DNA Data Definitions

Dave Love, <d.love@dl.ac.uk>

Draft, 2002-01-31

Contents

1 Preamble

The DNA project has chosen to pass data between stages in home-brewed XML which has to describe the arguments and results of RPCs.[1]

This is a specification of data passed between stages of automated data collection and processing, complete and detailed enough to write the necessary interfaces. It's more-or-less influenced by scraps of examples taken from minutes of meetings, which don't always provide enough information for an implementation, but altered where it seems appropriate and somewhat extended. I've proceeded mostly bottom-up, starting with the data collection parts.

Currently, there are many unresolved issues—see the '[*Fixme*: ...]'s throughout. *This still needs work before it's really ready for comment.*

1.1 Scope

It's meant to be general enough to cover arbitrary PX beamlines and data processing programs—not just Mosflm. The programs (or suites) known to be of interest are: Mosflm, d*TREK, XDS, Denzo, HKL2000, DPS, Strategy, Rotgen, Scala (presumably), Truncate (which appears at the end of DPS's chain; presumably we go that far). I'm less sure about variations of beamlines than programs.

Of these, I've looked to some extent at the input required by Mosflm, d*TREK, DPS and XDS. I've been guided mainly by the definitions for John Campbell's RDM stuff,[2], modified as appropriate, except that I assume the image headers should be the default source of information, overridden by other data only if necessary. [*Fixme*: I wonder if what d*TREK uses—is this Madnes-derived?—is a better basis, since it looks more general. I don't have time to study it enough.]

There's no specification here of control flow, specifically abort processing or fundamental failure of an RPC. This needs to be done out-of-band—i.e. not by passing XML documents—since the system may not be in a state to be able to process them. E.g., if we were using a proper HTTP system, an operation would be aborted simply by the client chopping the request; also there would be timeouts and extra support for error conditions from the protocol.

1.2 Presentation

This document is a *literate program*. 'Code' (DTD fragments) is embedded in a convenient way in the narrative about it, and the code and documentation are processed with tools, in this case `noweb` (`<URL:http://www.cs.virginia.edu/~nr/noweb/intro.html>`). The code *chunks* appear as sections of the form:

⟨*chunk name 1a*⟩≡
```
code ...
```

[1] This is couched in terms of remote procedure calls, even if they're actually local, since they're inter-process and have been defined to be networked.

[2] `<URL:http://www.dl.ac.uk/SRS/PX/jwc_progs/rdm_chap1.html>` RDM appears mainly to follow mosflm conventions.

These can be in arbitrary order appropriate to the exposition and are assembled into the appropriate output by the tool. They can be built up incrementally, typically by to the description of that part, with raw code and by using other chunks which are expanded in-place recursively. Chunks have various sorts of cross-referencing, though that's of limited use in this case.

1.3 Naming

When choosing XML element names, I've used '_' as a word separator rather than '-', which I prefer,[3] since that's what's been done before. I've tried to use verb phrases for RPC element names to make it clear what's the RPC and what's the result.

2 Data types and units

To specify the data properly, we need type information (typically with dimension/unit information for physical quantities). While a description of XML structure in terms of a DTD can't make use of this,[4] it acts as documentation and might be usable if the DTD is transformed to something else which can express the constraints it implies.

2.1 Basic types

The types of the data described below are abbreviated for convenience as follows:

$\mathbb{R}$ Single-precision real number in a format acceptable to `scanf(3)`. (This should also cover Fortran fixed and free format, which is typically more general.)

$\mathbb{Z}$ Integer (positive, negative or zero) in decimal.

$\mathbb{N}$ Natural number (an integer ≥ 0) in decimal.

$\mathbb{N}_1$ Natural number > 0 in decimal.

UTF-8 UTF-8-encoded Unicode text[5] (ASCII-compatible).

All non-ASCII text is UTF-8. In non-ASCII, non-UTF-8 locales, non-ASCII text must be converted to UTF-8 for exchange. Rationale: this avoids passing charset information around, and we can't assume are operating in the same locale[6]. I take it for granted that non-ASCII filenames and such should be allowed, especially en Fran cais.

Enumeration Taken from a set of symbolic values, e.g. detector types. Treated individually and written as '⟨*alternative*⟩ | ⟨*alternative*⟩ ...' in the text.

[3] It's easier to type and is compatible with the SGML reference concrete syntax.

[4] Except that it can express enumerated values iff they're treated as attributes. The non-uniformity involved isn't worth it.

[5] `<URL:http://www.unicode.org>`

[6] Concerning locale processing, see `locale(7)` or general references on software localization.

The corresponding entities for the DTD are as follows. Note that XML doesn't allow `CDATA` content, i.e. entity references are expanded, so `&` and `<` need to be escaped if they occur in the data.

4a ⟨*type entities* 4a⟩≡ (17e)

```
<!ENTITY % CDATA "(#PCDATA)"> <!-- Can't use CDATA content in XML -->
<!ENTITY % real "%CDATA;">
<!ENTITY % integer "%CDATA;">
<!ENTITY % nat "%CDATA;">
<!ENTITY % nat_one "%CDATA;">
<!ENTITY % utf-8 "(#PCDATA)">
<!ENTITY % enum "%CDATA;">  <!-- arbitrary enumeration -->
```

Defines:
`CDATA`, never used.
`enum`, used in chunks 6b, 10, and 16a.
`integer`, never used.
`nat`, used in chunk 12.
`nat_one`, used in chunks 6d and 12e.
`real`, used in chunks 4b and 7–9.
`utf-8`, used in chunks 5a, 12c, 14, and 16b.

The compound types we're interested in are tuples—pairs, triplets &c of data—modelled as nested elements. A coordinate, simply expressed as two real numbers might thus have type $(\mathbb{R}, \mathbb{R})$.[7] The type descriptions are often qualified with a numerical condition, e.g. an open range $[a..b]$. Repetition of a type once or more is indicated by '$^+$'.

2.2 Units

In practice, we need to give most of the data a *type* which includes physical units, perhaps measured according to a certain convention. Thus we have element contents which may be 'typed' as follows, all basically $\mathbb{R}$:

mm A distance in mm (> 0).

Å A wavelength in Å (> 0).

° An angle in degrees. This must have a specified convention.

s A time in seconds (> 0).

The corresponding entities for the DTD are as follows (restricting ourselves to ASCII names, though XML doesn't impose that restriction).

4b ⟨*unit entities* 4b⟩≡ (17e)

```
<!ENTITY % mm "%real;">
<!ENTITY % Angstroems "%real;">
<!ENTITY % degrees "%real;">
<!ENTITY % s "%real;">
```

Defines:
`Angstroems`, used in chunk 8c.
`degrees`, used in chunks 8f, 10, 12f, and 13.
`mm`, used in chunks 6, 7, and 9a.
`s`, used in chunks 12d and 18c.
Uses `real` 4a.

[7] In formal descriptions, compound types may be written as Cartesian products of the sets involved: $\mathbb{R} \times \mathbb{R}$.

2.3 Extensibility—keywords

To allow for some (private) extensibility within this framework, we have the possibility of providing extra information through arbitrary keyworded data in various places. This is just a list of textual key/value pairs. Type: (utf-8,utf-8)$^+$.

5a ⟨*keywords* 5a⟩≡ (17e)

```
<!ELEMENT keywords (key, value)+>
<!ELEMENT key %utf-8;>
<!ELEMENT value %utf-8;>
```

Defines:
`key`, never used.
`keywords`, used in chunks 11b and 15a.
`value`, never used.
Uses `utf-8` 4a.

3 'Beamline' parameters

This describes data returned in response to a trivial RPC requesting them:

5b ⟨*RPCs* 5b⟩≡ (17e) 15a▷

```
<!ELEMENT get_beamline_parameters EMPTY>
```

Defines:
`get_beamline_parameters`, used in chunk 5.

Detector and source info are included in the returned data.

Much of this can be optional, since it can 'normally' be read from the image headers. We need to be able to specify it separately, anyhow. Some is compulsory because we need to know parameters to take initial data, e.g. the wavelength, distance and detector radius determine the maximum resolution, which will be a data-taking parameter.[8]

There's a lot more here than is always necessary, e.g. just to find the current ϕ, but it shouldn't be expensive to generate the info and ignore most of it returned from the call.

The following chunk is used in the test document (§7). Alternations for all the top-level elements[9] are added as they're defined.

5c ⟨*top-level elements* 5c⟩≡ (18a) 11c▷

```
get_beamline_parameters
```

Uses `get_beamline_parameters` 5b.

We'll have top-level examples as we go.

5d ⟨*example* 5d⟩≡ (18a) 11d▷

```
<get_beamline_parameters/> <!-- RPC -->
```

Uses `get_beamline_parameters` 5b.

[8] One could argue that it's OK to take a dummy image to extract this stuff from the header.
[9] These are the ones corresponding to individual transactions, which will appear in the `DOCTYPE` declaration of a document.

3.1 Detector

We'll have a compulsory group for detector information.

6a ⟨*beamline* 6a⟩≡ (17e) 8b▷
⟨*detector* 6b⟩

3.1.1 Detector intrinsics

This covers things more-or-less intrinsic to the detector.

Detector type We need to know the detector type since mosflm, at least, can't deduce the information purely from the image header, even for SMV v. MAR, for instance. It's not clear to me whether this is best done in terms of actual detector models or more generically by image format, if that's more-or-less self-describing (or both). For now, let's enumerate detector types which, presumably, define the image formats too. These are the ones I know about:

`Q4` ADSC Quantum-4.

`MAR-CCD`

`MAR-345`

[*Fixme*: There should be a type *generic* in the case where we need to deal with images without a self-describing header we can grok. Perhaps it's better merely to imply that if the detector type is missing.]

6b ⟨*detector* 6b⟩≡ (6a) 6c▷
```
<!ELEMENT detector_type %enum;>
```
Defines:
`detector_type`, used in chunk 11.
Uses `enum` 4a.

Radius We need this to set the distance for a given resolution. In principle, perhaps, it's derived from the detector type, but that may be a generic image type rather than a specific detector, and we don't want to have to take data to find it out, even if it is in the headers. Type: mm (> 0).

6c ⟨*detector* 6b⟩+≡ (6a) ◁6b 6d▷
```
<!ELEMENT radius %mm;>
```
Defines:
`radius`, used in chunks 8a and 11.
Uses `mm` 4b.

Overload Any pixel value greater than or equal to this value is to be treated as overloaded. Don't treat overloads if this is omitted. Type: $\mathbb{N}_1$. Optional.

6d ⟨*detector* 6b⟩+≡ (6a) ◁6c 7a▷
```
<!ELEMENT overload %nat_one;>
```
Defines:
`overload`, used in chunk 11a.
Uses `nat_one` 4a.

Gain The conversion efficiency (pixel counts per photon). Type: $\mathbb{R}$ (> 0). Default: 1. Optional, but important for statistical purposes.[10]

7a ⟨*detector* 6b⟩+≡ (6a) ◁6d 7b▷

```
<!ELEMENT gain %real;>
```

Defines:
`gain`, used in chunk 11a.
Uses `real` 4a.

Correction info We need to know whether or not images are distortion-corrected and, if not, the calibration file names.[11] [*Fixme*: Sort this out in case it's needed.]

In d*TREK (following Stanton, I think), the distortion types are: 'simple' and 'interpolation tables' (with a base file name). The mask/non-uniformity specification is 'none', 'simple mask', or 'dark image and non-uniformity'.

For spiral-scanned image plates we need the radial and tangential offsets, 'Roff' and 'Toff', both of type: mm, optional.

7b ⟨*detector* 6b⟩+≡ (6a) ◁7a 7c▷

```
<!-- Fixme: correction-info -->
<!ELEMENT roff %mm;>
<!ELEMENT toff %mm;>
```

Defines:
`roff`, used in chunk 11a.
`toff`, used in chunk 11a.
Uses `mm` 4b.

Dark image A file name for the current dark image for corrections. Type: UTF-8. Optional. [*Fixme*: Does this ever need to be specified separately from the general correction info?]

3.2 Detector geometry

This group describes the geometry of the detector (as opposed to the goniometer) and things related to the image on it.

Distance The source/detector distance. Type: mm (> 0). Compulsory for the same reason as detector size.

7c ⟨*detector* 6b⟩+≡ (6a) ◁7b 7d▷

```
<!ELEMENT distance %mm;>
```

Defines:
`distance`, used in chunk 11.
Uses `mm` 4b.

Beam centre The centre of the diffraction pattern (x, y) in detector coordinates.[12] Type: (mm,mm). [*Fixme*: In mosflm convention?]

7d ⟨*detector* 6b⟩+≡ (6a) ◁7c 8a▷

```
<!ELEMENT x %mm;>
<!ELEMENT y %mm;>
<!ELEMENT beam_centre (x, y)>
```

Defines:
`beam_centre`, used in chunk 11.
`x`, used in chunks 8a and 11d.
`y`, used in chunks 8a and 11d.
Uses `mm` 4b.

[10] Note that this probably shouldn't be independent of distortion corrections, which have a smoothing effect.
[11] Note that mosflm can't use uncorrected data.
[12] Known to be crucial for the analysis.

Beam stop shadow This defines a circular area to omit from processing in terms of a centre and radius in detector coordinates. The convention for x and y is the same as for the beam centre. Type: (mm,mm,mm). Optional.

8a ⟨*detector* 6b⟩+≡ (6a) ◁7d 11a▷

```
<!ELEMENT beam_stop (x, y, radius)>
```

Defines:
`beam_stop`, used in chunk 11a.
Uses `radius` 6c, `x` 7d, and `y` 7d.

3.3 Source

This group describes the source. [*Fixme*: I think some of the parameters here are appropriate for synchrotrons only, but do we need anything special for lab sources?]

8b ⟨*beamline* 6a⟩+≡ (17e) ◁6a 9d▷

⟨*source* 8c⟩

Wavelength Type: Å.

8c ⟨*source* 8c⟩≡ (8b) 8d▷

```
<!ELEMENT wavelength %Angstroems;>
```

Defines:
`wavelength`, used in chunks 9c and 11d.
Uses `Angstroems` 4b.

Polarization The beam polarization factor. Type: $\mathbb{R}([0.0..1.0])$. [*Fixme*: d*TREK also has a 'vector normal to the plane of polarization'.]

8d ⟨*source* 8c⟩+≡ (8b) ◁8c 8e▷

```
<!ELEMENT polarization %real;>
```

Defines:
`polarization`, used in chunk 9c.
Uses `real` 4a.

Dispersion $\delta\lambda/\lambda$. Assumed zero if omitted. Type: $\mathbb{R}$. Optional. [*Fixme*: Is this signed?]

8e ⟨*source* 8c⟩+≡ (8b) ◁8d 8f▷

```
<!ELEMENT dispersion %real;>
```

Defines:
`dispersion`, used in chunk 9c.
Uses `real` 4a.

Divergence A pair of horizontal and vertical dispersions. Type: $(^\circ,^\circ) > 0$. Assumed zero if omitted. Optional. [*Fixme*: d*TREK has 'source rotation's.]

8f ⟨*source* 8c⟩+≡ (8b) ◁8e 9a▷

```
<!ELEMENT divergence (div_x, div_y)>
<!ELEMENT div_x %degrees;>
<!ELEMENT div_y %degrees;>
```

Defines:
`divergence`, used in chunk 9c.
`div_x`, never used.
`div_y`, never used.
Uses `degrees` 4b.

Correlated dispersion [*Fixme*: ?? 'a correlated component of the wavelength dispersion' according to RDM]. Type: $\mathbb{R}$. Optional.

Slits The collimator size, determining the beam spot size via its distance and beam divergence. A single number, assuming a round or square collimator.[13] Type: mm (> 0). Optional.

[*Fixme*: Check this. It came from ESRF (Sean?). Doesn't it make more sense to specify an actual spot size? If not, don't we need another number for the collimator distance?]

9a ⟨*source* 8c⟩+≡ (8b) ◁8f 9b▷

```
<!ELEMENT slits %mm;>
```

Defines:
`slits`, used in chunk 9c.
Uses `mm` 4b.

Beam FOM This is a number providing some sort of figure of merit for the beam, presumably at least a measure of its intensity. Type: $\mathbb{R}$. Optional. [*Fixme*: Does this need more than a single number?]

9b ⟨*source* 8c⟩+≡ (8b) ◁9a 9c▷

```
<!ELEMENT beam_fom %real;>
```

Defines:
`beam_fom`, used in chunk 9c.
Uses `real` 4a.

9c ⟨*source* 8c⟩+≡ (8b) ◁9b

```
<!ELEMENT source (wavelength, polarization?, dispersion?,
                  divergence?, slits?, beam_fom?)>
```

Uses `beam_fom` 9b, `dispersion` 8e, `divergence` 8f, `polarization` 8d, `slits` 9a, and `wavelength` 8c.

3.4 Goniometer

This group describes the rotation setup. [*Fixme*: Is 'goniometer' the right term?]

[*Fixme*: This description from RDM may need modification.] In contrast:

- DPS uses: 'spindle axis' (clockwise/anticlockwise), 'parallel with' (horizontal/vertical); 'detector rotations' x (beam), y, z (spindle).
- d*TREK uses: 'detector swing'; rotations: 'rotz', 'rotx/swing', 'roty'; translations: 'transx', 'transy', 'transz/distance'.
- XDS uses: DIRECTION_OF_DETECTOR_X-AXIS, DIRECTION_OF_DETECTOR_Y-AXIS, INCIDENT_BEAM_DIRECTION.

9d ⟨*beamline* 6a⟩+≡ (17e) ◁8b 11b▷
⟨*goniometer* 10a⟩

[13] Says Miri.

Type Either simple or three-circle. Type: enumerated simple | 3-circle. Default: simple (implying rotation axis ϕ). Optional. [*Fixme*: Is there a need for 3-circle, or anything more general?]

10a ⟨*goniometer* 10a⟩≡ (9d) 10b▷

```
<!ELEMENT goniometer_type %enum;>
```

Defines:
`goniometer_type`, used in chunk 10g.
Uses `enum` 4a.

Orientation The direction (perpendicular to the beam) of the axis around which the crystal is to be rotated during data collection. It is defined looking along the beam direction as 'right', 'left', 'up' or 'down'. Type: enumerated right | left | up | down. Default: right. Optional.

10b ⟨*goniometer* 10a⟩+≡ (9d) ◁10a 10c▷

```
<!ELEMENT goniometer_orientation %enum;>
```

Defines:
`goniometer_orientation`, used in chunk 10g.
Uses `enum` 4a.

Rotation axis Axis around which three-circle goniometer rotates. Type: enumerated $\phi \,|\, \omega$. Default: ω. Optional.

10c ⟨*goniometer* 10a⟩+≡ (9d) ◁10b 10d▷

```
<!ELEMENT rotation_axis %enum;>
```

Defines:
`rotation_axis`, used in chunk 10g.
Uses `enum` 4a.

ϕ Current ϕ rotation angle. Compulsory, in case we need to know it without taking data. Type: ° ([−720.0..720.0]).

10d ⟨*goniometer* 10a⟩+≡ (9d) ◁10c 10e▷

```
<!ELEMENT phi %degrees;>
```

Defines:
`phi`, used in chunks 10g, 11d, and 16e.
Uses `degrees` 4b.

κ Current κ setting. Ignored for type 'simple'. Type: ° ([−720.0..720.0]). Default: 0. Optional.

10e ⟨*goniometer* 10a⟩+≡ (9d) ◁10d 10f▷

```
<!ELEMENT kappa %degrees;>
```

Defines:
`kappa`, used in chunk 10g.
Uses `degrees` 4b.

ω Current ω setting. Ignored for type 'simple'. Type: ° ([−720.0..720.0]). Default: 0. Optional.

10f ⟨*goniometer* 10a⟩+≡ (9d) ◁10e 10g▷

```
<!ELEMENT omega %degrees;>
```

Defines:
`omega`, used in chunk 10g.
Uses `degrees` 4b.

10g ⟨*goniometer* 10a⟩+≡ (9d) ◁10f

```
<!ELEMENT goniometer (goniometer_type?, goniometer_orientation?,
    rotation_axis?, phi, kappa?, omega?)>
```

Defines:
`goniometer`, used in chunk 11.
Uses `goniometer_orientation` 10b, `goniometer_type` 10a, `kappa` 10e, `omega` 10f, `phi` 10d, and `rotation_axis` 10c.

3.5 Queries

[*Fixme*: What exactly is 2θ, and does it need to be specified?]

3.6 Detector element

The entire detector group is:

11a ⟨*detector* 6b⟩+≡ (6a) ◁8a

```
<!ELEMENT detector (detector_type, radius, distance, beam_centre,
                    overload?, gain?, roff?, toff?, beam_stop?)>
```

Defines:
`detector`, used in chunk 11.
Uses `beam_centre` 7d, `beam_stop` 8a, `detector_type` 6b, `distance` 7c, `gain` 7a, `overload` 6d, `radius` 6c, `roff` 7b, and `toff` 7b.

3.7 Beamline group

11b ⟨*beamline* 6a⟩+≡ (17e) ◁9d

```
<!ELEMENT beamline (source, detector, goniometer, keywords?)>
```

Defines:
`beamline`, used in chunk 11.
Uses `detector` 11a, `goniometer` 10g, and `keywords` 5a.

11c ⟨*top-level elements* 5c⟩+≡ (18a) ◁5c 15b▷

```
|beamline
```

Uses `beamline` 11b.

11d ⟨*example* 5d⟩+≡ (18a) ◁5d 15c▷

```
<beamline> <!-- result -->
  <source><wavelength>0.97</wavelength></source>
  <detector>
    <detector_type>Q4</detector_type>
    <radius>95.0</radius>
    <distance>120</distance>
    <beam_centre><x>91.2</x><y>93.4</y></beam_centre>
  </detector>
  <goniometer><phi>0.</phi></goniometer>
</beamline>
```

Uses `beam_centre` 7d, `beamline` 11b, `detector` 11a, `detector_type` 6b, `distance` 7c, `goniometer` 10g, `phi` 10d, `radius` 6c, `wavelength` 8c, `x` 7d, and `y` 7d.

4 Data collection RPC

This describes an RPC on the data acquisition system to collect a range of images. [*Fixme*: There seem to be different possible data collection modes, e.g. *dose* mode. Do we need to cover those?]

[*Fixme*: This doesn't consider setting beamline parameters. Should that be considered as part of the data collection—which seems appropriate—or done separately? The possibilities for setting things are at least: wavelength, distance, slits.]

4.1 Data collection parameters

Scan id [*Fixme*: Can it be 0?]

12a ⟨*data acq* 12a⟩≡ (17e) 12b▷

```
<!ELEMENT id %nat;> <!-- fixme -->
```

Defines:
`id`, used in chunks 15 and 16.
Uses `nat` 4a.

Start number This is the number at which to start labelling images in the scan. Type: $\mathbb{N}$. [*Fixme*: Should this be $\mathbb{N}_1$?] Optional, default 1.

12b ⟨*data acq* 12a⟩+≡ (17e) ◁12a 12c▷

```
<!ELEMENT start_number %nat;> <!-- fixme -->
```

Defines:
`start_number`, used in chunk 15a.
Uses `nat` 4a.

Project id [*Fixme*: I'm not sure what this is]

12c ⟨*data acq* 12a⟩+≡ (17e) ◁12b 12d▷

```
<!ELEMENT project_id %utf-8;> <!-- fixme -->
```

Uses `utf-8` 4a.

Exposure time The collection time per image. Type: s (> 0). [*Fixme*: Optional in dose mode?]

12d ⟨*data acq* 12a⟩+≡ (17e) ◁12c 12e▷

```
<!ELEMENT exposure_time %s;>
```

Defines:
`exposure_time`, used in chunks 15 and 17.
Uses `s` 4b.

Number of passes This is a number of passes to make through the oscillation angle to account for beam fluctuations etc. Type: $\mathbb{N}_1$.

12e ⟨*data acq* 12a⟩+≡ (17e) ◁12d 12f▷

```
<!ELEMENT passes %nat_one;>
```

Defines:
`passes`, used in chunks 15 and 17.
Uses `nat_one` 4a.

Oscillation angle The oscillation angle ($\delta\phi$) for each image in a scan. Type: $^\circ$ (> 0).

12f ⟨*data acq* 12a⟩+≡ (17e) ◁12e 13▷

```
<!ELEMENT oscillation_angle %degrees;>
```

Defines:
`oscillation_angle`, used in chunks 15 and 17.
Uses `degrees` 4b.

Start and end ϕ The start and end angles for a scan. In principle,

$$\phi_{\text{end}} = \phi_{\text{start}} \pm n(\delta\phi) \ ,$$

where n is the number of images in the scan, and the '$\pm$' accounts for taking the modulus of the oscillation angle. In practice, we have to be careful, because rounding errors could lead to an off-by-one error in the number of points if n is computed from the range and increment naïvely. I assume that the range should be taken as a real limit (since there may be physical constraints). In that case n should be computed as

$$\lfloor (|\phi_{\text{end}} - \phi_{\text{start}}| + \Delta)/\delta\phi \rfloor \ ,$$

where Δ is a fudge factor depending on the floating point precision.

Type: °. [*Fixme*: Clarify how the angles are related to the goniometer parameters.]

[*Fixme*: Is there never an overlap in ϕ between frames?]

13 ⟨*data acq* 12a⟩+≡ (17e) ◁12f 14▷

```
<!ELEMENT phi_start %degrees;>
<!ELEMENT phi_end %degrees;>
```

Defines:
`phi_end`, used in chunk 15.
`phi_start`, used in chunk 15.
Uses `degrees` 4b.

Sequence increment [*Fixme*: Do we also want the sequence increment that d*TREK allows?]

4.2 Image filename template

This has previously been put in a group broken into a directory and a filename template part. Specifying the directory is redundant—the system can keep track of a separate directory if necessary.

The template is in the form of a fully-qualified UTF-8 POSIX pathname[14] (not intended to name an existing file). If it becomes necessary to generalize the description of image locations, say to URLs, a pseudo root directory could be defined to cover that, e.g. `/URL:http://...`.

It is of the form

⟨*directory*⟩⟨*file base*⟩⟨*replacement*⟩⟨*file tail*⟩

where:

⟨***directory***⟩ is a directory name (including the trailing slash) or null;

⟨***file base***⟩ is the start of a file name template common through a run. It may not contain `#` characters.

It may contain a run of `?` characters to be replaced by the scan id (4.3). Rationale: the template is constant in a data collection specification covering several scans;

⟨***replacement***⟩ is a run of `#` characters to be replaced by a formatted integer run number with leading zeroes to pad the field to the number of characters represented by the `#`s, and

⟨***file tail***⟩ is the end of the file name common to a run (normally the extension, including the dot). It may not contain `#` or `?` characters.

When ⟨*replacement*⟩ and any `?` field in ⟨*file base*⟩ are filled in, the result is a path name to which to write the current image in the scan sequence. It's unspecified whether or not existing files may be overwritten, since this is presumably a function of a specific data acquisition setup.

14 ⟨*data acq* 12a⟩+≡ (17e) ◁13

```
<!ELEMENT template %utf-8;>
```

Defines:
`template`, used in chunks 15 and 17.
Uses `utf-8` 4a.

[14] I.e. the directory separators are '/', even on DozeN'T et al, should that ever be appropriate.

4.3 Whole RPC

The data collection has been defined to be split into multiple *scans*.[15] [*Fixme*: I don't remember the rationale for this.]

We'll add optional keywords to the elements defined above in case there's any other information that needs passing.

15a ⟨*RPCs* 5b⟩+≡ (17e) ◁5b 16f▷

```
<!ELEMENT scan (id, exposure_time, oscillation_angle, phi_start,
                phi_end, passes, template, start_number?, keywords?)>
<!ELEMENT collect (scan)+>
```

Defines:
`collect`, used in chunk 15.
`scan`, used in chunk 15c.
Uses `exposure_time` 12d, `id` 12a, `keywords` 5a, `oscillation_angle` 12f, `passes` 12e, `phi_end` 13, `phi_start` 13, `start_number` 12b, and `template` 14.

15b ⟨*top-level elements* 5c⟩+≡ (18a) ◁11c 16d▷

```
|collect
```

Uses `collect` 15a.

15c ⟨*example* 5d⟩+≡ (18a) ◁11d 16e▷

```
<collect> <!-- RPC (2 images for autoindexing) -->
  <scan>
    <id>1</id>
    <exposure_time>2.0</exposure_time>
    <oscillation_angle>1.0</oscillation_angle>
    <phi_start>0</phi_start>
    <phi_end>1</phi_end>
    <passes>1</passes>
    <template>index0_#.img</template>
  </scan>
  <scan>
    <id>2</id>
    <exposure_time>2.0</exposure_time>
    <oscillation_angle>1.0</oscillation_angle>
    <phi_start>90</phi_start>
    <phi_end>91</phi_end>
    <passes>1</passes>
    <template>index0_#.img</template>
  </scan>
</collect>
```

Uses `collect` 15a, `exposure_time` 12d, `id` 12a, `oscillation_angle` 12f, `passes` 12e, `phi_end` 13, `phi_start` 13, `scan` 15a, and `template` 14.

[15] Previously referred to as a 'data set'. I've used the d*TREK terminology, which is clearer and consistent with SR data-taking generally.

4.4 Results

A result seems to be expected for each scan in a collection, comprising the scan id, a status and an error message. [*Fixme*: I'd have thought feedback would be required after every image in a scan, though this is obtainable out-of-band (at least with the ADSC system).]

Status I guess this should be of type: enum OK | error. [*Fixme*: Is there any reason to require this? The presence or absence of an error message can convey the information.]

16a ⟨*results* 16a⟩≡ (17e) 16b▷

```
<!ELEMENT status %enum;>
```

Defines:
`status`, used in chunk 16.
Uses `enum` 4a.

Message The error message is text for human-readable display. Type: utf-8. Optional, since it's irrelevant if status=OK.

16b ⟨*results* 16a⟩+≡ (17e) ◁16a 16c▷

```
<!ELEMENT error_message %utf-8;>
```

Defines:
`error_message`, used in chunk 16.
Uses `utf-8` 4a.

16c ⟨*results* 16a⟩+≡ (17e) ◁16b 17b▷

```
<!ELEMENT collect_result (id, status, error_message?)>
```

Defines:
`collect_result`, used in chunk 16.
Uses `error_message` 16b, `id` 12a, and `status` 16a.

16d ⟨*top-level elements* 5c⟩+≡ (18a) ◁15b 16g▷

```
|collect_result
```

Uses `collect_result` 16c.

16e ⟨*example* 5d⟩+≡ (18a) ◁15c 17a▷

```
<collect_result> <!-- result -->
<id>1</id><status>error</status>
<error_message>phi out of range</error_message>
</collect_result>
```

Uses `collect_result` 16c, `error_message` 16b, `id` 12a, `phi` 10d, and `status` 16a.

5 Data Collection Query

There seems to be a short-term need to extract information from existing data acquisition interfaces comprising: exposure time, oscillation angle, number of passes, project id and image name template. These are defined as above.

16f ⟨*RPCs* 5b⟩+≡ (17e) ◁15a

```
<!ELEMENT get_dc_parameters EMPTY>
```

Defines:
`get_dc_parameters`, used in chunks 16g and 17a.

16g ⟨*top-level elements* 5c⟩+≡ (18a) ◁16d 17c▷

```
|get_dc_parameters
```

Uses `get_dc_parameters` 16f.

17a ⟨*example* 5d⟩+≡ (18a) ◁16e 17d▷

```
<get_dc_parameters/> <!-- RPC -->
```

Uses `get_dc_parameters` 16f.

The result is as follows. [*Fixme*: Should any/all of these be optional in this context?]

17b ⟨*results* 16a⟩+≡ (17e) ◁16c

```
<!ELEMENT dc_parameters (exposure_time, oscillation_angle, passes,
                         project_id, template)>
```

Defines:
`dc_parameters`, used in chunk 17.
Uses `exposure_time` 12d, `oscillation_angle` 12f, `passes` 12e, and `template` 14.

17c ⟨*top-level elements* 5c⟩+≡ (18a) ◁16g

```
|dc_parameters
```

Uses `dc_parameters` 17b.

17d ⟨*example* 5d⟩+≡ (18a) ◁17a

```
<dc_parameters> <!-- result -->
<exposure_time>2</exposure_time>
  <oscillation_angle>1.0</oscillation_angle>
  <passes>1</passes>
  <project_id>?</project_id> <!-- fixme -->
  <template>/data1/dl/foo_?_###.img</template>
</dc_parameters>
```

Uses `dc_parameters` 17b, `exposure_time` 12d, `oscillation_angle` 12f, `passes` 12e, and `template` 14.

6 Assembled DTD

17e ⟨*data-def.dtd* 17e⟩≡

```
<?xml version="1.0"?>
⟨type entities 4a⟩
⟨unit entities 4b⟩
⟨keywords 5a⟩
⟨beamline 6a⟩
⟨data acq 12a⟩
<!-- RPCs -->
⟨RPCs 5b⟩
<!-- RPC results -->
⟨results 16a⟩
```

7 XML test/example &c

Here's an example/test document to check the DTD and examples. We'll fiddle the DTD to allow including various top-level elements in it.

18a ⟨*example.xml* 18a⟩≡

```
<?xml version="1.0"?>
<!DOCTYPE test SYSTEM "data-def.dtd"
  [<!ELEMENT test (⟨top-level elements 5c⟩)+>]>
<test>
⟨example 5d⟩
</test>
```

It's checked with the following script, using the parser from the OpenSP tools (`<URL:http://openjade.sourceforge.net/>`). The `SGML_CATALOG_FILES` value is system-dependent, obviously. Pass it option `-s` to suppress parsed output.

18b ⟨*xml-nsgmls* 18b⟩≡

```
#! /bin/sh
SP_CHARSET_FIXED=YES SP_ENCODING=XML \
  SGML_CATALOG_FILES=/usr/share/sgml/declaration/xml.soc \
  onsgmls -wxml "$@"
```

The Makefile to generate the files from the source is the following. `noweave` generates documentation from the `noweb` source, and `notangle` generates (generalized) 'code'.

18c ⟨*Makefile* 18c⟩≡

```
%.tex: %.nw; noweave -delay -index $<  | cpif $@

%.dvi: %.tex;
        latex $<
        grep 'LaTeX Warning:.*Rerun' $*.log >/dev/null && latex $< || true
        grep 'LaTeX Warning:.*Rerun' $*.log >/dev/null && latex $< || true

%.pdf: %.tex;
        pdflatex $<
        grep 'LaTeX Warning:.*Rerun' $*.log >/dev/null && pdflatex $< || true
        grep 'LaTeX Warning:.*Rerun' $*.log >/dev/null && pdflatex $< || true

%.html: %.nw; noweave -html -filter l2h -index $< > $@

%.ps: %.dvi; dvips -o $@ $<

all: data-def.tex data-def.dtd example.xml

data-def.dtd: data-def.nw
        notangle -Rdata-def.dtd $< | cpif $@

example.xml: data-def.nw
        notangle -Rexample.xml $< | cpif $@

check: example.xml data-def.dtd
        xml-nsgmls -s $<

Makefile: data-def.nw; notangle -t8 -RMakefile $< | cpif $@
```

Uses `s` 4b.

8 Indexes

Indexing is by sub-page number, i.e. '2b' is the second chunk on page 2.

8.1 Code Chunks

8.2 Identifier definitions and usage

Definition points are underlined.