LHCb

Job Configuration Data Production Bookkeeping

LHCb Data Management Project

Scenarios and Requirements

Issue:	1
Revision:	1
Reference:	LHCb yy-xxx COMP
Created:	20 November 2001
Last modified:	<Unknown>

Prepared By:	LHCb data management working group J.Closier, M.Frank, E.van Herwijnen, F.Jacot-Loverre, P.Mato, S.Ponce Editor: M.Frank

Abstract

This document collects together requirements and example scenarios/use-cases for the management of datasets produced by LHCb software components. Three loosely connected domains in this area were identified:

- the job or process configuration area,
- the data production domain
- and the bookkeeping domain.

These domains were identified and the requirements defined with the help of use case studies. The use cases were collected from a number of possible clients. The use cases also serve for the extraction of further requirements and for the testing of the functionality and robustness of the system architecture.

This document will be under constant revision for the foreseeable future.

Document Status Sheet

Table 1 Document Status Sheet

1. Document Title: Job Configuration, Data Production and Bookkeeping			
2. Document Reference Number: LHCb yy-xxx COMP			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
1	1	20 Nov. 2001	Initial version

Table of Contents

1 Introduction

This is an informal document whose aim is to capture the essential requirements of the LHCb data management. This document is an attempt to cover the functionality needed to perform the LHCb data challenge, which is planned at the end of 2002. During this data challenge, Monte-Carlo events will be generated, reconstructed and analysed. In order to first start such an event data production, tools must be available to facilitate the different data handling steps, but also to monitor the progress of the production etc. The scope of this document should cover the handling of the production of these datasets and the handling of produced datasets themselves.

The document consists of three sections: Use Cases, Requirements and Tools.

An attempt was done to deduce the system requirements from use cases. Clearly the search for representative use cases by definition cannot be exhaustive. Therefore additional use cases may be added in the course of the project as missing functionality becomes evident. The use cases were deduced from a number of users of the existing data production and bookkeeping environment. These users see mainly the deficiencies of the existing environment. Hence, functionality of the existing tools may not be neglected.

After the analysis of the use cases, we deduced a set of requirements, which must be fulfilled by the different domains we identified.

During the analysis of the use cases and the deduction of the requirements of the individual domains it became obvious, that the job cannot be handled by data bases alone. Typically users interact with several domains. To ease this interaction, emphasis must be put on tools, which we listed in a separate chapter.

We have grouped the use cases according to the type of individual who is most affected by or interested by that particular usage. We have identified the main groups of people who will work with the framework, as follows:

1. *Physicist users*, who need to obtain information about datasets available for physics analysis.
2. *Application managers*, who have to deploy software packages and applications.
3. *Data production managers*, who are responsible for collaboration wide data reprocessing activities such as Monte-Carlo productions, data reprocessing etc.

These roles are not exclusive to individual persons, and of course, the same people may perform different roles at different times.

These users will interact with 3 different domains:

1. The *Configuration domain*. This includes both, the configuration of the software in terms of packages, libraries and executables, but also the parameters used at run-time, which define the behaviour of the software.
2. The *Data Production domain*. Only very few users will interact with the data production environment: this area typically is under the control of data production managers, who perform data manipulations like reprocessings, and Monte-Carlo data generation for the benefit of the entire collaboration.

3. The *Bookkeeping domain*. Whenever information about certain dataset is required, it can be looked up. This includes also the search for datasets available for physics analysis.

We assumed that any application used for the processing of data resulting from particle collisions uses the Gaudi framework [1]. In Gaudi, data processing can be expressed as a sequence of executing algorithms, which have tools and services at their disposal.

A seperate user interface of Gaudi to the Grid, called Ganga, will be developed seperately. Gang is not part of this project, but rather will Ganga use the component delivered. Ganga will then e.g. gather the parameters necessary to configure the software, calculate the resources required by the job and finally submit the job to an optimal compute element. Ganga will interact with the bookkeeping database to locate any required input dataset on the Grid. Finally, for production work, Ganga will keep track of jobs in the data production database.

For the simple reason that in LHCb we are not starting from a blank sheet, but rather from an existing environment, we included use cases as well as problems with the current situation if these problems are known.

Throughout the document emphasis was put on capturing the ideas rather than the formal expression.

2 Use Cases

2.1 Physicist Users

The “physicist user” develops data analysis code in order to extract physics parameters from the data. To do so he has to instrument his data analysis program and then process datasets he obtained from the bookkeeping facility. In order to run a data analysis program he has to solve several problems:

- Decide which code versions should be used:
    - From his own analysis code repository
    - From the LHCb specific packages
    - From Gaudi or other LHC wide available code (CLHEP,...)
- Decide the run-time configuration of the selected components:
    - Select components (services, algorithms and tools) from the chosen packages
- Configure the selected component with the corresponding options
- Select input data to be processed and submit the processing job.

In order to illustrate these actions a set of use cases was collected, of which we hope, that the reflect the daily work of the “physicist user”.

2.1.1 Use Cases

S-PU-1 Configure a new analysis program.

It should be possible to select individual algorithms and tools from a palette in order to build a chain of algorithms, which form his analysis program.

- A chain of algorithms can represent a new analysis program. These algorithms have sub-algorithms, tools, services and other objects at their dispense, which allow reusing them in a flexible way.
- Because it is typically a miracle to know which algorithms, tools etc. and in which DLL they are implemented, he wants to explore known components libraries by browsing the library i.e. getting all factories.
- The physicist wants to build “his” data processing chain and assign the properties of the identified objects. He wants to do this with an intuitive tool, maybe even graphically. To build the processing chain, he selects the algorithms to be executed in the desired order from a palette. The palette must contain available components (algorithms, tools) which are accessible in
    - The users code repository (i.e. his “mycmt” directory)
    - The LHCb and Gaudi components

- When he has build his chain of algorithms, he wants to perform a basic check whether the order in which he wants to execute algorithms satisfies certain criteria such as the availability of input data etc.
- Finally he wants to save the configuration he obtained.
- He then wants to submit a test job of 10 events of type B->pi pi for testing purposes.
- The next day he wants to read back the configuration saved previously, modify it and save again.
- Finally after the setup is finished, we wants to analyse all data of a given year using his analysis program.

S-PU-2 Sharing an existing data analysis.

Analyses are not only executed in the context to one single user, but also shared e.g. within a given analysis group. The analysis group wants to define the preselection criteria, which have to be applied to preselect events which correspond to a analysis channel. The entire setup of this preselection must be stored.

- Our example physicist wants to pick up this pre-configured analysis component and use it as a well defined data preselection before he intends to execute his private selection program.
- To do so, he wants to pick this pre-configured component from the palette available and modify his private analysis program.
- He then wants to save the obtained configuration.
- Finally after some small tests, he wants to run over a large Monte-Carlo dataset, which he retrieves from the bookkeeping environment.

S-PU-3 Deploy group analysis tag creation program.

This is the same problem as Brunel application manager faces. The analysis program for creating an event tag collection would be a quasi-official program under the management of a “Tag-Manager”. In order to deploy an analysis tag program, the same criteria apply as in S-PU-1 In addition the creator of the group analysis program must

- Add the tag creation program as a collaboration application. This includes the registration with the configuration database.
- Add all parameters needed by the application and the components used in the application to the configuration database.

S-PU-4 Configuration of an interactive analysis.

One user wants to start an interactive analysis by simply plugging together existing algorithms and executing them. For this purpose he uses a dummy package, which only contains the Gaudi bootstrap code. Then, of course he wants to configure the newly created application and instrument this application with algorithms, tools etc. Finally he wants to save the configuration.

This use case is identical to S-PU-1 with the special case that the user’s code repository is empty.

S-PU-5 Reuse of Reconstruction code for Analysis.

A physicist wants to reuse certain tools and algorithms which typically run in the reconstruction program for his data analysis program. However, in the data analysis environment the options for the algorithms differ slightly from the options used in the reconstruction. The database should allow him to automatically pick up the correct options according to his selection.

- He selects the algorithms he wants to execute from the LHCb code repository.
- He specifies that the components are executed in the Analysis environment. This triggers, that the parameters of these components are selected accordingly.
- He saves the obtained configuration locally without affecting the collaboration wide configuration database and reuses this configuration for submitting the data analysis job.

S-PU-6 Retrieval of datasets for physics analysis.

The physicist wants to access the datasets, which contains the information he asks for.

- He wants to select datasets according to several criteria, which are:
    - Predefined by the tool such as “B-> J/Psi... Monte Carlo created with Brunel v182”
    - Defined by a set of generator parameters.
- He also wants to possibly restrict his selection to datasets, where certain configuration parameters of Brunel v128 have a certain value.
- Via a display program (Web for the time being), he is able to do selections on different channels and to get a list of the data found. In the existing implementation this is a card file in the format requested by the data analysis program.
- At the same moment he also wants to retrieve basic information about the resulting datasets such as the total number of events.

NB: For the time being, the card file contains a list of JOBIDs and the programs must retrieve the associated storage device by another request to the bookkeeping database.

S-PU-7 Retrieve additional information about a dataset.

The results of a data analysis job has shown results, which cannot be explained. To understand the difference to the expectation, the physicist has the suspicion, that e.g. the Monte-Carlo event generation was performed with incorrect parameters. He wants to inspect the parameter set, which was used to produce the dataset in question. Using a display program (e.g. WWW browser), he is able to retrieve all relevant information from the individual processing steps, which was used to produce this dataset:

- Code configuration.
- Parameter configuration.
- Input datasets.
- Log files.

S-PU-8 Small user production

The physicist wants to test his selected event sample against a particular theoretical model. This requires that he produces 1000 Events with special generator settings.

- In order to save work he uses official settings for the simulation program version 123. He only overrides the settings he wants to change and saves the resulting configuration.
- Then he picks up the default settings for the reconstruction program version 234.
- Using these two configurations he builds and submits his production job.

2.2 Application Managers

Application managers are librarians who deploy either new versions of applications such as the reconstruction program parts of it like the tracking package(s). The deployment of a package includes setting up the default configuration options to execute the components of the package. The application manager only picks these settings and releases the application.

In order to test the new version of the released code configuration a test must be performed, which requires the execution of a few events.

2.2.1 Current Situation and Problems

Job options go along with the code in the form of text files. This heavily restricts the use: A new version of Brunel must be released if the job options change.

Currently this is not yet an issue, but it will be difficult to have different versions of Brunel according e.g. to the year of data taking, since probably different version of options are needed. To do this in a more general way, the application must be able to adapt itself to changing conditions, depending on the input data.

At some point also options must change according to the input data being processed (conditions db issue?).

2.2.2 Application Configuration

In Gaudi any application can be expressed as a sequence of executing algorithms, which have a set of tools and services at their disposal. All three, algorithms, tools and services require configuration through external parameters at run-time. These parameters actually give the executable code of the application its personality.

A *Configuration* of an application is defined by

- The code executed at *run-time*. This code is not necessarily the code used to build the executable, because many images are loaded at execution time. Nevertheless the code management tool which also defines the run-time environment has all knowledge about possibly used sub-packages.
- Options of algorithms, tools and services.

The handling of event data inside Gaudi uses a blackboard-like data store. In this approach the data flow is given by algorithms interacting with the data store. However, a logical data flow occurs as well, because algorithms simply require certain input data, which are produced by other algorithms. The execution sequence of these algorithms defines both, the mapping between the real and the logical data flow and the dependencies between these algorithms. Hence, the configuration must deal with:

- The mapping between logical and real data flow. This way there is some possibility to check whether an algorithm can be executed at all in a given environment.
- Dependencies between algorithms

2.2.3 Use Cases

S-PD-1 New Brunel Release

- Brunel options should be independent of the code. However, there is a dependency on the code in the sense, that options may or may not be existent depending on the version of the code.
- The Brunel application manager wants to configure an existing executable with several sets of job options. These different sets could correspond e.g. to LHCb classic, LHCb light, etc. He sees a configuration as an existing version of the software and the corresponding parameters. When creating a new configuration he wants to pick up default values and then only incrementally modify to achieve the wanted result.
- On demand, he wants to access the entire configuration with all options. The configuration options must be accessible in a well understandable form, grouped e.g. by algorithm, service etc.
- Similar to the current situation, where the package coordinators maintain the job options files, the options are edited by the package coordinators (typically one per package). A released Brunel configuration then is based on a slice through all algorithms/services/tools used by Brunel and a slice of the packages used.

Summary:

- Sub detector package coordinators edit options.
- A release is a slice using one version of all necessary options of all packages used.
- Necessary means that it must be possible to also generate a view of all application objects, which have options.
- Several of the above releases may use the same software configuration.

S-PD-2 Switch “On” or “Off” certain branches in the reconstruction

Example: Run only level 1 code:

A user wants to execute modified reconstruction code. To speed up the execution for his study, which deals only with trigger level 1 analysis, he wants to switch off any later data reconstruction.

At some point the dependencies must be evaluated, that the level1 code can only be executed if the level 0 code was executed before. At least relevant information must be present, if level 0 information should be used e.g. in “Super level 1".

This also corresponds to another set of options to be used, e.g. when a new reconstruction run must be configured, which is specialized e.g. on level 1 trigger.

S-PD-3 Publish reconstruction code for the use in analysis programs

The package manager wants to make an algorithm in a component library available for the usage within the data analysis framework. He knows that the parameter setup is slightly different. In order to ease the use of his clients he wants to supply two versions of default parameters: one for the use within the reconstruction program and another for the use within the analysis program.

S-PD-4 Access to event data for application tests.

Program developers have very similar needs as physicists performing data analysis. Their selection is often smaller than for the data analysis, and they sometimes want to have a look to some details (as: log files, etc.). They would be happy to be able to do their selections on the bookkeeping database directly in Gaudi. This feature typically is used just to check that programs basically work. For this purpose they would like to access event data by specifying a statement like '10 events of type B->pi pi'.

2.3 Data Production Managers

2.3.1 Current Situation and Problems

As of now the information of the job configuration needed to run a production is partially embedded in java servlets in the form of code and must be supplied through specially adapted WWW pages Eric has built himself. This schema is a bit error prone due to typos etc. Individuals typically pass a set of generator parameters to the production manager. This information includes:

- The definition of the content of the output stream(s).
- Special generator parameters.
- Sometime people supply built executables.
- Special cards files, special cdf files etc.

The production manager typically uses either prepared scripts or for special cases writes extra scripts, to submit the jobs to the batch system(s). Ideally he does not want to worry about all the different parameters, because all this inhibits and interrupts proper job submission.

S-DPM-1 Configuration of a B->J/Psi Monte-Carlo production.

The data production manager can select a predefined configuration of parameters, which generate the proper events. These parameters were agreed e.g. by the physics group. He picks them up from a database using a logical identifier. The same holds for detector simulation and reconstruction parameters. These parameters will not change for the entire production.

Using these generation parameters he can focus on the real task, which is the definition of proper output files, batch farms to which the jobs can be submitted etc, but this is then to be solved in the production system.

S-DPM-2 Generate a production

The physics group asks to produce 500K events of physics channel 61. The RAWH must be produced with SICBMC version A and the DST must be produced with Brunel version C. This production must be ready in one-week time and the DATA files must be available at CERN.

- Get the right version of the programs
- Generate the job for the production for all the steps
- Submit the jobs
- Check the jobs when they are finished
- Transfer the data files
- Update the bookkeeping
- Save the production stream

S-DPM-3 Reprocessing of a part of a production

A given production has produced DST files with a wrong version of Brunel. We have to redo this production with the correct version of Brunel. Restore the previous version of this production and launch only the Brunel part.

- Restore the production stream
- Get the right program version for Brunel
- Submit the jobs concerning the DST part
- Check the jobs when they are finished
- Transfer the data files
- Update the bookkeeping
- Save the production stream

S-DPM-4 Monitoring a production

A production failed because some jobs crashed. Find the cause of these crashes and resubmit only these jobs.

- Monitor the status of the jobs launch by a production
- Check in which phase they are and the number of events already produced
- Check output of jobs which crash
- Resubmit after correcting the problem, if possible.

S-DPM-5 Retrieval of datasets as input for a new production run.

The production manager needs to find and identify the datasets he needs to perform e.g. a reprocessing. When their production is ended and checked, he must update the bookkeeping database with the new datasets and has to make the newly created data available to the collaboration.

3 Requirements

We make no attempt to be exhaustive, relying instead on an "on-the-fly" analysis of the scenarios. We listed here also requirements, which were not immediately obvious from the use cases, but which became clear in the course of the discussions. For this reason not all requirements are necessarily connected to a specific use case.

3.1 Job Configuration

3.1.1 Definitions

Code: The term "Code" includes both, the code itself, the files that allow to compile it (e.g. requirements in the case of CMT but also fragments) and all parameters defined there. These parameters may define that such algorithm is or is not included into the binaries. The term code configuration is used to name these parameters.

Application: The term "Application" defines the code (as described above) and all configuration files that go with it and actually give to the program its specialisation. As a result, several applications may correspond to one single code. As an example, since the list of algorithms used in a reconstruction program is part of the configuration, Brunel could be reduced to a set of configuration files.

3.1.2 Edit Capabilities

UR-1 Identify/edit/add a code configuration in the global configuration database.

`Required by: S-PU-3, S-PU-4, S-PD-1`

UR-2 Identify a code configuration in the global configuration database and extract and save the selected configuration locally. Local configurations can also be re-opened and modified.

`Required by: S-PU-1, S-PU-2, S-PU-4, S-PU-5, S-PD-1`

UR-3 Identify applications (=code configuration + Parameters). One or several applications may correspond to one code configuration.

`Required by: S-PU-3`

UR-4 Add applications. The addition could be done by designing a brand new application or by slightly changing an existing one.

`Required by: S-PD-1`

UR-5 Edit application configurations in a convenient way

`Required by: S-PU-1, S-PU-2`

UR-6 It must be possible to browse an application configurations.

- Set/Edit parameters of Algorithms, Services and Tools, etc.

- View the packages in use by a given code configuration.

`Required by: S-PU-1, S-PU-2, S-PD-1`

UR-7 In order to build an applications dynamically, it must be possible to browse the content of a package. Doing so, it must be possible to access Algorithms, Services and Tools, which are implemented by a given package.

`Required by: S-PU-1, S-PD-1`

UR-8 Import Algorithms, Tools etc. into a given configuration.

`Required by: S-PU-1, S-PU-2, S-PU-4, S-PU-5`

UR-9 It must be possible to deal with several sets of parameters for a given Gaudi Service, Tool or Algorithm etc.

`Required by: S-PD-3, S-DPM-1, S-DPM-2, S-DPM-3`

UR-10 Bookkeeping tools must allow to convert a selected set of datasets into input, which can be understood by the data analysis program(s)

`Required by: (nearly all use cases)`

3.1.3 Access Control - Privileged Users

UR-11 Official (public) code configurations may only be edited by

- Package managers
- Application managers

The configuration tools must facilitate this functionality for these users and at the same time apply the restrictions to all other users.

`Required by: S-PU-3`

UR-12 Official (public) applications may only be edited by

- Package managers
- Application managers

The configuration tools must facilitate this functionality for these users and at the same time apply the restrictions to all other users.

`Required by: S-PU-3`

UR-13 The configuration tools must allow read-only access for any member of the LHCb collaboration.

`Required by: S-PU-1, S-PU-2`

3.1.4 Access Control - Non-Privileged Users

UR-14 Individual users must be able to save private configurations

`Required by: S-PU-1`

UR-15 Individual configurations may inherit from official configurations

`Required by: S-PU-8`

UR-16 It must be possible to read private configurations, modify private configurations and save them.

`Required by: S-PU-1`

3.1.5 Compatibility Checks

These actual implementation of these requirements may go to the "Desirables" section and is not mandatory from "first-day".

UR-17 Data flow requirements must be fulfilled. This means that input data required by an Algorithm must be present at execution time

- Input data objects of algorithms
- Output data objects of algorithms

`Required by: S-PU-1`

UR-18 Rule checking

- Dependencies between algorithms and services

`Required by: S-PU-1`

3.2 Data Production

3.2.1 General Functionality

UR-19 The data production manager must be able to override default configuration parameters.

`Required by: S-DPM-1, S-DPM-2, S-DPM-3`

UR-20 The data production manager must be able to access these parameters until a production run is finished

`Required by: S-DPM-1`

UR-21 When production parameters get changed by the application manager, the data production managermust be able to check the consistency of the resulting parameters set.

`Required by: S-DPM-1, S-DPM-2, S-DPM-3, S-DPM-4`

UR-22 The data production environment must be able to generate the production scripts for jobs submitted according to the program configurations (code version + parameter configuration)

`Required by: S-DPM-2`

UR-23 According to the size of the production the tools must be able to split jobs into junks, which can easily be handled.

UR-24 It must be possible to merge datasets e.g. for mini-DST production

`Required by: S-DPM-3`

UR-25 When a production is to be submitted, it must be possible to specify the input datasets if any.

`Required by: S-DPM-3`

UR-26 It must be possible to delete intermediate datasets, which should not be kept.

UR-27 Datasets are not only binary data, but also e.g. histogram or log files. The production environment must be able to handle all these datasets.

UR-28 Output datasets must be transferred to a destination specified by the production manager.

`Required by: S-DPM-2, S-DPM-3, S-DPM-5`

3.2.2 Building Production Runs

UR-29 Data production typically is not defined by one single execution run, but rather a workflow given by the execution of a sequence of executing tasks. It must be possible to build such workflows specifying the individual steps and identify the workflow by a name. It must not be necessary to specify all parameters for all tasks, but rather take advantage of a “palette” of existing steps.

UR-30 When building a new workflow, it should be possible to inherit from an existing workflow and modify the newly created one.

UR-31 The data production tools should be able to interact with the bookkeeping database and retrieve from this data-base input dataset specifications a given production run (e.g. reprocessing).

`Required by: S-DPM-3`

3.2.3 Control Requirements

UR-32 The production manager must be able to launch new productions to specified production centres. The centres themselves are defined elsewhere, e.g. by the GRID environment.

UR-33 The production manager should be able to kill jobs corresponding to a production run.

`Required by: S-DPM-4`

UR-34 The production manager should be able to Cancel jobs in two ways:

- Letting them finish the current step, but not automatically proceeding to the next step of the workflow.
- Immediately stop the current step.

`Required by: S-DPM-4`

UR-35 It must be possible to put jobs in a “hold” state and re-schedule them in case another production request with higher priority must be worked on.

`Required by: S-DPM-4`

UR-36 Starting from a given checkpoint, the production manager must be able to restart a whole production run or individual jobs

`Required by: S-DPM-4`

UR-37 It must be possible to split up production runs into "handy" junks.

UR-38 It must be possible to specify several input datasets and merge them into one output dataset.

3.2.4 Monitoring Issues

UR-39 The production manager must be able to interrogate the submission information software configuration of a running job in terms of

- Configuration parameters
- Additional parameters specified by the production environment
- Once submitted also the state of the production job is of interest
- Event number currently processed
- Workflow state
- Any objects published by the job for monitoring issues

`Required by: S-DPM-4`

UR-40 In order to start a production run, monitoring of available resources (CPUs, network etc.) should be available through the data production tools.

3.2.5 Users other than Production Managers

UR-41 Users, which do not use the production environment as production managers, will also have to start small production runs. A tool must exist, which allows starting small productions or data processing jobs.

UR-42 The tool must be able to split the requested data processing into suitable junks.

UR-43 These users also want to benefit from monitoring and control facilities.

3.2.6 Conditional Requirements

UR-44 The job submission and the storage of datasets could be managed by the GRID. The tools must use an interface which bridges the LHCb specific software from the GRID middle ware such as GLOBUS [2].

UR-45 According to the design principles of the Gaudi framework, should both, tools and API technology independent. The implementation should be flexible enough to cope with database implementations other than ORACLE, what is currently used.

3.3 Bookkeeping

UR-46 The data production manager must be able to select input datasets for a given production run.

UR-47 After finishing a data production run, the data production manager must be able to update the bookkeeping database with new information in order to make the produced datasets available to the physicists for data analysis.

`Required by: S-DPM-2, S-DPM-3, S-DPM-5`

UR-48 It must be possible to retrieve both, datasets, which correspond to collections of events and collections of event tags

`Required by: S-PU-1, S-PD-4, S-DPM-5`

UR-49 Users must be able to retrieve datasets according to

- Predefined select statements
- Specify themselves SQL query statements

`Required by: S-PU-6`

UR-50 After a select was successfully issued, it must be possible to refine the selection with parameters, which were used to produce the dataset. If such a refinement is requested, the user must have a choice to apply the selected criterion or not.

`Required by: S-PU-6`

UR-51 Users must also be able to exercise dataset queries in order to

- Information, which was used to produce these datasets, such as log files.
- The number of events the resulting query request corresponds to

`Required by: S-PU-6`

UR-52 To facilitate the interactive access to the data a suitable display must be provided, which allows presenting the retrieved data in an intuitive and convenient manner.

`Required by: S-PU-6`

UR-53 The attributes of a given dataset cannot be foreseen. For this reason the bookkeeping domain must allow to add or remove attributes to datasets.

UR-54 Different datasets may have different attributes.

UR-55 It must be possible to display and extract the entire configuration information, which was used to produce a dataset. If several steps were necessary to produce a dataset, the information should be presented accordingly.

`Required by: S-PU-7`

UR-56 Tools must exist, which allow to easily update the bookkeeping information with information located in the data production domain after a production run was successfully finished and the produced datasets should be made available to the collaboration.

UR-57 Bookkeeping tools must allow to convert a selected set of datasets into input, which can be understood by the data analysis program(s)

`Required by: S-PU-1, S-PU-2, S-PU-6`

4 List of Required APIs and Tools

When analysing the use cases of the different types of users it became evident, that a large part of the required functionality actually has to be implemented in tools, which have a persistent backend, the database.

In any case the tools this section we try to identify necessary tools, which help the users to perform the actions we identified in the use cases. The difference between a tool and a programmatic interface (API) is not strictly defined **-** with the help of an API the tools actually can be implemented more easily. In this case however, more emphasis must be put on the definition of the API, because it will be public and hence must be more intuitive than calling sequences, which are internal to a tool.

4.1 Job Configuration Domain

Table 1 Tools and APIs needed to configure a data processing job.

Tool Functionality	**Users**	**Tool/ Gui**	**API**
Explore, edit, save and re-read the options of a given application/package/component	Physicist users Application Mgr Production Mgr	yes	yes
Deploy options of a component/package/application to the common configuration database of the LHCb collaboration.	Application Mgr	yes	
Extract and visualize parameter configuration from component/package/application and update configuration	Physicist users Application Mgr Production Mgr	yes	yes
Extract and visualize code configuration of a component/package/application from the configuration database	Physicist users Application Mgr Production Mgr	yes	
Verify code and parameter configuration settings	Physicist users Application Mgr Production Mgr	yes	
Explore component DLL	Physicist users	yes	

4.2 Data Production Domain

Table 2

Tool Functionality	Users	Tool/ Gui	API
Job preparation	Physicist users Production Mgr	Yes	
Job submission	Production Mgr	yes	
Monitoring	Production Mgr Physicist users	yes	yes
User job submission	Physicist users	yes	

4.3 Bookeeping Domain

Table 3 Tools and APIs needed to fulfil bookkeeping requirements.

Tool Functionality	Users	Tool/ Gui	API
Selection of datasets	Physicist users Production Mgr		yes
Display of dataset selections	Physicist users Production Mgr	yes	
Commit updates to bookkeeping database	Production Mgr	yes	yes

4.4 Tools and APIs Spanning Several Domains

Table 4 Tools and APIs

Tool Functionality	Users	Tool/ Gui	API
Job submission requires bookkeeping and grid tools and production stuff...			

5 References

[1] Barrand G. et al., *GAUDI - A software architecture and framework for building LHCb data processing applications,*
Proc. of CHEP 2000, Comp.Phys.Comm. Vol 140,1-2.
http://lhcb-comp.web.cern.ch/lhcb-comp/General/Publications/longpap-a152.pdf

[2] The Globus Project, http://www.globus.org