1 Introduction

International Lattice Data Grid (ILDG) is a grid for sharing QCD configurations worldwide. Following the action plan decided at the first workshop in Edinburgh in December 2002, the metadata working group was organised in mid-January 2003. This document summarises proposals of the working group for 1) QCDML (QCD Markup Language) which is an XML schema for the ILDG, and 2) standard binary format for exchanging configurations.

2 QCDML

2.1 Strategy

Collaborations planning to submit QCD configurations to the ILDG will mark up and XML instance document (ID) according to the QCDML XML schema for each configuration. The QCDML documents are stored in databases. Researchers issue a query to databases and get information of matched configurations such as simulation parameters. The QCDML document includes a pointer to the actual configuration, which enables researchers to retrieve configurations using ftp-like protocol. Detail of the procedure will be discussed and proposed by the middleware working group of the ILDG project.

QCDML defines a minimal set of XML elements necessary for exchanging configurations and researchers are usually interested in. However, the definition of minimal has the caveat that it is consistent with general usage, and critically, extensibility to data other than gauge configurations such as quark propagators and hadron correlators\(^1\). This is not to say that the schema covers other data, but rather the structure is sufficiently general that further extension is quick and easy.

An important issue is standardisation of notion and terminology. Suppose that we search quenched QCD configurations generated with Wilson’s plaquette action. One may record the name of the action in the action element using different names such as “plaquette-action” or “Wilson-action”. When a query is issued with the name “plaquette-action”, one may not be able to retrieve configurations with the action name recorded.

\(^1\)This was discussed at ILDG2 and most people wanted this feature
as “Wilson-action”. The metadata working group propose candidates for values of elements and try to improve serviceability.

We propose that the QCDML consists of four parts; 1) Physics, 2) Simulation, 3) Management, and 4) Pointer. Each part of the QCDML will be described in subsequent subsections.

2.2 Physics Part

There are two alternative proposals. The first involves abdicating responsibility for the definition of the name of the action to a “Human Readable document”. The second has the definition in the schema, but as a result is longer. XML is clearly not suited to mathematical definitions, and this is what defines the action. However, the name of the action can be defined. Some more general points are discussed here, but they are relevant to the description of the action.

An object orientated approach can be used to define the action using the substitution group for elements. There is a much larger functionality in XML schema for elements than there is for attributes. For this reason we advocate the use of elements in general, and only use attributes if there is a specific reason for a particular datum. An abstract element of the general type which describes the structure of the data is defined in the schema, this is like a virtual class in c++. A specific element, which inherits the general properties of the general type, but has some of these properties extended or restricted is then substituted. The modular properties of XML schema are now used. The schema itself has the general structure, but specific modules are included to define named pieces. In our case the action.

Why is this a good idea? The name of the action (or anything else for that matter) is defined in the namespace of the schema. This means all XML IDs are only valid if they use the correct name. An unconstrained string attribute can take any value. It is quite possible to have the same value in the string for two different objects, or have two different values in the string for the same object. This would render searches useless, and is quite plausible for actions of increasing complexity. The “Human readable document” is not tied to the XML ID nor the schema. Its only use is to describe what the defined name means to a human, e.g the maths.

An objection to the OO approach is that every time someone uses a new action the schema has to be adjusted. Yes, indeed of course it must. XML is called XML because it will change. The way to manage this is have a general schema (which won’t change when you add a new action) and included modules which are quite short but are extended for the new action.
This isn’t a great deal of work. No more than adding stuff to the “Human readable document”.

Another objection might be that it is tedious to write out all this XML. There are several points here. Yes it is, but it is worth doing. It’s not that tedious, and besides it could be possible to have an ILDG metadata IO API that is available which would cut down but (of course) not eliminate this work. The OO approach schema would be much longer than the attribute-name schema, but the XML ID’s that have to be written by the API are not that much longer. If we try and be too minimal then we will only create more work for ourselves later.

There are two proposals. Tomoteru’s which seeks to avoid extra work load by having a minimal set of marked up tags, but without defined names. The UKQCD schema takes the OO approach, which will involve extra work load but not any great extent. Names would be defined and this approach could save work load in the future. There are other parts of schema to be decided but the issues will be similar.

The first proposal is documented in a previous version and the UKQCD schema itself exits with documentation and can be viewed by following links from the ILDG web pages. The metadata working group has to decide which approach to adopt. If there is no overall consent then the working group must except the majority view and construct XSLT translations if the ILDG schema does not satisfy their internal collaboration requirements.

### 2.3 Simulation Part

Simulation part of the QCDML consists of 1) algorithm, 2) machine, 3) code, 4) series and 5) trajectory or sweep sections.

#### 2.3.1 algorithm part

#### 2.3.2 machine part and code part

A primary purpose of elements in these part is to facilitate backtracking problems which occurred during data production and which have only been detected later, we allow (and expect) those who submit configurations to ILDG to add to each configuration information about which machine and code was used to generate this particular data.

For various reasons it is hard (or even impossible) to describe all machines used for lattice gauge theory calculations with only a few parameters:

1. A “machine” might not be a well-defined entity (see PC-farms).
2. It is possibly subject to relatively frequent changes.

3. I might be built from heterogeneous components (this can even be the case for a Cray/T3E).

We therefore concentrate on information that would allow the person, who submitted a particular configuration, to identify the machine used to generate that configuration. This could be achieved by providing information shown by the following sample.

```xml
<machine>
  <name>iras</name>
  <institution>CCP, Univ. of Tsukuba</institution>
  <type>Hitachi SR8000</type>
  <comment>PU8 partition (optional)</comment>
</machine>
```

The `<type>` element shows machine hardware type, such as Hitachi SR8000, QCDOC, apeNEXT, PC-cluster. The optional `<comment>` tag is used to give supplementary information such as the used partition.

Detail of simulation code will also not be necessary, because it will be used for backtracking by contributor of the configuration. We propose to use the following

```xml
<code>
  <name>PHMC_HMC</name>
  <version>ver5.41-SR_21.0</version>
  <date>2003-03-01</date>
  <precision value="double"/>
</code>
```

The `<precision>` element indicates whether the simulation code is coded in “single” or “double” precision. This element may be searched by researchers.

### 2.3.3 series part and trajectory_or_sweep part

In the simulation part, there are two important elements `<series>` and `<trajectory_or_sweep>`. The latter specifies trajectory or sweep number in a series of the Markov chain. We may execute simulations in parallel with the same parameter set. Therefore, only the `<trajectory_or_sweep>` tag is not sufficient to identify the configuration. The `<series>` tag distinguishes different runs. A sample of these part is
2.4 Management Part

The management part of the QCDML plays two important roles. One is for traceability and validation of the configuration. The other is to give the most important information of the “collaboration” name and the “project name”

The elements belonging to the second category are shown by an example

<collaboration> ABC Collaboration </collaboration>
<project> three floavor full QCD with RG and NP clover action </project>
<reference> hep-lat/0000000 (optional) </reference>

When researches in the lattice QCD community want to get some configurations, they usually know who generated the configurations (<collaboration>) and why or how the configurations were generated (<project>). They probably have read a paper (<reference>) and are interested in using these configurations. Therefore these elements are most frequently searched for. (Researches do not have to remember collaboration name and project name. User interface of the search engine can be designed in a way to show a list of collaboration names first, then project names of the collaboration.)

Optional reference is used to provide researches information which is not suitable to record in the QCDML. For example, suppose that one generates configurations by the hybrid Monte Carlo, store them each 10 trajectory, and put them all to the ILDG. These configurations may not be well decorrelated. One may be interested in expected auto-correlation-time so that he/she wants to retrieve only uncorrelated configurations. Such information is not suitable to be recorded in the QCDML document, because it is information for a set of configurations. Contributor don’t have to list up all references. One or two basic references are sufficient.

The rest of tags in this part of the QCDML is for traceability. For any configuration provided in ILDG, it should be possible to trace its history and the persons involved in generating and handling of the data. We should foresee the possible case where a configuration or other data provided in ILDG turn out to be corrupted (e.g. due to hardware defects on the machine used for the simulation or program bugs). In these cases configurations will have to be replaced or withdrawn, that might already have been used by others.
Thus our aim is to be able to trace the generation of any configuration, its submission to ILDG and all possible later changes. Each of these operations can be described the following triplet of information:

1. What kind of operation (generation, submission, change, withdrawal)?
2. Who executed this operation (name, institution)?
3. When was this operation executed?

A sample QCDML document looks like

```xml
<action type="insert">
  <actor>
    <name> A. Aabbcc </name>
    <institution> CCP, Univ. of Tsukuba </institution>
  </actor>
  <date> 2003-04-08 </date>
  <comment> First submission (optional) </comment>
</action>
```

Values of `<action>` tags are one of “generate” to indicate generation of configuration (this is optional), “insert” to indicate submission of configuration (required), “update” to replace the configuration (required when appropriate) and “withdraw” to remove the configuration from the ILDG (required when appropriate). When a contributor changes the configuration, `<action>` part has to be added without removing predecessors of `<action>` part.

In order to maintain the version number of the QCDML document, we use `<revision>` tag. Starting from 1 (one), we add one to the value of this tag when the document is changed. For example,

```xml
<revision> 2 </revision>
```

means that the document is changed once. QCDML documents will be stored in several databases. The `<revision>` tag might to be used to keep consistency among databases.

The last element in this part `<crc32>` is a checksum of the configuration calculated by the 32-bit cyclic redundacy code algorithm. We are planning to distribute configurations in DIME encapsulated file of configuration itself and the corresponding QCDML documents. We also allow to use different binary format of configurations group by group. (See Sec.3 for details.) The checksum is calculated for the binary part of the DIME file and for the original binary format.
2.5 Pointer Part
Tags in the pointer part show where you can find the configuration, and how you translate the binary format (see also Sec.QCDML:Format). This part depends on proposals by the middleware working group on how we manage and transfer configurations. Our current proposal is shown by a sample:

\(<\text{site}>\) ftp://qcdgpc.rccp.tsukuba.ac.jp/\(<\text{/site}>\)
\(<\text{file}>\) NF3/ProductRun/RC16x32/b1.90_csw1.715_kud0.13640_ks0.13580-C/000880.DIME \(<\text{file}>\)
\(<\text{c_library}>\) NF3/Translation/ToSTDConf.c \(<\text{c_library}>\)

3 Standard Binary Format

3.1 Strategy
The “standard format” of QCD configuration is an abstract format for exchanging QCD configurations. Several collaborations have already archived a lot of configurations with different formats. Each format is usually chosen for convenience. Each collaboration may want to continue to use their own format, because each computer has its own preferred format for which a QCD program on the computer works effectively. It is therefore not realistic to convert huge amount of data to some specific format; otherwise each site may have to store configurations in their own format and in the “standard“ format.

The metadata working group regulates the standard format as a reference format and proposes two methods to convert a configuration in one format to the one in the standard format via C-library (see Sec. 3.3), and from one format to the other using BinX technology (see Sec. 3.4).

3.2 Standard format of QCD configuration
X A gauge configuration is a set of SU(3) matrices assigned to the links of a four dimensional hypercube. If \(x(n)\) is a value of a quark field at site \(n = (n_1, n_2, n_3, n_4)\),

\[
x^\dagger(n)U_\mu(n)x(n + \hat{\mu}) \equiv \sum_{i,j=1}^{3} [x^\dagger(n)]_i[U_\mu(n)]_{ij}[x(n + \hat{\mu})]_j
\]

is gauge invariant.

Standard format of a QCD configuration is a sequence of 8-byte double precision real numbers coded in 64-bit IEEE numerical format. Endian is specified. See Sec. 3.3. The order of data is specified by the C statement
double U[t][z][y][x][mu][i][j][ri] ;
    in U[NT][NZ][NY][NX][4][2][3][2]

The last index [ri] refers to real(ri=0)/imaginary(ri=1) part of the complex
number, [i][j] corresponds to the definition above, and mu=1,2,3,4 refer to
x, y, z, and t directions respectively. We store only the first 2 rows of the
3x3 matrix. The third row can be calculated from the unitarity condition.
The format is equivalent to the fortran-77 notation

    complex*16 U(3,2,4,NX,NY,NZ,NT)

To put the format concretely,

1. 2x3 matrix data are stored in the following order:
   Re U_{11}, Im U_{11}, U_{12}, U_{13}, U_{21}, U_{22}, U_{23}
   (We subtract 1 from indices in actual C-program.)

2. Data at the link in \( \mu = 1 \) direction appear first, then \( \mu = 2, 3 \) and 4
   follow. (We subtract 1 from indices in actual C-program.)

3. Space-time direction \( x \) moves faster, followed by \( y, z, t \) directions.
   (In actual C-program, \( x \) runs from 0 to NX-1. Other directions are
   similar.)

Note that each configuration file transferred from ILDG may have other
header information such as group name, ID, checksums and simulation pa-
rameters. The “Standard format“ does not regulate such information, be-
cause the header information is different from group to group (some groups
may not have included any information in the binary file). Instead we pro-
pose a method to supply such information. See Sec. 3.5.

3.3 C interface to the standard format

Each collaboration planning to donate configurations to the ILDG decides
the name of their own format and provides a small C library to read con-
figurations in the standard format. The library consists of a single routine
“ILDG_read_conf“:

    void ILDG_read_conf(file, NX,ix0,ix1,
                        NY,iy0,iy1,
                        NZ,iz0,iz1,
                        NT,it0,it1,
                        endian,config)
char* file;
int NX,NY,NZ,NT;
int ix0,ix1,iy0,iy1,iz0,iz1,it0,it1;
int endian;
double* config;
{
    printf("Configuration will be read from %s\n",file);

    /* open file, read config, set config, close file */
}

This routine is used to read link variables in a hypercubic region (ix0–ix1)*(iy0–iy1)*(iz0–iz1)*(it0–it1). endian=0 (little endian) and endian=1 (big endian) specify the endian of the numerical format. (Therefore each collaboration have to write an endian conversion routine.) As an example, calling this routine as

    main()
    {
        int NX=8,NY=8,NZ=8,NT=16;
        int endian=1;
        double U[8][4][4][4][2][3][2];

        ILDG_read_conf("test-file", NX,0,3,
                        NY,4,7,
                        NZ,4,7,
                        NT,0,15,
                        endian,U)
    }

reads link variables in the region (0-3)*(4-7)*(4-7)*(0-15) of the whole lattice (0-7)*(0-7)*(0-7)*(0-15) and stores them in the array U[8][4][4][4][2][3][2] in big-endian format. Note that for an array A[N] in C, the index runs from 0 to N-1.

The name of the format and the pointer to the C library will be stored as tags in the QCDML document for the configuration, such as

    <original_file_format>
    group-A-format
    </original_file_format>
    <pointer_to_convert_program>
Researchers who want to use the configuration may have to write a short program to convert the configuration in the standard format to their own favourite format, if necessary.

A possible drawback of the C interface above is that the conversion program requires huge memory of order of configuration size to twice the configuration size, e.g. \( \approx 350\text{MB} \) to \( 700\text{MB} \) for a \( 24^3 \times 48 \) lattice. This issue may cause a memory bottleneck, but cannot be avoided if one wants to convert data between two arbitrary formats. Extension of the C interface may be necessary for parallel computers constructed with nodes with small memory. However, we propose the C interface above, because it is simple and expected size of full QCD configurations public to the ILDG will be smaller than \( 32^3 \times N_t \) for forthcoming several years, which can be handled by high-end PC with memory of \( \approx 2\text{GB} \).

### 3.4 BinX description of the format

BinX is an XML schema to describe format of binary file and has been developed recently by the edikt project (e-science Data information and knowledge transfer - www.edikt.org). A software to convert one binary format to the other will be available in May, 2003. The metadata working group has decided to employ this technology to describe binary formats of QCD configurations, which enables us to convert formats without referring the standard format.

Each collaboration donating configurations to the ILDG will need to describe their own format by BinX. One can define one’s own data types, the obvious example being an SU(3) matrix, as well as translate automatically between different formats. There is also a mechanism for describing an array split across several files. This is certainly a very useful feature.

The BinX description of the standard format is

```xml
<dataset>
  <definitions>
    <TypeDef typeName="complexDouble">
      <struct>
        <IeeeDouble-64 varName="Real"/>
        <IeeeDouble-64 varName="Imaginary"/>
      </struct>
    </TypeDef>
  </definitions>
</dataset>
```
Is the order of the name “t,z,y,x,mu” compatible with the C-statement U[NT][NZ][NY][N][4]?

3.5 Distributing configurations

When we share configurations, identification is an important issue. In particular, each configuration should be accompanied with fundamental parameters such as lattice size and checksums of the configuration. We do not include such information into the standard binary format, because such information will be described in the QCDML document for the configuration.

In order to keep identification of the configuration, we encapsulate the configuration and the QCDML document into one file and distribute it via ILDG. DIME (Direct Internet Message Encapsulation) is a candidate to do this, because file size is not increased by the encapsulation and one does not have to unpack files before reading binary file.

more description is required

if we employ DIME, C-library has to seek the file to find the
origin of configuration binary, following the format of DIME. How does the BinX work for the DIME encapsulated files?