, and therefore it has to be removed so that no new results are produced with it. The use case can then be:

1. ESM publishes a revocation of a given tag;
2. The appropriate grid service takes whatever action is appropriate following this notification. In practice this means that the corresponding tag can be removed from the list of available tags and that the space possibly occupied by the corresponding software on all the centres can be reclaimed.
3. Jobs requesting the deleted tag will fail due to inability to find the requested software.

This scenario raises a host of question about the handling of already running jobs using the tag being removed or, even worse, dependency chains of jobs supposed to be all run with this tag. Figuring out when a piece of software is "clear" to be removed will be difficult. At this moment we regard this as an implementation issue, even if we are fully aware that it may have long-ranging implications.

Other scenarios can be imagined, however these are almost all obtainable by the previous two, even if this may not be, by far the best way to do that.

[3] In fact often the ESM will received the code with the appropriate tag from the developers.
[4] In this context we assume that the validation procedure is mandatory, however, as said above, we make no assumption on its content or thoroughness.

Tag Failure

The ability to install the software on-demand by a job is essential, even if not used by all scenarios described below. Failures will certainly occur. We believe a mechanism is needed to indicate these failures to prevent yet another black-hole mechanism from occurring. One can imagine a jobs repeatedly being sent to some site at which the installation repeatedly fails, resulting in the resources once again being presented to the WMS as available to run the job. This can be prevented if such an installation failure is recorded (along with the offending tag(s)) in the information system, allowing jobs to match against "all systems for which the installation has not failed". Naturally, there should be a mechanism to bring the site back from such a blacklist.

Publishing the tag for a site

We referred above (and later in the document) to e.g. an SGM "publishing" that a tag is installed at a site. We make no specific recommendations about *where* this tag is published, as long as the information is available to all those people and services needing it. Currently the tags are published in the information system of a site's computing resources; other possibilities are that the VO itself operates a service or server (possibly visible in the information system) where this information is published; it could also be published by (or could be queried from) a package-management service.

4 Software implementation issues

In the following we will discuss various issues about software installation. We do not want to dictate implementation aspects, but rather to indicate what are current practices of experiments in this area.

4.1 Heterogeneity

There are two main aspects affecting the software installation procedures: heterogeneity of local configuration and of hardware and Operating System (OS). Here is how these are usually handled.

4.1.1 Local configuration

The software can be installed in different places on the different computing centres. Experiment software is usually "relocatable" in the sense that the top-level directory can change from system to system. There is a standard way to locate a software package. How this is done is an implementation issue. Examples are:

1. A procedure that accepts as argument a tag and returns the top directory where the software is located such as:

   findpkg <package> <tag>

2. An environment variable whose name contains package and tag name in a standard format, pointing to the top directory (something like VO_ALICE_ALIROOT_V3_4).

Note that these methods could point to the place where the software is installed, or where it could be installed on demand.

What can be more problematic is the location of different elements of the OS. For instance the location of a compiler may vary on different systems. As long as all the system-installed software is reachable via the standard command search path, set prior to the start of the job, the jobs can use standard command-discovery mechanisms (like unix 'which') to find the locations.

One point closely related is the user environment. Once the software is installed, the user job landing to a WN usually needs a running environment to be set in order to use it. As an example there are environment variables such as $PATH, $LD_LIBRARY_PATH and so on, or tools and scripts that can set these for the user. Usually the desired environment is obtained by executing a script referring to the experiment software. The above locator mechanism should be enough to determine the location of the script. This should be part of the standard prologue for all jobs requesting a given software.

One issue that has been debated at length is whether the installed software is on a shared area accessible to all Worker Nodes (WN) or it is installed on each WN. In reality this is not a concern for the users, as read-only files compose the software. Whatever mechanism ensuring that the software is located in the place found via the agreed mechanism is acceptable, irrespectively whether this is a shared directory or a local directory replicated on each WN or any solution in between.

One subtle point to consider in the case where the software is installed on each WN is the coverage of both installation and validation. Current practice effects installation and validation by submitting jobs to the relevant site. In order to reach all worker nodes, one must either flood the site with jobs (ensuring that all WNs are touched) or there must be some dedicated service running on the site that knows how to distribute the software to all worker nodes. Also, if validation fails on just one WN, should we consider the installation failed at the whole centre?

Another important point to consider with respect to WN-local installation is what happens if the system software of a node is reinstalled, e.g. following a maintenance intervention, or if new nodes are added. Shall we require that the installation and validation of all software be run? Whose responsibility is this? Should this be done preventively or only when a job requesting a given package is scheduled for this WN? We believe that most of these questions will be answered only when more practical experience will be acquired.

4.1.2 Hardware/OS

This is a more difficult problem to solve, and it depends on the installation strategy chosen by the experiments. Two extreme solutions, which have been implemented and seem to work, are:

- Prepare a completely self-contained tarball with binaries. Of course a different tarballs may have to be prepared for different combinations of hardware/OS. This works, however sometimes it may be necessary to include part of the system software such as compiler shared libraries.

- Send the code to be compiled. This of course may insure a better adaptation and optimisation to the local system, but it is much more dependent on system configuration. Just to mention one issue, making sure that the right version of all the compilers and interpreters needed is present, or making sure that the code works with all combinations available, may not be trivial.

For the moment the heterogeneity of the Grid is still limited, however it is probably bound to grow with time, so it will be important to be able to handle it transparently. It will of course help greatly if care is taken to make the code of the experiments as portable as is reasonably possible.

In any case the local hardware and OS should be transparent to the user when locating and using a package; transparent in this sense means that the user should not have to specify OS and architecture information when requesting a tag for a given package.

4.2 Installation on demand and pre-installation

It is generally accepted that there are two main strategies for software installation, on demand and pre-installation.

Installation on-demand

The WMS schedules a job for a given CE based on an optimisation procedure. If the requested tag is not installed, the software installation on the given node or site is performed before the job is run. The WMS might do this before sending the job, or the installation might be triggered when the job arrives at the site via some service. In order to avoid massively parallel installations, the installed software should be available to all users of that CE that have, in principle, the right to use this software. This may turn out to be quite complicated in practice, however, in the scope of this document, we will regard this aspect as an implementation issue.

In the simplest case a job can contains a prologue that checks the existence of a given software tag and then, if not present, install the software, either in user space or in a common area dedicated to this. If the installation is done in a common area, there are open questions to address such as the ability of a normal user to write in this common area, and her ability to register the package as present on the site, to prevent other jobs from reinstalling the same software in the same common area (and possibly overwriting open files).

A more elegant approach would be to have a “service” that can install software on demand and on behalf of a user or of the WMS. This service should be able to run the experiment specific validation procedure. In case this fails the installation should be declared failed.

Such a service should be able to determine whether an incoming installation request refers to a package that is not yet installed and validated, but is in the process of being installed due to an earlier request. For example the CE could maintain a queue of installation requests. Each new request would then be compared with the ones already in the queue to avoid two jobs requesting the same package from triggering two installations within the same area, with unpredictable results. We realise that here we are trespassing into implementation, but we want to give a concrete example of the functionality we need. Any other implementation providing similar functionality would of course be acceptable.

The requesting jobs should be withheld from execution till its required software is successfully installed. The service should intercept jobs upon their arrival to a CE, check their software requirements, fulfil them and only then let the job be scheduled for execution. Another option is to use the same mechanism as the data file prefetch used by the WMS, as is explained below in 4.5.

It is important to integrate job matchmaking and the on-demand installation, as it may turn out that a requested package is not available for the given system configuration. In such cases, jobs must not be scheduled to the sites for which the software is unavailable, according to the tag database.

The service should be intelligent enough to prevent deletion of a package that is actively being used by one or more running jobs. This could be handled either internally by the service (reference counts), or by users requesting "pinning" of the package for some amount of time.

In case the amount of resources used during installation is not negligible, a job requesting on-demand installation may have to wait for these resources to be available, independently on the amount of resources requested by the job itself. To control this it is probably necessary to be able to specify in the JDL a maximum time that the job is ready to wait for software installation before entering execution. It will be up to the WMS to use this information during the matchmaking procedure.

Pre-installation

In the above install-on-demand scenario, the primary function of the ESM is to officially bless the experiment software for use on the grid. In the pre-installation scenario, the ESM is an actor who causes the software to become physically present on the various resource centres. The simplest realisation of this is that the ESM sends installation batch jobs to the target centres. To avoid everybody being able to alter the software, the ESM is supposed to have special privileges. This may also allow her jobs to be given higher priority on the sites, as they are not supposed to take a large quantity of resources, and it is important that they are allowed to complete quickly, as they are in the position of enabling or denying access to a site for a VO.

The current LCG system does not have VO roles. This requires undesirable workarounds such as asking ESMs to have multiple certificates. Otherwise the ESM is either a privileged user, and then she has a very small quota for running fast jobs, and cannot run normal jobs, or she is a normal user, in which case her installation jobs can sit for long time in queues competing with everybody else, and deploying a new version of software may be very long and painful.

Again a more elegant solution would be to have a Grid enabled service that could be asked to install software by the ESM on target CE's. In the above scenario, this is achieved inserting in the installation queue of each target CE the request for the given tag. Jobs requesting this tag can then be sent immediately, and they will be retained on the destination CE till the installation ends successfully.

It is somewhat implicit in a pure pre-installation scenario that, once the software is installed by the ESM, only she can remove it. Another possibility would be to introduce a "pin time", i.e. a lapse of time during which the software is guaranteed to be on that CE, after which the local sysadmins can remove it.

Virtual File Systems

For completion we mention a third option that may be interesting in the long run – to place the software in virtual file system. One might liken this to AFS for the Grid, however from a user point of view the FS is read-only; only the ESM needs to have write access. A nice example of this philosophy is the Zero-install system (http://zero-install.sourceforge.net/).

4.3 Resources needed by installation

As we said above, installation jobs are supposed not to need a large quantity of resources. However that assumption may turn out to be false. In case of installation from source, compilation of large quantities of code at high optimisation levels can be very expensive in CPU time and memory. Verification of the installed software can also be very expensive in resources. If this is the case, the installation would have to be scheduled taking into account the resources needed.

The indication of the resources required by the installation can be given directly by the ESM. This is obvious in the case where the installation is performed via a batch job, less evident when it is done via a "service". Another source could be the Logging and Bookkeeping (L&B) service, where the information on the resource consumption of previous installations and verifications is stored.

Another issue to consider is that of the resources required for storing the installed software. The extreme on-demand scenario requires only enough resources to store the software being used. This may not be true if the software installed on demand remains on the site after having being used. The pre-installation scenario assumes that software will not disappear from a site unless specifically requested by the ESM. In both cases site software space-management tools may be needed in the future.

4.4 Package dependencies

Usually a software package needs other packages in order to work correctly. In most cases not all versions of the needed packages are acceptable. This creates a tree of dependencies. Installation of a package at a CE can be successful only if all dependencies are recursively satisfied.

If the information provided by the ESM when publishing a new tag contains also the dependencies, then the installation service can take care of inspecting the dependencies and recursively install all the needed packages. There are several packages that perform these functions, however these are not Grid-enabled. A very nice example, simple and powerful is the fink software maintenance tool (http://fink.sourceforge.net/).

Of course we are aware of the fact that installation of a new version of a given package performed in order to satisfy the dependencies may conflict with existing software already installed. This may open the way to have several versions of the same packages installed, which will complicate considerably the scenario described above.

4.5 Software and files

In this section we describe a possible implementation issue. We realise this is formally outside the scope of a requirement document, but we report it because some of the authors felt it relevant to the discussion. It is tempting, and indeed very elegant, to consider the following scenario. A given software tag is described in the File Catalogue (FC) as a collection of files. These files will share the software tag as metadata. When the WMS schedules a job to a site, the request for a given tag is translated into the request for an additional set of files. On-demand installation corresponds to the WMS (centrally or local to the site) arranging for the transfer of the requested files. Pre-installation corresponds to manual replication of these files to a given site. We think this option should be considered very attentively for its simplicity and because the whole machinery of the Data Management System (DMS) could be used in this case. We are however aware of the following issues:

1. Software files should go into a specific, CE dependent, location. This may be difficult to communicate to the DMS. Point 3 can be a solution to this.
2. Software needs to be installed in a posix-accessible location, and replicated files are normally installed on an SE, whose filesystems may not be accessible from WNs via posix calls.
3. The concept of replica postprocessor should be introduced, i.e. a procedure that runs after that a set of files has been replicated. This is necessary for software installation and validation on the CE.

All these points argue for a package management service running on each site, using the DMS as an underlying system for package storage and transport, tag publishing, and dependency-metadata support. We think worthwhile to consider this mechanism.

4.6 Software publication format

We have used several times the term "the ESM publishes the software to the Grid", however we never said how this is going to happen. This has to be negotiated between the experiments and the provider of the installation services. In general the ESM will indicate the location of the files and the corresponding tag, an *installation* and a *validation* script. If a general service is introduced, then all these components need some kind of standardisation, which however does not seem to be a major problem, once the right conventions are discussed and agreed between experiments.

5 User requirements

Given the above description, we are in the position to provide some more detailed user requirements for software installation.

1. There should be a Grid enabled mechanism allowing installation of the experiment software at a site, where by installation we intend:
    a. Retrieval of the software files from a place indicated by the ESM;
    b. Copy of the files on the target CE;
    c. Run of the installation scripts provided by the ESM;
    d. Run of the verification scripts provided by the ESM;
    e. In the case of ESM-driven preinstallation, a successful completion of these four steps should result in a record being published in the information system indicating that the software associated with the tag indicated by the ESM, has been successfully installed on the target CE. This tag should be available to all authorised users and to the WMS as part of the matchmaking procedure.

    The above is just an example, even if we think it is quite general. Other installation procedures, conforming to the site policy and negotiated with the VO, leading to the same results, would be acceptable. It would be highly desirable that, before executing a., the installation mechanism could

    i. Check that all needed dependencies are satisfied;
    ii. Trigger installation of needed tags that are not installed on the target CE;
2. There must be a standard way to locate a software package, i.e. to find out the top directory where the software is located, or where it could be installed on demand.
3. If the validation script runs successfully, subsequent jobs scheduled for the target CE should find the installed software in a directory that

should be possible to determine via a standard procedure starting from the package name and version tag.

4. It should be possible to restrict usage of the installation mechanism only to the ESM of each experiment.
5. It should be possible for the installation mechanism to be invoked on demand either by a user job, even if the user has no right to do that directly, or by the WMS. Experiments will have to choose either this installation method or the previous one, as they are mutually exclusive.
6. The installation mechanism should be such that, either invoked by the ESM or by the users, as mentioned before, access to the experiment software is strictly controlled. Each action should leave the software in a well-define state, which ensure either successful operation of the jobs or a clear corrective action to be taken.
7. The installation mechanism should have priority on normal user jobs, to ensure that software installation of a new tag can proceed quickly.
8. There should be the possibility via the same mechanism to revoke a tag. This implies removal of the tag from the information system and flagging the space occupied by the corresponding files as available for reclaim.
9. The information about packages needs to be encoded in a fashion that allows the JDL matching mechanism to do version-inequality arithmetic on packages, e.g. python>2.2.4
10. It must be possible to publish package-validation failures in a way that the WMS can avoid repeatedly sending jobs to sites where the installation and validation has failed.