System Kernel

Iain Last, Pavel Binko

1.0 The System Kernel.

This is a collection of very basic utilities, not associated to any package in particular, which will ensure cohesion between the various parts of the LHCb software. The system kernel may also define some conventions e.g. when and how to use exceptions.

2.0 Responsibilities

- The LHCb common header files. These will define system wide macros, constants, typedefs, common error codes, etc.
- Base classes & Interfaces that will be used by any components of the LHCb software.
- Ensuring RTTI is possible on any given platform. This shall be the only functionality of the kernel.
- Minimising effect of any changes to base components on end user.

2.1 The Header files.

There are two groups of header files in the **Kernel** package:

1) Definition files - *Constants.h, Kernel.h, StatusCodes.h*

2) Base classes and interfaces.

The ***Kernel.h*** file includes the files required for standard type definitions, RTTI and exception handling. It may well be included by every header and implementation file so therefore should include as few files as possible to minimise dependencies.

The ***StatusCodes.h*** file defines a **StatusCode** type and common error codes that should be used when a status is queried. These are defined in a **StatusCodes** namespace so that a successful operation from and LHCb component would return StatusCodes::SUCCESS.

The LHCb codes are given in the table below:

Status	*Description*
SUCCESS	Operation was successful.
FAILURE	Operation failed with no specific reason.

TABLE 1. List of LHCb status codes.

The ***Constants.h*** file defines the a **Constants** namespace and provides definitions of the following commonly used mathematical values.

Constant	*Value*
E	e
LOG2E	$\log_2 e$
LOG10E	$\log_{10} e$
LN2	ln 2
LN10	ln 10
PI	π
PI_2	$\pi / 2$
PI_4	$\pi / 4$
ONE_PI	π^{-1}
TWO_PI	$2\ \pi^{-1}$
TWO_SQRTPI	$2\ \pi^{-1/2}$
SQRT2	$2^{1/2}$
SQRT1_2	$2^{-1/2}$

TABLE 2. Mathematical Constants.

In addition there are other files implementing the defines mentioned in Section 2.5 e.g. Point3D.h will include HepPoint3D and provide a define such as *#define Point3D HepPoint3D<double>*. These defines will be stored in the **tools** package.

Define	*Implements*
LorentzRotation	HepLorentzRotation
LorentzVector	HepLorentzVector
Normal3D	HepNormal3D
Plane3D	HepPlane3D
Point3D	HepPoint3D
Rotation	HepRotation
ThreeVector	HepThreeVector
Transform3D	HepTransform3D
Vector3D	HepVector3D

TABLE 3. List of LHCb include files implementing third party classes.

2.2 Built-in Types.

We should not use built-in types for portability reasons. Instead we should define specific typenames which have a well defined size:

- **char, int32, int16, uint32, uint16**
- **float64, float32**

These correspond to the Objectivity types.

2.3 Base Classes & Interfaces.

A set of base classes and interfaces shall be provided which must be inherited by each class of a given type **e.g.** service classes will inherit the service base class and implement the service interface etc. These abstract classes will have virtual inheritance and virtual destructors.

Type	*Interface*	*Package*
Algorithm	IAlgorithm	Algorithms
Convertor	IConvertor	Convertors
Interface	IInterface	Kernel
Manager	IManager	Managers
Property	IProperty	Kernel
Selector	ISelector	Selectors
Service	IService	Services

TABLE 4. List of types and their interfaces.

Type	*Base Class*	*Package*
Data	IDataObject	Kernel
IDataObject	IdObject	Kernel

TABLE 5. List of types and their base classes.

N.B. Each of these base classes will, for the time being, inherit the *SimpleRTTIObject* class.

2.4 RTTI (*R*un-*T*ime *T*ype *I*nformation.)

Run-time type information works on polymorphic classes and will be used in the cases where one needs to test whether or not a cast can be made from a lower level (inherited) object to a more derived object but not necessarily the most derived object. Note in the case where an object inherits multiple interfaces (does this ever occur?) it will be necessary to use RTTI to identify the interfaces rather than virtual functions.

2.5 Change Impact Management.

Where reasonable the kernel system may provide a substitute class / typedef / define for an external component. e.g. It may be desirable to have a define for HepPoint3D in case at

some time in the future HepPoint3D will be templated and needs to be instantiated as HepPoint<double>. This will minimise the number of alterations to the source code should such a change occur.

2.6 Makefiles and/or Projects.

The kernel should provide makefiles, developer studio projects and tools to document the source code.

3.0 To Be Done.

- Decide on portable names for integers, floats etc.
- Decide on whether to add any functions to base classes.