Low-Level Data Containers

Morgan C White

Los Alamos National Laboratory

Submitted to the December 2013

Meeting of the WPEC Subgroup 38

Abstract

The successor to the ENDF-102 data format has been discussed many times. The NEA WPEC SG38 has been established to make the latest attempt. This set of viewgraphs contains proposed definitions for low-level primitives to be used by the new formats. Hopefully, lessons learned from 50 years of experience with ENDF have guided the suggestions made in the current proposal. These are not final and debate during the upcoming meeting is required to deliver an initial draft specification of the requirements, documentation and coding for low-level primitives to be used by the new nuclear data format.

Make it easy for the user to obtain, understand and use nuclear data.

A Set of Overarching Goals

- **Any piece of data shall be retrievable by its unique URL**
  - Uniform Resource Locator, a string used to identify a data node
- **Its URL should provide clear meaning as to what data**
- **A data type shall conform to its defined API**
  - Application Programing Interface, strictly defined uses of the data
- **Any data that conforms to an API belongs to its family**
  - Enable extensions by design. That is, if it looks like a duck and quacks like a duck, it's a duck.

If a user understands their question, they should recognize the data needed to answer it.

***May, Should and Shall – Syntax defined by http://www.ietf.org/rfc/rfc2119.txt**

Low Level Data Containers Requirements

- **Data shall be stored within a hierarchal structure**
  - Each node should be named to convey the kind of data
  - Node names should use a restricted character set
    - Proposed [A-Za-z]{1}[A-Za-z0-9_-]{0,*}
- **Nodes may contain only ‘text’ or ‘numeric’ data**
  - Only two fundamental low-level types
  - Numeric data are given as an ND-array of IEEE754 doubles
    - A numeric node shall have meta-data defining the ‘shape’, e.g. ‘(175,175,5)’
  - Nodes may be links to one or more internal or external nodes
  - Nodes may be empty
    - ‘0’ is a data value. But if you look at 0 things, you do not see anything. Likewise, an empty node has intrinsic meaning.
- **Metadata should be used for clarification of the data**
  - Metadata should be about the data, not data themselves

***We will come back to ‘text’**

‘Higher’ Level Data Containers Requirements

Goal: Distill the essence of each type of data.

- **Build more complex data types from lower-level types using a hierarchal structure; but do not use more complex structures than necessary.**
  - By themselves, our two basic containers are not enough, but…
- **Do not introduce new data types where existing structures already capture this essence.**
  - Essence is often resolved by observing their use
  - If two things can be ‘used’ identically, they are the same
    - … a rose by any other name is still a rose…

Low Level Data Containers
API Requirements

- **Shall provide queries to discover and traverse the nodal structure and its meta-data**
- **Shall provide queries to retrieve meta-data attributes as strings**
- **Shall provide a query to return the shape of the numeric data, if a node contains such data**
- **Shall provide a query to return the 'data', either text or numeric, contained by a node**
- **Shall develop verification tests based on meeting specification defined by API**

What Is A Low-Level Container?

- **To decide *how* to store these data, one must first understand *what* needs to be stored**
- **To that end, we must first deconstruct the data as used in the current ENDF format**
  - That is, the first task is to store the data we have, additions should only come later

- **Are "low-level" containers cross-sections and emission distributions? x-y lines and matrices? or something even more basic?**

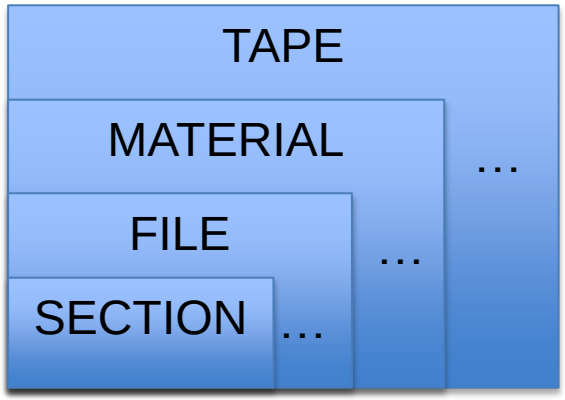
ENDF-102
Defines Five Basic (Low-Level) Records

Tremendous flexibility enabled by a small set of simple records…

- **TEXT Record**
  - [MAT, MF, MT/ HL] TEXT
  - READ(LIB,10)HL,MAT,MF,MT,NS
  - 10 FORMAT(A66,I4,I2,I3,I5)
- **CONT Record**
  - [MAT,MF,MT/C1,C2,L1,L2,N1,N2]CONT
  - READ(LIB,10)C1,C2,L1,L2,N1,N2,MAT,MF,MT,NS
  - 10 FORMAT(2E11.0,4I11,I4,I2,I3,I5)
- **LIST Record**
  - [MAT,MF,MT/ C1, C2, L1, L2, NPL, N2/ Bn] LIST
  - READ(LIB,10)C1,C2,L1,L2,NPL,N2,MAT,MF,MT,NS
  - 10 FORMAT(2E11.0,4I11,I4,I2,I3,I5)
  - READ(LIB,20)(B(N),N=1,NPL)
  - 20 FORMAT(6E11.0)
- **TAB1 Record**
  - [MAT,MF,MT/ C1, C2, L1, L2, NR, NP/xint/y(x)]TAB1
  - READ(LIB,10)C1,C2,L1,L2,NR,NP,MAT,MF,MT,NS
  - 10 FORMAT(2E11.0,4I11,I4,I2,I3,I5)
  - READ(LIB,20)(NBT(N),INT(N),N=1,NR) xint
  - 20 FORMAT(6I11)
  - READ(LIB,30)(X(N),Y(N),N=1,NP) y(x)
  - 30 FORMAT(6E11.0)
- **TAB2 Record**
  - [MAT,MF,MT/ C1, C2, L1, L2, NR, NZ/ Zint]TAB2
  - READ(LIB,10)C1,C2,L1,L2,NR,NP,MAT,MF,MT,NS
  - 10 FORMAT(2E11.0,4I11,I4,I2,I3,I5)
  - READ(LIB,20)(NBT(N),INT(N),N=1,NR) xint
  - 20 FORMAT(6I11)

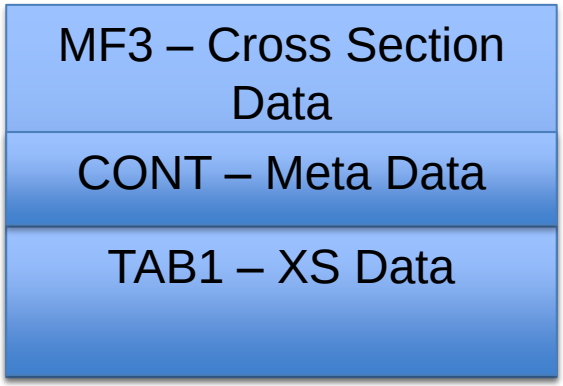
But 80-column punch card format drives unused values within records and yet we still find values that are “overloaded” (contain multiple sub-values).

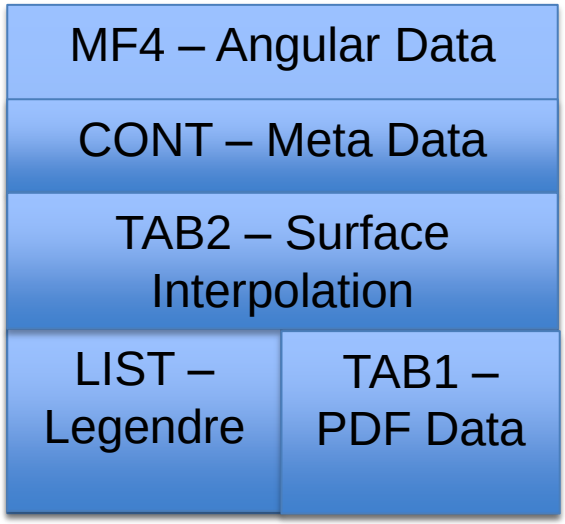
But Higher Level Structures Use These Records In An Inconsistent Manner

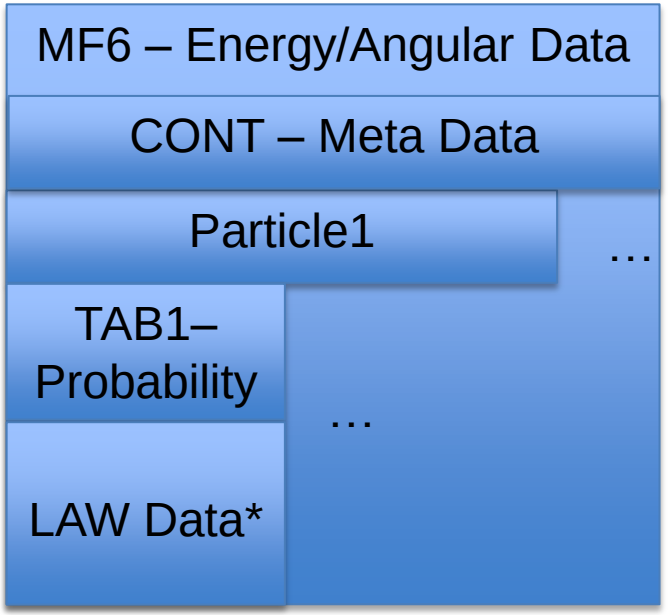


Sequential nature of records drives context driven structure that *requires external knowledge* to read

MF5 is similar to MF6 but constrained to a single emission particle







Distribution data gets steadily more complex.

*LAW Data include complex structures with tabular and function forms

Details of the MF4, MF5, MF6 Scattering Distributions

It is easy to lose track of the 'big picture' when one has to trudge through all the weeds.

MF4 Angular Distribution

```
1 x CONT
1 x LIST
IF LVT == 0
        NONE
ELIF LVT == 1
        1 x TAB2
        NE x LIST
ELIF LVT == 2
        1 x TAB2
        NE x TAB1
ELIF LVT == 3
        1 x TAB2
        NE x LIST
        1 x TAB2
        NE2 x TAB1
```

LVT = CONT.1 L2
NE = TAB2.1 N2
NE2 = TAB2.2 N2

MF5 Energy Distribution

```
1 x CONT
NK x SUBSECTIONS
   1 x TAB1
   IF LF == 1
      1 x TAB2
      NE x TAB1
   ELIF LF == 5 or LF == 11
      2 x TAB1
   ELIF LF == 7 or LF == 9
         or LF == 12
      1 x TAB1
```

NK = CONT.1 L2
LF = TAB1.1 L2
NE = TAB2.1 N2

MF6 Energy/Angle Distribution

```
1 x CONT
NK x SUBSECTIONS
   1 x TAB1
   IF LAW == 0 or LAW == 3 or LAW == 4
      NONE
   ELIF LAW == 1 or LAW == 2 or LAW == 5
      1 x TAB2
      NE x LIST
   ELIF LAW == 6
      1 x CONT
   ELIF LAW == 7
      1 x TAB2
      NE x SUBSECTION
         1 x TAB2
         NMU x TAB1
```

NK = CONT.1 L2
LAW = TAB1.1 L2
NE = TAB2.1 N2
NMU = TAB2.2 N2

ENDF-102 Cannot Be Read Without External Knowledge Of Its Structure

- **Each of the five basic records can be easily read**
  - In fact, attempting to interpret each sequential record on a tape by 'reading' it as a [TAB1, TAB2, LIST, CONT, or TEXT] record – in that order – and taking the first valid product will result in a list of ENDF records that comprise the 'tape'

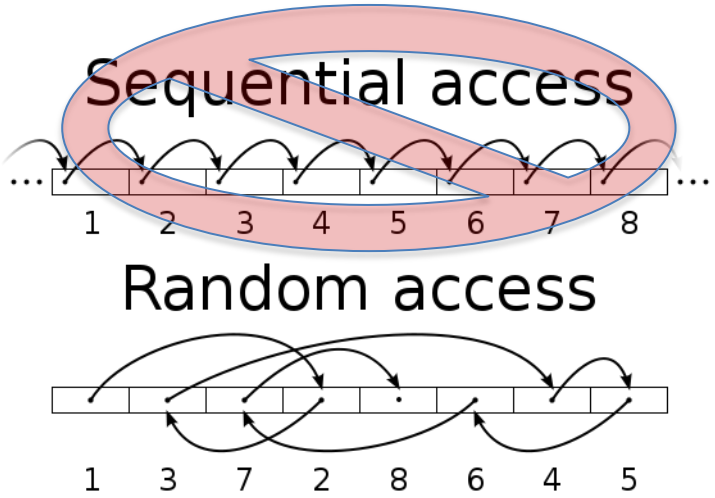
    BUT it will likely be incorrect!!

- **Sequential ENDF records cannot be recognized solely on their own merits**

  That is what is wrong with punch cards.

- **Hierarchal nodes stand on their own –and– provide additional meaning though structure.**

Reading Data Should Not Require External Knowledge Instead Having Meaning Through Self-Describing Structures That Can Be Accessed Directly



- **Should not require prior knowledge to read current record nor know what it is.**
- **Sequential access for "huge" files wastes time (minutes to hours) for small queries.**

- **Requires a "Uniform Resource Locator" (URL) that uniquely identifies each "chuck" of data.**
- **Enables explicit links helping remove redundancies and enabling better checking.**

Requirement: Basic data records will be stored in unique nodes with direct access by URL. Basic records URL text shall be used to convey the meaning of the values stored.

Generic Containers Seen Within ENDF

- **Text – Documentation such as code inputs, comparisons, notes, warnings, references, … we will always need a generic text container**
  - Requirement: All text will use the UTF-8 Encoding. A single basic record will be provided for storing text.
  - The closer the text is to the data to which it pertains, the more likely it is to have the desired impact.
  - The actual text may use plain text, html, latex, …
    - I believe we will struggle with text and should try anything
- **Numbers – with their meta-data**
  - …

Potential Basic Numeric Records

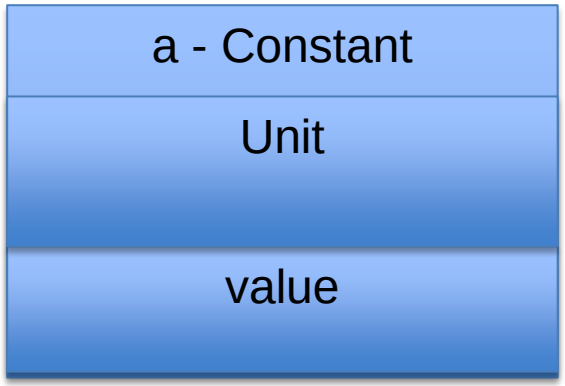
- **There are a number of potential number types…**
  - constants – ubiquitous throughout data files
  - x-y pairs – cross-sections, PDFs, parameters…
  - n-tuples – resonance parameters
  - matrices – covariance or scattering data
- **However, these are all specific cases of a generalized array**
  - A constant is an array with shape (1); x-y pairs (2,n); n-tuples (n-values,m-rows); matrices {(n,m) (n,m,p) …}

There only needs to be one generic numeric record.

Numerical Data Records

- **Requirement: All numeric data shall be stored in a general purpose array of IEEE754 'doubles'. Any restrictions beyond dimensionality may be imposed by the higher level data structure of which the array is a part. The node containing such numeric data shall include the attribute 'shape' that contains values defining the shape of the ndarray, e.g. 'shape=(175,175,5)'. Nodes that lack the 'shape' attribute are implicitly 'text' containers.**

Constants

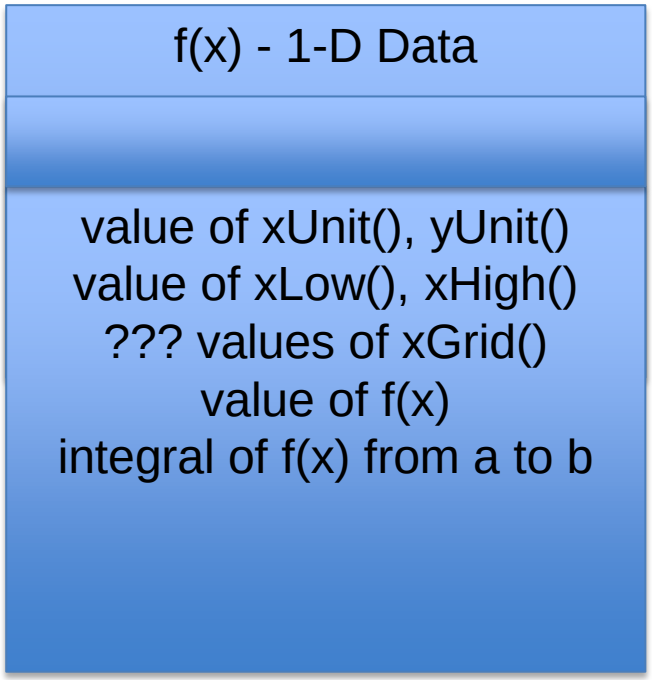


- **The simplest of numeric values**
- **Has an associated scale - unit**
- **In use, is simply a value**

Unlike ENDF102 constants, meaning is given by node and data is directly retrievable using URL.

No limitations on structure allow for as few or as many constants as are needed without resorting to either overloaded or redundant values.

Lines
Abstract Class



- **The core essence of cross sections, variable multiplicity, nubar, probability distributions, and many other higher-level data forms**
- **Three essential functions:**
  - Return the xLow, xHigh range of validity
  - Return a value given the dependent variable
  - Return the integral from a to b
- **Dependent and independent variables have associated scales**
- **It is often desired, though rarely if ever required, to know the xGrid**
  - Thus, maybe, one wants a function to return these values, if present

Structure Versus Form

- **The key is to focus on the structure**
  - Structure is the arrangement of the elements
  - So far no examples have been given
  - In the end, this should be implemented in many forms

- **Form is the concrete implementation**
  - XML provides a human and machine readable
    - This advantage is enough to make it valuable as the primary form
    - But it is too slow being a sequential
  - HDF5 provides a high-performance machine form

xy1D
XML Example

```
<xy1D
    interpolation='lin,lin',
    xUnit='keV',
    shape='(2,3)'>
  0.1    1.1
  1.2    1.1
  5.8    1.5
</xy1D>
```

Enumerated INT types

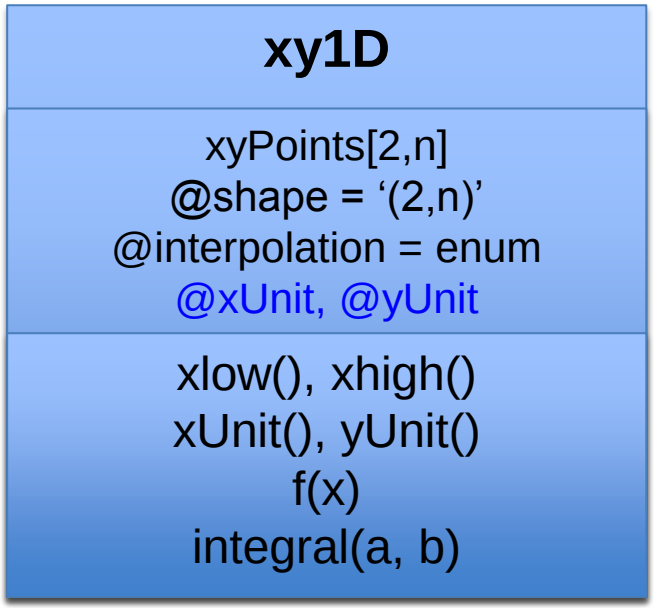
Value for 'xUnits' and 'yUnits' are inherited from container but may be overridden at this level.

Explicitly marks data node and provides shape, i.e. (2, n xy-pairs)

xy1D[0,:] – slice with x-values [x0,x1,x2]
xy1D[1,:] – slice with y-values [y0,y1,y2]
xy1D[0,1], xy1D[1,1] – values x1,y1

Using 'c' notation with arrays starting at zero [0…]

xy1D
UML Class



- **The shape and interpolation attributes are required**
- **xUnit and yUnit are optional attributes whose values are inherited from the higher level data container, if absent**

- **'interpolation' – TEXT value**
  - ENDF INT parameter as defined by ENDF-102 Section 0.5.2.1
    - INT equal 1 – 'leftStair'
    - INT equal 2 – 'lin,lin'
    - INT equal 3 – 'lin,log'
    - INT equal 4 – 'log,lin'
    - INT equal 5 – 'log,log'
    - INT equal 6 – 'cpInt'
- **xyPoints[2,n] array doubles**
  - x values must be monotonic
  - Restrict values based on INT
    - E.g. no negative values for logarithmic interpolations

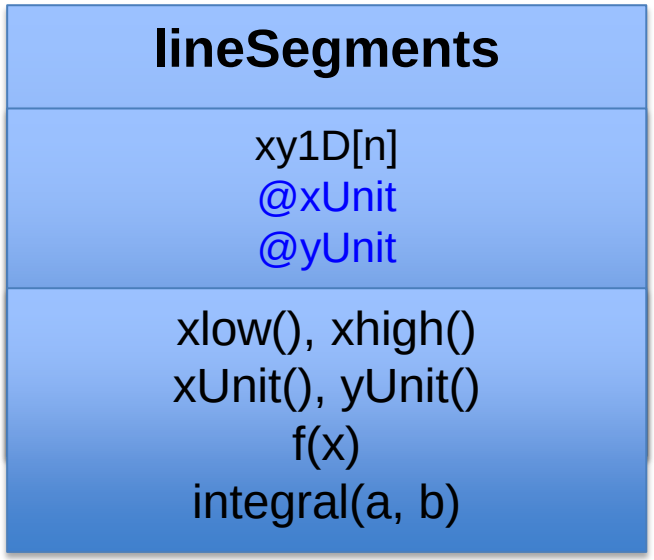
lineSegments XML Example

```
<lineSegments
    yUnit='millibarns'>
 <xy1D> [region 1] </xy1D>
 <xy1D> [region 2] </xy1D>
 ...
 <xy1D> [region n] </xy1D>
</lineSegments>
```

Value for 'xUnits' and 'yUnits' are inherited from container but may be overridden at this level.

n xy1D sub-containers
n shall be 2 or more

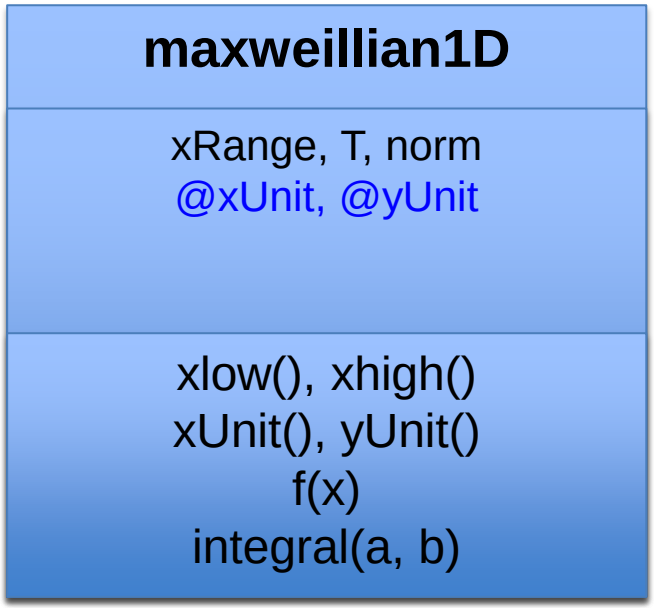
lineSegments UML Class



- **'lineSegments' used to form line from set of xy1D**
  - Adjacent lineSegment objects must contain identical end points
  - List of lineSegment objects must be given in ascending order

But this is not the right generic structure.

maxweillian1D
UML Class



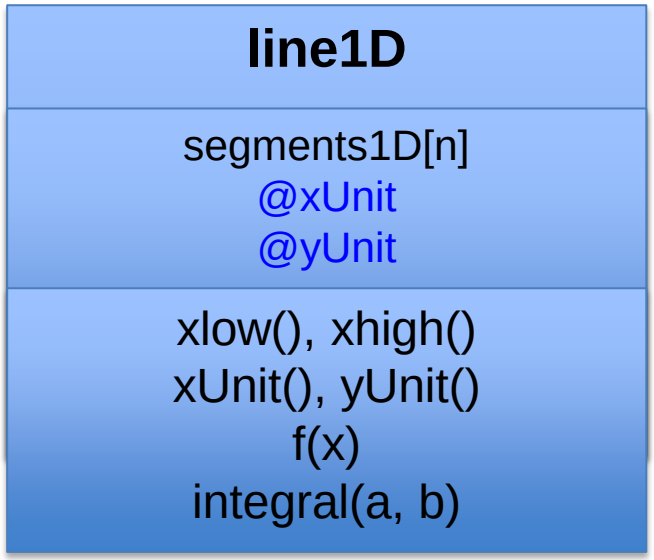
Maxweillian, Watt, Madland-Nix, Evaporation, Polynomial, Legendre, and other functional forms that 'are' simple lines can all be used interchangeably with xySegment. If it looks like a duck and quacks like a duck, it's a duck.

- **Shall perform all the same functions as any generic xy1D**
- **Example of how any concrete class implementing essential functions of abstract class may be used interchangeably**

maxweillian1D
XML Example

```
<maxweillian1D xUnit='MeV'>
 <xrange shape='(2)'> 0.0001  20. </xrange>
 <temperature shape='(1)'> 1.23 </temperature>
 <normalization shape='(1)'> 1. </normalization>
</maxweillian1D>
```

line1D
UML Class



- **The ‘line1D’ is now a generic container that stitches together any suite of ‘segments’ that themselves implement the line1D essential functions**

crossSection XML Example

```
<crossSection>
 <line1D> <axes>
  <axis index="0" label=”Energy" unit="eV” />
  <axis index="1" label=“Cross Section" unit="b"/></axes>
  <xy1D interpolation=‘log,log’ shape=‘(2,2)’>
   1e-5   10000.
   100.   1.       </xy1D>
  <maxweillian1D xUnit=‘MeV’>
   <xrange shape=‘(2)’> 0.0001  20. </xrange>
   <temperature shape=‘(1)’> 1.23 </temperature>
   <normalization shape=‘(1)’> 1. </normalization>
  </maxweillian1D> </line1D>
</crossSection>
```

crossSection
XML Example With RR Link

```
<crossSection>
 <line1D> <axes>
  <axis index="0" label=”Energy" unit="eV” />
  <axis index="1" label=“Cross Section" unit="b"/></axes>
  <xy1D interpolation=‘log,log’ shape=‘(2,2)’>
   1e-5   10000.
   1e-2   1.
  </xy1D>
  <rrLink xlink=‘URL/path/to/RR/component’ /> ADD xrange?
  <xy1D interpolation=‘lin,lin’ shape=‘(2,1027)’>
  ...
  </xy1D> </line1D>
</crossSection>
```

resonanceParameters XML Example

```
<resonanceRegion xUnit='keV'>
 <material>
  <component>
   <z shape='(1)'> 11 </z> <a shape='(1)'> 22 </a>
   <awr shape='(1)'> 21.8055 </awr>
   <abundance shape='(1)'> 1.0 </abundance>
   <resonanceParameters shape='(3,6)'>
    0.1   2. 3. 4. 5. 6.
    1.2   3. 4. 5. 6. 7.
    5.8   4. 5. 6. 7. 8.
   </resonanceParameters>
  </component>
 </material>
</resonanceRegion>
```

Next Steps

- **Examine existing ENDF102 and GND structures**
  - looking for commonalities
  - generalize families, and
  - capture key use cases into formal documentation
- **Must do this with use cases in mind**
  - Lends itself to concurrent analysis of API and QA
- **Expect refinements to continue as we gain experience and better recognize patterns**