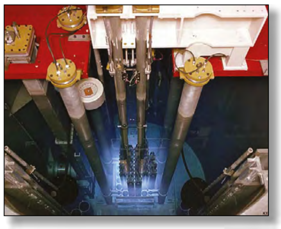
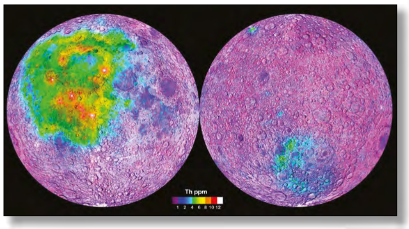
Nuclear Science
2023

Specifications for the Generalised Nuclear Database Structure (GNDS)

Version 2.0

OECD
BETTER POLICIES FOR BETTER LIVES

NEA
NUCLEAR ENERGY AGENCY

Nuclear Science

Specifications for the Generalised Nuclear Database Structure

Version 2.0

© OECD 2023
NEA No. 7647

NUCLEAR ENERGY AGENCY
ORGANISATION FOR ECONOMIC CO-OPERATION AND DEVELOPMENT

Cover photos: Frontier supercomputer (Oak Ridge National Laboratory/CC BY 2.0); Triga Reactor Core (United States Department of Energy); Lunar thorium concentrations (NASA).

Foreword

Complete data representing the relevant nuclear physics are required for the simulation of nuclear systems. Collectively, the discipline is known as nuclear data. These data must be stored in a standardised format and, since the release of the American ENDF/B-VI library and its associated ENDF-6 format in 1990, the ENDF-6 format has become a *de facto* worldwide standard. This format is based on legacy technology from an era when computers utilised punch-card readers and as a result the formats introduce artificial limitations, require legacy programming techniques to interface with the data and force new scientists and engineers to learn numerous Byzantine rules.

The NEA Working Party on International Nuclear Data Evaluation Co-operation (WPEC) was created in 1989, under the aegis of the Nuclear Science Committee (NSC),to improve the quality and completeness of nuclear data by bringing together represen--tatives of the major nuclear data evaluation projects of NEA member countries and of selected invitees. The WPEC community recognised the need for a new format that embraced modern computer programming paradigms, could address more so--phisticated user requirements and was maintained by the international community. Following the success of the WPEC subgroup 38 "Beyond the ENDF format: A modern nuclear database structure," which prepared the requirements for a replacement of the ENDF-6 format, WPEC established the Expert Group on the Recommended Definition of a General Nuclear Database Structure (GNDS). This Expert Group is tasked with defining the specifications of the GNDS, formalising these specifications and publishing them through the NEA.

Multiple versions of the GNDS specifications have been developed in parallel to the development of the requirements in an iterative process. The first public Version 1.9, which served effectively as a one-to-one translation of the ENDF-6 format as defined at the release of the 2018 ENDF/B-VIII.0 library, was published by the NEA in 2020. This document contains the specifications for Version 2.0, which is serves as a major update with 149 formally approved change requests made by the NEA Expert Group, bringing enhancements to how physics data are stored, new structures for uncertainty quantification, improvements to ensure complete compliance with the Subgroup 38 requirements, and a large number of quality-of-life changes.

Acknowledgements

The Generalised Nuclear Database Structure (GNDS) specifications are developed and maintained the by the members of the WPEC Expert Group on the Recommended Definition of a GNDS. The members of this Expert Group are listed in the table below, with specific contributions identified, including those that have directly authored the document (*), the Chair of the EG-GNDS (†) and the NEA Secretariat (‡).

Name	Country	Establishment
David BROWN†*	USA	Brookhaven National Laboratory
Jesse BROWN*	USA	Oak Ridge National Laboratory
Bret BECK*	USA	Lawrence Livermore National Laboratory
Jeremy Lloyd CONLIN*	USA	Los Alamos National Laboratory
Michael FLEMING‡*		OECD Nuclear Energy Agency
Godfree GERT*	USA	Lawrence Livermore National Laboratory
Wim HAECK*	USA	Los Alamos National Laboratory
Andrew HOLCOMB‡*		OECD Nuclear Energy Agency
Caleb MATTOON*	USA	Lawrence Livermore National Laboratory
Morgan WHITE*	USA	Los Alamos National Laboratory
Dorothea WIARDA*	USA	Oak Ridge National Laboratory
Teresa BAILEY	USA	Lawrence Livermore National Laboratory
Óscar CABELLOS	Spain	Universidad Politécnica de Madrid
Chris CHAMPMAN	USA	Oak Ridge National Laboratory
Mark CORNOCK	UK	Atomic Weaspons Establishment
Mireille COSTE-DELCLAUX	France	Commissariat à l’énergie atomique et aux énergies alternatives
Emmeric DUPONT	France	Commissariat à l’énergie atomique et aux énergies alternatives
Nathan GIBSON	USA	Los Alamos National Laboratory
Osamu IWAMOTO	Japan	Japan Atomic Energy Agency
Cédric JOUANNE	France	Commissariat à l’énergie atomique et aux énergies alternatives
Hyeong Il KIM	Korea	Korean Atomic Energy Research Institute
Alexander KONOBEEV	Germany	Karlsruhe Intitute of Technology
Fausto MALVAGI	France	Commissariat à l’énergie atomique et aux énergies alternatives
Jordan MCDONNELL	USA	Oak Ridge National Laboratory
Mark PARIS	USA	Los Alamos National Laboratory

Vladimir SOBES	USA	University of Tennessee, Knoxville
Martin STALEY	USA	Los Alamos National Laboratory
Alexey STANKOVSKIY	Belgium	Studiecentrum voor Kernenergie
Jean-Christophe SUBLET		International Atomic Energy Agency
Kenichi TADA	Japan	Japan Atomic Energy Agency
Ian THOMPSON	USA	Lawrence Livermore National Laboratory
Jason THOMPSON	USA	Naval Nuclear Laboratory
Kenji YOKOYAMA	Japan	Japan Atomic Energy Agency

This work was supported by the Nuclear Criticality Safety Program, funded and man--aged by the National Nuclear Security Administration for the United States Department of Energy. Additionally, this work was also supported by workers at several laboratories. Brookhaven National Laboratory is sponsored by the Office of Nuclear Physics, Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-98CH10886 with Brookhaven Science Associates, LLC. Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344. Oak Ridge National Laboratory is managed by UT-Battelle, LLC, for the US Department of Energy under contract DE-AC05-00OR22725 Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy under contract 89233218CNA000001.

List of abbreviations and acronyms

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AWE	Atomic Weapons Establishment (United Kingdom)
BNL	Brookhaven National Laboratory (United States)
BROND	Russian Evaluated Neutron Data Library
CEA	Commissariat à l’énergie atomique et aux énergies alternatives (France)
CENDL	China Evaluated Nuclear Data Library
CFPY	Cumulative Fission Product Yield
CM	Centre of Mass
CSEWG	Cross Section Evaluation Working Group (United States)
DOE	Department of Energy (United States)
DOI	Digital Object Identifier
ENDF	American Evaluated Nuclear Data File
ENDF-6	Evaluated Nuclear Data Format
ENSDF	Evaluated Nuclear Structure Data File
EXFOR	EXchange FORmat (Nuclear Reaction Physics Database)
FUDGE	For Updating Data and Generating Evaluations
GIDI	General Interaction Data Interface
GNDS	General Nuclear Data Structure
GPDC	General Purpose Digital Computer
HDF	Hierarchical Data Format
IAD	Interagency Data (United States)
IAEA	International Atomic Energy Agency
ICC	Internal Conversion Coefficients
IEEE	Institute of Electrical And Electronic Engineers

IFPY	Independent Fission Product Yield
ISO	International Organisation for Standardisation
JAEA	Japan Atomic Energy Agency
JEFF	Joint Evaluated Fission and Fusion File (NEA)
JENDL	Japanese Evaluated Nuclear Data Library
JSON	JavaScript Object Notation
KAERI	Korean Atomic Energy Research Institute
KERMA	Kinetic Energy Released per unit MAss
KIT	Karlsruhe Institute of Technology (Germany)
LANL	Los Alamos National Laboratory (United States)
LLNL	Lawrence Livermore National Laboratory (United States)
MD5	Message-digest Algorithm 5
MLBW	Multi-Level Breit Wigner
NIPNE	National Institute of Physics and Nuclear Engineering (Romania)
NNDC	National Nuclear Data Centre (United States)
NSC	Nuclear Science Committee (NEA)
ORNL	Oak Ridge National Laboratory (United States)
OSTI	Office of Scientific and Technical Information (United States)
PoPs	Properties of Particles
RUSFOND	Russian National Library of Neutron Data
SCK CEN	Studiecentrum voor Kernenergie Centre d’Étude de l’énergie Nucléaire (Belgium)
SI	Système Internationale for Units
SHA	Secure Hash Algorithm
SLBW	Single-Level Breit Wigner
TENDL	TALYS-based Evaluated Nuclear Data Library
T(N)SL	Thermal (Neutron) Scattering Law
UPM	Universidad Politécnica de Madrid (Spain)
URL	Uniform Resource Locator
URR	Unresolved Resonance Region
UTF	Unicode Transformation Format
W3C	World Wide Web Consortium
WPEC	Working Party on International Nuclear Data Evaluation Co-operation (NEA)

XML	eXtensible Markup Language

Contents

1. Introduction

This is the revision of the specifications for the Generalised Nuclear Database Structure (GNDS), corresponding to GNDS version 2.0. GNDS is designed to serve as a replacement for the Evaluated Nuclear Data Format (ENDF). Although GNDS supersedes the ENDF-6 format, GNDS should be viewed as a natural evolution of the ENDF-6 format, using modern computational tools to make nuclear data more accessible.

This "specifications" document provides the full documentation of all of the options for storing data in GNDS. The document follows in spirit of the ENDF-102 format manual (Cross Section Evaluation Working Group, 2018), but is not yet a replacement for a full User's Manual. In particular, this document does not provide any tutorial for generating or using GNDS files. Subsequent versions of this and other related documents will build on this work.

The separation of the format (GNDS) from a particular library, ENDF/B (the American nuclear data library) has a few implications. Firstly, the format no longer makes requirements on the evaluators regarding things like completeness or required energy ranges. This is now left to each data project from each region of the world to decide. Secondly, this means the format itself can be used to describe far more than just the traditional reaction data. In principal, other nuclear data library projects such as EXchange FORmat (EXFOR) (Otuka et al., 2014) and Evaluated Nuclear Structure Data File (ENSDF) (Tuli, 2001) could extend GNDS to serve their purposes.

The NEA Nuclear Science Committee (NSC) Working Party on International Nu--clear Data Evaluation Co-operation (WPEC) Expert Group on GNDS (EG-GNDS) drew inspiration from the ENDF-6 format for both for guidance and to ensure backwards compatibility (to the extent possible) in support of all nuclear data programmes, includ--ing the American Evaluated Nuclear Data File (ENDF) data project (Brown et al., 2018; Chadwick et al., 2006, 2011), Chinese Evaluated Nuclear Data Library (CENDL) (Ge et al., 2017), NEA Data Bank Joint Evaluated Fission and Fusion File (JEFF) (Plompen et al., 2020), Japanese Evaluated Nuclear Data Library (JENDL) (Shibata et al., 2011), Russian National Library of Neutron Data (RUSFOND) (Zabrodskaya et al., 2007) and TALYS-based Evaluated Nuclear Data Library (TENDL) (Koning et al., 2019). These specifications are therefore greatly informed by the ENDF-6 formats manual (Cross Section Evaluation Working Group, 2018), which have served as the *de facto* international standard for decades.

2.1 The hierarchy should reflect one's understanding of nuclear re--actions and decays, clearly and uniquely specifying all such data.
2.2 It should support storing multiple representations of these quan--tities simultaneously, for example evaluated and derived data.
2.3 It should support both inclusive and exclusive reaction data, that is discrete reaction channels as well as sums over multiple channels.
2.4 It should use general-purpose data containers suitable for reuse across several application spaces.
2.5 It should eliminate redundancy where possible.
2.6 As a corollary to requirements 2.1 and 2.2, multiple representa--tions of the same data should be stored as closely together in the hierarchy as feasible.

Figure 1.1: **Requirements of the Generalised Nuclear Database Structure**

As with the legacy ENDF-6 format, GNDS provides representations for cross sections and distributions, particle production from any reaction, photo- and electro-atomic interaction data, thermal neutron scattering data, and radionuclide production and decay data (including fission products). GNDS does this in a hierarchical, coherent and simpler manner.

Design philosophy

In designing GNDS, a format was sought that is both human and computer readable, enables both textual (e.g. eXtensible Markup Language [XML]) and binary (e.g. the Hier--archical Data Format [HDF]) representations, is extensible, makes provisions for both evaluated and processed data and supports multiple representations simultaneously (e.g. resonance parameters and reconstructed pointwise cross sections [0 Kelvin] and heated cross sections, simultaneously). To codify these goals, and to capture additional needs from the larger nuclear data community, the NEA NSC WPEC Subgroup 38 drafted a series of requirements (NEA, 2016a).

To understand the philosophy of the GNDS format and the developing support infrastructure, it is helpful to reiterate the main requirements of the GNDS format as described in the companion requirements document (NEA, 2016a) (using the numbering system from that document) in Figure 1.1.

Requirement 2.1 is reflected in the layout of both Properties of Particles (PoPs), the `reactionSuite` and other top-level nodes. A GNDS file can begin with one of several top-level nodes. All other nodes are nested inside these top-level nodes:

`reactionSuite`: Stores an evaluation of the reactions involving the combination of a projectile and target.

`covarianceSuite`: Stores a set of covariance sections. A `covarianceSuite` may be associated with a single `reactionSuite`, or it may contain cross-terms between more than one `reactionSuite`

`PoPs`: A database storing the properties of a collection of particles.

`fissionFragmentData`: Stores fission product yields from either spontaneous or particle induced fission. This node is maintained as a top-level node to support the legacy organization of data files containing fission product yield data from particle-induced fissions.

Requirements 2.2 and 2.6 are reflected in the `styles` element and its usage. **Requirement 2.3** is reflected in the `reactionSuite` and its children. **Requirement 2.4** is reflected in the usage of the standard data containers described in Part I. **Requirement 2.5** is maintained through good design and, in particular, through the use of a central `PoPs` data structure. This central `PoPs` data structure also lays the groundwork for a central, library-wide, particle property database.

Roles of WPEC Subgroups, EG-GNDS and the CSEWG Formats Committee

There are four groups who have had the largest impact on the drafting of these specifications:

WPEC Subgroup 38 SG-38 has been responsible for drafting the requirements for the top-level hierarchy for storing nuclear reaction data, hierarchy for storing particle/nucleus data and low-level data containers. Additionally, SG-38 spearheaded the drafting of this initial format specifications. Under the coordination of D. McNabb (LLNL), the success of SG-38 led to the formation of the EG-GNDS and the Subgroup 43.

WPEC Subgroup 43 The most important part of an nuclear data structure, such as GNDS, is the ecosystem built on the format, which includes the tools that ultimately bring the data into applications. In the case of the ENDF-6 format, these tools include standard code packages such as PREPRO, NJOY, the NNDC checking codes, AMPX and CALENDF, which perform hundreds of different operations depending on the specific needs of users. A good format can determine the data structures used to interact with it and these data structures are the components used to create new objects that users require. SG-43 was tasked with supporting the development of an Application Program Interface (API) for reading and writing data in the new structure, enabling data handling, processing, plotting and helping to assure the quality of data using the format(s). As of Spring 2022, there are several GNDS APIs in production:

- PoPs — properties of particles C++ API
- GIDI — I/O classes and routines for transport, C++
- MCGIDI — extensions to GIDI for MC transport
- HAPI — low level I/O API, includes HDF5 support

WPEC Expert Group on GNDS Under its mandate, EG-GNDS governs the development of the GNDS format. This role has two major aspects: 1) maintaining the specifi--cations and format development and 2) promoting its use through tutorials and outreach events, and the drafting of a user's guide.

CSEWG Formats Committee The Cross Section Evaluation Working Group (CSEWG) has a committee dedicated to nuclear data formats, which maintains the ENDF-6 format and actively participants in the development of GNDS through the NEA.

Layout of this document

The outline of this document is as follows. First the conventions used in the descrip--tions of the GNDS hierarchy in the Conventions (Chapter 2) are defined. Following this, an overview of general-purpose data types and containers, including covariance/uncer--tainty data in Part I, is provided. With this background, the main parts are elaborated upon: Data styles (Part II), Particle properties (Part III), and Reaction data (Part IV). There are also a series of appendices that provide a mapping between GNDS and the ENDF-6 format (Appendix A), an overview of the GNDS version history (Appendix B) and a comparison of different metadata schema (Appendix C).

2. Conventions

The basic structure and definitions of the GNDS hierarchical structure are now laid out, begining with the definitions of the allowed characters in both the specifications and the files. It is noted that although the examples given here are given in XML, the restrictions on various fields enable this hierarchy to be serialised in other heirarchical formats such as JSON, YAML, HDF5, etc.

The specifications below are described in terms of XML formatted data. The fundamental entity in XML is the node (element). A node has a node name, a list of attributes (i.e. meta-data) and text (also called body) and/or nested nodes. The node name and each attribute is a *scalar* valued data in that it corresponds to one and only one value.

For each container, the specifications shall define the node name, list of allowed attributes, list of allowed nested nodes and text (body). For each attribute, the specifications shall define the allowed values and whether the attribute is required or optional and, if relevant, if it has a default value. A required value does not have to be specified if it has a default value.

Fundementally each attribute or value within a nested node or body is given in one of the basic data types described in Chapter 3. The general-purpose data containers can be divided into several types.

scalar data containers: These containers store either text or a number. The basic containers are various number containers (Sections 4.1.1, 4.1.2, and 4.1.3) and the `string` container (Section 4.1.4).

array-like data containers: These containers are list or array like containers. They may contain additional information that describe an axis representing a variable. For example, for the function $x_0(x_1)$, the variables x_0 and x_1 each require axis information. The axes containers are `axis` (Section 5.1.2) and `grid` (Section 5.1.3) and `axes` (Section 5.1.1) while more general array-like containers are `values` (Section 5.2.1) `array` (Section 5.3.2) and `table` (Section 5.4.1)

functional containers: These containers store data representing single-valued[1] functions of the form $f(x)$, $f(x, y)$, $f(x, y, z)$, etc. or in the generic functional expression used in this document and n-dimensional function is $x_0(x_n, ..., x_2, x_1)$. These are known as 1-, 2-, 3- and n-dimensional functions (for $n > 3$). The functional containers are `XYs1d` (Section 6.1.1), `series` (Section 6.2), `XYs?d` for '?' an integer greater

[1] Except for the `regions` container which allows for a discontinuity.

than 1 (Section 6.3), `regions` (Section 6.4), and `gridded` (Section 6.5).

uncertainty containers: These containers allow for the storing of uncertainty data that is associated with a functional container.

documentation containers: These containers store the documentation for data in the GNDS formatted files.

Before one can delve into any of these topics, one must first define the most basic things, including what characters are allowed in the files, how one describes basic data types built with these characters and how one assembles full nodes in the heirarchy. Once this ground work is laid "abstract nodes" are described – these define features common to a class of nodes. Next, how to use these specifications in other, non-XML, heirarchichal languages are described. Finally, a primer on the Python regular expression syntax, used in many definitions in this document is provided.

2.1 Character encoding and various character sets

While many meta-languages can be used for storing the general-purpose data con--tainers, this document focuses on text-based storage which is platform independent and easily shareable. The general-purpose data containers use the UTF-8 character encoding (International Standards Organisation, 2017) for text[2]. UTF-8 was chosen because it is now the most commonly used character encoding for the World-Wide-Web (International Standards Organisation, 2017), because of its ability to represent characters from a wide variety of languages and because it is backwards compatible with the ASCII character set (Cerf, 1969).

This section defines various character sets that are subsets of the 128 characters of the ASCII character-encoding scheme. While most of these subsets, if not all, are well known, it is worth repeating them so their definitions are clear. The specifications in this document will in some places restrict the allowed set of characters that can be used to one or more of the following subsets:

2.1.1 Letter characters subset

In this document, characters from the `letters` subset along with several other character subsets are the only characters allowed for constructing node and attribute names (see Sections 3.1.1). The nodes and attributes use the ISO basic Latin alphabet for its letter subset. This consists of 26 uppercase and 26 lowercase letters. The uppercase letters are the following 26 ASCII characters: 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', and 'Z'. These characters are encoded in the ASCII character set as the base 10 integers 65 to 90, respectively. The lowercase letters

[2]Encoding for numbers can be meta-language specific. For example, some meta-languages use binary, and not text, encoding. However, when a number is represented in text, it must follow the format given in Sections 3.1 and 3.2.

are the following 26 ASCII characters: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’, ‘k’, ‘l’, ‘m’, ‘n’, ‘o’, ‘p’, ‘q’, ‘r’, ‘s’, ‘t’, ‘u’, ‘v’, ‘w’, ‘x’, ‘y’, and ‘z’. These characters are encoded in the ASCII character set as the base 10 integers 97 to 122, respectively.

2.1.2 Arabic digits character subset and numbers

A number can be an integer or a real number[3]. This section defines the characters needed for constructing a number.

The main characters in a number are the ten Arabic numerals which consist of the 10 characters: ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, and ‘9’. These characters are encoded in the ASCII character set as the base 10 integers 48 to 57, respectively.

In addition to the ten Arabic numerals, integers and real numbers need the plus (i.e. ‘+’) and minus (i.e. ‘−’) characters. These characters are encoded in the ASCII character set as the base 10 integers 43 and 45, respectively. For real numbers the period character (i.e. ‘.’) is needed. This character is encoded in the ASCII character set as the base 10 integer 46. The e-form (see Section 3.2.4) needs the additional characters ‘e’ and ‘E’ as defined in the letter subset.

2.1.3 Underscore

It is also useful to list the underscore character (i.e. ‘_’) which in the ASCII character set is the base 10 integer 95.

2.2 Denoting Nodes and Attributes

For the design of an XML general-purpose data container, four text types need to be defined. These types can be understood by noting the basic structure of an XML document. The main component of an XML document is an element. An element contains 1 to 3 parts and they are:

- The name of the element also called the node name, which herein is called the `nodeName`. All nodes have a `nodeName`.
- The attributes of the node, comprised of a list of 0 or more attributes. Each attribute is a name/value pair. Herein, the name is called the `attributeName` and the value is called the `"attributeValue"`.
- The body (or content) of the node. The body is optional. If present, it can contain text or other nodes. The text component of the body is herein called the `bodyText`. Currently, no container requires a body with mixed text and nodes.

[3]Octal and hexadecimal are currently not needed and are not defined in this document. Complex numbers are used but can be stored as two separate numbers representing the real and imaginary parts.

Hence, the four text types for an XML document are `nodeName`, `attributeName`, `"at-tributeValue"` and `bodyText`. Some simple XML examples are:

```
<nodeName/>
<nodeName attributeName="attributeValue"/>
<nodeName attributeName="attributeValue"></nodeName>
<nodeName attributeName="attributeValue">BODY</nodeName>
```

All `nodeName`, `attributeName` and `"attributeValue"` text types are case sensitive. For example, the only allowed Boolean true value is ‘true’ (see 3.3.5); neither ‘True’ nor ‘TRUE’ is an allowed value. There is one exception: in a floating point value (see Section 3.2.4) the exponent designator can be either an ‘e’ or ‘E’ (e.g. both ‘1e3’ and ‘1E3’ are allowed and are equivalent).

XML is very liberal in the characters that are allowed for `nodeName` and `attributeName`. To allow for better association between meta-languages’ names and programming variable names, it is best to restrict the allowed characters[4]. In addition, the allowed character set for `nodeName` and `attributeName` should probably be the same. Herein their character set will be generically called `XMLName`.

Links are often used to connect different parts of the database. Links are created using the XLink syntax (World Wide Web Consortium, 2010). An XLink is stored as an attribute with `attributeName` = ‘href’ and `"attributeValue"` containing the actual link. The XLink syntax supports linking within a file, linking to an external file, and using attribute values to select a specific node when multiple nodes share the same `nodeName`. Examples include:

```
<link href="/path/to/another/node"/>
<link href="externalFile#/path/within/external/file"/>
<link href="/path/to/node[@attributeName='attributeValue']"/>
```

In addition to alphanumeric characters, an XLink `"attributeValue"` may contain the characters ‘/’, ‘@’, ‘[’, ‘]’, ’_’ and ‘=’ as well as single and double quotes.

In these specifications, each node description is given its own subsection and one or more simplistic examples. The specification for each node will contain the format’s nodeName, abstract node, attributes and body (i.e. child nodes or text). Note that an abstract node is not an actual GNDS node, but describes attributes and/or body common to several nodes.[5] A sample “abstract node” specification is given below, in order to illustrate the how GNDS nodes are specified elsewhere in this document.

[4]See, for example, the “Best Naming Practices” in http://www.w3schools.com/xml/xml_elements.asp.
[5]In this sense, an abstract node is like an abstract base class in C++ or Python.

2.2.1 Abstract base type: `node`

The specification for each container will contain the container's nodeName, abstract node, attributes and body (i.e. child nodes or text). An abstract node[6] describes at--tributes and/or body common to several containers and, in itself, is not an actual node (container). The general specification looks like

Specifications for `node`

Node name: `node`
Abstract node: `abstract node's name`
Attributes: The list of additional allowed attributes is:

- **`attribute1`** [`Integer32`, **required**] Description of attribute.
- **`attribute2`** [`Float64`, optional, default is "1.0"] Description of attribute.
- **`attribute3`** [`Float64`, optional] Description of attribute.

Child nodes: The list of additional allowed child nodes is:

- **`childNode1`:** [**required**, must appear at least one time] Description of node.
- **`childNode2`:** [optional, when present, must appear one time] Description of node.
- **`childNodeWithChoice1`:** [optional, when present, * one of marked children must appear one time] Description of node.
- **`childNodeWithChoice2`:** [optional, when present, * one of marked children must appear one time] Description of node.

Body text: Description of body text.

XML Example(s) of `node`

```
<node
        attribute1="..."
        attribute2="..."
        attribute3="...">
    <childNode1>...</childNode1>
    <childNode2>...</childNode2>
    <childNodeWithChoice1>...</childNodeWithChoice1>
    <childNodeWithChoice2>...</childNodeWithChoice2>
    <![CDATA[SAMPLE BODY TEXT]]></node>
```

2.3 Abstract nodes

Now let us turn to some general kinds ("motifs") of nodes one will see in GNDS.

[6]In this sense, an abstract node is like an abstract base class in C++ or Python.

2.3.1 Abstract base type: `label`

Many nodes (containers) can reside in other nodes. When more than one node of the same type is embedded in another node, a unique way is needed to address a specific node. The label attribute provides this capability. An xlink can address a child node with a unique label even when embedded in a parent node containing more than one child nodes of the same type.

Specifications for `label`

Node name: `label`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**] The label is used to allow an xlink to uniquely address to the node (container).
Child nodes: This element has no child nodes

XML Example(s) of `label`

```
<label
        label="..."></label>
```

2.3.2 Abstract base type: `text`

This node stores a list of characters. This node can be used to store documentation, for example.

Specifications for `text`

Node name: `text`
Attributes: The list of additional allowed attributes is:
  `encoding` [`XMLName`, optional, default is “ascii”] One of ‘ascii’ or ‘utf8’. Defines the allowed character set for the body.
  `markup` [`attributeValue`, optional, default is “none”] One of ‘none’, ‘xml’, ‘xhtml’, ‘latex’, or other markups defined by a project.
  `label` [`XMLName`, optional] The label for the text node.
Child nodes: This element has no child nodes
Body text: The list of characters. The value for the encoding attribute defines the allowed character set. If the data are being stored in an XML file and if the body contains XML markup characters (e.g. ’<’, ’>’), then the characters must wrapped in special XML CDATA[7] section.

[7] An XML CDATA section is need whenever a string contains characters that are normally reserved for xml markup, such as ‘<’, ‘>’, ‘/’, and ‘&’.

XML Example(s) of `text`

- Example 1: Text representing the latex markup for "$\alpha \times x^{3/2}$".

```
<text markup="latex">$\alpha \times x^{3/2}$</text>
```

- Example 2: Same as example 1 but with the text wrapped using the special XML CDATA section (i.e. the text between '<![CDATA[' and ']]>').

```
<text markup="latex"><![CDATA[$\alpha \times x^{3/2}$]]></text>
```

2.3.3 Abstract base type: `physicalQuantity`

Within, for example, a particle property, each possible assignment is a physical quan--tity with a value, a unit, and optional documentation and uncertainty information. Each physical quantity is assigned a label, so that if the database contains multiple possible assignments for a particle's property one can uniquely identify each possible assignment by the corresponding label.

For most properties in the particle database, the `value` is a floating-point number. However, the database also supports storing values that are integers (e.g. for parity), fractions (for spin) and strings (for values like 'stable'). Since slightly different rules apply to each type of quantity, they are given unique abstract node names: `Float64`, `Integer32`, `Fraction32` and `String`. This section defines the abstract node used by the physical quantities.

Specifications for `physicalQuantity`

Node name: `physicalQuantity`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

`unit` [`XMLName`, optional] String name of the unit for this quantity. The unit can be omitted if the quantity is unitless.

`value` [`XMLName`, **required**] Determined by the sub-class. Value is usually a Float64, but may also be Integer32, Fraction32 or String.

Child nodes: The list of additional allowed child nodes is:

`documentation`: [optional, when present, must appear one time] Documentation specific to the physical quantity.

`uncertainty`: [optional, when present, must appear one time] The `uncertainty` node.

XML Example(s) of `physicalQuantity`

```
<physicalQuantity
        unit="..."
        value="...">
    <documentation>...</documentation>
    <uncertainty>...</uncertainty></physicalQuantity>
```

2.3.4 Abstract base type: `functional`

This section lists attributes and child nodes common to functional nodes (containers). These are the ‘outerDomainValue’ attribute and the child node `axes`.

Specifications for `functional`

Node name: `functional`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, optional] A label is required for a top-level functional node, but not for lower-dimensional functions appearing inside another functional node.
- **`outerDomainValue`** [`Float64`, optional] For an n-dimensional function, this is its associated x_{n+1} value. Required for all functional nodes that are nested inside higher-dimension functional nodes.

Child nodes: The list of additional allowed child nodes is:

- **`axes`:** [**required**, must appear one time] An `axes` node.
- **`values`:** [optional, when present, * one of marked children must appear one time] A list of numeric values. The meaning of the numbers depends on the type of function.
- **`functional`:** [optional, when present, * one of marked children must appear at least two times] A list of lower-dimensional functions contained inside this function.
- **`uncertainty`:** [optional, when present, must appear one time] The `uncertainty` node.

XML Example(s) of `functional`

```
<functional
        label="..."
        outerDomainValue="...">
    <axes>...</axes>
    <values>...</values>
    <functional>...</functional>
    <uncertainty>...</uncertainty></functional>
```

2.4 Other meta-languages

The discussion and specifications so far have used the XML meta-language. This section illustrates how other meta-languages can be used to express general-purpose data containers.

The XML meta-language has the following high level concepts:

1. three fundamental components - node, attribute and body. Recall, an attribute is a key with a value.
2. a node's body can contain text and/or other nodes.
3. text and child nodes can be interspersed in a node's body.
4. child nodes are ordered, so parsers reading the document can iterate over child nodes in the same order they appear in the document.
5. multiple nodes with the same `nodeName` can reside within the same parent node.
6. multiple attributes with the same attribute name cannot reside within the same parent node.

If a meta-language contains these concepts, expressing general-purpose data contain--ers in it should be easy. However, many meta-languages lack either the **attribute** in concept 1, or one or more of concepts 3, 4 or 5. For example, JSON does not support attributes, HDF5 does not preserve the order of child nodes, and neither JSON nor HDF5 support concepts 3 or 5.

In order to be more compatible with these meta-languages, the use of concept 3 is restricted in GNDS: each node shall contain *either* a list of child nodes *or* text body, not both. GNDS does not dis-allow the use of attributes or of multiple child nodes with the same name. These data types can still be translated to other meta-languages using fairly simple strategies:

1. for meta-languages that do not support attributes, either group all attributes together under a child node called **attributes** or convert each attribute to a child node and add a suffix like **_attr** to the node name.
2. for meta-languages that do not support multiple child nodes with the same name, add a unique suffix to each node name. For example, if multiple `reaction` elements appear in the file, they become `reaction0`, `reaction1`, etc. To avoid parsing strings to determine the original node name, a `nodeName` attribute (or child node) should also be added indicating the original unmodified node name.
3. for meta-languages that do not preserve the order of child elements, an attribute or child node with the (0-based) index should be added to the node. For example, in HDF5 the attribute **nodeIndex** could be added to each child in a group.

For example, the following XML:

```
<employees>
    <employee>
        <name first="Doc" last="Jones"/></employee>
    <employee>
        <name first="Grumpy" last="Smith"/></employee>
    <employee>
        <name first="Happy" last="Earp"/></employee></employees>
```

could be expressed in XML - although not recommend - as:

```
    <employees>
        <employee0 nodeName="employee">
            <name>
                <attributes>
                    <first>Doc</first>
                    <last>Jones</last></attributes></name></employee1>
        <employee1 nodeName="employee">
            <name>
                <attributes>
                    <first>Grumpy</first>
                    <last>Smith</last></attributes></name></employee2>
        <employee2 nodeName="employee">
            <name>
                <attributes>
                    <first>Happy</first>
                    <last>Earp</last></attributes></name></employee3>
                                                        </employees>
```

Or, in JSON as either:

```
    { "employees" : {
        "employee0" : {
            "nodeName" : "employee",
            "name" : {
                "attributes" : {
                    "first" : "Doc",
                    "last" : "Jones" } } },
        "employee1" : {
            "nodeName" : "employee",
            "name" : {
                "attributes" : {
                    "first" : "Grumpy",
                    "last" : "Smith" } } },
        "employee2" : {
            "nodeName" : "employee",
            "name" : {
                "attributes" : {
                    "first" : "Happy",
                    "last" : "Earp" } } } } }
```

where all attributes are collected in a child node named `attributes`, or as

```
    { "employees" : {
        "employee0" : {
            "nodeName" : "employee",
            "name" : {
                "first_attr" : "Doc",
                "last_attr" : "Jones" } },
        "employee1" : {
            "nodeName" : "employee",
            "name" : {
                "first_attr" : "Grumpy",
                "last_attr" : "Smith" } },
        "employee2" : {
            "nodeName" : "employee",
            "name" : {
                "first_attr" : "Happy",
                "last_attr" : "Earp" } } } }
```

where attributes are converted into child nodes, with the suffix `_attr` added to indicate that they are attributes.

2.4.1 Hybrid evaluations using multiple meta-languages

Different meta-languages have different strengths and weaknesses. For example, XML provides a compact way to store complex hierarchies, but no standard way to store large arrays of numbers in binary form. HDF5 excels at storing large numeric arrays, but is not well suited for storing deeply-nested hierarchies (creating a new node in HDF5 consumes significant memory due to constant overhead costs, and deeply-nested files slow down file access). The optimal solution may be to use a hybrid storage approach, with the data hierarchy expressed in XML or JSON, but large data arrays stored in HDF5 with links connecting the two files. Some initial experiments suggest that this option provides significant advantages in reducing file size and improving access times.

2.5 Python regular expression syntax

This section describes some of the Python regular expression syntax. A regular expres--sion is a “sequence of characters that forms a search pattern”. For this document, a regular expression representing a form is compared to a string and if they match, the string is properly formatted for that form. For the limited use of regular expression syntax in this document, it is sufficient to consider a regular expression built using 4 nodes: a repetition character, an expression, the or-ing operator and escape sequences. Here an expression can be another regular expression.

An expression can be:

- a single character, except a few special characters which need to be escaped. For this discussion, the important special characters are '.', '[', ']', '(', ')', '|', '?', '*' and '+'.
- any character between the '[' and ']'. For example, the expression representing any of the first 6 lower case English alphabet characters (LCEA) can be the string '[abcdef]' or '[defbca]' (i.e. order does not matter). This can be shortened using the '-' character to '[a-e]'. Independent of the number of characters between the '[' and ']', only one character in the string is compared to the list of characters between the '[' and ']'. For example, the regular expression 'c[aeiou]t' matches 'cat' and 'cot' but not 'coat' as the sub-expression '[aeiou]' is only compared to the second character.
- The period character '.' matches any character.
- The regular expression between the '(' and ')'. For example, the string '(ab?cd+)'.

For this document, the following repetition characters and their expressions are needed. If 'C' is an expression then the repetition expressions are:

'C?': 0 or 1 of the preceding 'C' expression. For example, 'a?' will match 0 or 1 'a' characters.
'C+': 1 or more of the preceding 'C' expressions. For example, 'a+' will match 1 or more 'a' characters.
'C*': 0 or more of the preceding 'C' expressions. For example, 'a*' will match 0 or more 'a' characters.
'C{m}': m of the preceding 'C' expressions. For example, 'a{4}' will match 4 'a' characters.
'C{m,n}': m to n of the preceding 'C' expressions. For example, 'a{2,4}' will match 2, 3 or 4 'a' characters.

The or-ing character is the '|' character. For example, the regular expression 'a|b' will match either the string 'a' or 'b'. The or-ing character has the lowest precedence so the expression 'abc|def' will match either 'abc' or 'def'.

An escape sequence starts with the backslash character (i.e. '\') followed by a character designating a special mode. For this document, the only relevant escape sequence is '\.' which allows for the matching of the period character (i.e. '.').

A simple regular expression example to match any number of the form '#', '#.', '#.#' or '.#' where # is an integer number of arbitrary size (e.g. '324'). The first three are matched by the regular expression '[0-9]+\.?[0-9]*'. The last representation is matched by the regular expression '\.[0-9]+'. Combining these with the or-ing character yields the expression '[0-9]+\.?[0-9]*|\.[0-9]+' which matches all four number representations.

Part I

STANDARD DATA TYPES

3. Basic data types

GNDS data types are divided into several classes: text, numbers, enumerations and various convenience types. As a special case of an enumeration, units are also described below. In addition, some data types to consider adding to GNDS are listed.

The discussion only considers ASCII and UTF-8 (International Standards Organisa--tion, 2017) representations of the various data types and their representation in XML. Binary representations of numbers are not defined in this document as one is only concerned with their ASCII representation in a file.

3.1 Description of text types

This section defines the `XMLName`, `attributeValue` and `bodyText` types and additional text types that are useful for the **table** and **values** containers. These additional types are `UTF8Text`, `printableText`, `IntegerTuple`, `quotedText` and `tdText`.

3.1.1 Basic data type: `XMLName`

XMLName represents the set and sequence of characters that are allowed for node and attribute names. Some meta-languages and programming languages (e.g. C and Python) allow a name to start with the underscore character. While other programming languages (e.g. FORTRAN) do not. For maximum compatibility an XMLName text shall not start with an underscore character.

Type name: `XMLName`
Allowed values: XMLNames shall begin with a character from the ISO basic Latin alphabet. All other characters shall be from the following: ISO basic Latin alphabet, Arabic numerals, and/or an underscore. There is no limit on the length of a name, except that it shall contain at least 1 character.

3.1.2 Basic data type: `attributeValue`

This represents the set and sequence of characters that are allowed for attribute values.

Type name: `attributeValue`
Allowed values: The allowed values for an attribute will depend on the attribute/node and, in general, should be specified by the project defining the attribute/node. A project can use any UTF-8 character in a value deemed necessary. However, some general rules apply:

- If a value of an attribute is an Integer32, UInteger32, Float64 or other defined type then it shall follow the specification for an Integer32, UInteger32, Float64 or the other defined type, respectively (see Sections 3.2.1, 3.2.2 and 3.2.4).
- If any part of the value of an attribute is an Integer32, UInteger32, Float64 or other defined type then that part shall follow the specification for an Integer32, UInteger32, Float64 or the other defined type, respectively. For example, if an attribute value contains a Float64 with units, as in 'mass="3.2 kg"', the numeric part of the attribute value shall follow the Float64 specification.

3.1.3 Basic data type: `bodyText`

In general, any UTF-8 character is allowed. However, the allowed characters for a node's body can be limited by its parent node's specifications.

Type name: `bodyText`
Allowed values: Any UTF-8 character is allowed.

3.1.4 Basic data type: `UTF8Text`

This is text composed of any sequence of UTF-8 characters.

Type name: `UTF8Text`
Allowed values: Any sequence of 0 or more UTF-8 characters.

3.1.5 Basic data type: `printableText`

This is text composed of only the printable ascii characters.

Type name: `printableText`
Allowed values: Any sequence of the ascii characters between the space character (decimal 32) to the tilde character ('~' or decimal 126) inclusive.

3.1.6 Basic data type: `quotedText`

This represents a UTF8Text string that is contained between two matching quote characters. The allowed quote characters are the ascii double quote character (i.e. " or decimal 34) and the ascii single quote character (i.e. ' or decimal 39).

Type name: `quotedText`
Allowed values: A UTF8Text string contained between matching single or double quote characters. For example, the quoted string “abc 123 xyz” is expressed as

```
"abc 123 xyz"
```

or

```
'abc 123 xyz'
```

This is not the same as

```
'abc 123 xyz '
```

as leading and trailing spaces are part of the string.

3.1.7 Basic data type: `tdText`

This represents a UTF8Text string that is contained between the XML start and end elements that define a standard html table cell (i.e. “<td>” and “</td>”).

Type name: `tdText`
Allowed values: The XML element “<td>” and the UTF8Text string it contains. For example, the quoted string “abc 123 xyz” is expressed as

```
<td>abc 123 xyz</td>
```

This is not the same as

```
<td> abc 123 xyz</td>
```

as leading and trailing spaces are part of the string.

3.2 Description of number types

This section defines some common number types relevant for computer programming and storage. The types defined here are the commonly used number types. Additional types can be defined by each project. The following contains some rules that a project should consider when defining number types. For many computer languages it is important to define the form and size allowed for various types of numbers.

In general, an ASCII integer should not start with the ‘0’ character unless its value is zero. Disallowing a ‘0’ character as the first character in a non-zero integer is desirable since some programming languages interpret an integer starting with a 0 as an octal value. For example, in Python 2 the command ‘int(077)’ returns the base 10 value 63.[8]

[8] To underline the potential for confusion, note that the command ‘int(“077”)’ returns the base 10 value 77 and ‘eval(“int(%s)” % “077”)’ returns the base 10 value 63. Even more problematically, in Python 3 the command ‘int(077)’ raises a SyntaxError exception. Octal values in Python 3 must start with ‘0o’, as in ‘int(0o77)’.

Furthermore, it is best that an ASCII representation of an integer not contain a decimal point (i.e. '.') or be stored using the floating-point e-form. For example, the integer value '123456' should not be represented in any of the forms shown in Table 3.1. There are two reasons for not allowing the decimal period (i.e. '.') or the e-form when storing integers in ASCII form. Firstly, programming languages have functions for converting an ASCII string to an integer, such as the `int` function in Python. In general, these functions fail when the string representation of an integer contains a '.' or is an e-form. A few examples are given below for the programming languages Python, C and FORTRAN.

Table 3.1: Examples of invalid integers.

Invalid representation	Reason
'123456.'	'.' not allowed.
'1.23456e5'	'.' and e-form not allowed.
'12345600e-2'	e-form not allowed.

Python programming example:

```
    k = int( "120." )       # A Python Exception is raised by the int function.
```

C programming example:

```
    int i = -1, j = -1, k = -1;
    k = sscanf( "120. 14", "%d %d", &i, &j );    // Only the first value is converted
                                                 // (i.e. k = 1, i = 120 and j = -1).
    k = sscanf( "120e-1 14", "%d %d", &i, &j );  // Only one value is converted and it
                                                 // is incorrect (i.e. k = 1, i = 120
                                                 // and j = -1).
```

FORTRAN programming example: The FORTRAN programming language does appear to translate integers containing a '.' or an e-form representation correctly provided the integer value can be expressed in the bits of the floating-point representation.

The second reason to avoid using '.' or an e-form in integer values is that some programmers use floating-point types to manipulate integers in their codes. Often, this yields floating-point values that are not integers. For example, consider the following Python code:

```
    a = 5
    b = 7
    inv_a = 1. / 5
    i = 7
    c = b * a * i * ( inv_a / i )
    print( "%.17e" % c, c / b - 1, int( c ) )
```

which prints,

```
    7.00000000000000089e+00 2.22044604925e-16 7
```

while replacing the assignments for ‘a’ and ‘i’ with ‘a = 3’ and ‘i = 11’, it prints

```
    6.99999999999999911e+00 -1.11022302463e-16 6
```

Mathematically both values are exactly 7, but as can be seen from the examples above, not all calculated values are exactly 7, and the latter assignments yield 6 when converted to an integer.

In the ENDF/B-VII.0 (Chadwick et al., 2006) release, there are many instances where integer multiplicities are stored as non-integer values (i.e. floating point representation). For example, in the ^{242}Am evaluation the multiplicity for neutrons for reaction MT 16 (n,2n) has 37 energy dependent values for the multiplicity. All the values use a special FORTRAN ‘e’-less format which looks like the **e-form** (see Section 3.2.4) but does not contain the ‘e’ (e.g. ‘1.999963+0’ instead of ‘1.999963e+0). None of these multiplicities can be converted to an integer using the Python command ‘int(m)’ where m is one of the values even when the ‘e’ is inserted back into the string. Only four converted to 2 with the Python command ‘int(float(m))’ - with the ‘e’ reinserted - while all the others convert to 1.

While it is impossible to restrict coders from working with non-integer types in their codes, the onus of converting an integer from a code’s representation to ASCII (the writer) or vice-versa (the reader) should fall on the writer of the number. This makes the proper interpretation of the integer trivial for the many readers.

Furthermore, if a reader wishes to input an integer as a float, the integer value will be properly represented as long as the value has less digits than the number of significant digits in the float. For example, converting the integer ‘123456’ to a Float64 (see below) and then to an Integer32 will produce the correct value, but converting ‘123456789’ to a Float32 and then to an Integer32 will not. FORTRAN converts the string ‘123456789’ to 123456792 without raising a warning[9].

If integer representations are restricted to not containing leading zeros (unless the value is equal to zero) and the floating-point form is not allowed, then each positive integer has two possible representations (since the ‘+’ is optional), while each negative integer has exactly one representation.

[9]All Integer32 values can be represented exactly as Float64 values.

3.2.1 Basic data type: `Integer32`

`Integer32` represents the allowed set and sequence of characters, and values that are allowed for a 32-bit signed integer.

Type name: `Integer32`
Allowed values: Any integer in the range [-2^{31} to 2^{31}) – note that the lower limit is inclusive and the upper limit is exclusive. The minimum allowed value (i.e. $-2^{31} = -2147483648$) is defined to be **Integer32_Min** and the maximum allowed value (i.e. $2^{31} - 1 = 2147483647$) is defined to be **Integer32_Max**. The Python regular expression (see Section 2.5) for an **Integer32** shall be

```
`[+-]?([1-9][0-9]*|0+)'
```

with the restriction that the value shall be in the range [**Integer32_Min**, **Integer32_Max**].
C programming equivalent: int32_t

3.2.2 Basic data type: `UInteger32`

This represents the allowed set and sequence of characters, and values that are allowed for a 32-bit unsigned integer.

Type name: `UInteger32`
Allowed values: Any integer in the range [0 to 2^{32}). An unsigned integer is represented in the same way as a signed integer, with the exception that a minus sign (e.g. '-' is not allowed). The minimum allowed value (i.e. 0) is defined to be **UInteger32_Min** and the maximum allowed value (i.e. $2^{32} - 1$) is defined to be **UInteger32_Max**. The Python regular expression (see Section 2.5) for an **UInteger32** shall be

```
`+?([1-9][0-9]*|0+)'
```

C programming equivalent: uint32_t

3.2.3 Basic data type: `Fraction32`

This represents the allowed set and sequence of characters, and values that are allowed for a faction given as the ratio of two Integer32 values.

Type name: `Fraction32`
Allowed values: All Fraction32 are represented as 'i1 / i2' or 'i3' where i1, i2 and i3 are Integer32s. Spaces are allowed around all Integer32 values. Example are '1/2', ' 1 / 2 ', '5 / 32 ' and '2'.

3.2.4 Basic data type: `Float64`

This represents the allowed set and sequence of characters, and values for a 64-bit floating point number.

Type name: `Float64`
Allowed values: Any normalised or denormalised IEEE-754 binary64 (Institute of Elec--trical and Electronics Engineers, 2008) value. The Python regular expression (see Section 2.5) for a Float64 shall be

```
'[+-]?([0-9]+\.?[0-9]*|\.[0-9]+)([eE][+-]?[0-9]+)?'
```

with the restriction that the value shall only be in the range for a normalised or denormalised IEEE-754 binary64 value. The number of significant digits (base 10) in a normalised IEEE-754 binary64 value is about 16, and is fewer for a denormalised value. The string representation of a Float64 value can have more significant digits than are supported by a normalised or denormalised IEEE-754 binary64 value; however, these additional digits will generally be ignored when the value is read into a Float64 variable.
C programming equivalent: double

3.3 Description of other "convenience" data types

3.3.1 Basic data type: `whiteSpace`

This represents the allowed set and sequence of characters for a white space.

Type name: `whiteSpace`
Allowed values: Any of the characters shown in Table 3.2

Table 3.2: Valid white spaces and their ASCII characters.

character name	escape sequence	ASCII decimal value
space	‘ ’	32
tab	‘\t’	9
linefeed	‘\n’	10
carriage control	‘\r’	13

3.3.2 Basic data type: `date`

This represents the allowed formats for a date which may also include a time or a time with offset from UTC information. The allowed formats are a subset of the ISO-8061 ‘extended’ formats (International Standards Organisation, 2019b).

Type name: `date`
Allowed values: One of the following ISO-8061 ’extended’ formats

- date only as ‘YYYY-MM-DD’ (e.g. ‘2015-02-01’),
- date and time as ‘YYYY-MM-DDThh:mm:ss’ (e.g. ’2015-02-01T16:55:18’) and
- date and time with offset from UTC as ‘YYYY-MM-DDThh:mm:ss±hh:mm’ (e.g. ’2015-02-01T16:55:18+01:00’)

where

- YYYY represents the year and must be in the range [0000,9999],
- MM represents the month and must be in the range [01,12],
- DD represents the day of the month and must be in the range [01,28], [01,29], [01,30] or [01,31], depending on the value specified for YYYY-MM,
- hh represents the hour and must be in the range [00,23],
- mm represents the minute and must be in the range [00,59] and
- ss represents the second and must be in the range [00,60] where 60 is only allowed to denote an added leap second.

All characters must be ASCII. In particular, the ’minus sign’, for representing a negative offset from UTC, must be the ASCII hyphen-minus character (i.e. 45 decimal). All other ISO-8601 formats are not allowed. For example, ‘YYYYMMDD’ (e.g. ’20150201’) is not allowed. Date ranges are not supported.

3.3.3 Basic data type: `checksum`

A checksum (also called a ‘hash’) is a hexadecimal number computed from a file, string or other data. The checksum serves as a unique digital ‘fingerprint’ that can be used to detect whether data has changed since it was computed.

Type name: `checksum`
Allowed values: Any combination of digits 0-9 and letters a-f. The length depends on the algorithm used to compute the checksum: 32 characters for MD5, 40 for SHA-1. The following regular expression can be used to check for a valid MD5 or SHA-1 checksum: ‘[0-9a-f]{40}|[0-9a-f]{32}’.

3.3.4 Basic data type: `IntegerTuple`

This is text node composed of a comma-delimited list of integers. It is used to denote starting indices within an array. The integer type is Integer32.

Type name: `IntegerTuple`
Allowed values: A list of Integer32 values separated by a comma (i.e., ‘,’). Any number of printable white spaces are allow before and after each Integrer32 value.
For example, the array index [3][4][7] can be represented as the IntegerTuple ‘3, 4, 7’.

3.3.5 Basic data type: `Boolean`

This represents the allowed set and sequence of characters that represent the Boolean `"true"` and `"false"` values in general-purpose data containers.

Type name: `Boolean`
Allowed values: The allowed strings are ‘true’ and ‘false’. For a table cell, the allowed values are ‘`<true/>`’ and ‘`<false/>`’.
C programming equivalent: bool
Comments: the standard XML attribute value representation for Boolean values true and false are ‘true’ and ‘false’ respectively.

3.3.6 Basic data type: `Empty`

This is the token that states that the value of the cell in a table is empty.

Type name: `Empty`
Allowed values: The ASCII string ‘`<td/>`’.

3.4 Description of enumerated types

3.4.1 Basic data type: `parity`

Parity is a particle property related to the (anti)-symmetry of the wave function to reflection.

Type name: `parity`
Allowed values: [‘+1’, ‘-1’]

3.4.2 Basic data type: `frame`

Tells what reference frame a projectile, product, distribution or double-differential cross section is given in.

Type name: `frame`
Allowed values: [‘lab’, ‘centerOfMass’]

3.4.3 Basic data type: `algorithm`

Two algorithms for computing hash sums are currently supported: the MD5 and SHA-1 algorithms.

Type name: `algorithm`
Allowed values: [‘md5’, ‘sha1’]

3.4.4 Basic data type: `interpolation`

Rule for interpolating between two points used to specify a function.

Type name: `interpolation`
Allowed values: [‘flat’, ‘charged-particle’, ‘lin-lin’, ‘lin-log’, ‘log-lin’, ‘log-log’]

GNDS employs the ENDF-6 interpolation scheme as it is quite general. In “traditional” ENDF-6 interpolation, each axis has an interpolation rule that defines how to interpolate its variable along its axis. Consider, for example, the function $f(x, y)$ with linear (‘lin’) interpolation along the x-axis, logarithm (‘log’) interpolation along the y-axis and ‘flat’[10] interpolation along the dependent axis. Between any two consecutive x values, the interpolation for the dependent value is ‘flat’ as it also is for any two consecutive y values. There is, in this scheme, no way to have a different interpolation rule for dependent axis depending on which independent axis is being interpolated (e.g. for the x- and y-axis in this example). In fact, since the interpolation for the dependent axis is ‘flat’, there is no need to specify the interpolation along the $x-$ and y-axes.

In the ENDF-6 interpolation scheme, interpolation rules are defined for each inde--pendent axis, and those rules specify the interpolation rule for both that independent axis and for the dependent axis while moving along that independent axis. No in--terpolation is defined for the dependent axis as its interpolation is given for each independent axis. This scheme allows for a different interpolation rule for the depen--dent axis depending on which independent axis it is being interpolated along. In this scheme, two interpolation values are typically required for each independent axis: one for the independent axis and one for the dependent axis (e.g. `"lin-lin"` or `"log-lin"`).[11]

For a dataset representing $f(x, y)$, the ENDF-6 interpolation allows, for example, the x-axis interpolation rule to be `"log-lin"` while the y-axis rule is `"lin-lin"`. That is, for $x_i \leq x \leq x_{i+1}$ and for the $y = y_j$, $f(x, y_i)$ is interpolated as

$$f(x, y_i) = f(x_i, y_j)\ \mathrm{exp}\left(\log\left(\frac{f(x_{i+1}, y_j)}{f(x_i, y_j)}\right)\left(\frac{x - x_i}{x_{i+1} - x_i}\right)\right) \tag{3.1}$$

while for $y_j \leq y \leq y_{j+1}$ and for the $x = x_i$, $f(x_i, y)$ is interpolated as

$$f(x_i, y) = (f(x_i, y_{j+1}) - f(x_i, y_j))\ \left(\frac{y - y_i}{y_{j+1} - y_j}\right) + f(x_i, y_j) \quad . \tag{3.2}$$

[10]Flat interpolation is also called ‘constant’ or ‘histogram’ interpolation.
[11]The first string in a name such as `"log-lin"` refers to the dependent axis (the $y-$ axis) and the second to the independent axis (the $x-$ axis).

Many of the containers have an interpolation attribute. In general, the value for an interpolation attribute can be specified by each format project. However, several predefined values are listed in Table 3.3.

Table 3.3: Predefined interpolation strings. The first four strings contain two sub-strings separated by the dash character (e.g. '-'). The sub-string to the left of the dash is the interpolation for the dependent axis while the sub-string to the right of the dash is the interpolation for the independent axis.

string	definition
"lin-lin"	The independent and dependent axes are linearly in--terpolated.
"lin-log"	The independent axis is logarithmically interpolated and the dependent axis is linearly.
"log-lin"	The independent axis is linearly interpolated and the dependent axis is logarithmically.
"log-log"	The independent and dependent axes are logarithmi--cally interpolated.
"flat'	The dependent value between consecutive indepen--dent values is constant, equal to the dependent value at the lower of the two independent value.
"charged-particle"	Special rule for interpolating charged-particle reac--tions near threshold. See description below.

A special `chargedParticle` interpolation law (equivalent to INT=6 in ENDF-6), is defined for charged-particle cross sections and is based on the limiting forms of the Coulomb penetrabilities for exothermic reactions at low energies and for endothermic reactions near the threshold.

Consider interpolating a cross section for a reaction (with a given Q value and threshold energy Ξ) between two energies E_1 and E_2 with $E_1 \neq E_2$. The cross section at E_1 and E_2 are σ_1 and σ_2 respectively. For an energy E where $T < E_1 \leq E \leq E_2$, the INT=6 interpolation rule is

$$\sigma(E) = \frac{A}{E} \exp\left[-\frac{B}{\sqrt{E-T}}\right] \tag{3.3}$$

where

$$T = \begin{cases} 0 & \text{for } Q > 0 \\ \Xi & \text{for } Q \leq 0 \end{cases} \tag{3.4}$$

and

$$A = \exp\left[\frac{B}{\sqrt{E_1 - T}}\right] \sigma_1 E_1 \tag{3.5}$$

and

$$B = \frac{\ln\left(\frac{\sigma_2 E_2}{\sigma_1 E_1}\right)}{\frac{1}{\sqrt{E_1 - T}} - \frac{1}{\sqrt{E_2 - T}}}. \tag{3.6}$$

Formulated this way, B diverges if either σ_1 or $\sigma_2 = 0$ because of the logarithm in the definition of B. Furthermore, A, B and $\sigma(E)$ all diverge as $E \to T$.

Let $x(E) = 1/\sqrt{E-T}$, $x_1 = x(E_1)$ and $x_2 = x(E_2)$ and define

$$\alpha(E) = \frac{x(E) - x_1}{x_2 - x_1}. \tag{3.7}$$

In terms of these, the cross section may be written

$$\sigma(E) = \frac{1}{E}(\sigma_2 E_2)^{\alpha(E)}(\sigma_1 E_1)^{1-\alpha(E)}. \tag{3.8}$$

Since $0 \leq \alpha(E) \leq 1$ for all E such that $E_1 \leq E \leq E_2$, one also has $0 \leq 1 - \alpha(E) \leq 1$. Formulated this way, one avoids taking any logarithms. One only raises the cross section values at certain points to fractional powers. If either σ_1 or σ_2 are zero, one no longer need to evaluate anything as the result is clearly zero. Note this reformulation does not help with the singularity as $E \to T$ already present in the original formulation of INT=6 as this singularity is now contained in the definition of $x(E)$.

This interpolation method should only be used for E close to T. At higher en--ergies, non-exponential behaviour will normally begin to appear, and linear-linear interpolation is more suitable.

3.4.5 Basic data type: `interpolationQualifier`

Additional rule for interpolating between two N-dimensional functions inside an (N+1)-dimensional function.

Type name: `interpolationQualifier`
Allowed values: [‘direct’, ‘unitBase’, ‘correspondingEnergies’, ‘correspondingPoints’]

In addition to an interpolation attribute, the nuclear data community requires an interpolation qualifier to produce physical results when interpolating a distribution (e.g. $P(E'|E)$) where the domain of the outgoing energy E' changes with incident energy. Directly interpolating between the outgoing distributions at two different incident energies produces unphysical results unless the domains of the two outgoing distribu--tions match. More generally, interpolation qualifiers determine how to appropriately interpolate between two N-dimensional functions where $N \geq 1$.

To illustrate the use of interpolation qualifiers, consider an energy distribution $P(E'|E)$ at two different incident energies E_1 = 10 MeV and E_2 = 12 MeV. Assume that both distributions start at outgoing energy E' = 0, but the 10 MeV distribution extends up to E' = 1.2 MeV while the 12 MeV distribution extends to E' = 2.8 MeV. The challenge is to find an intermediate curve at incident energy E_i, e.g. at 11 MeV.

The simplest interpolation qualifier is ‘direct’, wherein the intermediate curve is produced by directly interpolated between the two bounding curves using the `in--terpolation` rule. Direct interpolation has a significant problem: the domain of an intermediate curve between two bounding curves is always the max domain of the two bounding curves. In the example, that means the interpolated curve at incident energy E_i = 10.1 MeV extends up to 2.8 MeV, which likely violates conservation of energy.

A better option is to use the `unitBase` interpolation qualifier. In this case, the outgoing energy distributions at 10 MeV and 12 MeV are both mapped onto the unit interval [0,1], then interpolated on that interval using the `interpolation` rule. The resulting curve is then mapped back onto a new domain whose limits are determined by interpolating between the domain min and max respectively of the two original distributions. This solves the main issue with direct interpolation: here the outgoing distribution at incident energy 10.1 MeV would only extend to 1.28 MeV (assuming "lin-lin" `interpolation`).

`unitBase` interpolation performs very well when the two bounding curves are both fairly smooth, but it can still cause unphysical results when the spectra contain a combination of smooth and sharp features. For example, if an outgoing photon spec--trum includes both a continuum and discrete lines, unitBase interpolation may end up 'doubling' the discrete lines (i.e. turning one discrete line into two different lines) if the domain of the continuum changes between incident energies. This effect is generally strongest at the midpoint between the two bounding curves.

Two other interpolation qualifiers attempt to improve on unitBase for probability densities. In brief, the `correspondingEnergies` option means to first compute N equally-probable bins from each bounding curve (i.e. N bins such that the integral of the curve over each bin = $\frac{1}{N}$), then use `unitBase` to interpolate between each pair of corresponding bins. This option is supported by ENDF-6, and depending on the value of N can do a better job than `unitBase` at interpolating complicated spectra.

The `correspondingPoints` option is another alternative recently proposed by G. Hed--strom. It means first computing the cumulative distribution function (cdf) for both bounding curves, then forming a union grid of all points in the two cdfs, and using `unitBase` to interpolate between each set of points on the union grid. This method (described further in (Hedstrom et al., 2016)) is considered more robust than `corre--spondingEnergies` (G. Hedstrom points out that `correspondingEnergies` produces the same result as `correspondingPoints` in the limit $N \to \infty$).

While the `interpolationQualifier` could in principle be added to the `interpolation` attribute (e.g. 'unitBase,lin-lin'), this would significantly increase the number of allowed interpolation values. The `interpolationQualifier` is therefore stored as a separate attribute, and the two attributes together fully specify how data should be interpolated.

3.4.6 Basic data type: `dateType`

Metadata describing important dates in the generation or maintainance of a dataset. See Table C.3 for a description of the allowed values.

Type name: `dateType`

Allowed values: [‘accepted’, ‘available’, ‘copyrighted’, ‘collected’, ‘created’, ‘issued’, ‘submitted’, ‘updated’, ‘valid’, ‘withdrawn’]

3.4.7 Basic data type: `contributorType`

Metadata describing various contributions to a maintainance/distribution/etc. of a dataset which are not strictly authorship contributions. See Table C.2 for a description of the allowed values.

Type name: `contributorType`
Allowed values: [‘ContactPerson’, ‘DataCollector’, ‘DataCurator’, ‘DataManager’, ‘Dis--tributor’, ‘Editor’, ‘HostingInstitution’, ‘Producer’, ‘ProjectLeader’, ‘ProjectManager’, ‘ProjectMember’, ‘RegistrationAgency’, ‘RegistrationAuthority’, ‘RelatedPerson’, ‘Researcher’, ‘ResearchGroup’, ‘RightsHolder’, ‘Sponsor’, ‘Supervisor’, ‘WorkPack--ageLeader’, ‘Other’]

3.4.8 Basic data type: `relationType`

Metadata describing how one dataset is related to another. See Table C.5 for a descrip--tion of the allowed values.

Type name: `relationType`
Allowed values: [‘IsCitedBy’, ‘Cites’, ‘IsSupplementTo’, ‘IsSupplementedBy’, ‘IsContin--uedBy’, ‘Continues’, ‘Describes’, ‘IsDescribedBy’, ‘HasMetadata’, ‘IsMetadataFor’, ‘HasVersion’, ‘IsVersionOf’, ‘IsNewVersionOf’, ‘IsPreviousVersionOf’, ‘IsPartOf’, ‘HasPart’, ‘IsPublishedIn’, ‘IsReferencedBy’, ‘References’, ‘IsDocumentedBy’, ‘Docu--ments’, ‘IsCompiledBy’, ‘Complies’, ‘IsVariantFormOf’, ‘IsOriginalFormOf’, ‘IsIden--ticalTo’, ‘IsReviewedBy’, ‘Reviews’, ‘IsDerivedFrom’, ‘IsSourceOf’, ‘IsRequiredBy’, ‘Requires’, ‘Obsoletes’, ‘IsObsoletedBy’]

3.5 Units

In order for data with units to be shared among various projects, an accepted set of symbols for units must be used. In addition, a way to combine units, unit prefixes and symbols for fundamental constants must be defined. This section list recommended symbols for fundamental constants and units, unit prefixes and ways to combine units[12].

Recommend prefixes are listed in table 3.4. It is recommended that only one prefix be prepended to a unit. For example, prefixing the symbol for gramme ‘g’ with the kilo

[12]These recommended prefixes, fundamental constants and units are based on the Python module Scientific/Physics/PhysicalQuantities.py written by Konrad Hinsen with contributions from Greg Ward and Berthold Hoellmann.

Table 3.4: This table lists the recommended unit prefixes, and their factors and symbols. One prefix may be appended to the beginning of any base unit.

Prefix	Factor	Symbol
yotta	10^{24}	Y
zetta	10^{21}	Z
exa	10^{18}	E
peta	10^{15}	P
tera	10^{12}	T
giga	10^{9}	G
mega	10^{6}	M
kilo	10^{3}	k
hecto	10^{2}	h
deka	10	da
deci	10^{-1}	d
centi	10^{-2}	c
milli	10^{-3}	m
micro	10^{-6}	mu
nano	10^{-9}	n
pico	10^{-12}	p
femto	10^{-15}	f
atto	10^{-18}	a
zepto	10^{-21}	z
yocto	10^{-24}	y

'k' (i.e. 'kg') but not two 'k's (i.e. 'kkg'). Instead of 'kkg' use 'Mg'.

The recommend way to combine units is to use the asterisk '*' for multiplication, the slash '/' for division and two asterisks '**' as the power operator. For example, the unit for mass can be 'g', 'kg', 'eV/c**2' or 'MeV/c**2', and the unit for energy can be 'J' or 'kg*m**2/s**2'. Spaces are to be ignored; hence, 'kg*m**2/s**2' and 'kg * m**2 / s**2' are equivalent, Table 3.6 list the seven SI units and Table 3.7 list the some derived SI units.

It is recommended that units represent the physical quantity that they describe. For example, a mass can have a unit of 'kg', 'oz' or 'MeV/c**2'. Note that the common nuclear physics convention of expressing masses in 'MeV' (rather than 'MeV/c**2') should not be allowed.

3.6 Some additional types to consider

Below is a list of types that are supported by most, if not all, programming languages but are currently are not used by the GPDCs. They are listed so as not to be ignored.

Table 3.5: This table lists the Fundamental constants and their recommended symbols.

Fundamental constant	symbols
speed of light	c
permeability of vacuum	mu0
permittivity of vacuum	eps0
gravitational constant	Grav
Planck constant	hplanck
Planck constant / 2pi	hbar
elementary charge	e
Avogadro number	Nav
Boltzmann constant	k

Table 3.6: This table lists the seven SI units, and their measures and recommended symbols.

Unit	Measure	Symbol
metre	length	m
kilogramme	mass	kg
second	time	s
Ampere	electrical current	A
Kelvin	thermodynamic temperature	K
mole	amount of substance	mol
candela	luminous intensity	cd

Table 3.7: This table lists derived SI units, and their admixtures and recommended symbols.

Unit	Admixture	Symbol
Hertz	1/s	Hz
Newton	m*kg/s**2	N
Pascal	N/m^2	Pa
Joule	N * m	J
Watt	J/s	W
Coulomb	s * A	C
Volt	W/A	V
Farad	C/V	F
Ohm	V/A	ohm
Siemens	A/V	S
Weber	V * s	Wb
Tesla	Wb/m^2	T
Henry	Wb/A	H
Lumen	cd * sr	lm
Lux	lm/m^2	lx
Becquerel	1/s	Bq
Gray	J/kg	Gy
Sievert	J/kg	Sv

3.6.1 Basic data type: `Integer64`

This represents the allowed set and sequence of characters, and values for a 64-bit signed integer.

Type name: `Integer64`
Allowed values: Any integer in the range [-2^{63} to 2^{63}). The minimum allowed value (i.e. -2^{63}) is defined to be **Integer64_Min** and the maximum allowed value (i.e. $2^{63} - 1$) is defined to be **Integer64_Max**. The Python regular expression (see Section 2.5) for an **Integer64** shall be

```
`[+-]?([1-9][0-9]*|0+)'
```

with the restriction that the value shall be in the range [**Integer64_Min**, **Integer64_-Max**].
C programming equivalent: int64_t
Implementation status: Not implemented as of GNDS-1.9
Comments: Currently not needed so not defined.

3.6.2 Basic data type: `Float32`

Floating point number that can be stored at reduced precision

Table 3.8: This table lists some standard units and their recommended symbols.

Unit	symbol
time	
minute	min
hour	h
day	d
week	wk
year	yr
length	
inch	inch
foot	ft
yard	yd
(British) mile	mi
Nautical mile	nmi
Angstrom	Ang
light year	lyr
Astronomical unit	au
Bohr radius	Bohr
area	
are (100 m^2)	a
acre	acres
barn	b
volume	
litre	l
deci litre	dl
centi litre	cl
milli litre	ml
teaspoon	tsp
tablespoon	tbsp
fluid ounce	floz
cup	cup
pint	pt
quart	qt
US gallon	galUS
British gallon	galUK

Unit	symbol
mass	
atomic mass units	amu
ounce	oz
pound	lb
ton	ton
force	
dyne (cgs unit)	dyn
energy	
erg (cgs unit)	erg
electron volt	eV
Wavenumbers/inverse cm	Hartree
Kelvin as energy unit	Ken
thermochemical calorie	cal
thermochemical kilocalorie	kcal
international calorie	cali
international kilocalorie	kcali
British thermal unit	Btu
power	
horsepower	hp
pressure	
bar (cgs unit)	bar
standard atmosphere	atm
torr = mm of mercury	torr
pounds per square inch	psi
angle	
degrees	deg
radian	rad
steradian	sr
temperature	
degrees Rankine	degR
degrees Celsius	degC
degrees Fahrenheit	degF

Type name: `Float32`
Allowed values: n/a
Implementation status: Not implemented as of GNDS-1.9
Comments: Since only ASCII representation of data types is being considered here, and since Float32 is a subset of Float64, this is probably not needed.

3.6.3 Basic data type: `Octal`

Typical octal number

Type name: `Octal`
Allowed values: n/a
Implementation status: Not implemented as of GNDS-1.9
Comments: Currently not needed so not defined.

3.6.4 Basic data type: `Hexadecimal`

Typical hexadecimal number

Type name: `Hexadecimal`
Allowed values: n/a
Implementation status: Not implemented as of GNDS-1.9
Comments: Currently not needed so not defined.

4. Scalar container nodes

4.1 Scalar quantities in GNDS

Techopedia[13] explains the concept of a scalar as:

> In C programming languages, scalar data types (such as char, int and float) are commonly used. However, scalar data types also may be scalar variables - basic variables used in practical extraction and report language. They are either strings that include symbols and letters, or numbers with exponents, integers and decimal values.
>
> Scalar is also a common concept in mathematics and physics. In mathe--matics, scalars are used as vector components, as well as in modules and normed vector spaces. In physics, a scalar function gives a single variable value for all points in space and measures temperature, charge variations, etc. It is a physical quantity that is not changed by the rotations and transla--tions of coordinate systems. Scalar field data is visualised as a set of discrete sampled values.

In this chapter, the scalar quantities used in GNDS will be detailed.

4.1.1 General purpose type: `double`

Stores a exttttFloat64 quantity along with a unit, optional label and uncertainty

Specifications for `double`

Node name: `double`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:
    `label` [`XMLName`, optional] Unique label

[13]Techopedia - Where Information Technology and Business Meet, www.techopedia.com/definition/16441/scalar.

`unit` [`XMLName`, optional, default is "'' (no label)"] e.g. 'kg' or 'm'
`value` [`Float64`, **required**] numeric value

Child nodes: The list of additional allowed child nodes is:
`uncertainty`: [optional, when present, must appear one time] Stores the uncer--tainty on the parent `double`.

XML Example(s) of `double`

```
<double
        label="..."
        unit="..."
        value="...">
    <uncertainty>...</uncertainty></double>
```

4.1.2 General purpose type: `integer`

The `integer` physical quantity is used to store quantities that can only be integers. It is similar to the `double`, except that the value must be an integer (e.g. value='1' rather than value='1.0'). Uncertainties are also treated differently: rather than an `uncertainty` node, tentative assignments are indicated by storing each possible assignment along with a likelihood.

Specifications for `integer`

Node name: `integer`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:
`label` [`XMLName`, optional, default is "'' (no label)"] Unique label
`unit` [`XMLName`, optional, default is "'' (i.e. unitless)"] e.g. 'kg' or 'm'
`value` [`Integer32`, **required**]
Child nodes: This element has no child nodes

XML Example(s) of `integer`

```
<integer
        label="..."
        unit="..."
        value="..."></integer>
```

4.1.3 General purpose type: `fraction`

The `fraction` physical quantity is used to store values that can only be integers or fractions. It is nearly identical to the `integer`, but the value in this case may either be an integer or a ratio of integers separated by the '/' character, as in '3/2'. Values like '1.5' are not allowed.

Specifications for `fraction`

Node name: `fraction`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:
 `label` [`XMLName`, optional, default is "'' (no label)"] Unique label
 `unit` [`XMLName`, optional, default is "'' (i.e. unitless)"] e.g. 'J/s'
 `value` [`Fraction32`, **required**] numeric value
Child nodes: This element has no child nodes

XML Example(s) of `fraction`

```
<fraction
        label="..."
        unit="..."
        value="...">
</fraction>
```

4.1.4 General purpose type: `string`

The `string` physical quantity is used to store quantities that cannot be represented numerically. The main examples are half-lives for stable particles, which are best represented as the string 'stable'.

Specifications for `string`

Node name: `string`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:
 `label` [`XMLName`, optional] Unique label
 `unit` [`XMLName`, optional] A valid unit
 `value` [`XMLName`, **required**] contains a string composed of printable ascii characters (e.g. 'stable').
Child nodes: This element has no child nodes

XML Example(s) of `string`

```
<string
        label="..."
        unit="..."
        value="..."></string>
```

4.1.5 General purpose type: `link`

Used to explicitly link to another part of the GNDS document (or an external document). For example, if both axis grids in a `gridded2d` container are identical, the `link` can be used rather than repeating the grid values.

Specifications for `link`

Node name: `link`
Attributes: The list of additional allowed attributes is:
 `href` [`bodyText`, **required**] The URL or xpath-like string pointing to the referred-to element
Child nodes: This element has no child nodes

XML Example(s) of `link`

```
<link
        href="..."></link>
```

5. Array-like container nodes

5.1 Axes and related objects

5.1.1 General purpose type: `axes`

Many of the data containers represent functions that have independent and dependent axes. The `axes` node provides a way to assign a label and unit to each axis. If an `axes` node is present, each independent and dependent axis must have an `axis` or `grid` node. The `axis` or `grid` nodes are indexed 0 to n where n is the number of independent axes. For the function $x_0(x_n, ..., x_1)$, index 0 is for dependent axis x_0, 1 is for independent axis x_1, ... and n is for the independent axis x_n. Note: Two types of axes nodes are allowed:

- One type gives an xlink to another axes node
- The other lists the axes explicitly.

Specifications for `axes`

Node name: `axes`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:
  `href` [`UTF8Text`, optional] A the URL, in xpath syntax, of another `axes` element
Child nodes: The list of additional allowed child nodes is:
  `axis`: [optional, when present, must appear at least one time] This node stores an `index`, `label` and `unit` for an axis.
  `grid`: [optional, when present, must appear at least one time] Stores a list of boundaries or points along this axis, e.g. to define group boundaries for grouped data.

XML Example(s) of `axes`

One of two forms

```
<axes href="/link/to/another/axes/node"/>
```

or a list of `<axis>` and `<grid>` elements. For example,

```
<axes>
   <axis>...</axis>
   <grid>...</grid></axes>
```

5.1.2 General purpose type: `axis`

This node stores an `index`, `label` and `unit` for an axis.

Specifications for `axis`

Node name: `axis`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:
- **`index`** [`Integer32`, optional] An integer that indicates which independent/dependent axis this `axis` node belongs to as defined in Section 5.1.1.
- **`label`** [`XMLName`, **required**] A unique label
- **`unit`** [`XMLName`, optional] The unit for the axis.

Child nodes: This element has no child nodes

XML Example(s) of `axis`

Example 1: An axes node for multiplicity as a function of energy (i.e. $m(E)$):

```
<axes>
   <axis index="1" label="energy_in" unit="eV"/>
   <axis index="0" label="multiplicity" unit=""/></axes>
```

Example 2: An axes node for $P(\mu|E)$:

```
<axes>
   <axis index="2" label="energy_in" unit="eV"/>
   <axis index="1" label="mu" unit=""/>
   <axis index="0" label="P(mu|energy_in)" unit=""/></axes>
```

5.1.3 General purpose type: `grid`

The `grid` node is like the `axis` but adds a list of values that the function is evaluated at for the independent axis defined by the `grid`. The `grid` has the same three attributes (`index`, label, and unit) as the axis node and adds the attributes style and interpolation. In addition, the grid node has a values child node that contains the list of values the function is evaluated at or parameters the function is specified at as indicated by the style attribute.

Specifications for grid

Node name: grid
Abstract node: axis
Attributes: The list of additional allowed attributes is:

- **index** [Integer32, optional] An integer that indicates which independent/dependent axis this grid node belongs to.
- **interpolation** [interpolation, optional, default is "lin-lin"] Defines the interpolation to be used between consecutive domain points along this axis. Note, this attribute is required when style is points or boundaries and interpolation is other than lin-lin.
- **label** [XMLName, optional] A unique label
- **style** [UTF8Text, optional] A string denoting the type of grid associated with this axis. Allowed values are 'none', 'points', 'boundaries' and 'parameters'.
- **unit** [XMLName, optional] The unit for the axis.

Child nodes: The list of additional allowed child nodes is (only one of them is allowed):

- **values:** [optional, when present, * one of marked children must appear one time] List of values defining the grid.
- **link:** [optional, when present, * one of marked children must appear one time] Link to a list of values defining the grid.

XML Example(s) of grid

Example 1: In this example, all energies E contain the same list of μ values, hence, a grid node can be used for the E and μ axes. An axes node for $P(\mu|E)$ where the x_2 (i.e. E) and x_1 (i.e. μ) have style="points" and the x_2 axis has "log,lin" interpolation:

```xml
<axes>
    <grid index="2" label="energy_in" unit="eV" style="points"
          interpolation="log,lin"><values> ... </values></grid>
    <grid index="1" label="mu" unit="" style="points">
          <values> ... </values></grid>
    <axis index="0" label="P(mu|energy_in)" unit=""/></axes>
```

Example 2: An axes node for $P(\mu|E)$ where $P(\mu|E)$ is given on the x_2 (i.e. E) grid 1e-5, 1e-4, 1e-2, 1, 10 and 20 MeV and x_1 (i.e. μ) is represented with a Legendre expansion of $P(\mu|E)$ for orders 0 to 7:

```xml
<axes>
```

```
    <grid index="2" label="energy_in" unit="MeV" style="points">
      <values> 1e-5 1e-4 1e-2 1 10 20 </values></grid>
    <grid index="1" label="l" style="parameters">
      <values valueType="Integer32"> 0 1 2 3 4 5 6 7 </values></grid>
    <axis index="0" label="P(mu|energy_in)" unit=""/></axes>
```

5.2 List of values

5.2.1 General purpose type: `values`

Data containers have meta-data and data. In most cases the data have one common piece of meta-data, which is the `type` of data. For, this reason, many of the containers store their data in a `values` node where, if needed, `type` is stored. This node also allows for compressing leading and trailing zero data. If the `start` attribute is defined, it specifies the first non-zero data value. If the sum of the `start` and the number of data values is less than the value of the `length` attribute, then the the `length` value represents the total number of data values including the non-specified leading and trailing zeros. All data in this node must be of the same type and separated by one or more white spaces.

Specifications for `values`

Node name: `values`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

`valueType` [`UTF8Text`, optional, default is "Float64"] Specifies the type of data in the body (e.g. Integer32, Float64). Only one type of data can be stored in each instance of a values node.

`start` [`Integer32`, optional, default is "0"] For start="N", the first N values are zero and are not stored.

`length` [`Integer32`, optional] The total number of data values including leading and trailing zero values that are not stored. This attribute should only be used when the sum of `start` and the number of listed values do not add to the total number of data values. This should only happen when there are trailing zeros not listed in the Body text.

`href` [`UTF8Text`, optional] Link to another `values` node or to a dataset in an external file such as HDF5. If the href points to an external file, the link should be of the form 'href="FileID#/path/in/external/file"', where 'FileID' is the label of one of the external files listed in the `externalFiles` section.

`startIndex` [`Integer32`, optional] Optionally used along with an `href` to indicate that a subset of the externally-defined dataset should be used. This option must only be used if an `href` and `count` are also defined. Data from the external dataset between startIndex (inclusive) and startIndex + count (exclusive) are selected.

count [Integer32, optional] Optionally used along with an `href` to indicate that a subset of the externally-defined dataset should be used. This option must only be used if an `href` and `startIndex` are also defined. Data from the external dataset between startIndex (inclusive) and startIndex + count (exclusive) are selected.

Child nodes: This element has no child nodes

Body text: Contains a space-delimited ordered list of data for the parent node, or an href pointing to the data.

XML Example(s) of `values`

Example 1: values node representing the floating point numbers 1.2, 3.2 and 2.3.

```
<values valueType="Float64"> 1.2 3.2 2.3</values>
```

Example 2: `values` node representing the floating point numbers 0, 0, 0, 0, 0, 1.2, 3.2 and 2.3.

```
<values start="5" valueType="Float64"> 1.2 3.2  2.3</values>
```

Example 3: Same as Example 2 but with the default `valueType`.

```
    <values start="5"> 1.2 3.2  2.3</values>
```

Example 4: `values` node representing the floating point numbers 0, 0, 0, 0, 0, -1.2, 7.3, 2.3, -11, 0, 0 and 0.

```
<values start="5" length="12">-1.2 7.3 2.3 -11</values>
```

Example 5: `values` node with an href to an external dataset.

```
<values href="HDF#/externalDataset"/>
```

Example 6: `values` node with an href to a slice of an external dataset.

```
<values href="HDF#/externalDataset" startIndex="1073" count="1022"/>
```

5.3 Arrays

The array node stores numeric data on a multi-dimensional grid. An array of dimension n is a list containing zero or more n-1 dimensional arrays which must all have the same shape (see definitions below). Each n-1 dimensional array in turn contains a list of n-2 dimensional arrays, and so on down to the 0-dimensional array which is a number.

For an n-dimensional array with N_i grid points on the i^{th} dimension (where i ranges from 1 to n), there are $N_\mathrm{n} \times N_{\mathrm{n}-1} \times \cdots \times N_2 \times N_1$ total values of the array. The most commonly used array is the matrix (i.e. the 2-dimensional array).

Arrays are used frequently in nuclear data, whenever the underlying data are stored on a uniform grid. For example, a covariance matrix can be stored in a 2-dimensional array while a transfer matrix (for use in deterministic transport codes) requires a 3-dimensional array. While the simplest way to store an array is to explicitly list all the values, substantial space savings can often be achieved by using various compression strategies. One example where compression is valuable in a nuclear data evaluation is the ^{232}Th resonance parameter covariance matrix, MF=32 MT=151, found in the ENDF-VII.1 (Chadwick et al., 2011) neutron sub-library. The matrix shape is 2781×2781, yet only about 9000 of these values (0.1% of the matrix) are non-zero. If one assumes that each zero is stored using two units of ASCII storage including the separator (i.e. ‘ 0’), and also assume that each non-zero value takes up a total of 24 units, then the zero values will require about $2781 \times 2781 \times 2 \,/\, (9000 \times 24)$, or about 71, times more memory than the non-zero values occupy. It is therefore beneficial that the array data node be designed to allow for storing various compression schemes.

5.3.1 Definitions

shape: the shape of an n-dimensional array is a list of n integers representing the number of values (i.e. length) along each dimension. For example, shape=‘4,3,6’ describes a 3-dimensional array with 4 values along the first dimension, 3 values along second dimension and 6 values along the third dimension. An array with shape=‘5’ represents a 1-dimensional list with indices ranging from 0 to 4 inclusive. The length of the `shape` is equal to the number of dimensions in the array.

size: The size of an array is the total number of values it contains. The size is equal to the product of the length of each dimension: for an array with shape=‘4,6,3’, the size is $4 \times 6 \times 3$ or 72.

storage order: The storage order determines how an array is laid out in contiguous memory. For example, the 2×3 array

$$C = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

could be laid out either with the rows stored one after another as ‘1 2 3 4 5 6’ (this is known as `"row-major"` storage order) or with the columns stored one after another as ‘1 4 2 5 3 6’ (`"column-major"` storage order). More generally, for an n-dimensional array, row-major storage means that the right-most array index (see **array indexing** description below) changes fastest in contiguous memory, followed by the second-to-right-most and so on. Column-major storage means that the left-most index changes fastest, followed by second-to-left-most and so on.

flattened array: Flattening an n-dimensional array means converting it into a 1-dimensional array with the same total number of array elements as the original. The order of those elements depends on the array’s storage order. For example, flattening array C (above) produces a 1-d array with six elements. Those elements are 1 2 3 4 5 6 if the storage order is row-major, or 1 4 2 5 3 6 if the storage order is column-major.

array indexing: A list of integer indices can be used to point to any value in an array, or to a sub-array within an array. For an n-dimensional array, the index list contains from 1 to n distinct indices (e.g. $[i_\mathrm{n}, i_{\mathrm{n}-1}, ..., i_2, i_1]$). Each item in the index list must be a non-negative integer, and must be less than the length of the array along that dimension. Indexing is 0-based. For compactness, arrays are often stored as a list of values with implicit demarcation. For example, an array of shape=‘3,4’ that uses `"row-major"` storage order may store its 12 values as the string ‘v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12’ (here v1 symbolically represents the first value and so on). The first value v1 is indexed as array[0][0], the second value v2 as array[0][1] and the last value v12 as array[2][3]. Indexing this array with only the index 1, as in array[1], yields the 1-dimensional array ‘v5, v6, v7, v8’ of shape=‘4’. This is the same array indexing notation used for the C and Python programming languages.

The node pointed to by an array index does *not* depend on storage order: in the example above, array[1,2] points to v7 no matter whether the array is stored as `"row-major"` or `"column-major"` as the storage order only describes how the data are stored.

Two indexing notations are used and are equivalent. The first notation lists the indices as comma separated values enclosed between ‘[’ and ‘]’. For example, array[3, 2, 4]. The second notation lists each index enclosed between ‘[’ and ‘]’ (this is the notation used in the C programming language). The prior example in this latter notation is array[3][2][4].

storage: An array may contain many values that are 0, and/or it may be symmetric or anti-symmetric about the diagonal. In these cases, the array can often be stored more efficiently using sparse, symmetric and/or diagonal compression. The array node includes optional `"compression"`, `"symmetry"` and `"permutation"` attributes describing the compression strategies (if any) used to reduce storage size.

compression: The following `"compression"` types are defined:

none: every value is listed explicitly (unless the `"symmetry"` attribute is other than ‘none’; see below).

diagonal: if most non-zero values in an array lie on or near the diagonal, the array can often be stored more efficiently by traversing along the direction of the array diagonal. The simplest example is when *all* off-diagonal nodes in the array are zero, so that only diagonal values need to be specified. For an n-dimensional array with N values along each dimension, only the

values with indices $[i_n, i_{n-1}, \ldots, i_2, i_1]$ where $i_1 = i_2 = \ldots = i_n$ are stored. For example, to store a diagonal array with shape = '4,4,4', only the values with indices [0,0,0], [1,1,1], [2,2,2], [3,3,3] need to be stored.
Diagonal storage is made more general by permitting the storage of off-diagonal vectors that lie parallel to the **main** diagonal, along with starting indices for each vector. For arrays whose non-zero values lie near the diagonal[14], diagonal storage can be an optimal compression scheme. For example, the 2-dimensional array

$$C = \begin{array}{cccc} 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 5 & 6 \\ 0 & 0 & 0 & 7 \end{array}$$

can be stored using diagonal compression as two vectors: [1 3 5 7] starting at indices [0,0] and [2 4 6] starting at indices [0,1].
This array can be stored on disc using two `values` nodes. One has its `label` attribute set to `"startingIndices"`, and contains the starting indices with length equal the number of diagonal vectors times the number of array dimensions. The other has no `label` attribute and contains the diagonal vectors with length equals the sum of the lengths of all diagonal vectors. In the example above, the startingIndices are [0 0 0 1] (indicating that data start on indices [0, 0] and [0, 1]), and the values are [1 3 5 7 2 4 6]. For higher dimensions see the examples in Section 5.3.2.
While diagonal compression is most often used for arrays with the same length along each dimension, it can apply to other array shapes as well. For example, if the only non-zero nodes in a 5x3 array are at [0,0], [1,1], [2,2], it could be stored using diagonal compression.

flattened: A general method for storing any sparse n-dimensional array is to start by flattening the array so that there is only one index and then to define three separate `values` nodes as: 1) starting indices in the new flat--tened array, 2) the number of values (`lengths`) that will be given following each starting index, and 3) the list of values, whose length is equal to the sum of the `lengths` values. For example, consider the following matrix:

$$C = \begin{array}{rrrrr} 1 & -1 & 0 & -3 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{array}$$

This matrix can be stored in a flattened `array` using three separate `values` nodes as (assuming row major storage order):

- starts = 0, 12, 21, 24
- lengths = 7, 7, 1, 1
- values = 1, -1, 0, -3, 0, -2, 5, 4, 6, 4, -4, 0, 2, 7, 8, -5

[14]For a 2-d array (i.e. a matrix) this is called a banded matrix.

The first seven items in `values` correspond to the seven nodes starting at index 0 of the flattened array, the next seven items correspond to the seven nodes starting at index 12, and the last two items correspond to nodes 21 and 24 respectively. Together with the `shape` and `storageOrder` attributes, these three `values` nodes completely describe any n-dimensional array. In this example, some zeros appear in the `values` array. This is not required, but if two non-zero values are separated by a single 0 this helps achieve greater compression.

embedded: Some arrays can be efficiently represented by breaking them into multiple sub-arrays. For example, if the original array is

$$C = \begin{matrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 & 10 \\ 0 & 0 & 11 & 12 & 13 \end{matrix} \tag{5.1}$$

it can be represented as two separate blocks:

$$C_1 = \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \quad \text{and} \quad C_2 = \begin{matrix} 5 & 6 & 7 \\ 8 & 9 & 10 \\ 11 & 12 & 13 \end{matrix}$$

where C_1 starts at indices (0,0) in the original array, and C_2 starts at indices (2,2).

The `"embedded"` compression scheme allows decomposing arrays in this fashion. An embedded n-dimensional array with shape = $(M_n, M_{n-1}, ..., M_1)$ contains 1 or more n-dimensional sub-arrays. Each sub-array has its own shape $(m_n, m_{n-1}, ..., m_0)$ along with starting indices $(s_n, s_{n-1}, ..., s_0)$ indicating where it begins in the `"full"`. The shape and starting indices of sub-arrays are constrained such that $0 \leq s_j \leq M_j - m_j$ along each dimension j (in other words, sub-arrays must be small enough to fit inside the full array after being offset by their starting indices).

For an n-dimensional array C containing K sub-arrays (labelled $c_0, c_1 ..., c_{K-1}$), C can be computed as follows:

$$C[i_n, i_{n-1}, ..., i_0] = \sum_{k=0}^{K-1} w(k)\; c_k[i_n - s_n, i_{n-1} - s_{n-1}, ..., i_0 - s_0]$$

where the weight function $w(k)$ is equal to 1 if sub-array k satisfies the condition $0 \leq i_j - s_j \leq m_j$ along each dimension j, or equal to 0 otherwise. Any node of C not covered by any of the sub-arrays is equal to 0. Sub-arrays cannot overlap.

Each sub-array can store data using any of the `"compression"`, `"symmetry"` and `"permutation"` attributes. Also, no restriction is placed on the storage order used by each sub-array. However, each sub-array must contain the same data type as the parent array.

Embedded arrays are quite flexible, and offer many different ways of decomposing an array into constituents. The choice of the best way to decompose any given array is left up to the user.

symmetry: An n-dimensional array is symmetric if

- the value at location [$i_n, i_{n-1}, \cdots, i_2, i_1$] is the same for all permutations of the indices $i_1, i_2, \cdots, i_n$.

For a 2-dimensional array, this reduces to the requirement that

- array[i_2, i_1] = array[i_1, i_2] for all combinations of i1 and i2.

An n-dimensional array is anti-symmetric if the value is multiplied by -1 when indices i1 and i2 are interchanged (for i2 != i1).
For a 2-dimensional array, this reduces to the requirement that

- array[i_2, i_1] = -array[i_1, i_2] for all combinations of i1 and i2 where i2 != i1.

Symmetric and anti-symmetric arrays can be stored in a compact form by only listing one permutation of the indices. Two common ways of storing symmetric arrays are `"symmetric lower-diagonal"`, where only the nodes on and below the main diagonal are stored (i.e. indices with $i_M \geq i_{n-1} \geq ... \geq i_2 \geq i_1$), and `"symmetric upper-diagonal"`, where only nodes on and above the main diagonal are stored (i.e. indices with $i_M \leq i_{n-1} \leq ... \leq i_2 \leq i_1$).
For example, the following is a symmetric array with shape='4,4'. In this example, array indices are listed explicitly:

```
second index --> | 0  1  2  3
first index     ---------------
    |           0| 1  2  4  7
    |           1| 2  3  5  8
   \|/          2| 4  5  6  9
                3| 7  8  9  0
```

This array can be stored using triangular lower-diagonal symmetry as the list 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 (assuming the storage order is `"row-major"`). That is, the lower-diagonal triangle is:

```
1
2  3
4  5  6
7  8  9  0
```

The anti-symmetric array:

```
 0  -1   2
 1   0   1
-2  -1   0
```

can be stored using the upper-diagonal as the list -1, 2, 1 . The zeros on the diagonal are not stored.
The `symmetry` attribute specifies whether only the *lower* or *upper* hyper-triangle is stored. The `permutation` attribute specifies whether the array is symmetric (*permutation="+1"*) or anti-symmetric (*permutation="-1"*). If an array has only non-zero values in the *lower* or *upper* hyper-triangle, it can be stored compactly by setting `symmetry` appropriately and with *permutation="none"*.

5.3.2 General purpose type: `array`

An array consists of attributes indicating the dimension, compression strategy, etc. and a list of one or more `values` nodes that depend on the value of the `compression` attribute. The `label` attribute of each `values` node designates the type of data stored in that node. For the `values` node containing the data of the array, no label shall be specified. The allowed **values** nodes for different compressions are:

compression='none': in this case the array only contains a single `values` node that contains the array data. The `label` attribute of the `values` node shall not be used. The number of data in the `values` node depends on the array shape and the values of the `symmetry` and `storageOrder` attributes.

compression='diagonal': if all off-diagonal nodes are zero, then the diagonal array only needs to contain a `values` node, with size = N (the smallest array dimension). If some off-diagonal nodes are included, then the array contains two `values` nodes. One node has label="startingIndices" and contains the starting indices. The other node contains the array data and its `label` attribute must not be used.

compression='flattened': in this case the array contains three `values` nodes. One node contains a list of starting indices in the flattened array and has `label`="starts". Another node contains the number of values given after each 'start' and has `label`="lengths". The third node contains the array data and its `label` attribute shall not be used. The `starts` and `lengths` both shall have data type Integer32, and must have the same number of values.

compression='embedded': contains 0 or more child `array` nodes. Each child array shall have the same data type as the parent array, must have an `offset` attribute, and must obey size restrictions as described in Section 5.3.1.

Specifications for `array`

Node name: `array`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`shape`** [`IntegerTuple`, **required**] a comma-separated list of integers representing the length of the array along each dimension. The integer type is Integer32.
- **`compression`** [`UTF8Text`, optional, default is "'none'"] A flag indicating whether a sparse storage scheme is being used, and if so which scheme is used. Allowed value is either 'none', 'diagonal', 'flattened' or 'embedded'. Projects can also define their own compression schemes.
- **`symmetry`** [`UTF8Text`, optional, default is "'none'"] Allows for storing values in only the upper- or lower-diagonal hyper-triangle. Allowed values are 'none', 'lower' and 'upper'. If `"permutation"` is 'none', non-stored values are zero. Otherwise, they are determine by the `"permutation"` attribute.
- **`permutation`** [`UTF8Text`, optional, default is "'none'"] If `"symmetry"` is other then 'none', this attribute specifies whether the permutation of two indices is +1 or -1. Allowed values are 'none', '+1' and '-1'.
- **`storageOrder`** [`UTF8Text`, optional, default is "row-major"] Indicates whether the

data are stored in row-major or column-major order (i.e. whether the last or first index is varying the fastest). Allowed values are 'row-major' or 'column-major'.

`offset` [`IntegerTuple`, optional] Required if this array is nested inside an `"embed-ded"` array; not allowed otherwise; Gives the starting indices for a sub-array nested inside an array that uses `"embedded"` compression. A comma-separated list of integers equal in dimension to the parent array. The integer type is Integer32.

Child nodes: The list of additional allowed child nodes is:

`values`: [optional, when present, may appear any number of times] Containers holding the array values. The number of `values` nodes depends on the `com-pression` attribute. For compression of 'embedded', 'none', or 'flattened' the number of `values` nodes are 0, 1, or 3 respectively. For 'diagonal' compression, either 1 or 2 `values` nodes may be present.

`array`: [optional, when present, may appear any number of times] Containers holding sub-arrays embedded in a larger array. The number of `array` nodes depends on the `compression` attribute. For compression of 'embedded', 0 or more are allowed. For compression of 'none', 'diagonal' or 'flattened' no `array` nodes are allowed.

XML Example(s) of `array`

The following are sample arrays showing how data are stored for several combinations of `compression`, `symmetry` and `permutation` attributes. Newlines are used in some of these examples to help the reader visualise the arrays, but newlines are not required:

Example 1: an array used to store 1-dimensional data:

```
<array shape="5">
    <values> 3.14 1.59 2.65 3.589 7.93 </values></array>
```

Example 2: a 2-dimensional array. In this example the `compression` and `symmetry` attributes can be omitted since they are both equal to the defaults:

```
<array shape="4,4" compression="none" symmetry="none">
  <values>
    1.1  2.7  3.6  0.0
    2.7  4.8  0.7  0.2
    3.6  0.7  5.4  0.1
    0.0  0.2  0.1  3.6</values></array>
```

The same data can be stored more compactly by taking advantage of symmetry:

```
<array shape="4,4" compression="none" symmetry="lower"
    storageOrder="row-major">
```

```
      <values>
        1.1
        2.7  4.8
        3.6  0.7  5.4
        0.0  0.2  0.1  3.6</values></array>
```

Example 3:

```
    <array shape="4,4" compression="none" symmetry="none">
      <values>
        2.3    0    0    0
        0    4.6    0    0
        0      0  5.4    0
        0      0    0  6.8</values></array>
```

This can be stored more compactly using the `compression="diagonal"` as:

```
    <array shape="4,4" compression="diagonal">
      <values> 2.3 4.6 5.4 6.8 </values></array>
```

Example 4: 3-dimensional array with most non-zero nodes near the diagonal. Stored first without any compression as:

```
    <array shape="3,3,3">
      <values>
        1 2 0
        0 3 0
        0 0 0

        0 0 0
        0 2 3
        0 0 4

        0 0 0
        0 0 0
        0 0 3</values></array>
```

The same data can also be stored using diagonal compression as:

```
    <array shape="3,3,3" compression="diagonal">
      <values label="startingIndices" dataType="Integer32">
         0 0 0
         0 0 1
         0 1 1 </values>
      <values>
```

```
        1 2 3  <!-- diagonal array starting at 0,0,0 -->
        2 3    <!--             -"-             0,0,1 -->
        3 4    <!--             -"-             0,1,1 --></values></array>
```

Note that the main diagonal (starting at 0,0,0) has three values, while the other diagonals have 2 values.

Example 5: Sparse 3-dimensional array where all non-zero nodes are found in one corner.

```
    <array shape="2,8,15" storageOrder="row-major"
          compression="flattened">
      <values label="starts" dataType="Integer32">97 111 221 235</values>
      <values label="lengths" dataType="Integer32">8 9 4 4</values>
      <values>
       12.2 6.431  0.983   2.121     8.7205 0.511   1.24e-3 4.24e-5
       8.25 2.154  1.232   0.963     3.126  7.82e-2 6.18e-4 9.29e-6 1.23e-8
      2.843 0.9261 6.23e-3 7.4502e-6
      3.126 1.5723 0.3421  4.167e-4</values></array>
```

The same array can also be stored using "embedded" compression:

```
    <array shape="2,8,15" compression="embedded">
      <array shape="1,2,9" offset="0,6,6">
        <values>
         0.0  12.2   6.431   0.983    2.121  8.7205  0.511   1.24e-3 4.24e-5
         8.25 2.154  1.232   0.963    3.126  7.82e-2 6.18e-4 9.29e-6 1.23e-8
            </values></array>
      <array shape="1,2,5" offset="1,6,10">
        <values>
         0.0   2.843  0.9261 6.23e-3  7.4502e-6
         3.126 1.5723 0.3421 4.167e-4 0.0 </values></array></array>
```

5.4 Tables

The **table** node stores spreadsheet-style data as a list of M rows by N columns. The table node is like a 2-dimensional array of shape ”M,N”. However, there are several differences between a 2-dimensional array and a table:

- a table allows more data types; in particular, it allows for non-numeric data types
- a table allows for an empty cell
- a table allows for mixed data types
- sparse representations for tabular data are not supported.

For a table, the intersection of a row and a column is called a cell and stores a single datum. A datum can be a Integer32, Float64, string, empty (indicated by ‘<td/>’), valid

XML text or any other type defined by a project. Each column has a header that consists of a column index, label, a unit and optionally a dataType.

Each cell can contain one of the following five data types (or other types defined by a project):

Integer32: Any valid Integer32 value. For ASCII, the data are stored without quotes (e.g. 3245 and not "3245").

Float64: Any valid Float64 values. For ASCII, the data are stored without quotes (e.g. 3.245e-4 and not "3.245e-4").

string: Any string enclosed by the xml start node '<td>' and its end node '</td>'. For example,
<td>**abcd**</td>
or
<td>**Even spaces and quotes (i.e. " and ') are allowed.**</td>.

empty: The string <**td**/> is used to denote a cell that is empty (i.e. has no value). Note that while <td/> is valid XML text, it is treated here as an independent type.

Boolean values: The Boolean values for true and false are '<true/>' and '<false/>' re--spectively. While these are XML text, they are treated as an independent type.

XML text: This is any valid XML text that starts with the <**td**> node and ends with the </**td**> node. Valid XML text must have matching start/end nodes. An example of valid XML text is:

```
        <td><area><width value="12.3" unit="cm"/>
            <height value="1.3"><unit>mm</unit></height></area></td>
```

An example of invalid XML text is:

```
        <td><area><width value="12.3" unit="cm"/>
            <height value="1.3"><unit>mm</unit></area></td>
```

The latter example is missing the matching end node for the "<height>" start node. Another example shows valid XML that is however not a valid table entry:

```
        <area><width value="12.3" unit="cm"/>
            <height value="1.3"><unit>mm</unit></height></area>
```

This example is the same as first example, but is missing the <td> start and </td> end nodes.

5.4.1 General purpose type: `table`

Specifications for `table`

Node name: `table`

Attributes: The list of additional allowed attributes is:
`columns` [`Integer32`, **required**] The number of columns of the table.
`rows` [`Integer32`, **required**] The number of rows of the table.

`storageOrder` [`XMLName`, optional, default is “row-major”] Allowed value is one of ‘row-major’ and ‘column-major’

Child nodes: The list of additional allowed child nodes is:

`columnHeaders`: [**required**, must appear one time] Contains a list of N ‘header’ nodes where N is the number of columns.

`data`: [**required**, must appear one time] The data for the table, consisting of M $\times$ N cells.

XML Example(s) of `table`

```
<table
        columns="..."
        rows="..."
        storageOrder="...">
    <columnHeaders>...</columnHeaders>
    <data>...</data></table>
```

5.4.2 General purpose type: `columnHeaders`

Contains a list of N ‘header’ nodes where N is the number of columns.

Specifications for `columnHeaders`

Node name: `columnHeaders`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`column`: [**required**, must appear at least one time] Container which holds the name, units, etc. describing a column of data.

XML Example(s) of `columnHeaders`

```
<columnHeaders>
    <column>...</column></columnHeaders>
```

5.4.3 General purpose type: `column`

Container which holds the attributes that describe a column of data including the name, units, etc.

Specifications for `column`

Node name: `column`
Attributes: The list of additional allowed attributes is:

- `index` [`Integer32`, **required**] An integer value in the range 0 to (N-1) that represents the order of the column.
- `name` [`XMLName`, **required**] A string representing the name of the column.
- `unit` [`XMLName`, optional] The unit of the data in the column.
- `types` [`XMLName`, optional] Comma separated list of valid GNDS types denoting what types of elements are allowed in each cell in the column.

Child nodes: This element has no child nodes

XML Example(s) of `column`

```
<column
        index="..."
        name="..."
        unit="..."
        types="...">
</column>
```

5.4.4 General purpose type: `data`

The data for the table, consisting of M $\times$ N cells.

Specifications for `data`

Node name: `data`
Attributes: The list of additional allowed attributes is:

- `sep` [`UTF8Text`, optional, default is “whiteSpace”] Valid options are “whiteSpace”, “,”, “td”, “tr” or “tc”. The standard rules for “whiteSpace” and “comma” separators apply. The other separator options are described below.

Child nodes: This element has no child nodes
Body text: The layout of the table data depends on the values of the `storageOrder` and `sep` attributes.

- **sep=‘whiteSpace’ or ‘,’** In this case the table has no child node, and the body of the table consists of a **sep**-separated list containing M $\times$ N values. The order of data depends on the `storageOrder`. This option can only be used if all data in the table are numeric types.
- **sep=‘td’** In this case every cell in the table is explicitly contained inside ‘<td>’ and ‘</td>’ (or ‘<td/>’, to indicate an empty cell). The order of data depends on the `storageOrder`. This option can handle multiple data types, but if only a few cells contain non-numeric data the following two options may be more concise.

storageOrder='row-major', sep='tr' In this case each row of the table is stored inside '<tr sep="..."> ' and '</tr>'. Each row can define its own separator, which may be 'whiteSpace' (the default), 'td' or ','. Any row containing non-numeric data must have 'sep="td"'.

storageOrder='column-major', sep='tc' In this case each column of the table is stored inside '<tc sep="...">' and '</tc>'. Each column can define its own separator, which may be 'whiteSpace' (the default), 'td' or ','. Any column con--taining non-numeric data must have 'sep="td"'. This storage is recommended whenever some columns contain only numeric data while others contain mixed data types, since it helps reduce the size of the numeric columns.

XML Example(s) of `data`

```
<data
        sep="...">
    <![CDATA[SAMPLE BODY TEXT]]></data>
```

5.4.5 Detailed `table` Examples

- In GNDS, resonance parameters are stored in a table. In the resolved region, each row is one resonance, and the columns store the energy and partial widths of the resonance.
  A sample table storing nuclear resonance parameters might look like the following:

  ```
      <table columns="4" rows="4">
        <columnHeaders>
          <column index="0" label="energy" units="eV"/>
          <column index="1" label="gamma + Cl36 width" units="eV"/>
          <column index="2" label="n+Cl35 width" units="eV">
          <column index="3" label="H1+S35 width" units="eV">
        </columnHeaders>
        <data>
            54932.0      0.36726    46.4424       0.0
            68236.16     0.39336    217.904     1e-05
            115098.0     0.739      4.30778       0.0
            182523.0     0.74515    1759.74       0.4
        </data>
      </table>
  ```

  In this example, each row corresponds to a single resonance, and each column corresponds to one parameter (such as the central energy, width, spin, etc.) of that resonance.
- In EXFOR, experimental data and uncertainties are stored in tables. Each row typically corresponds to the measurement at a single incident energy, and columns contain information like cross section, ratio to standard, and uncertainty.

6. Functional container nodes

6.1 One dimensional functionals (e.g. `XYs1d`)

6.1.1 General purpose type: `XYs1d`

This node stores a single-valued function $x_0(x_1)$ (i.e. $y(x)$) as a tabulated list of (x_i, y_i) pairs with $x_i < x_{i+1}$. It may appear on its own (i.e. when storing an energy-dependent cross section or multiplicity), or it may appear inside of another functional data con--tainer such as `regions1d` or `XYs2d`.

Specifications for `XYs1d`

Node name: `XYs1d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:

- **`index`** [`Integer32`, optional] Integer index, used when the `XYs1d` appears inside a `regions1d`.
- **`interpolation`** [`interpolation`, optional, default is "lin-lin"] The rule for interpo--lating y along the x axis.
- **`label`** [`XMLName`, optional] Unique label. When the `XYs1d` appears as a form, the label indicates the associated style.
- **`outerDomainValue`** [`Float64`, optional] When an XYs1d appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.

Child nodes: The list of additional allowed child nodes is:

- **`axes`:** [optional, when present, must appear one time] The `axes` element describing the x and y directions.
- **`uncertainty`:** [optional, when present, must appear one time] Uncertainty and/or covariance for this one-dimensional function.
- **`values`:** [**required**, must appear one time] Contains the list of the n pairs of x_i, y_i as the list $x_0\ y_0\ x_1\ y_1\ ...\ x_{n-1}\ y_{n-1}$.

XML Example(s) of `XYs1d`

Example 1: A tabulated $y(x)$ with (x, y) points (1e-5, 0), (1e-3, 0.12), (0.1, 2e-5), (1, 3.14e-5) and (2e7, 4.3e-12).

```
<XYs1d interpolation="lin-lin">
    <values>
        1e-5 0  1e-3 0.12  0.1 2e-5  1 3.14e-5  2e7 4.3e-12</values></XYs1d>
```

Example 2: Same as Example 1, but with axes specified, and default value for interpo--lation.

```
<XYs1d>
    <axes>
       <axis index="1" label="energy_in" unit="eV"/>
       <axis index="0" label="cross section" unit="b"/></axes>
    <values>
        1e-5 0  1e-3 0.12  0.1 2e-5  1 3.14e-5  2e7 4.3e-12</values></XYs1d>
```

6.1.2 General purpose type: `Ys1d`

This node stores a tabulated representation of the single-valued function $x_0(x_1)$. This is like the `XYs1d` node (see Section 6.1.1) except that the independent axis must be a `grid` node (see Section 5.1.3) that either stores a list of x_i values or links to another grid that stores the list. This node is useful when many tabulated $x_0(x_1)$'s contain the same x_i points. Only one of the `Ys1d` node needs to store the x_i values and the others can link to it, allowing for reduced redundancy and storage. The x_i values must statisfy $x_i < x_{i+1}$. The y_i are stored in the `values` child node.

Specifications for `Ys1d`

Node name: `Ys1d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:
- **`interpolation`** [`interpolation`, optional, default is "lin-lin"] The rule for interpo--lating y rule along the x axis.
- **`label`** [`XMLName`, optional] Unique label. When the `Ys1d` appears as a form, the label indicates the associated style.

Child nodes: The list of additional allowed child nodes is:
- **`axes`:** [**required**, must appear one time] The `axes` element describing the x and y directions.
- **`values`:** [**required**, must appear one time] Contains the list of y_i values.

XML Example(s) of `Ys1d`

Example 1: A tabulated $y(x)$ with (x, y) points (1e-5, 0), (1e-3, 0.12), (0.1, 2e-5), (1, 3.14e-5) and (2e7, 4.3e-12), and another tabulated $y(x)$ with (x, y) points (1e-5, 0), (1e-3, 0.2), (0.1, 3.1), (1, 0.12) and (2e7, 6.3e-2).

```
<data>
    <Ys1d label="f1" interpolation="lin-lin">
        <axes>
            <grid index="1" label="energy_in" unit="eV">
                <values> 1e-5 1e-3 0.1 1 2e7 </values></grid>
            <axis index="0" label="cross section" unit="b"/></axes>
        <values> 0  0.12  2e-5  3.14e-5  4.3e-12</values></Ys1d>
    <Ys1d label="f2" interpolation="lin-lin">
        <axes>
            <grid index="1" href="/data/XYs1d[@label=`f1']/axes/grid"/>
            <axis index="0" label="cross section" unit="b"/></axes>
        <values> 0  0.12  2e-5  3.14e-5  4.3e-12</values></Ys1d></data>
```

6.1.3 General purpose type: `xs_pdf_cdf1d`

For Monte-Carlo sampling it is often useful to store both the probability density (pdf) and cumulative probability density (cdf) on the same grid. The sampling routine can then draw a random number in the range [0,1), do a bisection search through the cdf to determine which interval the random number lies in, and then use the pdf to sample within that interval. The `xs_pdf_cdf1d` container provides a way to store these three related quantities together. The number of values in the `xs`, `pdf` and `cdf` must be the same.

Specifications for `xs_pdf_cdf1d`

Node name: `xs_pdf_cdf1d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:

`outerDomainValue` [`Float64`, optional] When the xs_pdf_cdf1d container appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.

Child nodes: The list of additional allowed child nodes is:

`xs`: [**required**, must appear one time] Defines the common list of x values for the pdf and cdf.

`pdf`: [**required**, must appear one time] Stores the pdf evaluated at each corre--sponding x value.

`cdf`: [**required**, must appear one time] Stores the cdf evaluated at each corre--sponding x value.

XML Example(s) of `xs_pdf_cdf1d`

```
<xs_pdf_cdf1d
        outerDomainValue="...">
    <xs>...</xs>
    <pdf>...</pdf>
    <cdf>...</cdf></xs_pdf_cdf1d>
```

6.1.4 General purpose type: `xs`

Stores the common list of x values for use in an `xs_pdf_cdf1d`.

Specifications for `xs`

Node name: `xs`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `values`: [**required**, must appear one time]

XML Example(s) of `xs`

```
<xs>
    <values>...</values></xs>
```

6.1.5 General purpose type: `pdf_in_xs_pdf_cdf1d`

Stores the value of the pdf at each x in an `xs_pdf_cdf1d`.

Specifications for `pdf_in_xs_pdf_cdf1d`

Node name: `pdf_in_xs_pdf_cdf1d`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `values`: [**required**, must appear one time]

XML Example(s) of `pdf_in_xs_pdf_cdf1d`

```
<pdf_in_xs_pdf_cdf1d>
    <values>...</values></pdf_in_xs_pdf_cdf1d>
```

6.1.6 General purpose type: `cdf`

Stores the value of the cdf at each x in an `xs_pdf_cdf1d`.

Specifications for `cdf`

Node name: `cdf`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `values`: [**required**, must appear one time]

XML Example(s) of `cdf`

```
<cdf>
    <values>...</values></cdf>
```

6.2 One dimensional series

These nodes store a function $x_0(x_1)$ (i.e. $y(x)$) as coefficients of a polynomial sequence. That is, for coefficients C_i and polynomial sequence $P_i(x)$, the function $y(x)$ is defined as

$$y(x) = \sum_{i=\text{lowerIndex}}^{\text{upperIndex}} C_i\, P_i(x) \tag{6.1}$$

Predefined polynomial sequences are listed in Table 6.1.

Table 6.1: List of predefined polynomial sequences.

Name	defaults			series
	lower-Index	domain-Min	domain-Max	
polynomial1d	0	$-\infty$	∞	$y(x) = \sum_{i=\text{lowerIndex}}^{\text{upperIndex}} C_i\, x^i$
Legendre	0	-1	1	$y(x) = \sum_{i=\text{lowerIndex}}^{\text{upperIndex}} C_i\, P_i(x)$ where $P_i(x)$ is the Legendre polynomial of order i.
constant1d	0	$-\infty$	∞	$y(x) =$ const

This is a 1-dimensional functional $x_0(x_1)$ container (i.e. it is a representation of a function of the form $x_0(x_1)$ or $y(x)$).

All one-dimensional series have the same outline:

```
<series1d label=""
          outerDomainValue=""
          lowerIndex=""
          domainMin=""
          domainMax=""> . . . </series1d>
```

6.2.1 General purpose type: `Legendre`

Specifications for `Legendre`

Node name: `Legendre`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, optional] Unique label
- **`outerDomainValue`** [`Float64`, optional] When a Legendre series appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.
- **`lowerIndex`** [`Integer32`, optional, default is "0"] Used if the first `lowerIndex` coeffi--cients are all zero. The upperIndex is equal to lowerIndex plus the number of coefficients in the `values` node, so it is not stored.
- **`domainMin`** [`Float64`, optional, default is "-1"] $y(x_1)$ is only valid (i.e. defined) over the domain `domainMin` $\leq x_1 \leq$ `domainMax`. `domainMin` must be less than or equal to `domainMax`.
- **`domainMax`** [`Float64`, optional, default is "1"] See `domainMin`.

Child nodes: The list of additional allowed child nodes is:

- **`values`:** [**required**, must appear one time] The specifications of the `values` node are given in Section 5.2.1. The numeric values are the coefficients C_i listed by consecutive order i starting with `lowerIndex`.

XML Example(s) of `Legendre`

```
<Legendre>
    <values> 1 0.1 -0.02 1.3e-5</values></Legendre>
```

6.2.2 General purpose type: `polynomial1d`

Specifications for `polynomial1d`

Node name: `polynomial1d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:

`label` [`XMLName`, optional] Unique label

`outerDomainValue` [`Float64`, optional] When a polynomial1d appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.

`lowerIndex` [`Integer32`, optional, default is “0”] Used if the first `lowerIndex` coeffi--cients are all zero. The upperIndex is equal to lowerIndex plus the number of coefficients in the `values` node, so it is not stored.

`domainMin` [`Float64`, **required**] $y(x_1)$ is only valid (i.e. defined) over the domain `do--mainMin` $\leq x_1 \leq$ `domainMax`. `domainMin` must be less than or equal to `domainMax`.

`domainMax` [`Float64`, **required**] See `domainMin`.

Child nodes: The list of additional allowed child nodes is:

`axes`: [**required**, must appear one time] The `axes` element describing the x and y directions.

`uncertainty`: [optional, when present, must appear one time] Uncertainty and/or covariance for this one-dimensional function.

`values`: [**required**, must appear one time] The specifications of the `values` node are given in Section 5.2.1. The numeric values are the coefficients C_i listed by consecutive order i starting with lowerIndex.

XML Example(s) of `polynomial1d`

The polynomial series

$$f(x) = 1 + 0.1x - 0.02x^2 + 1.3 \times 10^{-5}x^4 \tag{6.2}$$

can be represented as

```
<polynomial1d>
    <values> 1 0.1 -0.02 0 1.3e-5</values></polynomial1d>
```

6.2.3 General purpose type: `constant1d`

Describes a constant one-dimensional function, $f(x) = C$.

Specifications for `constant1d`

Node name: `constant1d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:

`value` [`Float64`, optional] The value of the data.

`label` [`XMLName`, optional] Unique label

`outerDomainValue` [`Float64`, optional] When a constant1d appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.

domainMin [Float64, **required**] $y(x_1)$ is only valid (i.e. defined) over the domain `domainMin` $\leq x_1 \leq$ `domainMax`. `domainMin` must be less than or equal to `domainMax`.

domainMax [Float64, **required**] See `domainMin`.

Child nodes: The list of additional allowed child nodes is:

axes: [**required**, must appear one time] The `axes` element describing the x and y directions.

XML Example(s) of `constant1d`

```
<constant1d
        value="..."
        label="..."
        outerDomainValue="..."
        domainMin="..."
        domainMax="...">
    <axes>...</axes></constant1d>
```

6.3 Multi-dimensional functionals (e.g. `XYsNd` with $N > 1$)

These nodes store the n-dimensional function $x_0(x_n, ..., x_1)$ for $n > 1$ as a list of (n-1)-dimensional functions $x_0(x_{n-1}, ..., x_1)$. Each (n-1)-dimensional function stores its associated x_n value in its **value** attribute. The embedded $x_0(x_{n-1}, ..., x_1)$ nodes are sorted by increasing x_n with $x_{n,i} < x_{n,i+1}$ (i.e. each x_n must be unique and ascending).

The outline for such containers is

```
 <XYsNd label=""
        outerDomainValue=""
        interpolation=""
        interpolationQualifier=""> . . . </XYsNd>
```

Within an `XYsNd` node, the list of (n-1)-dimensional functions are collected inside a `function#ds` node (where '#' = n-1) as described below. These are not the same as `regions#d` nodes as will be described in sections 6.4.1, 6.4.2, etc. To illustrate the difference, contrast a `function1ds` node with a `regions1d` node:

- `regions1d` can be used to store a 1-dimensional function like a cross section or multiplicity. The `functions1d` node *only* appears inside a `XYs2d` container.
- When `regions1d` nodes appear inside a higher-dimensional container, each one has an associated outerDomainValue. Each one is the one-dimensional curve at one point inside the higher-dimensional space. The functions1d does not have an outerDomainValue, instead it contains the list of one-dimensional functions.

In the specifications below, x_0 = dependent variable, x_1 = lowest-dimension independent variable, x_2 = next-lowest-dimension independent variable, etc.

6.3.1 General purpose type: `function1ds`

This node stores a list of 1-dimensional functions.

Specifications for `function1ds`

Node name: `function1ds`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`functional`:** [optional, when present, must appear at least one time] Any valid 1-dimensional function that is consistent with the quantity being represented.

XML Example(s) of `function1ds`

```
<function1ds>
    <functional>...</functional></function1ds>
```

6.3.2 General purpose type: `function2ds`

This node stores a list of 2-dimensional functions.

Specifications for `function2ds`

Node name: `function2ds`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`functional`:** [optional, when present, must appear at least one time] Any valid 2-dimensional function that is consistent with the quantity being represented.

XML Example(s) of `function2ds`

```
<function2ds>
    <functional>...</functional></function2ds>
```

6.3.3 General purpose type: `function3ds`

This node stores a list of 3-dimensional functions.

Specifications for `function3ds`

Node name: `function3ds`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`functional`:** [optional, when present, must appear at least one time] Any valid 3-dimensional function that is consistent with the quantity being represented.

XML Example(s) of `function3ds`

```
<function3ds>
    <functional>...</functional></function3ds>
```

6.3.4 General purpose type: `XYs2d`

This node stores a 2-dimensional function $x_0(x_2, x_1)$ as a list of 1-dimensional functions at various x_2 values. Each 1-dimensional function stores its x_2 point in its `value` attribute. The attributes `interpolation` and `interpolationQualifier` instruct how to interpolate x_2 and x_1 between two adjacent 1-dimensional functions.

Specifications for `XYs2d`

Node name: `XYs2d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:

- **`index`** [`Integer32`, optional] Integer index, required when the `XYs2d` appears inside a `regions2d`.
- **`interpolation`** [`interpolation`, optional, default is “lin-lin”] The rule for interpo--lating x_0 along the x_2 axis.
- **`interpolationQualifier`** [`XMLName`, optional] The interpolation qualifier.
- **`outerDomainValue`** [`Float64`, optional] When an XYs2d appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.

Child nodes: The list of additional allowed child nodes is:

- **`axes`:** [optional, when present, must appear one time]
- **`function1ds`:** [**required**, must appear one time] Contains the list of 1-dimensional functions $x_0(x_1)$ that represent $x_0(x_2, x_1)$ at various points along x_2.
- **`uncertainty`:** [optional, when present, must appear one time] Uncertainty and/or covariance for this two-dimensional function.

XML Example(s) of `XYs2d`

As an example, for an $x_0(x_2, x_1)$ function for $P(\mu|E)$, the XML representation could look like:

```
<XYs2d interpolation="lin-lin">
    <axes>
        <axis index="2" label="energy_in" unit="eV"/>
        <axis index="1" label="mu"/>
        <axis index="0" label="pdf(mu|energy_in)"/></axes>
    <function1ds>
        <XYs1d outerDomainValue="1e-05" interpolation="lin-lin">
            <values> ... </values></XYs1d>
        <XYs1d outerDomainValue="1e-01" interpolation="lin-lin">
            <values> ... </values></XYs1d>
        <XYs1d outerDomainValue="2e7" interpolation="lin-lin">
            <values> ... </values></XYs1d></function1ds>
    <uncertainty> ... </uncertainty></XYs2d>
```

6.3.5 General purpose type: `XYs3d`

This node stores a 3-dimensional function $x_0(x_3, x_2, x_1)$ as a list of 2-dimensional func--tions at various x_3 values. Each 2-dimensional function stores its x_3 point in its `value` attribute. The attributes `interpolation` and `interpolationQualifier` instruct how to interpolate x_3 and x_0 between two adjacent 2-dimensional functions.

Specifications for `XYs3d`

Node name: `XYs3d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:

- **`interpolation`** [`interpolation`, optional, default is "lin-lin"] The rule for interpo--lating x_0 along the x_3 axis.
- **`interpolationQualifier`** [`XMLName`, optional] The interpolation qualifier.

Child nodes: The list of additional allowed child nodes is:

- **`axes`:** [optional, when present, must appear one time]
- **`function2ds`:** [**required**, must appear one time] Contains the list of 2-dimensional functions that represent $x_0(x_3, x_2, x_1)$ at various points along x_3 as a $x_0(x_2, x_1)$ functions.
- **`uncertainty`:** [optional, when present, must appear one time] Uncertainty and/or covariance for this three-dimensional function.

XML Example(s) of `XYs3d`

An example for the XML representation of a $x_0(x_3, x_2, x_1)$ function for $P(\mu, E'|E)$ could look like the following:

```
<XYs3d interpolation="log-log">
  <axes>...</axes>
  <function2ds>
    <XYs2d outerDomainValue="1e-5" interpolationQualifier="unitBase">
      <function1ds>
        <XYs1d  outerDomainValue="-1" interpolation="lin-log">
          <values> ... </values></XYs1d>
        <XYs1d outerDomainValue="-0.5" interpolation="lin-log">
          <values> ... </values></XYs1d>
        ...
        <XYs1d outerDomainValue="1" interpolation="lin-log">
          <values> ... </values></XYs1d></XYs2d></function1ds>
    ...
    <XYs2d outerDomainValue="2e7" interpolationQualifier="unitBase">
      <function1ds>
        <XYs1d outerDomainValue="-1" interpolation="lin-log">
          <values> ... </values></XYs1d>
        <XYs1d outerDomainValue="-0.5" interpolation="lin-log">
          <values> ... </values></XYs1d>
        ...
        <XYs1d outerDomainValue="1" interpolation="lin-log">
          <values> ... </values></XYs1d></XYs2d></function1ds></function2ds>
    <uncertainty> ... </uncertainty></XYs3d>
```

In this example, the `interpolationQualifier` attributes modify how x_0 is interpo-lated along x_3. The “log-log” interpolation rule applies to x_0 along the x_3 axis, and the “lin-log” rules apply to x_0 along the x_1 axis.

6.4 Multi-dimensional functionals broken up into different re-gions

These nodes store an n-dimensional function as a list of adjoining n-dimensional functions where each function represents a different region of the highest dimensional axis. The dependent values for the adjoining functions can be discontinuous and their interpolation rules can be different. Here, the word **adjoining** means that for the highest dimensional axis, the upper domain boundary for region i must be the same as the lower domain boundary for region $i + 1$. That is, there can be neither gaps nor overlaps in the domain.

The outline of a general `regions` container is as follows:

```
<regions label=""
        outerDomainValue=""> . . . </regions>
```

6.4.1 General purpose type: `regions1d`

Specifications for `regions1d`

Node name: `regions1d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, optional] A unique label
- **`outerDomainValue`** [`Float64`, optional] When an `regions1d` appears inside a higher-dimensional container, the `outerDomainValue` corresponds to the next higher dimension.

Child nodes: The list of additional allowed child nodes is:
- **`axes`**: [optional, when present, must appear one time]
- **`function1ds`**: [**required**, must appear one time] Contains the list of 1-dimensional functions that represent $x_0(x_1)$. The upper x_1 value for a function in the list must be equal to the lower x_1 value for the next function in the list.
- **`uncertainty`**: [optional, when present, must appear one time] Uncertainty and/or covariance for this one-dimensional function.

XML Example(s) of `regions1d`

```
<regions1d
        label="..."
        outerDomainValue="...">
    <axes>...</axes>
    <function1ds>...</function1ds>
    <uncertainty>...</uncertainty></regions1d>
```

6.4.2 General purpose type: `regions2d`

Specifications for `regions2d`

Node name: `regions2d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, optional] A unique label

outerDomainValue [Float64, optional] When a regions2d appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.

Child nodes: The list of additional allowed child nodes is:

axes: [optional, when present, must appear one time]

function2ds: [**required**, must appear one time] Contains the list of 2-dimensional functions that represent $x_0(x_2, x_1)$. The upper x_2 value for a function in the list must be equal to the lower x_2 value for the next function in the list.

uncertainty: [optional, when present, must appear one time] Uncertainty and/or covariance for this two-dimensional function.

XML Example(s) of `regions2d`

The following is an example for the use of the `regions2d` node representing a function $P(E'|E)$ (an $x_0(x_2, x_1)$ function).

```xml
<regions2d>
  <axes href="/templates/axes/energy">
  <function2ds>
    <XYs2d interpolation="lin-lin" index="0">
      <function1ds>
        <XYs1d outerDomainValue="1e-05">
          <values> ... </values></XYs1d>
        ...
        <XYs1d outerDomainValue="2">
          <values> ... </values></XYs1d></function1ds></XYs2d>
    <XYs2d interpolation="log-log" index="1">
      <function1ds>
        <regions1d outerDomainValue="2">
          <function1ds>
            <XYs1d index="0"><values> ... </values></XYs1d>
            <XYs1d interpolation="log-log" index="1">
              <values> ... </values></XYs1d>
            <XYs1d index="2"><values> ... </values></XYs1d></function1ds></regions1d>
        ...
        <XYs1d outerDomainValue="1e6">
          <values> ... </values></XYs1d></function1ds></XYs2d>
    <XYs2d index="2">
      <function1ds>
        <XYs1d outerDomainValue="1e6"><values> ... </values></XYs1d>
        ...
        <XYs1d outerDomainValue="2e7">
          <values> ... </values></XYs1d></function1ds></XYs2d></function2ds>
    <uncertainty> ... </uncertainty></regions2d>
```

In this example, `"lin-lin"` is used for the incident energy axis (i.e. x_2) for $10^{-5} \leq x_2 \leq 2$ and $10^6 \leq x_2 \leq 2 \times 10^7$ (i.e. within the first and last `XYs2d` nodes), and `"log-`

log" interpolation is used for $2 \leq x_2 \leq 10^6$ (i.e. within the middle `XYs2d`). The mid--dle `XYs2d` node possesses a `regions1d` node. Furthermore, `"log-log"` interpolation is used for the outgoing energy axis (i.e. x_1) inside the /regions2d/XYs2d[@index=``1''] /regions1d[@outerDomainValue=``2'']/XYs1d[@index=``1''] node.

6.4.3 General purpose type: `regions3d`

Specifications for `regions3d`

Node name: `regions3d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, optional] A unique label
- **`outerDomainValue`** [`Float64`, optional] When a regions3d appears inside a higher-dimensional container, the outerDomainValue corresponds to the next higher dimension.

Child nodes: The list of additional allowed child nodes is:
- **`axes`**: [optional, when present, must appear one time]
- **`function3ds`**: [**required**, must appear one time] Contains the list of 3-dimensional functions that represent $x_0(x_3, x_2, x_1)$. The upper x_3 value for a function in the list must be equal to the lower x_3 value for the next function in the list.
- **`uncertainty`**: [optional, when present, must appear one time] Uncertainty and/or covariance for this three-dimensional function.

XML Example(s) of `regions3d`

```
<regions3d
        label="..."
        outerDomainValue="...">
    <axes>...</axes>
    <function3ds>...</function3ds>
    <uncertainty>...</uncertainty></regions3d>
```

6.5 Multi-dimensional functionals given on a regular grid

An n-dimensional **gridded** node stores tabulated data representing a function of n independent variables (i.e. $x_0(x_n, ..., x_1)$) where the data are given on a grid for each independent axis. These nodes are composed of an n-dimensional array node and an **axes** node composed of n **grid** nodes for each of the independent axis plus an additional **axis** node for the dependent axis. The **style** attribute for each independent **grid** shall be either *points*, *boundaries* or *parameters* and each **grid** shall contain a **values** node whose data have the following interpretation:

points: a list of ascending x_i values where the function is evaluated. Behavior of the dependent value (i.e. x_0) between consecutive x_i values is determined by the interpolation for that axis.

boundaries: a list of ascending x_i values where the function is constant between two consecutive values. No interpolation shall be supplied for this style as it is always `"flat"`.

parameters: each value corresponds to a coordinate for a polynomial sequence. For example, the parameters may be Legendre orders (possibly starting with the $l > 0$). No interpolation shall be supplied for this style.

For independent axis i that is represented with a **grid** node, and with its **values** node having size N (i.e. when leading and trailing zeros are included, its has N values), the length N_i of dimension i in the array depends on the **style** attribute for that axis (i.e. dimension). For *points* and *parameters* styles, $N_i = N$. For the *boundaries* style, $N_i = N - 1$.

An example of a use for a gridded node in nuclear data is the covariance matrix, where each matrix node stores, for example, the covariance between a cross section at two different ranges of incident particle energies. The array of covariances by itself is insufficient to fully describe the data; the list of incident energy ranges is also needed.

The outline of a general `gridded` container is as follows:

```
<gridded label="">
        . . . </gridded>
```

6.5.1 General purpose type: `gridded1d`

A one-dimensional function defined on a grid or set of groups.

Specifications for `gridded1d`

Node name: `gridded1d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:
`label` [`XMLName`, optional] A unique identifier.
Child nodes: The list of additional allowed child nodes is:
`array`: [**required**, must appear one time] An `array` node representing the x_0 values.
`axes`: [**required**, must appear one time] An `axes` element containing the `grid` elements.

XML Example(s) of `gridded1d`

```
<gridded1d
       label="...">
```

```
    <array>...</array>
    <axes>...</axes></gridded1d>
```

6.5.2 General purpose type: `gridded2d`

A two-dimensional function defined on a grid or set of groups.

Specifications for `gridded2d`

Node name: `gridded2d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:
`label` [`XMLName`, optional] An unique identifier.
Child nodes: The list of additional allowed child nodes is:
`array`: [**required**, must appear one time] An `array` node representing the x_0 values.
`axes`: [**required**, must appear one time] An `axes` element containing the `grid` elements for the independent axes and an axis node for the dependent axis.

XML Example(s) of `gridded2d`

In nuclear data a common use for a gridded node is for storing covariance matrix data. If the rows and columns of the matrix correspond to the same ranges of incident energy, the second axis grid can link to the first (i.e. it can use the same `boundaries` defined in the first):

```
<gridded2d>
    <axes>
      <grid index="2" label="rows" unit="eV" sytle="boundaries">
          <values> 1e-5 1e5 2e+7</values></grid>
      <grid index="1" label="columns" href="grid[@index='2']"/>
      <axis index="0" label="covariances" unit=""/></axes>
    <array shape="2,2" symmetry="lower">
      <values>
        0.23
        0.09 0.18</values></array></gridded2d>
```

6.5.3 General purpose type: `gridded3d`

A three-dimensional function defined on a grid or set of groups.

Specifications for `gridded3d`

Node name: `gridded3d`
Abstract node: `functional`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, optional] An unique identifier.
Child nodes: The list of additional allowed child nodes is:
  `array`: [**required**, must appear one time] An `array` node representing the x_0 values.
  `axes`: [**required**, must appear one time] An `axes` element containing the `grid` elements.

XML Example(s) of `gridded3d`

Another example is the transfer matrix used in deterministic transport codes. In this example, the transfer matrix is stored as a 3-d array where the array dimensions correspond to (in order): incident energy group 'energy_in', outgoing energy group 'energy_out', and Legendre order l. The first two axis (corresponding to incident and outgoing energies) are `boundaries`-style grids, while the third independent axis (cor- -responding to l) is a `parameters`-style grid. No interpolation is allowed along any of the independent axes in this example since 1) boundaries-style axes do not require interpolation (they implicitly use flat interpolation), and 2) there is no interpolation between Legendre orders.

```
<gridded3d>
    <axes>
      <grid index="3" label="energy_in" unit="MeV" style="boundaries">
        <values> 1.3068e-9 2.0908e-8 ... 20.0 </values></grid>
      <grid index="2" label="energy_out" href="axis[@index='0']"/>
      <grid index="1" label="Legendre order" style="parameters">
        <values valueType="Integer32"> 0 1 2 3 4 5 6 7 8 </values></grid>
      <axis index="0" label="matrix elements" unit="b"/></axes>
    <array shape="87,87,9" compression="flattened">
      <values valueType="Integer32" label="starts"> 0 87 ... 67338 </values>
      <values valueType="Integer32" label="lengths"> 6 5 ... 3 </values>
      <values>
        3.145 1.23e-4 -7.4e-6 8.9e-7 6.8e-9 9.2e-11
        2.843 2.46e-4 -6.4e-6 3.2e-8 -7.8e-11
        ...
        1.2e-4 0.23 0.784</values></array></gridded3d>
```

7. Uncertainties

7.1 The `uncertainty` node

Each scalar data type and functional data container described in Chapters 4 and 6 may contain an `uncertainty` node. This node may contain a scalar uncertainty (described in the next section), it may contain a covariance (described in Chapter 25), or it might contain a link to an uncertainty or covariance described elsewhere.

7.1.1 General purpose type: `uncertainty`

Stores the uncertainty for a scalar quantity or a functional data container.

Specifications for `uncertainty`

Node name: `uncertainty`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`standard`**: [optional, when present, must appear one time] Stores a normally-distributed uncertainty.
- **`logNormal`**: [optional, when present, must appear one time] Stores a log-normal-distributed uncertainty.
- **`confidenceIntervals`**: [optional, when present, must appear one time] Stores a list of intervals along with the confidence that the 'true value' lies within each interval.
- **`pdf`**: [optional, when present, must appear one time] Stores a probability distribu--tion as a 1-dimensional function.
- **`XYs1d`**: [optional, when present, must appear one time] Store the uncertainty for a 1-dimensional function. Uncertainties are assumed to be normally distributed. If uncertainties are correlated, use the `covariance` option instead.
- **`polynomial1d`**: [optional, when present, must appear one time] Stores uncertain--ties for each coefficient in a polynomial expansion. Uncertainties are as--sumed to be normally distributed. If coefficients are correlated, use the `covariance` option instead.

covariance: [optional, when present, must appear one time] Stores a covariance matrix (or a link to a covariance matrix), usually for a 1-dimensional function.

listOfCovariances: [optional, when present, must appear one time] Stores a list of covariance matrices (or a list of links to covariance matrices). This option is generally used for higher-dimensional functions. For example, if an angular distribution $P(\mu|E)$ is stored as an energy-dependent list of `Legendre` expansions, the full covariance may be decomposed into energy-dependent covariance matrices for each combination of L_i and L_j. The `listOfCovariances` lists all of these covariances.

XML Example(s) of `uncertainty`

```
<uncertainty>
    <standard>...</standard>
    <logNormal>...</logNormal>
    <confidenceIntervals>...</confidenceIntervals>
    <pdf>...</pdf>
    <XYs1d>...</XYs1d>
    <polynomial1d>...</polynomial1d>
    <covariance>...</covariance>
    <listOfCovariances>...</listOfCovariances></uncertainty>
```

7.2 Uncertainties for scalar quantities

A scalar quantity (e.g. a length) has a mean value and an uncertainty (and a unit). Typically, the uncertainty is given as a single value representing a 1-σ uncertainty for a normal distribution (e.g. 12.3 ± 1.2 *cm*). At times, two uncertainty values are given, one representing the uncertainty below the mean value and another the uncertainty above the mean value (e.g. $12.3^{+1.4}_{-0.8}$ *cm*). Each uncertainty value may be associated with a different distribution. For example, the -0.8 distribution may be for a log-normal distribution while the 1.4 is for a normal distribution. Furthermore, an uncertainty value may be given as a percent (e.g. 12.3 ± 12% *cm*). The *General-Purpose Data Containers* support a similar structure for a function's uncertainty where the uncertainty is expressed as one or more functions. For example, for $f(x) \pm d(x)$ the function $d(x)$ represents the uncertainty of $f(x)$ and for $g(x)^{+u(x)}_{-l(x)}$ the functions $l(x)$ and $u(x)$ represent the lower and upper uncertainties, respectively. Each uncertainty function is called an uncertainty component for a function. That is, $g(x)$ has the two uncertainty components $l(x)$ and $u(x)$.

GNDS provides several ways to store uncertainty information for a scalar quantity. These include the `standard`, `logNormal`, `confidenceIntervals` and `pdf` nodes.

7.2.1 General purpose type: `standard`

Used to store the standard deviation of a standard Gaussian uncertainty distribution. The mean value and unit are stored higher up in the hierarchy, for example in the parent `double` node.

Specifications for `standard`

Node name: `standard`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `double`: [**required**, must appear one time] Stores σ for the normal distribution.

XML Example(s) of `standard`

```
<standard>
    <double>...</double></standard>
```

7.2.2 General purpose type: `logNormal`

Used to store the standard deviation σ of a log-normal uncertainty distribution, $X = e^{\mu+\sigma Z}$. The mean value and unit are stored higher up in the hierarchy, for example in the parent `double` or `averageEnergy` node.

Specifications for `logNormal`

Node name: `logNormal`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `double`: [**required**, must appear one time] Stores σ for the log-normal distribution.

XML Example(s) of `logNormal`

```
<logNormal>
    <double>...</double></logNormal>
```

7.2.3 General purpose type: `confidenceIntervals`

Stores a list of intervals along with the confidence that the 'true value' lies within each interval. The mean value and unit are stored higher up in the hierarchy, for example in the parent `double` or `averageEnergy` node.

Specifications for `confidenceIntervals`

Node name: `confidenceIntervals`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
`interval`: [**required**, must appear at least one time] Stores one confidence interval.

XML Example(s) of `confidenceIntervals`

```
<confidenceIntervals>
    <interval>...</interval></confidenceIntervals>
```

7.2.4 General purpose type: `interval`

Stores a list of intervals along with the confidence that the 'true value' lies within each interval.

Specifications for `interval`

Node name: `interval`
Attributes: The list of additional allowed attributes is:
`confidence` [`Float64`, **required**] A value between 0 and 1, indicating how confident the evaluator is that the 'true' value lies between mean - lower and mean + upper.
`lower` [`Float64`, **required**] Value to subtract from the mean to obtain the lower limit for this interval.
`upper` [`Float64`, **required**] Value to add to the mean to obtain the upper limit for this interval.
Child nodes: This element has no child nodes

XML Example(s) of `interval`

```
<interval
        confidence="..."
        lower="..."
        upper="...">
</interval>
```

7.2.5 General purpose type: `pdf`

Stores an explicit probability distribution for the parent quantity.

Specifications for `pdf`

Node name: `pdf`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`XYs1d`:** [optional, when present, must appear one time] Probability distribution stored as an `XYs1d`.
- **`regions1d`:** [optional, when present, must appear one time] Probability distribution stored as a `regions1d`.

XML Example(s) of `pdf`

```
<pdf>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></pdf>
```

7.3 Uncertainties for one-dimensional functional containers

Along with a one-dimensional function $f(x)$, it is sometimes possible to provide an un--certainty estimate of the form $\delta(x) = g(x)$. This form of uncertainty is less general than a covariance matrix, providing no information about how $f(x)$ is correlated between different values of x. However, it can still be useful as a way to capture the overall variability of $f(x)$.

Two types of one-dimensional uncertainties are supported by GNDS-1.9: `XYs1d` and `polynomial1d`. The specifications for the two elements are identical to those listed in Chapter 6, but the interpretation of the data is different. The `XYs1d` uncertainty stores the one-sigma interval of a probability distribution (assumed to be Gaussian) as a function of x. The `polynomial1d` container stores the one-sigma uncertainty in each polynomial coefficient.

7.4 Covariances for one-dimensional functional containers

A covariance matrix is generally more useful than an uncertainty band for a one-dimensional function $f(x)$, since the covariance contains information about how the uncertainties at different values of x are correlated.

7.4.1 General purpose type: `covariance`

Stores a covariance matrix (or a link to a covariance matrix), usually for a 1-dimensional function.

Specifications for `covariance`

Node name: `covariance`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, optional] Unique label for this covariance (required if it resides in a `listOfCovariances`).
- **`href`** [`bodyText`, optional] The URL or xpath-like string pointing to the referred-to element. Often points to a covariance inside the `covarianceSuite`.

Child nodes: This element has no child nodes

XML Example(s) of `covariance`

```
<covariance
        label="..."
        href="..."></covariance>
```

7.4.2 General purpose type: `listOfCovariances`

Stores a list of covariance matrices (or a list of links to covariance matrices). This option is generally used for higher-dimensional functions. For example, if an angular distribution $P(\mu|E)$ is stored as an energy-dependent list of `Legendre` expansions, the full covariance may be decomposed into energy-dependent covariance matrices for each combination of L_i and L_j. The `listOfCovariances` lists all of these covariances.

Specifications for `listOfCovariances`

Node name: `listOfCovariances`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`covariance`: [**required**, must appear at least one time] Covariance matrix (or link to a covariance matrix).

XML Example(s) of `listOfCovariances`

```
<listOfCovariances>
    <covariance>...</covariance></listOfCovariances>
```

8. Documentation nodes

8.1 Structure of documentation

The documentation is contained in a `documentation` element, possibly containing author information, metadata, the title, abstract, body and bibliography elements.

8.1.1 Documentation type: `documentation`

Specifications for `documentation`

Node name: `documentation`
Attributes: The list of additional allowed attributes is:

- **`doi`** [`UTF8Text`, optional] The Digital Object Identifier (DOI) is a unique identifier affixed to a document or dataset (International Standards Organisation, 2019a). A DOI is a character string that uniquely identifies a piece of information and is associated with the URL where the information may be found.
- **`publicationDate`** [`date`, optional] The evaluation date in the form specified in section 3.3.2. This is equivalent to the DataCite.org `PublicationYear` or the ENDF-6 format's ENDATE field. If the evaluation is not published, this attribute is optional.
- **`version`** [`UTF8Text`, optional] Version of the current evaluation or document, using whatever scheme is appropriate for the library to which the (partial) evaluation belongs

Child nodes: The list of additional allowed child nodes is:

- **`authors`:** [**required**, must appear one time] The list of authors
- **`contributors`:** [optional, when present, must appear one time] The list of additional contributors
- **`collaborations`:** [optional, when present, must appear one time] The name(s) of the collaboration(s) of the author(s) (e.g. CIELO or n_TOF)
- **`dates`:** [**required**, must appear one time] The list of important dates in the development of the present evaluation.
- **`copyright`:** [optional, when present, must appear one time] Copyright information
- **`acknowledgements`:** [optional, when present, must appear one time] Statements acknowledging contributions of some other sort (e.g. funding) not otherwise

covered here

`keywords`: [optional, when present, must appear one time] A collection of NSR, PACS or PhySH keywords associated with this evaluation

`relatedItems`: [optional, when present, must appear one time] A list of datasets related to the current one.

`title`: [**required**, must appear one time] The title of the evaluation

`abstract`: [optional, when present, must appear one time] An abstract summariz--ing the evaluation

`body`: [**required**, must appear one time] The main body of text describing the evaluation. For translated ENDF-6 data, the entire MF1/MT451 text goes here.

`computerCodes`: [optional, when present, must appear one time] The list of pro--grams used to perform this evaluation

`experimentalDataSets`: [optional, when present, must appear one time] List of references to the experimental data sets used and any corrections/rescalings used in the evaluation process

`bibliography`: [optional, when present, must appear one time] The list of refer--ences used in the evaluation documentation

`endfCompatible`: [optional, when present, must appear one time] Documentation restricted to plain ascii with 66 characters per line, for backwards compatibil--ity with ENDF-6.

XML Example(s) of `documentation`

```xml
<documentation
        doi="..."
        publicationDate="..."
        version="...">
    <authors>...</authors>
    <contributors>...</contributors>
    <collaborations>...</collaborations>
    <dates>...</dates>
    <copyright>...</copyright>
    <acknowledgements>...</acknowledgements>
    <keywords>...</keywords>
    <relatedItems>...</relatedItems>
    <title>...</title>
    <abstract>...</abstract>
    <body>...</body>
    <computerCodes>...</computerCodes>
    <experimentalDataSets>...</experimentalDataSets>
    <bibliography>...</bibliography>
    <endfCompatible>...</endfCompatible></documentation>
```

8.2 Significant Dates

In addition to the publication date, there are many other events, and hence dates, that should be recorded within an evaluation. Table C.3 provides a list of attributes that denote these events. These `dateType` attributes are recommended for use in GNDS files.

8.2.1 Documentation type: `dates`

List of important dates in the development of the current evaluation.

Specifications for `dates`

Node name: `dates`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- `date`: [**required**, must appear at least one time] Each event in the lifecycle of the evaluation has its own `date` entry.

XML Example(s) of `dates`

```
<dates>
    <date>...</date></dates>
```

8.2.2 Documentation type: `date`

An important date in the development of the current evaluation.

Specifications for `date`

Node name: `date`
Attributes: The list of additional allowed attributes is:
- `dateType` [`dateType`, **required**] An enumeration describing the nature of the date in question.
- `value` [`date`, **required**] The evaluation date in the form specified in section 3.3.2. Date ranges are not supported.

Child nodes: This element has no child nodes

XML Example(s) of `date`

```
<date
        dateType="..."
        value="..."></date>
```

8.3 Authorship

Every evaluation has its creator(s) and this markup provides mechanisms for recognizing them and/or assigning blame. Use the `authors` element to list the authors of the current document/evaluation. Each author should be given their own `author` element providing their name and other pertinent information.

Smaller contributions to the creation and/or maintainance of an evaluation can be denoted with the `contributor` markup. The `contributor` markup provides and additional attribute, the `contributorType` that provides additional information about the nature of the contribution. Examples of additional contributions denoted by the `contributorType` attribute include `"ContactPerson"`, `"DataCurator"`, `"DataManager"`, `"Distributor"`, `"Editor"`, `"Sponsor"` and `"Other"`. The full list is given in Appendix C Table C.2 which denotes the `contributorType` allowed by the DataCite.org schema (DataCite Metadata Working Group, 2021a, 2021b).

Within the `author` or `contributor` nodes, use:

- the `name` attribute to provide an author's name.
- the `email` attribute to provide an author's email.
- the `note` node to denote things like the corresponding author, the author is retired, or other contact information, etc.
- the `affiliations` to collect all of an author's institutional or collaborative affiliations.

8.3.1 Documentation type: `authors`

Specifications for `authors`

Node name: `authors`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `author`: [**required**, must appear at least one time] Each author has their own `author` node providing contact information, etc.

XML Example(s) of `authors`

```
<authors>
    <author>...</author></authors>
```

8.3.2 Documentation type: `author`

Specifications for `author`

Node name: `author`
Attributes: The list of additional allowed attributes is:
- `name` [`UTF8Text`, **required**] Name of author
- `orcid` [`UTF8Text`, optional] The author's 16-digit ORCID ID number.
- `email` [`UTF8Text`, optional] Email address of author

Child nodes: The list of additional allowed child nodes is:
- `affiliations`: [optional, when present, must appear one time] Institutional affili--ation(s) of the author
- `note`: [optional, when present, must appear one time] Any special note pertaining to the author

XML Example(s) of `author`

```
<author
        name="..."
        orcid="..."
        email="...">
    <affiliations>...</affiliations>
    <note>...</note></author>
```

8.3.3 Documentation type: `contributors`

Specifications for `contributors`

Node name: `contributors`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- `contributor`: [**required**, must appear at least one time] Each contributor has their own `contributor` node providing contact information, etc.

XML Example(s) of `contributors`

```
<contributors>
    <contributor>...</contributor></contributors>
```

8.3.4 Documentation type: `contributor`

Specifications for `contributor`

Node name: `contributor`
Attributes: The list of additional allowed attributes is:
- **`name`** [`UTF8Text`, **required**] Name of contributor
- **`contributorType`** [`contributorType`, **required**] An enumeration describing the na--ture of the contribution
- **`orcid`** [`UTF8Text`, optional] The contributor's 16-digit ORCID ID number.
- **`email`** [`UTF8Text`, optional] Email address of contributor

Child nodes: The list of additional allowed child nodes is:
- **`affiliations`**: [optional, when present, must appear one time] Institutional affili--ation(s) of the contributor
- **`note`**: [optional, when present, must appear one time] Any special note pertaining to the contributor

XML Example(s) of `contributor`

```
<contributor
        name="..."
        contributorType="..."
        orcid="..."
        email="...">
    <affiliations>...</affiliations>
    <note>...</note></contributor>
```

8.3.5 Documentation type: `note`

Extra note that the authors may want to include.

Specifications for `note`

Node name: `note`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `note`

```
<note/>
```

8.3.6 Documentation type: `affiliations`

Specifications for `affiliations`

Node name: `affiliations`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`affiliation`:** [**required**, must appear at least one time] Each affiliation is described in an `affiliation` element.

XML Example(s) of `affiliations`

```
<affiliations>
    <affiliation>...</affiliation></affiliations>
```

8.3.7 Documentation type: `affiliation`

Specifications for `affiliation`

Node name: `affiliation`
Attributes: The list of additional allowed attributes is:

- **`name`** [`UTF8Text`, **required**] The name of the affiliate.
- **`href`** [`UTF8Text`, optional] The URL of the affiliate.

Child nodes: This element has no child nodes

XML Example(s) of `affiliation`

```
<affiliation
        name="..."
        href="..."></affiliation>
```

8.4 Collaborations

Collaboration is of course essential in highly technical work where people with different skillsets can come together to achieve something greater than any one collaborator could achieve. Often collaborations form with a more formal structure and naming to address specific longer term problems. Authors may belong to one of these formal collaborations and the `collaboration` markup provides a way to denote them.

8.4.1 Documentation type: `collaborations`

Specifications for `collaborations`

Node name: `collaborations`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `collaboration`: [**required**, must appear at least one time] Each collaboration is specified using a `collaboration` node.

XML Example(s) of `collaborations`

```
<collaborations>
    <collaboration>...</collaboration></collaborations>
```

8.4.2 Documentation type: `collaboration`

Specifications for `collaboration`

Node name: `collaboration`
Attributes: The list of additional allowed attributes is:

- `name` [`UTF8Text`, **required**] The name of the collaboration.
- `href` [`UTF8Text`, optional] The URL of the collaboration web page.

Child nodes: This element has no child nodes

XML Example(s) of `collaboration`

```
<collaboration
        name="..."
        href="..."></collaboration>
```

8.5 The title, abstract and body

Following the LaTeX organisation of a report, proper documentation should have a title, abstract and body. In the legacy ENDF-6 format, the title and abstract are absent.

However, the "meat" of the documentation is the `body` and ENDF-6's entire MF1/MT451 file is equivalent to the `body` element.

8.5.1 Documentation type: `title`

Use this markup to provide the title of the evaluation/document.

Specifications for `title`

Node name: `title`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `title`

```
<title/>
```

8.5.2 Documentation type: `abstract`

If needed, use this markup to provide a short summary of the documentation.

Specifications for `abstract`

Node name: `abstract`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `abstract`

```
<abstract/>
```

8.5.3 Documentation type: `body`

Use this markup for the main text of the documentation. An ENDF-6's entire MF1/MT451 file is equivalent to this element.

Specifications for `body`

Node name: `body`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `body`

```
<body/>
```

8.5.4 Documentation type: `endfCompatible`

Documentation in ENDF-6 is stored in MF1 MT451. This section contains free-form text which cannot be broken up into the various sections supported by the `documentation` node without manual intervention. The `endfCompatible` node is provided to support backwards compatibility and translating back to ENDF-6. This node is expected to eventually be deprecated, but in the meantime evaluators should consider providing a short summary of the documentation in ENDF-6 compatible form.

Specifications for `endfCompatible`

Node name: `endfCompatible`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `endfCompatible`

```
<endfCompatible/>
```

8.5.5 Markup inside `text` elements

Often times the text within a `text` element contains unicode or special markups. This is true for non-English author names and for scientific texts with equations and figures. Also, authors may just want the flexibility of writing documentation or other things using a more familiar format (e.g. Markdown). The `text` element described in the (WPEC Subgroup #38, 2016) provides the required markup. Simple text that is not structured should use `encoding="utf-8"` (the markup of `"text"` is implied), for example:

```
<name><text encoding="utf-8">Pavel Obložinský</text></name>
```

Alternatively, one could use LaTeX:

```
<name><text markup="latex"><![CDATA[Pavel Oblo\v{z}insk\'{y}]]></text></name>
```

For longer, structured, text, use formats such as `"markdown"`, `"html"`, `"xml"` or `"latex"`. In all cases, the text inside the tags must be escaped with CDATA tags.

8.6 Other metadata

Often there is a need for providing more information than the above markups allow. Information such as copyrights, keywords, and Document Object Identifiers (DOI) need a home in the documentation hierarchy as well. This section describes such markups.

8.6.1 Documentation type: `copyright`

The `copyright` element provides a way for evaluators to add or link to a copyright statement for the (partial) evaluation.

Specifications for `copyright`

Node name: `copyright`
Abstract node: `text`
Attributes: The list of additional allowed attributes is:
  `href` [`UTF8Text`, optional] A link to the text of the copyright statement
Child nodes: This element has no child nodes

XML Example(s) of `copyright`

```
<copyright
        href="..."></copyright>
```

8.6.2 Documentation type: `version`

Used to state the version of a document or code using whatever markup is appropriate. Usually versions are simple numbers (e.g. subversion revision numbers), but they can be much more complex (MacOS Sierra for instance). Given this, a `text` low level container is used to store the version information itself. When used at the `documentation` level, it refers to the version of the evaluation and is thus equivalent to ENDF-6's NMOD flag.

Specifications for `version`

Node name: `version`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `version`

```
<version/>
```

8.6.3 Documentation type: `acknowledgements`

A collection of acknowledgement statements.

Specifications for `acknowledgements`

Node name: `acknowledgements`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `acknowledgement`: [**required**, must appear at least one time] Statement acknowl--edging contributions of some other sort (e.g. funding) not otherwise covered.

XML Example(s) of `acknowledgements`

```
<acknowledgements>
    <acknowledgement>...</acknowledgement></acknowledgements>
```

8.6.4 Documentation type: `acknowledgement`

This element allows one to acknowledge contributions of some other sort (e.g. funding) not otherwise covered. All United States Department of Energy funded work require a statement of the funding used and this statement should be placed here.

Specifications for `acknowledgement`

Node name: `acknowledgement`
Abstract node: `text`

Attributes: The list of additional allowed attributes is:

`label` [`XMLName`, **required**] The *unique* reference to the acknowledgement.

Child nodes: This element has no child nodes

XML Example(s) of `acknowledgement`

```
<acknowledgement
        label="..."></acknowledgement>
```

8.6.5 Documentation type: `keywords`

Both the American Physical Society and the National Nuclear Data Center use keyword--ing systems to categorise documents. The `keywords` element allows us to collect these keywords together.

The National Nuclear Data Center uses the Nuclear Science References (NSR) key--wording system. The American Physical Society's Physics and Astronomy Classification Scheme (PACS) system is another keywording system for categorising documents. The APS has stated that this keyword system is obsolete and will be replaced with the Physics Subject Headings (PhySH) system (American Physical Society, 2021).[15] Note that it is unlikely that older papers will be recoded or that older papers will have their PACS codes removed. Therefore one expects that the present markup will be relevant for the forseeable future.

Specifications for `keywords`

Node name: `keywords`

Attributes: This element has no attributes

Child nodes: The list of additional allowed child nodes is:

`keyword`: [**required**, must appear at least one time] A NSR, PACS or PhySH keyword associated with this evaluation

XML Example(s) of `keywords`

```
    <keywords>
        <keyword type="NSR">...</keyword>
        <keyword type="PACS">...</keyword>
        <keyword type="PhySH">...</keyword>
    </keywords>
```

[15]For more information, see https://journals.aps.org/PACS and https://journals.aps.org/authors/physh.

8.6.6 Documentation type: `keyword`

A NSR, PACS or PhySH keyword associated with this evaluation

Specifications for `keyword`

Node name: `keyword`
Abstract node: `text`
Attributes: The list of additional allowed attributes is:

- **`type`** [`XMLName`, **required**] The name of the keywording scheme used, either “NSR”, “PACS” or “PhySH”

Child nodes: This element has no child nodes

XML Example(s) of `keyword`

```
    <keyword type="NSR">
        <text>NUCLEAR REACTIONS 10B(16O, 16O), E=24 MeV;
              measured reaction products. Comparison with
              available data.</text></keyword>
```

8.6.7 Documentation type: `relatedItems`

List of datasets related to the present one.

Specifications for `relatedItems`

Node name: `relatedItems`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`relatedItem`:** [**required**, must appear at least one time] A dataset related to the present one.

XML Example(s) of `relatedItems`

```
<relatedItems>
    <relatedItem>...</relatedItem></relatedItems>
```

8.6.8 Documentation type: `relatedItem`

A dataset related to the present one.

Specifications for `relatedItem`

Node name: `relatedItem`
Attributes: The list of additional allowed attributes is:

- **`name`** [`UTF8Text`, **required**] The name of the related item.
- **`href`** [`UTF8Text`, optional] The URL of the related item.
- **`relationType`** [`relationType`, optional] The nature of the relationship of the present set with the related item.

Child nodes: This element has no child nodes

XML Example(s) of `relatedItem`

```
<relatedItem
        name="..."
        href="..."
        relationType="...">
</relatedItem>
```

8.7 How was this file generated: codes and their inputs

Often computer codes are used to either model aspects of an evaluation or to fit data used in the evaluation process. This markup should provide a way to not only reference the code, but this markup also should provide sufficient information to acquire the *exact* code used in the evaluation with the inputs needed to reproduce the evaluation. As a code may need several input files, the option for this with the `inputDecks` element is provided.

8.7.1 Documentation type: `computerCodes`

Lists the programs used while preparing the evaluation, along with code inputs.

Specifications for `computerCodes`

Node name: `computerCodes`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `computerCode`: [**required**, must appear at least one time] Each set of code and inputs has a `computerCode` node.

XML Example(s) of `computerCodes`

```
<computerCodes>
    <computerCode>...</computerCode></computerCodes>
```

8.7.2 Documentation type: `computerCode`

Contains the information pertaining to a code used in the evaluation process and the inputs for that code.

Specifications for `computerCode`

Node name: `computerCode`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`name`** [`UTF8Text`, **required**] The name of the code, using all the same restrictions, etc. as the `name` markup introduced above in the `author` markup. This includes allowing unicode program names (such as CEA's GALILEÉ processing code (Coste-Delclaux, 2009)).
- **`version`** [`XMLName`, **required**] The version of the code, with all the restrictions and allowances of the `version` markup introduced above.

Child nodes: The list of additional allowed child nodes is:

- **`executionArguments`**: [optional, when present, must appear one time] The com--mand line arguments used when invoking this code.
- **`codeRepo`**: [optional, when present, must appear one time] Information for access--ing the source code repository for this program.
- **`note`**: [optional, when present, must appear one time] Any note describing extra information needed to run the code in the manner needed to reproduce the evaluation.
- **`inputDecks`**: [optional, when present, must appear one time] The list of input files for the code used.
- **`outputDecks`**: [optional, when present, must appear one time] The list of output files generated by the code.

XML Example(s) of `computerCode`

```
<computerCode
        name="..."
        version="...">
    <executionArguments>...</executionArguments>
    <codeRepo>...</codeRepo>
    <note>...</note>
    <inputDecks>...</inputDecks>
    <outputDecks>...</outputDecks></computerCode>
```

8.7.3 Documentation type: `codeRepo`

The `codeRepo` element provides a mechanism for referencing the exact version of a computer code used in the evaluation. This markup is meant to point to the exact revision of a code in a generally available code repository.

Specifications for `codeRepo`

Node name: `codeRepo`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`revisionSystem`** [`XMLName`, **required**] The source control format (e.g. `"git"`, `"subver--sion"` or `"mercurial"`).
- **`href`** [`XMLName`, **required**] Hyperlink to the source code repository.
- **`revisionID`** [`XMLName`, **required**] Revision identifier. Could be a tag, svn revision number or git commit hash for example.

Child nodes: This element has no child nodes

XML Example(s) of `codeRepo`

```
<codeRepo
        revisionSystem="..."
        href="..."
        revisionID="...">
</codeRepo>
```

8.7.4 Documentation type: `executionArguments`

Use this markup to list any special command line specifics used to invoke this code.

Specifications for `executionArguments`

Node name: `executionArguments`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `executionArguments`

```
<executionArguments/>
```

8.7.5 Documentation type: `inputDecks`

Lists the inputs for a given code.

Specifications for `inputDecks`

Node name: `inputDecks`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `inputDeck`: [**required**, must appear at least one time] Inputs for a `computerCode` node.

XML Example(s) of `inputDecks`

```
<inputDecks>
    <inputDeck>...</inputDeck></inputDecks>
```

8.7.6 Documentation type: `inputDeck`

The `inputDeck` contains the actual code inputs. The name of the element was chosen to avoid any confusion with other uses of the word “input” while evoking the mental image of a stack of punch cards such as used in the early days of nuclear data.

Specifications for `inputDeck`

Node name: `inputDeck`
Abstract node: `text`
Attributes: The list of additional allowed attributes is:

- `filename` [`XMLName`, optional] File name (possibly including relative path) for this input deck.

Child nodes: This element has no child nodes

XML Example(s) of `inputDeck`

```
<inputDeck
        filename="..."></inputDeck>
```

8.7.7 Documentation type: `outputDecks`

Lists the output files for a given code.

Specifications for `outputDecks`

Node name: `outputDecks`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `outputDeck`: [**required**, must appear at least one time] Inputs for a `computerCode` node.

XML Example(s) of `outputDecks`

```
<outputDecks>
    <outputDeck>...</outputDeck></outputDecks>
```

8.7.8 Documentation type: `outputDeck`

The `outputDeck` contains the actual code outputs.

Specifications for `outputDeck`

Node name: `outputDeck`
Abstract node: `text`
Attributes: The list of additional allowed attributes is:
  `filename` [`XMLName`, optional] File name (possibly including relative path) for this output file.
Child nodes: This element has no child nodes

XML Example(s) of `outputDeck`

```
<outputDeck
        filename="..."></outputDeck>
```

8.8 Data sets used in this evaluation

Evaluations represent our best estimate for a particular fundamental value of a param- -eter. The most reliable evaluations are rooted in experimental values and extended with trustworthy models where experimental data is not available. For nuclear reaction data, one often relies on the EXFOR nuclear reaction data library (Otuka et al., 2014). Although the focus here is on the use of EXFOR reaction data, in the future a similar markup for XUNDL data sets may be considered for nuclear structure data.

While a markup for raw data could be provided, one is not provided for several reasons:

- This is the purview of EXFOR and therefore out of the scope the current project. Should the EXFOR project choose to join the GNDS effort, then this decision should be reconsidered.
- The number of possible variations in data is astronomical and cannot easily be denoted in this short document
- The number of possible variations in data is astronomical and designing a system to accommodate these variations is beyond the scope of this effort.

8.8.1 Documentation type: `experimentalDataSets`

Container for experimental data sets. Each type of experimental data will have its own top-level node. Currently, the only top-level node is the (`exforDataSets`) node. This top-level tag is so-named so that other child nodes can be added, say for XUNDL datasets or for simple tables of XYdY pairs for raw experimen without distorting the meaning of the top-level tag.

Specifications for `experimentalDataSets`

Node name: `experimentalDataSets`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
`exforDataSets`: [**required**, must appear one time] Node to contain all the exfor experimental datasets.

XML Example(s) of `experimentalDataSets`

```
<experimentalDataSets>
    <exforDataSets>...</exforDataSets></experimentalDataSets>
```

8.8.2 Documentation type: `exforDataSets`

Container storing a list of EXFOR datasets.

Specifications for `exforDataSets`

Node name: `exforDataSets`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

exforDataSet: [**required**, must appear at least one time] One for each EXFOR data set.

XML Example(s) of `exforDataSets`

```
<exforDataSets>
    <exforDataSet>...</exforDataSet></exforDataSets>
```

8.8.3 Documentation type: `exforDataSet`

Contains a reference to a specific EXFOR subentry (specified with the `SUBENT` attribute) and details of the retrieval and any corrections, rescalings, etc. This markup does not contain the full EXFOR entry or subentry – these are already stored in the master EXFOR database at the IAEA. This markup provides a scheme to reference the datasets only. Note that only the covariance generation and correction scripts provided by V. Zerkin in his EXFOR Web application (Zerkin & Pritychenko, 2018) are supported. Other schemes could be designed but for the present none have been enumerated. If the current markup is insufficient to describe modifications/corrections to a data set, the evaluator should consider documenting it in the main `body` element.

Specifications for `exforDataSet`

Node name: `exforDataSet`
Attributes: The list of additional allowed attributes is:

subentry [`XMLName`, **required**] The EXFOR 'SUBENT' (subentry) is an 8 character ascii string comprised of integers only. Each subentry corresponds to a unique dataset within the experiment detailed in the EXFOR entry.

retrievalDate [`date`, **required**] The date the EXFOR subentry was retrieved. The date must be in the form specified in the section 3.3.2.

Child nodes: The list of additional allowed child nodes is:

covarianceScript: [optional, when present, must appear one time] The script describing the generation of covariance data..

correctionScript: [optional, when present, must appear one time] The script describing any modifications to the data.

note: [optional, when present, must appear one time] A note describing extra information needed to reproduce the evaluation.

XML Example(s) of `exforDataSet`

```
<exforDataSet
        subentry="..."
        retrievalDate="...">
```

```
    <covarianceScript>...</covarianceScript>
    <correctionScript>...</correctionScript>
    <note>...</note></exforDataSet>
```

8.8.4 Documentation type: `covarianceScript`

V. Zerkin's EXFOR retrieval web application (Zerkin & Pritychenko, 2018) provides a mechanism for computing covariance matrices based off the statistical and systematic errors presented in an EXFOR entry. This markup provides the means to place this script directly in the documentation for future reference and for reconstructing the covariance used at a later date.

Specifications for `covarianceScript`

Node name: `covarianceScript`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `covarianceScript`

```
<covarianceScript/>
```

8.8.5 Documentation type: `correctionScript`

V. Zerkin's EXFOR retrieval web application (Zerkin & Pritychenko, 2018) provides a mechanism for rudimentary data correction of an EXFOR entry. This markup provides the means to place this script directly in the documentation for future reference and for reconstructing the covariance used at a later date.

Specifications for `correctionScript`

Node name: `correctionScript`
Abstract node: `text`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `correctionScript`

```
<correctionScript/>
```

8.9 Bibliographic elements

8.9.1 Documentation type: `bibliography`

The bibliography markup gives the evaluator the tools to provide references for this evaluation with a system as flexible as LaTeX or as simple as plain text. Our notation here is clearly inspired by LaTeX and should be familiar to many scientists and engineers. The `bibliography` element is the outer container for all of the `bibitem`s (in keeping with LaTeX notation).

Specifications for `bibliography`

Node name: `bibliography`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`bibitem`:** [**required**, must appear at least one time] Each entry in the bibliography has a unique `bibitem`.

XML Example(s) of `bibliography`

```
<bibliography>
    <bibitem>...</bibitem></bibliography>
```

8.9.2 Documentation type: `bibitem`

The `bibitem` element contains the actual bibliographic reference. The `label` attribute provides the landing point for hyperreferences to a `bibitem` and must be unique.

Specifications for `bibitem`

Node name: `bibitem`
Abstract node: `text`
Attributes: The list of additional allowed attributes is:

- **`xref`** [`XMLName`, **required**] The *unique* reference to this bibitem. When using LaTeXmarkup, it is natural to use this as the key used when cross referencing with LaTeX's `\cite{}` command.

Child nodes: This element has no child nodes

XML Example(s) of `bibitem`

```
<bibitem
        xref="..."></bibitem>
```

8.9.3 Markup inside a `bibitem`

The `format` attribute of the `text` element of a `bibitem` shall be used to specify the markup of the bibliographic reference. Allowed markups are `"ascii"`, `"bibtex"` (Patashnik, 1988), `"latex"` (Lamport, 1994), `"bibtexml"` (Gundersen & Hendrikse, 2007) or `"openXML"` (ECMA International, 2021). All must be escaped with CDATA except `"ascii"`. The use of `"bibtex"` or `"latex"` formatted `bibitem`s within LaTeX should be straightforward. Additional coding is required to translate `"bibtexml"` or `"openXML"` into LaTeX. That said, both `"bibtexml"` or `"openXML"` markups are easier to translate into other forms since they are already structured.

There are no restraints on the format of the references other than what is required by the inner `text` element's `format` attribute. It is up to each data project to maintain and enforce their referencing styles.

8.10 An example

```
<reactionSuite projectile="n" target="Fe56" evaluation="ENDF/B-VIII.0"
    format="2.0" projectileFrame="lab" interaction="nuclear">
  <styles>
    <evaluated label="eval" date="2016-07-01" library="ENDF/B" version="8.0.5">
      <temperature value="0" unit="K"/>
      <projectileEnergyDomain min="1e-5" max="2e7" unit="eV"/>

      <documentation doi="12.23134/8821.322" publicationDate="2016-11-01">
      <!--  The ENDF format can't handle all the fun
            options of GNDS.  We'll need styles on
            documentation as well and a tool to convert.-->

        <authors>
          <author name="M.W. Herman" email="someemail@labnl.gov">
            <affiliations>
              <affiliation name="Los Alamos National Laboratory"/>
              <affiliation name="NNDC, Brookhaven National Laboratory"/>
            </affiliations> </author>
          <author name="Pavel Obložinský" email="fakeemail@gmail.com">
            <affiliations>
              <affiliation name="NNDC, Brookhaven National Laboratory (retired)"/>
            </affiliations> </author>
          <!-- More authors here --> </authors>

        <contributors>
          <contributor contributorType="ContactPerson" name="John Doe" email=
          ↪  "fakeemail1@aol.com">
            <note>Translated from legacy ENDF-6 to GNDS</note><contributor/>
          <contributor contributorType="DataCurator" name="Jane Doe" email="fakeemail2@aol.com"
          ↪  ></contributors>

        <collaborations>
          <collaboration name="CIELO Pilot Project, WPEC/SG-40"
                         href="https://www.oecd-nea.org/science/wpec/sg40-cielo/"/>
        </collaborations>

        <dates>
          <date dateType="created">2016-11-01</date>
          <date dateType="collected">2015-02-01</date></dates>

        <copyright/>
        <!--  copyright is optional, there isn't one here. -->

        <title>
          <!-- allowed markups are "text", "markdown", "latex", "xhtml", ....
               All must be escaped with CDATA except "text" -->
              56Fe evaluation for ENDF/B-VIII.0 </title>

        <abstract>
            <![CDATA[abstract goes here]]></abstract>

        <body>
            <![CDATA[writeup goes here]]></body>

        <computerCodes>
          <computerCode name="EMPIRE" version="3.2 (Malta)">
            <codeRepo revisionSystem="subversion"
              href="https://ndclx4.bnl.gov/svn/empire/trunk"
              revisionID="345"/>
            <inputDecks>
              <inputDeck filename="input.dat">
                EMPIRE input here
              </inputDeck> </inputDecks> </computerCode>

          <computerCode name="ENDRES" version="1.8.7">
            <codeRepo revisionSystem="git"
              href="https://fake/url/here/endres.git"
              revisionID="[git hash here]"/>
            <executionArguments>ENDF.in 0 1</executionArguments>
            <note>
                ENDRES was modified from the trunk by blah blah...
              </note>
            <inputDecks>
              <inputDeck filename="ENDRES.in">
                ENDRES input here
              </inputDeck> </inputDecks> </computerCode> </computerCodes>

        <experimentalDataSets>
          <exforDataSets>
            <exforDataSet subentry="E1922008" retrievalDate="2014-05-01">
              <covarianceScript>
                <!-- A "gotcha" when nesting a new xml document
                   inside CDATA: no white-space is allowed before
                   the xml declaration. -->
                <![CDATA[<?xml version="1.0" encoding="UTF-8" ?>
```

```
<CovRecipe  DatasetID="E1922008"
  Created="2014-05-14 06:57:54"
  Software="Web-tool:2012.03.16">
  <Subent id="E1922008" x4upd="20060411" />
  <Reacode code="78-PT-0(P,X)79-AU-196,,SIG" />

  <defineErr name="Statistical" type="C5" cmd="Setup" src="Total" koeff="0.5" ></defineErr>
  <defineErr name="myErr-1" type="myErr" cmd="Setup" src="Systematic" koeff="0.5" ></defineErr>
  <defineErr name="myErr-2" type="myErr" cmd="Setup" src="Systematic" koeff="0.5" ></defineErr>

  <setCompressFactor Factor="2" lx0="58" lx1="29" />

  <addCovarFraction errName="Statistical" errType="C5" corrType="SERC" fracType="Uncorrelated"/>
  <addCovarFraction errName="myErr-1" errType="myErr" corrType="LERC" fracType="Fully-correlated"/>
  <addCovarFraction errName="myErr-2" errType="myErr" corrType="MERC" fracType="MERC-correlated"
      MercType="Lin" MercInterval="0.5" MercEnMin="8e6" MercEnMax="6.94e7" EnUnits="eV"/>
<myStamp>EXFOR-Web-Covariance-Recipe, V.Zerkin, IAEA-NDS, 2012-03-19.</myStamp>
</CovRecipe>]]></covarianceScript></exforDataSet>

          <exforDataSet subentry="30581004" retrievalDate="2016-10">
            <correctionScript>
              <![CDATA[
Applied corrections. Datasets: 1
1) EXFOR:#30581004 Corrected_Points:10 Deleted_Points:0
30581004 X4U:20090506; X4U:20090506; M0:exfor$20377002_fe56np; M1:recom$fe56np; dY=dY/Y; Y=Y/M0*M1; tmp0=dY
↪  </correctionScript>
          </exforDataSet> </exforDataSets></experimentalDataSets>

        <bibliography>
          <bibitem xref="pigni:2012">
              <!-- Allowed markups are "text", "bibtex", "latex",
              "bibtexml" or "openXML".  All must be escaped
              with CDATA except "text".  That said, Caleb argues
              that "bibtexml" or "openXML" structure markup
              should be allowed in lieu of <text/> elements.
              If we did this, it would elevate either "bibtexml"
              or "openXML" elements to be Low Level Data
              Containers.  Do we want this?-->
              <![CDATA[M.~T.~Pigni {\it et al.}, PHYSOR 2012 Conference
              Proceedings (2012).]]> </bibitem>

          <bibitem xref="Derrien1997">
                <![CDATA[
                O. Bouland, H. Derrien, N.~M. Larson, and L.~C. Leal,
                ``R-Matrix Analysis of the $^{240}$Pu Neutron Cross Sections
                in the Thermal to 5700-eV Energy Range,''
                {\sc Nuclear Sci. and Eng.} {\bf 127} (2) (1997). ]]> </bibitem>
        </bibliography>

        <endfCompatible>
          <!-- Note: encapsulating the legacy documentation in a <writeup>
               loses the full power of the <documentation> full markup,
               but hey, do you really want to parse the free text in an
               ENDF file?-->
          <![CDATA[
               26-Fe- 56 CIELO
               EVAL-FEB17
               IAEA Consortium----ENDF/B-VIII.beta
               MATERIAL 2631-----INCIDENT NEUTRON DATA------ENDF-6 FORMAT...]]>
          </endfCompatible>
      </documentation></evaluated></styles>

  <PoPs name="protare_internal" version="1.0" format="2.0">...</PoPs>
  <resonances>...</resonances>
  <reactions>...</reactions>
  <sums>...</sums>
</reactionSuite>
```

Part II

DATA STYLES

9. Introduction

One of the main goals of the Working Party on International Nuclear Data Evaluation Co-operation Subgroup 38 (SG38) was to support storing multiple types of data in the same file. This goal, summarised in requirement 2.2 of Figure 1.1, states that a GNDS file should be capable of storing various types of derived data (e.g. resonance parameters reconstructed to obtain full cross sections, Doppler broadened cross sections, grouped cross sections and transfer matrices, average energy deposited to products) alongside the original evaluated data. SG38 considered this capability a high priority in hopes that it would facilitate sharing of both evaluated and processed between institutions, making it simpler to compare processing code outputs and to export processed data from one institution for use with transport codes from another facility.

9.1 The `styles` node

Storing multiple types of data in the same file requires adding metadata to keep track of each data type. SG38 considered several methods for handling this, eventually con--verging on the `styles` node described in this chapter. This node contains descriptions of all different types of data that appear in a GNDS file as well as details of how one data type was derived from another, what codes were used to generate derived data types, and parameters like temperature, flux and group boundaries that are shared by many data containers within the file.

Each child node in the `styles` section also defines a unique label that can be used to refer to that data style. These labels are useful for differentiating between different styles of data inside nodes like the `crossSection`, `multiplicity` or `distribution`, and serve as a record of how different styles of data were derived.

As a simple example, consider a reaction cross section in the neutron resonance region, where the evaluator represents the cross section as resonance parameters and a background cross section (i.e. the `resonancesWithBackground` form). This data form is associated with the original evaluation and would share the same label as an `evaluated` style described below. After resonance parameters are reconstructed to obtain a point-wise linear representation of the cross section, that data would be added as another “form” inside the `crossSection`. That form would be associated with the `crossSectionReconstructed` style and would share the label of that style. An example appears in the reaction labelled ‘reaction1’ below:

```
<styles>
  <evaluated label="eval" ...>...</evaluated>
  <crossSectionReconstructed label="recon" derivedFrom="eval">...</crossSectionReconstructed>
</styles>
...
<reaction label="reaction1">
  <crossSection>
    <resonancesWithBackground label="eval">...</resonancesWithBackground>
    <XYs1d label="recon">...</XYs1d>
  </crossSection>
  ...
</reaction>
...
<reaction label="reaction2">
  <crossSection>
    <XYs1d label="eval">...</XYs1d>
  </crossSection>
  ...
</reaction>
```

The cross section for ‘reaction1’ has two different forms, and codes reading in this data from GNDS can request either style by the associated ‘label’. Next consider ‘reaction2’ from the same example: here the cross section has no `resonancesWithBack-ground` form (likely because this is a threshold reaction like (n,2n) that is not affected by resonance parameters). If a user requests the data corresponding to the “recon” style from reaction2, a code could either raise an error indicating that no such data was found, or it could consult the `styles` section, determine that the “recon” style was derived from the “eval” style and return the `XYs1d` labelled as “eval” instead.

Some styles of data are only meant to be used for particular applications. For ex-ample, while the `evaluated` style is meant to be general-purpose, a `heatedMultiGroup` style is specific to multi-group (generally deterministic) transport and activation/trans-mutation calculations. In order to keep these derived styles more manageable, they are grouped in Chapter 10 in this document.

9.1.1 Style type: `styles`

The `styles` node contains a list of `style` nodes. Each style describes either information about its evaluated data or information about its processed data. For example, the `heated` style (see Section 9.3.4) defines cross sections that have been heated (i.e. Doppler broadened) to a temperature T. The following is an example of a `heated` style:

```
    <heated label="heated1" derivedFrom="..." date="2016-01-01">
        <temperature value="600" unit="K"/></heated>
```

In this example, all cross section forms (i.e. representations) with their style attribute equal to `"heated1"` have been heated to 600 K on the date 2016-01-01. The `derivedFrom` attribute indicates the style before heating. In addition, the code that heated the cross sections and its input should be listed under a `code` element.

Specifications for `styles`

Node name: `styles`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`evaluated`**: [optional, when present, must appear at least one time] Style for original data created by the evaluator(s).
- **`crossSectionReconstructed`**: [optional, when present, must appear at least one time] Style for pointwise cross sections produced by reconstructing resonance parameters and combining results with background (if any).
- **`angularDistributionReconstructed`**: [optional, when present, must appear at least one time] Style for angular distributions produced by reconstructing reso--nance parameters.
- **`CoulombPlusNuclearElasticMuCutoff`**: [optional, when present, must appear at least one time] Style for Coulomb elastic scattering, avoiding divergence at small scattering angles.
- **`heated`**: [optional, when present, must appear at least one time] Style for cross sections that have been Doppler broadened to account for thermal motion in the target.
- **`averageProductData`**: [optional, when present, must appear at least one time] Style for average outgoing product energy and/or momentum data computed from the original (evaluated) distributions.
- **`MonteCarlo_cdf`**: [optional, when present, must appear at least one time] Style for data where cumulative probability densities (cdfs) have been pre-computed and stored alongside pdfs for faster Monte Carlo sampling.
- **`griddedCrossSection`**: [optional, when present, must appear at least one time] Style for data where all reaction cross sections are put on a union grid for more efficient Monte Carlo sampling.
- **`URR_probabilityTables`**: [optional, when present, must appear at least one time] Style for data cross section probability tables were generated for the unre--solved resonance region.
- **`heatedMultiGroup`**: [optional, when present, must appear at least one time] This style specifies parameters used for generating multi-grouped data. Some of the parameters specified include the multi-group boundaries for each particle processed via the `transportables` node, and the `flux` weight and `inverseSpeed` for the projectile.
- **`SnElasticUpScatter`**: [optional, when present, must appear at least one time] Style for data where a multi-group upscatter correction is applied to elastic scattering.

XML Example(s) of `styles`

```
<styles>
    <evaluated>...</evaluated>
    <crossSectionReconstructed>...</crossSectionReconstructed>
    <angularDistributionReconstructed>...</angularDistributionReconstructed>
    <CoulombPlusNuclearElasticMuCutoff>...</CoulombPlusNuclearElasticMuCutoff>
    <heated>...</heated>
    <averageProductData>...</averageProductData>
    <MonteCarlo_cdf>...</MonteCarlo_cdf>
    <griddedCrossSection>...</griddedCrossSection>
    <URR_probabilityTables>...</URR_probabilityTables>
    <heatedMultiGroup>...</heatedMultiGroup>
    <SnElasticUpScatter>...</SnElasticUpScatter></styles>
```

9.2 The `evaluated` data style

The `evaluated` style describes data as created by the evaluator(s). An evaluation may in some cases contain more than one `evaluated` style (with different labels for each). This may be useful, for example, when an evaluator makes major changes or additions to an existing evaluation and wishes to preserve the history of how the evaluation was performed. When doing so, the evaluator(s) and library maintainer(s) must ensure that there is no confusion between the current evaluation (and its documentation) and the older evaluation (and its documentation). The `documentation` child node in the `evalu-` `-ated` element provides a mechanism for capturing modest changes and corrections to the evaluation.

9.2.1 Style type: `evaluated`

This style denotes evaluated data (i.e. data created by the evaluator(s)). More than one `evaluated` style is allowed. Each `evaluated` style represents a revision of the evaluated data. If a style represents a revision, its `derivedFrom` must point to the prior `evaluated` style.

Specifications for `evaluated`

Node name: `evaluated`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`date`** [`date`, **required**] A valid date string representing when the data for this style were last released.
- **`label`** [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.

`derivedFrom` [`XMLName`, optional] The style that the data associated with this style were derived from. For example, multi-group data are often derived from a `heated` style.

`library` [`XMLName`, **required**] The name of the library for this evaluation (e.g. "ENDF/B", "JEFF", "JENDL").

`version` [`XMLName`, **required**] The version of this library's release (e.g. "8.0", "3.3").

Child nodes: The list of additional allowed child nodes is:

`projectileEnergyDomain`: [**required**, must appear one time] Lower and upper limits of projectile incident energy included in the evaluation.

`temperature`: [**required**, must appear one time] The temperature the data are evaluated at.

`documentation`: [**required**, must appear one time] Documentation as described in document (NEA, 2017). As the `evaluated` style describes the source of all derived data styles in this `reactionSute`, this element contains main docu--mentation for the entire `reactionSuite`. The `documentation` element provides the metadata needed for external citation of the present dataset.

XML Example(s) of `evaluated`

```
<evaluated
        date="..."
        label="..."
        derivedFrom="..."
        library="..."
        version="...">
    <projectileEnergyDomain>...</projectileEnergyDomain>
    <temperature>...</temperature>
    <documentation>...</documentation></evaluated>
```

9.2.2 Style type: `projectileEnergyDomain`

This element stores the energy domain of the projectile (in the lab frame) along with a unit. All `reaction` nodes in the evaluation should span *at least* this domain, although some may extend past the domain.

Specifications for `projectileEnergyDomain`

Node name: `projectileEnergyDomain`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

`max` [`Float64`, **required**] The maximum projectile energy for the evaluation.

`min` [`Float64`, **required**] The minimum projectile energy for the evaluation.

`unit` [`XMLName`, **required**] The unit for the projectile energy minimum and maxi--mum.

Child nodes: This element has no child nodes

XML Example(s) of `projectileEnergyDomain`

```
<projectileEnergyDomain
        max="..."
        min="..."
        unit="...">
</projectileEnergyDomain>
```

9.2.3 Style type: `temperature`

Specifies the average temperature of the target. In general the ‘evaluated’ style has a temperature of ‘0 K’, while derived data are Doppler broadened to higher temperature.

Specifications for `temperature`

Node name: `temperature`
Abstract node: `PhysicalQuantity`
Attributes: The list of additional allowed attributes is:
  `value` [`Float64`, **required**] Floating-point value of the temperature.
  `unit` [`XMLName`, **required**] Temperature unit, e.g. ‘K’ or ‘eV/k’.
Child nodes: This element has no child nodes

XML Example(s) of `temperature`

```
<temperature
        value="..."
        unit="..."></temperature>
```

9.3 Derived data styles

Several types of data can be derived from a nuclear reaction evaluation. Each type of derived data may have an associated style.

9.3.1 Style type: `crossSectionReconstructed`

This style denotes cross section data that have been reconstructed from resonance parameters.

Specifications for `crossSectionReconstructed`

Node name: `crossSectionReconstructed`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`date`** [`date`, **required**] A valid date string indicating the last time the data for this style were last updated.
- **`label`** [`XMLName`, **required**] A unique string identifier for the style.
- **`derivedFrom`** [`XMLName`, optional] Contains the label of another style from which reconstructed cross sections were derived. The derivedFrom style is usually 'evaluated'.

Child nodes: The list of additional allowed child nodes is:

- **`temperature`:** [optional, when present, must appear one time] The temperature the data are evalauted at.
- **`documentation`:** [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `crossSectionReconstructed`

```
<crossSectionReconstructed
        date="..."
        label="..."
        derivedFrom="...">
    <temperature>...</temperature>
    <documentation>...</documentation></crossSectionReconstructed>
```

9.3.2 Style type: `angularDistributionReconstructed`

This style denotes angular distribution data that have been reconstructed from reso--nance parameters.

Specifications for `angularDistributionReconstructed`

Node name: `angularDistributionReconstructed`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`date`** [`date`, **required**] A valid date string indicating the last time the data for this style were last updated.
- **`label`** [`XMLName`, **required**] A unique string identifier for the style.
- **`derivedFrom`** [`XMLName`, optional] Contains the label of another style from which reconstructed angular distribution were derived. The derivedFrom style is usually 'evaluated'.

Child nodes: The list of additional allowed child nodes is:

temperature: [optional, when present, must appear one time] The temperature the data are evalauted at.
documentation: [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `angularDistributionReconstructed`

```
<angularDistributionReconstructed
        date="..."
        label="..."
        derivedFrom="...">
    <temperature>...</temperature>
    <documentation>...</documentation></angularDistributionReconstructed>
```

9.3.3 Style type: `CoulombPlusNuclearElasticMuCutoff`

For Coulomb interactions the cross section is infinite due to a singularity at $\mu = 1.0$. Because of this, a cross section for Coulomb interactions only represents the integral of the $d\sigma(E, \mu)/d\mu$ from $\mu = -1.0$ to `"muCutoff"`.

Specifications for `CoulombPlusNuclearElasticMuCutoff`

Node name: `CoulombPlusNuclearElasticMuCutoff`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:
date [`date`, **required**] A valid date string representing when the data for this style were processed.
derivedFrom [`XMLName`, **required**] The style that the data associated with this style were derived from.
label [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.
muCutoff [`Float64`, **required**] The upper limit of μ.
Child nodes: The list of additional allowed child nodes is:
documentation: [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `CoulombPlusNuclearElasticMuCutoff`

```
<CoulombPlusNuclearElasticMuCutoff
        date="..."
        derivedFrom="..."
        label="..."
        muCutoff="...">
    <documentation>...</documentation></CoulombPlusNuclearElasticMuCutoff>
```

9.3.4 Style type: `heated`

This style represents data that have been heated to a particular temperature. For example, cross section may be heated (i.e. Doppler broadened).

Specifications for `heated`

Node name: `heated`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`date`** [`date`, **required**] A valid date string representing when the data for this style were processed.
- **`derivedFrom`** [`XMLName`, **required**] The style that the data associated with this style were derived from.
- **`label`** [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.

Child nodes: The list of additional allowed child nodes is:

- **`temperature`**: [**required**, must appear one time] The temperature the data for this style were heated to.
- **`documentation`**: [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `heated`

```
<heated
        date="..."
        derivedFrom="..."
        label="...">
    <temperature>...</temperature>
    <documentation>...</documentation></heated>
```

9.3.5 Style type: `averageProductData`

This style signifies data calulated for the `averageProductEnergy` and `averageProductMo-` `-mentum` components.

Specifications for `averageProductData`

Node name: `averageProductData`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`date`** [`date`, **required**] A valid date string indicating the last time the data for this style were last updated.
- **`label`** [`XMLName`, **required**] A unique string identifier for the style.
- **`derivedFrom`** [`XMLName`, optional] Contains the label of another style from which reconstructed cross sections were derived. The derivedFrom style is usually ‘evaluated’.

Child nodes: The list of additional allowed child nodes is:

- **`temperature`:** [**required**, must appear one time] The temperature the data for this style were heated to.
- **`documentation`:** [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `averageProductData`

```
<averageProductData
        date="..."
        label="..."
        derivedFrom="...">
    <temperature>...</temperature>
    <documentation>...</documentation></averageProductData>
```

10. Transport data styles

This chapter contains styles that describe data processed for use in Monte Carlo and deterministic transport codes.

10.1 Styles for continuous-energy Monte Carlo transport

Some data styles are specific to continuous-energy Monte Carlo transport. In particular, for efficient Monte Carlo sampling it is useful to convert all probability distributions to lin-lin interpolation and also pre-compute the cumulative probability density. It is also useful to define a union grid for all reaction cross sections, to avoid having to use a bisect (or other search) routine to evaluate the cross section for each reaction.

10.1.1 Style type: `MonteCarlo_cdf`

This style represents data that have distributions processed for use in Monte Carlo transport codes. Distribution $P(\mu, E'|E)$ must be converted to $P(\mu|E) \times P(E'|E, \mu)$, distri--bution $P(E', \mu|E)$ must be converted to $P(E'|E) \times P(\mu|E, E')$. Each function1d of $P(E'|E)$, $P(\mu, E)$, $P(E'|E, \mu)$ and $P(\mu|E, E')$ (including the $P(E'|E)$ in Kalbach-Mann) must be a `xs_pdf_cdf1d` node.

Specifications for `MonteCarlo_cdf`

Node name: `MonteCarlo_cdf`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- `date` [`date`, **required**] A valid date string representing when the data for this style were processed.
- `derivedFrom` [`XMLName`, **required**] The style that the data associated with this style were derived from.
- `label` [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.

Child nodes: The list of additional allowed child nodes is:

documentation: [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `MonteCarlo_cdf`

```
<MonteCarlo_cdf
        date="..."
        derivedFrom="..."
        label="...">
    <documentation>...</documentation></MonteCarlo_cdf>
```

10.1.2 Style type: `griddedCrossSection`

This style signifies that cross section data have been put on a common energy grid as used in Monte Carlo transport. The grid is given in the `grid` child node. Each `gridded--CrossSection` node represents the reaction data for one temperature. The temperature is specified by a `heated` style which must be in the `derivedFrom` lineage.

Specifications for `griddedCrossSection`

Node name: `griddedCrossSection`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **date** [`date`, **required**] A valid date string representing when the data for this style were processed.
- **derivedFrom** [`XMLName`, **required**] The style that the data associated with this style were derived from.
- **label** [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.

Child nodes: The list of additional allowed child nodes is:

- **grid**: [**required**, must appear one time] The common energy grid as used in Monte Carlo transport. See section 5.1.3.
- **documentation**: [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `griddedCrossSection`

```
<griddedCrossSection
        date="..."
        derivedFrom="..."
        label="...">
    <grid>...</grid>
    <documentation>...</documentation></griddedCrossSection>
```

10.1.3 Style type: `URR_probabilityTables`

Indicates that cross section probability tables ($P(\sigma|E)$) were generated for use in Monte Carlo transport.

Specifications for `URR_probabilityTables`

Node name: `URR_probabilityTables`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`date`** [`date`, **required**] A valid date string representing when the data for this style were processed.
- **`derivedFrom`** [`XMLName`, **required**] The style that the data associated with this style were derived from.
- **`label`** [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.

Child nodes: The list of additional allowed child nodes is:

- **`documentation`:** [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `URR_probabilityTables`

```
<URR_probabilityTables
        date="..."
        derivedFrom="..."
        label="...">
    <documentation>...</documentation></URR_probabilityTables>
```

10.2 Styles for deterministic and multi-group Monte Carlo trans--port

For deterministic and multi-group Monte Carlo transport the nuclear data are pro--cessed into groups in incident and outgoing energies. The `heatedMultiGroup` style specifies the multi-group boundaries as well as the projectile flux (which may be at a temperature other than that of the evaluated data) and inverse speeds needed for multi-group transport. Each `heatedMultiGroup` node represents the reaction data for one temperature. The temperature is specified by a `heated` style which must be in the `derivedFrom` lineage.

10.2.1 Style type: `heatedMultiGroup`

This style represents data that have been heated and multi-grouped. The temperature is inherited from the `derivedFrom` lineage. The multi-group boundaries are specified under a `transportables` node. The flux, which can be at a temperature relevant for the reaction data, is specified by the `flux` child node.

Specifications for `heatedMultiGroup`

Node name: `heatedMultiGroup`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

- **`date`** [`date`, **required**] A valid date string representing when the data for this style were processed.
- **`derivedFrom`** [`XMLName`, **required**] The style that the data associated with this style were derived from.
- **`label`** [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.

Child nodes: The list of additional allowed child nodes is:

- **`transportables`:** [**required**, must appear one time] Contains a list of `transportable` nodes. Each `transportable` specifies the multi-group boundaries for a speci--fied particle. There can be at most one `transportable` per particle id.
- **`flux`:** [**required**, must appear one time] The flux used when multi-grouping.
- **`inverseSpeed`:** [**required**, must appear one time] The multi-group inverse speeds v_g for group g. That is, $v_g = \int_g dE\, f(E)/v(E)$, where E is the projectile's energy, $f(E)$ is a flux, $v(E)$ is the projectile's velocity and the integral is over group g.
- **`documentation`:** [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `heatedMultiGroup`

```
<heatedMultiGroup
        date="..."
        derivedFrom="..."
        label="...">
    <transportables>...</transportables>
    <flux>...</flux>
    <inverseSpeed>...</inverseSpeed>
    <documentation>...</documentation></heatedMultiGroup>
```

10.2.2 Style type: `transportables`

Contains a list of particles (i.e. `transportable` nodes) for which multi-group processing was performed. Each particle's id is specified via the `label` attribute which must be unique for each transportable.

Specifications for `transportables`

Node name: `transportables`
Abstract node: `label`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

> `transportable`: [**required**, must appear at least one time] Each `transportable` spec--ifies multi-group parameters for each particle processed for multi-group transport.

XML Example(s) of `transportables`

```
<transportables>
    <transportable>...</transportable></transportables>
```

10.2.3 Style type: `transportable`

Contains a particle id and its associated multi-group boundaries.

Specifications for `transportable`

Node name: `transportable`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:

> `conserve` [`XMLName`, optional, default is "number"] Defines the outgoing particle's energy weight used when calculating a product's transfer matrices. Allowed values are `"number"` with a weight of 1 and `"energy"` with a weight of the product's outgoing energy.
>
> `label` [`XMLName`, **required**] The transportable particle id.

Child nodes: The list of additional allowed child nodes is:

> `multiGroup`: [**required**, must appear one time] Contains the multi-group bound--aries for the specified particle.

XML Example(s) of `transportable`

```
<transportable
        conserve="..."
        label="...">
    <multiGroup>...</multiGroup></transportable>
```

10.2.4 Style type: `multiGroup`

Contains a multi-group identifier and its multi-group boundaries.

Specifications for `multiGroup`

Node name: `multiGroup`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:
- `label` [`XMLName`, **required**] An identifier for the multi-group boundaries.

Child nodes: The list of additional allowed child nodes is:
- `grid`: [**required**, must appear one time] The axes information for the multi-group boundaries. See Section 5.1.3. The grid values are the multi-group boundaries.

XML Example(s) of `multiGroup`

```
<multiGroup
        label="...">
    <grid>...</grid></multiGroup>
```

10.2.5 Style type: `flux`

For `heatedMultiGroup` data, this node specifies the flux used to multi-group the data.

Specifications for `flux`

Node name: `flux`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:
- `label` [`XMLName`, **required**] An identifier string for the flux.

Child nodes: The list of additional allowed child nodes is:
- `XYs2d`: [**required**, must appear one time] The flux $f(E, \mu)$. The axes are incident particle energy, Legendre order and flux. See section 6.3.4.

XML Example(s) of `flux`

```
<flux
        label="...">
    <XYs2d>...</XYs2d></flux>
```

10.2.6 Style type: `inverseSpeed`

The multi-group inverse speeds v_g for group g. That is, $v_g = \int_g dE\, f(E)/v(E)$ where E is the projectile's energy, $f(E)$ is a flux, $v(E)$ the projectile's velocity and the integral is over group g.

Specifications for `inverseSpeed`

Node name: `inverseSpeed`
Abstract node: `label`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`gridded1d`:** [**required**, must appear one time] Contains the inverse speeds for each group (group boundaries are listed in the `multiGroup` ancestor).

XML Example(s) of `inverseSpeed`

```
<inverseSpeed>
    <gridded1d>...</gridded1d></inverseSpeed>
```

10.3 Thermal elastic upscatter style

10.3.1 Style type: `SnElasticUpScatter`

Like style `heatedMultiGroup` but with an upscatter correction included for elastic scat--tering.

Specifications for `SnElasticUpScatter`

Node name: `SnElasticUpScatter`
Abstract node: `label`
Attributes: The list of additional allowed attributes is:
- **`date`** [`date`, **required**] A valid date string representing when the data for this style were processed.
- **`derivedFrom`** [`XMLName`, **required**] The style that the data associated with this style were derived from.
- **`label`** [`XMLName`, **required**] A string identifier for the style and all data associated with it. The identifier must be unique among all the styles.
- **`upperCalculatedGroup`** [`Integer32`, optional] Index of the highest-energy group (counting from 0) impacted by the upscatter correction.

Child nodes: The list of additional allowed child nodes is:

- `documentation`: [optional, when present, must appear one time] Documentation for the style.

XML Example(s) of `SnElasticUpScatter`

```
<SnElasticUpScatter
        date="..."
        derivedFrom="..."
        label="..."
        upperCalculatedGroup="...">
    <documentation>...</documentation></SnElasticUpScatter>
```

Part III

PARTICLE PROPERTIES

11. Properties of Particles (PoPs) overview

The Properties of Particles (PoPs) database structure is designed to store reaction independent quantities like particle mass, spin, parity, half-life and decay information. A PoPs database can stand on its own or can be included inside an evaluation, storing information about all of the particles appearing in the evaluation.

The term ‘particle’ is used in PoPs to indicate both fundamental physical particles like the photon and collections of particles in a specific configuration (like nucleons, nuclei and excited-state nuclei). Each particle is described by a collection of particle properties. Some, like the mass and spin, apply to all types of particles, but other properties are specific to certain types of particles (e.g. ‘excitation energy’ applies to excited-state nuclei but not to fundamental particles).

Each particle in PoPs is assigned a unique id and can be looked up using that id. PoPs also supports creating alias particles that refer to other particles in the database. For example, an alias particle can be used to link the metastable particle id ‘Am242_m1’ to the actual particle id ‘Am242_e2’.

In order to organise different types of particles into a single hierarchy, the PoPs database supports grouping similar particles together. For example, all excited state nuclei with the same number of neutrons and protons can be stored together in PoPs under the same `isotope` node, and all isotopes can be stored together under the same `chemicalElement` node.

PoPs supports storing more than one possible assignment for each physical quantity. For example, if experimental evidence could support two (or more) different spin assignments for a particular nucleus, all possible assignments may be stored in PoPs, although one must be identified as the ‘default’ assignment.

In order to support various kinds of nuclear data evaluations, PoPs must be able to handle two different types of decay data. First (and arguably most useful) is when a decay explicitly lists its branching ratio along with all decay products, including their excited state energies or level indices if applicable. This gives the user full access to the details of how the decay proceeds, so if desired they can keep track of what intermediate particles were populated during the decay. This type of detailed decay spectrum is supported by both ENDF-6 (using the ‘Transition Probability Arrays’, MF=12 LO=2 option) and GNDS. However, it is not always possible to disentangle decay spectra into all of their individual contributions, especially when many different decay paths are possible. For this reason, both ENDF-6 and GNDS support storing decay spectra as probability distributions of outgoing particle energy (for each type of emitted particle). This option is similar to the ENDF decay sub-library and to MF=15 gamma-decay spectra in ENDF-6.

11.1 Particle naming schemes

Requirement #1 in reference (NEA, 2016b) states that particle ids and group symbols shall follow a naming convention. The convention shall be consistent, applying the same rules to each particle family where possible. It should be designed to give short, easily-recognised ids to the most commonly used particles.

While the convention is meant to be strictly adhered to, users will also be able to define aliases in case they prefer a different naming. The meanings of some common aliases are also specified, to ensure that they point to the expected particle.

Note that the precise definitions of particle names as defined by IUPAC Gold Book (Nic et al., 2018) is used. This avoids confusion as many of the particles named and used in GNDS have very similar but distinct meanings and usages:

chemical element A species of atoms; all atoms with the same number of protons in the atomic nucleus.
isotopes Nuclides having the same atomic number but different mass numbers.
nuclide A species of atom, characterised by its mass number, atomic number and nuclear energy state, provided that the lifetime of that state is long enough to be observable.
nucleus The positively charged central portion of an atom, excluding the orbital elec--trons.

11.1.1 Specifications

- The id for the photon is `"photon"`.
- The suffix `"_anti"` denotes an anti-particle. For example, `"n_anti"` is the anti-neutron. Chaining together multiple `"_anti"` suffixes is not allowed.
- Ids for commonly-used leptons are `"e-"` for the electron, `"e-_anti"` for the positron, `"nu_e-"` for the electron neutrino and `"nu_e-_anti"` for the electron anti-neutrino.
- The neutron and proton are baryons with ids `"n"` and `"p"`, respectively. Their anti-particles are `"n_anti"` and `"p_anti"`, respectively.
- Chemical element symbols are equal to the standard chemical symbol with the first letter capitalised. For example, the symbol for iron is `"Fe"`.
- Isotope symbols are composed of the corresponding chemical symbol plus the nucleon number A. For example, the iron isotope with 30 neutrons has 56 total nucleons and symbol `"Fe56"`.
- As a special case, ‘natural abundance’ isotopes are sometimes used in nuclear

reaction databases. Symbols for these isotopes follow the naming convention chemical symbol + "0", as in "Fe0" for natural iron. However, note that PoPs neither supports specifying the relative abundance of isotopes nor implies anything about those abundances.
- For a nuclide containing a nucleus in the ground state, the id is equal to the symbol of the corresponding isotope. For example, the ground state nuclide of iron 56 has id "Fe56". [16]
- For a nuclide with its nucleus in an excited-state, the id is composed of the parent isotope symbol plus the string "_e#", where "#" is an integer string equal to the level index. For example, the third excited state of "Fe56" has id "Fe56_e3". Note that the level ordering may be ambiguous, especially if the energy levels are degenerate, so care must be taken when defining excited nuclei with GNDS.
- The id for each nucleus is identical to the parent nuclide, except the first letter is changed to lower case. Thus the nucleus corresponding to nuclide "Fe56_e3" is "fe56_e3".
- Some common aliases are defined, along with the particle they point to.
  - Alias ids "gamma" and "x-ray" shall point to the photon.
  - Alias "e+" should be reserved for the positron.
  - Aliases "p+" and "p-" should be reserved for the proton and anti-proton respectively.
  - Some light ions are used frequently in nuclear reaction databases. Their defined aliases are "d" for the deuteron (aliased to "h2"), "t" for the triton (aliased to "h3"), "h" for the helion (aliased to "he3") and "a" for the alpha (aliased to "he4"). Note that the proton is defined as a baryon with id "p" rather than as a nucleus, so it does not require an alias.
  - Nuclear metastable states are aliases pointing to the appropriate nuclide node. A nuclear metastable id is composed of the parent isotope symbol plus the string "_m#", where "#" is an integer string equal to the metastable index. For example, the alias for the first metastable state of ^{242}Am is "Am242_m1".

For other particles not covered by the naming conventions above, some basic guide--lines for defining particle ids are given:

- use only the ascii character set,
- case matters (e.g. "Gamma" and "gamma" are distinct and legal ids),
- make the id recognisable to nuclear and particle physicists if possible (e.g. a good id for the J/ψ particle might be "J/psi").

11.2 Particle qualifiers

Users of particle databases may need information about electron configurations in atoms and molecules. For example, nuclear reactions and decays may result in one or more electrons being ejected from a subshell in the atom, leading to x-ray fluorescence

[16]Note that although the isotope and nuclide share the same id, there is no naming collision since the isotope is treated by PoPs as a particle *group* rather than as a particle.

as electrons cascade down from outer subshells to fill the vacancy.

Rather than assigning a unique id to every possible atomic configuration, the par--ticle database includes the ability to store 'qualifiers' that can be used to add extra information when referring to a particle. The main purpose of qualifiers (at least for now) is to add information about electron configurations to a `chemicalElement`, `isotope` or `nuclide` node. A qualifier might for example indicate that one K-shell electron is missing. The PoPs database may contain additional information about the associated binding energy and fluorescence for that electron vacancy.

Each qualifier defines a string that can be appended to a particle id inside braces '{' and '}'. More than one qualifier can be stored in the braces as a ';'-delimited list.

11.3 Common particle properties

According to requirement #5 in reference (NEA, 2016b), the particle database must support storing multiple possible assignments (physical quantites) for a property (e.g. mass, spin, lifetime). For example, an evaluator may not be able to firmly establish the spin and parity of a particle, but may be able to narrow the list down to a few possibilities. Each of these possibilities is stored in a **physical quantity** node (defined in section 2.3.3), and they are grouped together inside a **particle property**.

Particle properties in a PoPs database share some common characteristics. Each has a descriptive name based on the type of property (i.e. `mass`, `spin`, etc.). Each may contain more than one value, and each value may contain an uncertainty.

Table 11.1 list the 5 common physical properties and common physical quantity nodes and units. All of these use nodes which inherit from the `physicalQuantity` abstract node 2.3.3.

Table 11.1: Common particle properties and their recommended physical quantity nodes. The list of units is not comprehensive, but the actual unit must be consistent with examples in the list.

Property Name	child nodes	unit	comment
mass	`float64`	amu, eV/c^2	
spin	`integer32`, `fraction32`	$\hbar$	
parity	`integer32`	unitless	
half-life	`float64`, `string`	s, m, h	may be 'stable'.
charge	`integer32`, `fraction32`	e	fractional charges supported.

The following subsections describe the most common particle properties.

11.3.1 Particle property type: `mass`

Particle mass.

Specifications for `mass`

Node name: `mass`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`double`**: [optional, when present, must appear one time] Particle mass. Recom--mended units are ‘amu’ or ‘MeV/c/c’.

XML Example(s) of `mass`

```
<mass>
    <double>...</double></mass>
```

11.3.2 Particle property type: `charge`

Particle charge, may be integer or fraction.

Specifications for `charge`

Node name: `charge`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`integer`**: [optional, when present, * one of marked children must appear one time] Net charge. Recommended unit is the elementary charge ‘e’.
- **`fraction`**: [optional, when present, * one of marked children must appear one time] Fractional net charge. Recommended unit is the elementary charge ‘e’.

XML Example(s) of `charge`

```
<charge>
    <integer>...</integer>
    <fraction>...</fraction></charge>
```

11.3.3 Particle property type: `spin`

Particle spin. Recommended unit is 'hbar'.

Specifications for `spin`

Node name: `spin`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

- **`fraction`**: [optional, when present, must appear at least one time] Spin assign--ment. Recommended unit is 'hbar'. Multiple 'fraction' entries may be used to indicate uncertain spin assignments.

XML Example(s) of `spin`

```
<spin>
    <fraction>...</fraction></spin>
```

11.3.4 Particle property type: `parity`

Particle parity (unitless).

Specifications for `parity`

Node name: `parity`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

- **`integer`**: [optional, when present, must appear at least one time] Parity assign--ment (unitless). Multiple 'integer' entries may be used to indicate uncertain parity assignment.

XML Example(s) of `parity`

```
<parity>
    <integer>...</integer></parity>
```

11.3.5 Particle property type: `halflife`

Choices for the the string element are ‘stable’ or ‘unstable’, otherwise a halflife is given in the double element

Specifications for `halflife`

Node name: `halflife`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):
`string`: [optional, when present, * one of marked children must appear one time] Mainly used for stable particles, i.e. halflife = ‘stable’.
`double`: [optional, when present, * one of marked children must appear one time] Numeric value of halflife. Recommended unit = ‘s’.

XML Example(s) of `halflife`

```
<halflife>
    <string>...</string>
    <double>...</double></halflife>
```

11.3.6 Particle property type: `energy`

Excitation energy (for excited-state particles).

Specifications for `energy`

Node name: `energy`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
`double`: [optional, when present, must appear at least one time] Excitation energy. Recommended unit: ‘eV’, ‘MeV’, etc.

XML Example(s) of `energy`

```
<energy>
    <double>...</double></energy>
```

11.4 Examples of particle properties and physical quantities

If the recommended mass for a particle is the atomic mass (“1.0089 amu” with uncer--tainty “$\pm$ 1.1e-3 amu”), it can be stored in XML as:

```
<mass>
  <double label="AME" value="1.0089" unit="amu">
    <documentation>
      <body>
        <text>Atomic mass from Audi and Wapstra ...</text>
      </body>
    </documentation>
    <uncertainty>
      <standard>
        <double value="1.1e-3"/>
      </standard>
    </uncertainty>
  </double>
</mass>
```

In the example above, the first physical quantity inside the `mass` node is the default or ‘recommended’ value. The uncertainty has the same unit as the parent since no unit is specified.

If the same mass value were given with asymmetric uncertainties, it could be stored as:

```
<mass>
  <double label="AME" value="1.0089" unit="amu">
    <documentation>
      <body>
        <text>Atomic mass from Audi and Wapstra ...</text>
      </body>
    </documentation>
    <uncertainty>
      <confidenceIntervals>
        <interval confidence="0.68" lower="9e-4" upper="1.2e-3"/>
        <interval confidence="0.97" lower="2e-3" upper="2.4e-3"/>
      </confidenceIntervals>
    </uncertainty>
  </double>
</mass>
```

The particle structure supports listing multiple possible assignments for a particle property. In this case the property contains multiple `fraction` nodes (the first one is used by default). The `likelihood` attribute may also be used to establish the relative likelihood of each assignment. In the example below, the recommended spin is ‘1/2’, but other possible assignments include ‘3/2’ and ‘5/2’:

```
  <spin recommended="eval">
    <fraction label="eval" value="1/2" unit="hbar" likelihood="0.5"/>
    <fraction label="alt1" value="3/2" unit="hbar" likelihood="0.25"/>
    <fraction label="alt2" value="5/2" unit="hbar" likelihood="0.25"/>
  </spin>
```

11.4.1 Examples of atomic relaxation data and qualifiers

If a reaction or decay may leave behind a completely or partially ionised atom, the 'atomic' section (as defined in 12.1.13) is used to store properties of the ionised state. For example,

```
<chemicalElement symbol="U" name="Uranium" Z="92">
  <atomic>
    <configurations>
      <configuration subshell="1s1/2" electronNumber="2.0">
        <bindingEnergy>
          <double value="115610." unit="eV"/></bindingEnergy>
        <decayData>
          <decayModes>
            <decayMode index="0" mode="fluorescense">
              <probability>
                <double value="0.289261"/></probability>
              <decayPath>
                <decay index="0">
                  <products>
                    <product label="photon" pid="photon"/>
                    <product label="U{2p1/2}" pid="U{2p1/2}"/>
                  </products>
                </decay>
              </decayPath>
            </decayMode>
            <decayMode index="1" mode="fluoresence">
              <probability>
                <double value="0.462811"/></probability>
              <decayPath>
                <decay index="0">
                  <products>
                    <product label="photon" pid="photon"/>
                    <product label="U{2p3/2}" pid="U{2p3/2}"/>
                  </products>
                </decay>
              </decayPath>
            </decayMode>
          ...
          </decayModes>
        </decayData>
      </configuration>
      <configuration subshell="2p1/2" electronNumber="2.0">
```

```
        ...
      </configuration>
    </configurations>
  </atomic>
  <isotope symbol="U233" A="233">
    ...
</chemicalElement>
```

With these qualifiers defined, another part of the database could use a pid with a qualifier to indicate that one instance of the nuclide ‘U235_e2’ is missing a K-shell electron by adding a qualifier: ‘U235_e2{1s1/2}’. If it were missing electrons from *both* the K-shell and the L2-shell, the pid would contain multiple qualifiers separated by semi-colons, as in ‘U235_e2{1s1/2;2p1/2}. In this example the computed binding energy would not be correct since the interaction between the two electrons is missing. If that correction is important, another `atomicConfiguration` node should be added with both electrons removed.

11.4.2 Discussion

- The examples above do not deal with the possibility (raised earlier in the discus--sion of requirements) that one may need to support multiple *correlated* assign--ments, as in: ‘spin/parity = 1/2- or 3/2+’. This appears occasionally in RIPL and ENSDF, for example the 3.13 MeV excited state in Mn51 which has spin/parity = either 3/2- or 5/2+.
  One suggestion for handling this is to add an optional `useWith` child node to the `integer32` and `fraction32` specifications. This would be used to link each possible spin assignment to the corresponding parity. For example,

  ```
  <particle id="...">
    ...
    <spin>
      <fraction label="0" value="1/2" unit="hbar">
        <useWith xlink:href=`parity/integer[@label="0"]'/>
      </fraction>
      <fraction label="1" value="3/2" unit="hbar">
        <useWith xlink:href=`parity/integer[@label="1"]'/>
      </fraction>
    </spin>
    <parity>
      <integer label="0" value="-1" unit=""/>
      <integer label="1" value="+1" unit=""/>
    </parity>
  </particle>
  ```

  Another option (under the assumption that this issue only comes up with regards to spin and parity) would be to use a `spinParity` rather than separate `spin` and `parity` elements. That removes the need for the `useFor` element, but has other issues: sometimes only the spin or parity are firmly assigned, so easier to have them separated out.

12. PoPs database

12.1 Organisation of particles within PoPs

Here the top-level organisation of a PoPs database is presented. Different types of particles (such as baryons, leptons, nuclides and nuclei) are separated into different particle groups.

12.1.1 Particle property type: `PoPs`

This is the root node for a full particle database. Within the database, particles are organised into different families and/or groups. Each particle in the database must have a unique 'id'. A PoPs database may be a stand-alone file, or it may appear inside a reactionSuite.

Specifications for `PoPs`

Node name: `PoPs`
Root node: This node may be the root node of a GNDS file
Attributes: The list of additional allowed attributes is:

- **`name`** [`XMLName`, **required**] Name of the PoPs library, e.g. 'ENDF-VIII'
- **`version`** [`XMLName`, **required**] Library version / release number, e.g. 'VIII.1' or '8.1_-beta'
- **`format`** [`XMLName`, **required**] PoPs format version, e.g. '1.0.1'

Child nodes: The list of additional allowed child nodes is:

- **`styles`**: [optional, when present, must appear one time] Defines the styles of data appearing in the database. Only required in stand-alone databases.
- **`documentation`**: [optional, when present, must appear one time] Top-level docu--mentation applying to the entire PoPs database.
- **`aliases`**: [optional, when present, must appear one time] Defines 'alias particles' that point to other particles. Often used to identify metastable states, for example.
- **`gaugeBosons`**: [optional, when present, must appear one time] Contains definitions for gauge bosons, notably the photon.

leptons: [optional, when present, must appear one time] Contains definitions for leptons, notably the electron and electron anti-neutrino

baryons: [optional, when present, must appear one time] Contains definitions of baryons such as the neutron and proton.

chemicalElements: [optional, when present, must appear one time] Contains defi--nitions for chemical elements, which in turn contain a list of isomers, nu--clides and nuclei

unorthodoxes: [optional, when present, must appear one time] Contains defini--tions for 'unorthodox' particles, such as representative fission products.

XML Example(s) of `PoPs`

```
<PoPs
        name="..."
        version="..."
        format="...">
    <styles>...</styles>
    <documentation>...</documentation>
    <aliases>...</aliases>
    <gaugeBosons>...</gaugeBosons>
    <leptons>...</leptons>
    <baryons>...</baryons>
    <chemicalElements>...</chemicalElements>
    <unorthodoxes>...</unorthodoxes></PoPs>
```

12.1.2 Particle property type: `aliases`

Particles may sometimes be known by more than one name. Examples include the α particle, which is synonymous with the nucleus of a He4 atom, the photon which goes by several names including 'gamma' and 'x-ray', and isomeric nuclear states which can be referred to either by isomer index or by nuclear level index. The `aliases` node contains a list of `alias` and `metastable` nodes that allow for alternate ids to be assigned to a particle.

Specifications for `aliases`

Node name: `aliases`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

alias: [optional, when present, may appear any number of times] Defines an alias pointing to another particle in the database.

metaStable: [optional, when present, may appear any number of times] Defines a metastable particle, with a particle id like 'Al26_m1'.

XML Example(s) of `aliases`

```
<aliases>
  <alias id="gamma" pid="photon"/>
  <metaStable id="Am242_m1" pid="Am242_e2" metaStableIndex="1"/></aliases>
```

12.1.3 Particle property type: `alias`

Alias particle. The alias ‘id’ must not match the id of any other particle or alias in the database.

Specifications for `alias`

Node name: `alias`
Attributes: The list of additional allowed attributes is:
- `id` [`XMLName`, **required**] alias id. For example, alias ‘gamma’ and/or ‘x-ray’ may be used to refer to the photon.
- `pid` [`XMLName`, **required**] Equal to the ‘id’ of the nuclide this alias points to, for example ‘photon’ for alias ‘gamma’.

Child nodes: This element has no child nodes

XML Example(s) of `alias`

```
<alias id="gamma" pid="photon"/>
```

12.1.4 Particle property type: `metaStable`

A `metaStable` node is a special type of alias pointing to a relatively long-lived excited nuclear level. In addition to the **id** and **pid**, it also defines a meta-stable index that shall be ‘1’ for the lowest-energy metastable state, ‘2’ for the second-lowest, etc. Metastables are a useful way to identify nuclear levels that live long enough to accumulate and potentially change reaction rates during a radiation transport calculation. The defi--nition of metastable states is often problem-specific. For some applications such as inertial confinement fusion, a micro-second half-life can be long enough to qualify a state as meta-stable, while for other applications much longer half-lives are required. A particle database allows users to define their own `metaStable` nodes for more flexible control over what nuclear states to consider as meta-stable.

Specifications for `metaStable`

Node name: `metaStable`
Attributes: The list of additional allowed attributes is:

- **`id`** [`XMLName`, **required**] Metastable id, e.g. 'Am242_m1'.
- **`metaStableIndex`** [`Integer32`, **required**] Index for metastable states, equal to '1' for lowest-energy metastable, '2' for second metastable, etc.
- **`pid`** [`XMLName`, **required**] Equal to the 'id' of the nuclide this metastable points to, for example 'Am242_e2' for metastable 'Am242_m1'.

Child nodes: This element has no child nodes

XML Example(s) of `metaStable`

```
<metaStable id="Am242_m1" pid="Am242_e2" metaStableIndex="1"/>
```

12.1.5 Particle property type: `gaugeBosons`

Collection of gauge bosons.

Specifications for `gaugeBosons`

Node name: `gaugeBosons`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`gaugeBoson`:** [optional, when present, must appear at least one time] Defines one gauge boson particle.

XML Example(s) of `gaugeBosons`

```
<gaugeBosons>
    <gaugeBoson>...</gaugeBoson></gaugeBosons>
```

12.1.6 Particle property type: `gaugeBoson`

Defines a gauge boson, notably including the photon.

Specifications for `gaugeBoson`

Node name: `gaugeBoson`

Attributes: The list of additional allowed attributes is:
  `id` [`XMLName`, **required**] Particle id, e.g. ‘photon’ or ‘gluon’.
Child nodes: The list of additional allowed child nodes is:
  `charge`: [optional, when present, must appear one time]
  `halflife`: [optional, when present, must appear one time]
  `mass`: [optional, when present, must appear one time]
  `spin`: [optional, when present, must appear one time]
  `parity`: [optional, when present, must appear one time]
  `decayData`: [optional, when present, must appear one time] Description of nuclide decay.

XML Example(s) of `gaugeBoson`

```
<gaugeBoson
        id="...">
    <charge>...</charge>
    <halflife>...</halflife>
    <mass>...</mass>
    <spin>...</spin>
    <parity>...</parity>
    <decayData>...</decayData></gaugeBoson>
```

12.1.7 Particle property type: `leptons`

Collection of leptons.

Specifications for `leptons`

Node name: `leptons`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `lepton`: [**required**, must appear at least one time] Defines one lepton particle.

XML Example(s) of `leptons`

```
<leptons>
    <lepton>...</lepton></leptons>
```

12.1.8 Particle property type: `lepton`

Defines a lepton, including the electron, muon, tau, neutrinos and their anti-particles.

Specifications for `lepton`

Node name: `lepton`
Attributes: The list of additional allowed attributes is:
- `generation` [`XMLName`, optional] Lepton generation, i.e. '1' for the electron and '2' for muon.
- `id` [`XMLName`, **required**] Particle id, e.g. 'e-' for the electron.

Child nodes: The list of additional allowed child nodes is:
- `charge`: [optional, when present, must appear one time]
- `halflife`: [optional, when present, must appear one time]
- `mass`: [optional, when present, must appear one time]
- `spin`: [optional, when present, must appear one time]
- `parity`: [optional, when present, must appear one time]
- `decayData`: [optional, when present, must appear one time] Description of nuclide decay.

XML Example(s) of `lepton`

```
<lepton
        generation="..."
        id="...">
    <charge>...</charge>
    <halflife>...</halflife>
    <mass>...</mass>
    <spin>...</spin>
    <parity>...</parity>
    <decayData>...</decayData></lepton>
```

12.1.9 Particle property type: `baryons`

Contains definitions of baryons such as the neutron and proton.

Specifications for `baryons`

Node name: `baryons`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- `baryon`: [optional, when present, must appear at least one time] Defines a baryon (a composite particle made up of three quarks)

XML Example(s) of `baryons`

```
<baryons>
    <baryon>...</baryon></baryons>
```

12.1.10 Particle property type: `baryon`

Defines a baryon (a composite particle made up of three quarks)

Specifications for `baryon`

Node name: `baryon`
Attributes: The list of additional allowed attributes is:
  `id` [`XMLName`, **required**]
Child nodes: The list of additional allowed child nodes is:
  `charge`: [optional, when present, must appear one time]
  `halflife`: [optional, when present, must appear one time]
  `mass`: [optional, when present, must appear one time]
  `spin`: [optional, when present, must appear one time]
  `parity`: [optional, when present, must appear one time]
  `decayData`: [optional, when present, must appear one time] Description of nuclide decay.

XML Example(s) of `baryon`

```
<baryon
        id="...">
    <charge>...</charge>
    <halflife>...</halflife>
    <mass>...</mass>
    <spin>...</spin>
    <parity>...</parity>
    <decayData>...</decayData></baryon>
```

12.1.11 Particle property type: `chemicalElements`

Contains definitions for chemical elements, which in turn contain a list of isomers, nuclides and nuclei

Specifications for `chemicalElements`

Node name: `chemicalElements`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`chemicalElement`:** [optional, when present, must appear at least one time] Defines a chemical element (i.e. a group of particles with the same proton number Z).

XML Example(s) of `chemicalElements`

```
<chemicalElements>
    <chemicalElement>...</chemicalElement></chemicalElements>
```

12.1.12 Particle property type: `chemicalElement`

Defines a chemical element (i.e. a group of particles with the same proton number Z). A chemicalElement is not a particle and does not define a mass or decay properties. If a 'natural abundance' particle is required, it shall be entered in the database as a `nuclide` node with A = 0, as in 'Fe0'.

Specifications for `chemicalElement`

Node name: `chemicalElement`
Attributes: The list of additional allowed attributes is:

- **`symbol`** [`XMLName`, **required**] Standard chemical symbol, e.g. 'Au' or 'C'
- **`Z`** [`Integer32`, **required**] Proton number 'Z'
- **`name`** [`XMLName`, optional] Full name, e.g. 'Gold' or 'Carbon'

Child nodes: The list of additional allowed child nodes is:

- **`atomic`:** [optional, when present, must appear one time] Atomic properties of a chemical element (assumed to be independent of neutron number N).
- **`isotopes`:** [optional, when present, must appear one time]

XML Example(s) of `chemicalElement`

```
<chemicalElement
        symbol="..."
        Z="..."
        name="...">
    <atomic>...</atomic>
    <isotopes>...</isotopes></chemicalElement>
```

12.1.13 Particle property type: `atomic`

Stores atomic properties of a chemicalElement, including electron subshells and decay properties.

Specifications for `atomic`

Node name: `atomic`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `configurations`: [optional, when present, must appear one time]

XML Example(s) of `atomic`

```
<atomic>
    <configurations>...</configurations></atomic>
```

12.1.14 Particle property type: `configurations`

List of atomic electron configurations.

Specifications for `configurations`

Node name: `configurations`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `configuration`: [**required**, must appear at least one time]

XML Example(s) of `configurations`

```
<configurations>
    <configuration>...</configuration></configurations>
```

12.1.15 Particle property type: `configuration`

Specifications for `configuration`

Node name: `configuration`

Attributes: The list of additional allowed attributes is:

`subshell` [`attributeValue`, **required**] Electron subshell indicator, e.g. ‘1s1/2’.

`electronNumber` [`Float64`, **required**] Effective number of electrons in the given subshell.

Child nodes: The list of additional allowed child nodes is:

`bindingEnergy`: [**required**, must appear one time] Electron binding energy.

`decayData`: [optional, when present, must appear one time] Decay properties for electron vacancies in this subshell (i.e. atomic relaxation).

XML Example(s) of `configuration`

```
<configuration
        subshell="..."
        electronNumber="...">
    <bindingEnergy>...</bindingEnergy>
    <decayData>...</decayData></configuration>
```

12.1.16 Particle property type: `bindingEnergy`

Binding energy for an atomic or nuclear sub-shell.

Specifications for `bindingEnergy`

Node name: `bindingEnergy`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`double`: [optional, when present, must appear at least one time] Binding energy. Recommended unit: ‘eV’, or ‘keV’

XML Example(s) of `bindingEnergy`

```
<bindingEnergy>
    <double>...</double></bindingEnergy>
```

12.1.17 Particle property type: `isotopes`

List of isotopes for a chemicalElement

Specifications for `isotopes`

Node name: `isotopes`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `isotope`: [**required**, must appear at least one time]

XML Example(s) of `isotopes`

```
<isotopes>
    <isotope>...</isotope></isotopes>
```

12.1.18 Particle property type: `isotope`

Defines the total nucleon number ‘A’, and contains a list of nuclides with the same number of protons (Z) and neutrons (A).

Specifications for `isotope`

Node name: `isotope`
Attributes: The list of additional allowed attributes is:
    `A` [`Integer32`, **required**] Nucleon number ‘A’
    `symbol` [`XMLName`, **required**] Chemical element symbol + nucleon number A, e.g. ‘Fe56’
Child nodes: The list of additional allowed child nodes is:
    `nuclides`: [optional, when present, must appear one time]

XML Example(s) of `isotope`

```
<isotope
        A="..."
        symbol="...">
    <nuclides>...</nuclides></isotope>
```

12.1.19 Particle property type: `nuclides`

List of ‘nuclide’ nodes, including ground state and excited states

Specifications for `nuclides`

Node name: `nuclides`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `nuclide`: [**required**, must appear at least one time]

XML Example(s) of `nuclides`

```
<nuclides>
    <nuclide>...</nuclide></nuclides>
```

12.1.20 Particle property type: `nuclide`

A nuclide is a particle consisting of a nucleus (in ground or excited state) plus a full complement of Z electrons.

Specifications for `nuclide`

Node name: `nuclide`
Attributes: The list of additional allowed attributes is:
  `id` [`XMLName`, **required**] Nuclide id, of the form ‘Th232’ (ground state) or ‘Th232_e4’ (4th excited state)
Child nodes: The list of additional allowed child nodes is:
  `charge`: [optional, when present, must appear one time]
  `mass`: [optional, when present, must appear one time]
  `spin`: [optional, when present, must appear one time]
  `parity`: [optional, when present, must appear one time]
  `nucleus`: [optional, when present, must appear one time] Particle corresponding to this nuclide with all electrons removed.
  `decayData`: [optional, when present, must appear one time] Description of nuclide decay.
  `fissionFragmentData`: [optional, when present, must appear one time] Description of products, delayed neutrons, etc. produced by spontaneous fission.

XML Example(s) of `nuclide`

```
<nuclide
        id="...">
    <charge>...</charge>
    <mass>...</mass>
    <spin>...</spin>
    <parity>...</parity>
    <nucleus>...</nucleus>
    <decayData>...</decayData>
    <fissionFragmentData>...</fissionFragmentData></nuclide>
```

12.1.21 Particle property type: `nucleus`

Particle definition for a bare nucleus (no electrons).

Specifications for `nucleus`

Node name: `nucleus`
Attributes: The list of additional allowed attributes is:
- `id` [`XMLName`, **required**] nucleus id, of the form ‘th232’ (ground state) or ‘th232_e4’ (4th excited state)
- `index` [`Integer32`, **required**] Excited state index. ‘0’ = ground state, ‘1’ = 1st excited, etc.

Child nodes: The list of additional allowed child nodes is:
- `charge`: [optional, when present, must appear one time]
- `energy`: [optional, when present, must appear one time]
- `halflife`: [optional, when present, must appear one time]
- `spin`: [optional, when present, must appear one time]
- `parity`: [optional, when present, must appear one time]
- `decayData`: [optional, when present, must appear one time] Description of nucleus decay.
- `fissionFragmentData`: [optional, when present, must appear one time] Description of products, delayed neutrons, etc. produced by spontaneous fission.

XML Example(s) of `nucleus`

```
<nucleus
        id="..."
        index="...">
    <charge>...</charge>
    <energy>...</energy>
    <halflife>...</halflife>
    <spin>...</spin>
    <parity>...</parity>
    <decayData>...</decayData>
    <fissionFragmentData>...</fissionFragmentData></nucleus>
```

12.1.22 Particle property type: `unorthodoxes`

List of ‘unorthodox’ particles, such as representative fission products.

Specifications for `unorthodoxes`

Node name: `unorthodoxes`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `unorthodox`: [**required**, must appear at least one time]

XML Example(s) of `unorthodoxes`

```
<unorthodoxes>
    <unorthodox>...</unorthodox></unorthodoxes>
```

12.1.23 Particle property type: `unorthodox`

An unorthodox particle is particle-like but often cannot be assigned specific particle properties like spin and parity. Such particles show up in thermal neutron scattering law files (describing molecules with bulk properties like porosity) and as representative fission products. They usually only contain a `mass`, although some other properties may also be pertinent.

Specifications for `unorthodox`

Node name: `unorthodox`
Attributes: The list of additional allowed attributes is:
    `id` [`XMLName`, **required**] Unorthodox particle id.
Child nodes: The list of additional allowed child nodes is:
    `charge`: [optional, when present, must appear one time]
    `mass`: [optional, when present, must appear one time]

XML Example(s) of `unorthodox`

```
<unorthodox
        id="...">
    <charge>...</charge>
    <mass>...</mass></unorthodox>
```

12.1.24 Example

The following is a more detailed example of how particles are arranged in a PoPs database:

```
<PoPs name="ENDF" version="8.0.1" format="0.1">
  ...
  <chemicalElements>
    <chemicalElement symbol="Mn" name="Manganese" Z="25">
      <isotopes>
        <isotope symbol="Mn55" A="55">
          <nuclides>
            <nuclide id="Mn55">
              <mass>
                <double label="0" value="54.938045141" unit="amu">
                  <uncertainty>
                    <standard>
                      <double value="5.4e-4"/></standard></uncertainty>
                </double>
              </mass>
              <nucleus id="mn55" index="0">
                <mass>
                  <!-- optional, used if bare nuclear mass is measured --></mass>
                <energy>
                  <double label="0" value="0" unit="eV"/></energy>
                <spin>
                  <fraction label="0" value="5/2" unit="hbar"/>
                  <!-- other spin assignments if applicable --></spin>
                <parity>
                  <integer label="0" value="-1" unit=""/></parity>
                <halflife>
                  <string label="0" value="stable" unit="s"/></halflife>
              </nucleus>
            </nuclide>
            <nuclide id="Mn55_e1">
              <nucleus id="mn55_e1" index="1">
                <energy>
                  <double label="0" value="125.949" unit="keV"/></energy>
                <spin>
                  <fraction label="0" value="7/2" unit="hbar"/></spin>
                <parity>
                  <integer label="0" value="-1" unit=""/></parity>
                <halflife>
                  <double label="0" value="2.59e-10" unit="s"/></halflife>
              </nucleus>
            </nuclide>
            ...
          </nuclides>
        </isotope>
        ...
      </isotopes>
    </chemicalElement>
  </chemicalElements>
<PoPs>
```

12.2 PoPs decay sections

While some particles are stable or have very long half-lives, most particles are subject to radioactive decay. Within the PoPs database, decay data are collected together inside the `decayData` node. This node is capable of storing various types of decay information, ranging from explicit decay cascades (e.g. a gamma decay of an excited nuclear state leading to a lower excited state and a photon emission) to averaged quantities like measured outgoing energy spectra.

12.2.1 Particle property type: `decayType`

Within the decay data section, the `mode` attribute indicates the type of decay. This attribute can take from an enumerated set of values.

Type name: `decayType`
Allowed values: [‘SF’, ‘beta-’, ‘beta+’, ‘EC’, ‘electroMagnetic’, ‘IT’, ‘n’, ‘p’, ‘d’, ‘t’, ‘alpha’, ‘atomicRelaxation’]

Allowed values for the `mode` are described here:

SF	Spontaneous fission.
beta-	β-decay (emitting an electron, an anti-neutrino and a residual with one neutron converted to a proton).
beta+ or e.c.	$\beta+$ decay or electron capture. These two decay modes typically compete.
electroMagnetic	De-excitation of a nucleus through emission of a photon or conversion electron.
IT	Isomeric transition, special case of electroMagnetic.
n	Emission of a neutron.
p	Emission of a proton.
d	Emission of a deuteron.
t	Emission of a triton.
alpha	Emission of an α particle.
atomicRelaxation	Emission of atomic x-rays as electrons rearrange in the atom, for example after electron capture or photo-ionisation.

Some decays can be chained together, for example β-decay followed by the emission of one or more neutrons. These decay modes are indicated in PoPs by joining the decays together in a list, as in `mode="beta-,n,n"`.

12.2.2 Particle property type: `decayData`

Unstable particles undergo decay, emitting other particles in the process. The `decayData` node stores details about possible ways that a particle can decay. This node is designed to be flexible enough to describe decays with varying levels of detail. This flexibility is useful since some decays are very well known (including all the details of what particles and excited nucleus energy levels are involved), while other decays are more difficult to measure and may include a 'continuum' portion in addition to discrete particle emission.

Specifications for `decayData`

Node name: `decayData`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`decayModes`:** [optional, when present, must appear one time] List of decay modes.
- **`averageEnergies`:** [optional, when present, must appear one time] List of average energies for each type of emitted particle.

XML Example(s) of `decayData`

```
<decayData>
    <decayModes>...</decayModes>
    <averageEnergies>...</averageEnergies></decayData>
```

12.2.3 Particle property type: `averageEnergies`

For complex decays such as spontaneous fission, it is sometimes useful to store infor- -mation about the average energy for each category of outgoing particle.

Specifications for `averageEnergies`

Node name: `averageEnergies`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`averageEnergy`:** [**required**, must appear at least one time] Mean energy for one type of decay product.

XML Example(s) of `averageEnergies`

```
<averageEnergies>
    <averageEnergy>...</averageEnergy></averageEnergies>
```

12.2.4 Particle property type: `averageEnergy`

Stores the mean outgoing energy for one type of decay product, e.g. ‘n’, ‘photon’ or ‘e-’. Note that the type of product is identified by the ‘label’ attribute, but in the future this may change to a ‘pid’.

Specifications for `averageEnergy`

Node name: `averageEnergy`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, **required**] Name of outgoing particle.
- **`value`** [`Float64`, **required**]
- **`unit`** [`XMLName`, **required**] Energy unit, e.g. ‘eV’ or ‘MeV’.

Child nodes: The list of additional allowed child nodes is:
- **`uncertainty`**: [optional, when present, must appear one time]

XML Example(s) of `averageEnergy`

```
<averageEnergy
        label="..."
        value="..."
        unit="...">
    <uncertainty>...</uncertainty></averageEnergy>
```

12.2.5 Particle property type: `decayModes`

List of decay modes, each containing a probability and list of decay products.

Specifications for `decayModes`

Node name: `decayModes`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`decayMode`**: [**required**, must appear at least one time]

XML Example(s) of `decayModes`

```
<decayModes>
    <decayMode>...</decayMode></decayModes>
```

12.2.6 Particle property type: `decayMode`

Description of a single decay mode. Lists the type of decay, a probability, an optional decay Q-value, a ‘decay path’ showing how the decay proceeds and optionally a list of spectra for different types of decay products.

Specifications for `decayMode`

Node name: `decayMode`
Attributes: The list of additional allowed attributes is:

- `label` [`XMLName`, **required**] Unique label for this decay mode.
- `mode` [`decayType`, **required**] Type of decay, e.g. ‘electroMagnetic’, ‘beta+’, etc.

Child nodes: The list of additional allowed child nodes is:

- **`probability`**: [**required**, must appear one time] Probability that this decay mode occurs. Probability of all ‘decayMode’ nodes should sum to 1.0.
- **`internalConversionCoefficients`**: [optional, when present, must appear one time] Proportional to the probability that the decay proceeds through internal conversion.
- **`photonEmissionProbabilities`**: [optional, when present, must appear one time] Probability that photons are emitted as part of the decay.
- **`Q`**: [optional, when present, must appear one time] Decay Q-value.
- **`decayPath`**: [optional, when present, must appear one time] Lists specific decay products including excited states where possible.
- **`spectra`**: [optional, when present, must appear one time] Contains a list of outgo--ing energy spectra for various types of decay products.

XML Example(s) of `decayMode`

```
<decayMode
        label="..."
        mode="...">
    <probability>...</probability>
    <internalConversionCoefficients>...</internalConversionCoefficients>
    <photonEmissionProbabilities>...</photonEmissionProbabilities>
    <Q>...</Q>
    <decayPath>...</decayPath>
    <spectra>...</spectra></decayMode>
```

12.2.7 Particle property type: `probability`

Probability that a particular `decayMode` occurs.

Specifications for `probability`

Node name: `probability`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `double`: [**required**, must appear at least one time] Numeric value of the probability (unitless). Should be in the range (0, 1]

XML Example(s) of `probability`

```
<probability>
    <double>...</double></probability>
```

12.2.8 Particle property type: `internalConversionCoefficients`

Contains a list of coefficients corresponding to the probability that the decay proceeds by converting an electron from a particular electron shell.

Specifications for `internalConversionCoefficients`

Node name: `internalConversionCoefficients`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `shell`: [**required**, must appear at least one time] Internal conversion coefficient for a single electron shell.

XML Example(s) of `internalConversionCoefficients`

```
<internalConversionCoefficients>
    <shell>...</shell></internalConversionCoefficients>
```

12.2.9 Particle property type: `photonEmissionProbabilities`

Contains a list of probabilities that internal conversion for a particular electron shell will be accompanied by photon emission.

Specifications for `photonEmissionProbabilities`

Node name: `photonEmissionProbabilities`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `shell`: [**required**, must appear at least one time] Photon emission probabilities for internal conversion in specified electron shell.

XML Example(s) of `photonEmissionProbabilities`

```
<photonEmissionProbabilities>
    <shell>...</shell></photonEmissionProbabilities>
```

12.2.10 Particle property type: `positronEmissionIntensity`

Contains a list of intensities for transitions in which internal conversion for a particular electron shell is accompanied by photon emission.

Specifications for `positronEmissionIntensity`

Node name: `positronEmissionIntensity`
Attributes: The list of additional allowed attributes is:
  `value` [`Float64`, **required**] Intensity of positron emission.
Child nodes: The list of additional allowed child nodes is:
  `uncertainty`: [optional, when present, must appear one time]

XML Example(s) of `positronEmissionIntensity`

```
<positronEmissionIntensity
        value="...">
    <uncertainty>...</uncertainty></positronEmissionIntensity>
```

12.2.11 Particle property type: `shell`

Defines the probability (and optionally uncertainty) that a process like internal conver--sion or pair production occurs through a particular electron shell.

Specifications for `shell`

Node name: `shell`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] Label of the shell, e.g. ‘1s1/2’. This matches the Python regular expression ‘[0-9+][spdf][13579]+/2’.
- **`value`** [`Float64`, **required**] Probability associated with this shell (e.g. internal con--version coefficient or pair-production coefficient).
- **`unit`** [`XMLName`, optional, default is “ ”] Internal conversion or pair production coef--ficient unit and is usually dimensionless.

Child nodes: The list of additional allowed child nodes is:

- **`uncertainty`:** [optional, when present, must appear one time] Uncertainty on the probability for this shell.

XML Example(s) of `shell`

```
<shell
        label="..."
        value="..."
        unit="...">
    <uncertainty>...</uncertainty></shell>
```

12.2.12 Particle property type: `Q`

Q-value for this decay mode.

Specifications for `Q`

Node name: `Q`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`double`:** [**required**, must appear at least one time] Numeric value of the decay Q-value, recommended unit ‘eV’ or ‘MeV’.

XML Example(s) of `Q`

```
<Q>
    <double>...</double></Q>
```

12.2.13 Particle property type: `decayPath`

Shows how a decay proceeds. It contains one or more `decay` nodes, each representing one step in the decay. For example, the `decayPath` node for the beta-decay of a light, neutron-rich nucleus could contain multiple `decay` nodes, each representing a single beta-decay towards stability. In general only using one `decay` node per `decayPath` is recommended, then looking up each of the resulting products in PoPs to determine whether and how they decay. However, the `decayPath` is meant to also be general enough to handle data like the ENDF decay sub-library where the exact identity (including excitation energy) of each intermediate state may not be specified.

Specifications for `decayPath`

Node name: `decayPath`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `decay`: [**required**, must appear at least one time]

XML Example(s) of `decayPath`

```
<decayPath>
    <decay>...</decay></decayPath>
```

12.2.14 Particle property type: `decay`

Single step in a decay. Includes a decay type and a list of products.

Specifications for `decay`

Node name: `decay`
Attributes: The list of additional allowed attributes is:
    `index` [`Integer32`, **required**] Index, must be unique within the parent `decayPath` node.
    `mode` [`decayType`, optional] Type of decay, e.g. 'electroMagnetic'.
    `complete` [`Boolean`, optional, default is "false"] Whether the decay scheme is con- -sidered complete by the evaluator.
Child nodes: The list of additional allowed child nodes is:
    `products`: [optional, when present, must appear one time] List of decay products.

XML Example(s) of `decay`

```
<decay
        index="..."
        mode="..."
        complete="...">
    <products>...</products></decay>
```

12.2.15 Particle property type: `products`

List of decay products.

Specifications for `products`

Node name: `products`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `product`: [**required**, must appear at least one time] Single decay product.

XML Example(s) of `products`

```
<products>
    <product>...</product></products>
```

12.2.16 Particle property type: `product`

Single decay product.

Specifications for `product`

Node name: `product`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**] Unique decay product label.
  `pid` [`XMLName`, **required**] Decay product particle id.
Child nodes: This element has no child nodes

XML Example(s) of `product`

```
<product
        label="..."
        pid="..."></product>
```

12.2.17 Particle property type: `spectra`

Used to store outgoing energy probability distributions for each type of emitted particle. The use of `decayMode` elements is preferred where possible since that supports decays to specific final states, but the spectra node is still important for cases when specific decay paths are too complicated to measure.

Specifications for `spectra`

Node name: `spectra`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `spectrum`: [**required**, must appear at least one time] Spectrum for one type of outgoing particle.

XML Example(s) of `spectra`

```
<spectra>
    <spectrum>...</spectrum></spectra>
```

12.2.18 Particle property type: `spectrum`

Stores the outgoing spectrum for one type of emitted particle. The spectrum can be broken down into a sum of `discrete` and `continuum` nodes. The sum of all `discrete` node intensities plus the integral of the `continuum` node contribution should equal 1.0.

Specifications for `spectrum`

Node name: `spectrum`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**] Unique label for this spectrum. Usually identical to the pid attribute value.
  `pid` [`XMLName`, **required**] id of the particle corresponding to this decay spectrum.
Child nodes: The list of additional allowed child nodes is:

`continuum`: [optional, when present, must appear one time] Spectrum for contin--uum emission of particles.
`discrete`: [optional, when present, must appear at least one time] List of particles emitted with a specific 'discrete' energy.

XML Example(s) of `spectrum`

```
<spectrum
        label="..."
        pid="...">
    <continuum>...</continuum>
    <discrete>...</discrete></spectrum>
```

12.2.19 Particle property type: `continuum`

Stores the smooth part of the emitted particle spectrum as a 1-dimensional function of outgoing particle energy. If no discrete emission is present, the continuum portion is a probability density function and should integrate to one. If discrete emissions are present, the continuum should integrate to (1 - sum of discrete intensities).

Specifications for `continuum`

Node name: `continuum`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
`XYs1d`: [optional, when present, must appear one time] Continuum energy spec--trum stored as a 1-dimensional function.

XML Example(s) of `continuum`

```
<continuum>
    <XYs1d>...</XYs1d></continuum>
```

12.2.20 Particle property type: `discrete`

Stores a single discrete particle emission, including the energy of the emitted particle and the intensity of emission.

Specifications for `discrete`

Node name: `discrete`
Attributes: The list of additional allowed attributes is:

- **`type`** [`XMLName`, optional] Type of transition for beta and electron capture. For example, ‘allowed’, ‘first-forbidden’, etc.

Child nodes: The list of additional allowed child nodes is:

- **`energy`:** [**required**, must appear one time] Emitted particle energy.
- **`intensity`:** [**required**, must appear one time] Relative intensity of discrete radia--tion (usually normalised such that most intense branch = 1.0).
- **`internalConversionCoefficients`:** [optional, when present, must appear one time] Proportional to the probability that the decay proceeds through internal conversion.
- **`internalPairFormationCoefficient`:** [optional, when present, must appear one time] Proportional to the probability that an electron/positron pair are formed as part of the decay.
- **`photonEmissionProbabilities`:** [optional, when present, must appear one time] Probability that photons are emitted as part of the decay.
- **`positronEmissionIntensity`:** [optional, when present, must appear one time] In--tensity of transitions in which positrons are emitted as part of the decay.

XML Example(s) of `discrete`

```
<discrete
        type="...">
    <energy>...</energy>
    <intensity>...</intensity>
    <internalConversionCoefficients>...</internalConversionCoefficients>
    <internalPairFormationCoefficient>...</internalPairFormationCoefficient>
    <photonEmissionProbabilities>...</photonEmissionProbabilities>
    <positronEmissionIntensity>...</positronEmissionIntensity></discrete>
```

12.2.21 Particle property type: `discreteEnergy`

This is the energy of the discrete emitted particle.

Specifications for `discreteEnergy`

Node name: `discreteEnergy`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:

- **`value`** [`Float64`, **required**]
- **`unit`** [`XMLName`, **required**]

Child nodes: The list of additional allowed child nodes is:
  `uncertainty`: [optional, when present, must appear one time] Uncertainty on the intensity.

XML Example(s) of `discreteEnergy`

```
<discreteEnergy
        value="..."
        unit="...">
    <uncertainty>...</uncertainty></discreteEnergy>
```

12.2.22 Particle property type: `intensity`

Relative intensity of this decay branch (usually normalised so the most intense branch has intenisty = 1).

Specifications for `intensity`

Node name: `intensity`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:
  `value` [`Float64`, optional]
Child nodes: The list of additional allowed child nodes is:
  `uncertainty`: [optional, when present, must appear one time] Uncertainty on the intensity.

XML Example(s) of `intensity`

```
<intensity
        value="...">
    <uncertainty>...</uncertainty></intensity>
```

12.2.23 Particle property type: `internalPairFormationCoefficient`

Related to the probability of forming an electron/positron pair.

Specifications for `internalPairFormationCoefficient`

Node name: `internalPairFormationCoefficient`

Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:

- `value` [`Float64`, **required**]
- `unit` [`XMLName`, **required**, default is “ ”] Usually dimensionless

Child nodes: This element has no child nodes

XML Example(s) of `internalPairFormationCoefficient`

```
<internalPairFormationCoefficient
        value="..."
        unit="...">
</internalPairFormationCoefficient>
```

12.3 PoPs Examples

The neutron β-decay is a simple example showing how decay data are stored:

```
  <baryon id="n" name="neutron">
    ...
    <halflife>
      <double label="0" value="613.9" unit="s">
        <uncertainty>
          <standard>
            <double value="0.6"/></standard></uncertainty></double>
    </halflife>
    <decayData>
      <decayModes>
        <decayMode label="0" mode="beta-">
          <probability>
            <double label="0" value="1"/></probability>
          <Q>
            <double label="0" value="782.347" unit="keV">
              <uncertainty>
                <standard>
                  <double value="1e-3"/></standard></uncertainty></double>
          </Q>
          <decayPath>
            <decay index="0">
              <products>
                <product label="e-" pid="e-">
                <product label="H1" pid="H1"/>
                <product label="nu_e-_anti" pid="nu_e-_anti"/>
              </products>
            </decay>
          </decayPath>
          <spectra>
            <spectrum label="beta-" pid="e-">
              <discrete type="allowed">
                <intensity value="1.0"/>
                <energy value="782347.0" unit="eV">
                  <uncertainty>
                    <standard>
                      <double value="1.0"/></standard></uncertainty>
                </energy>
              </discrete>
            </spectrum>
          </spectra>
        </decayMode>
      </decayModes>
      <averageEnergies>
        <averageEnergy label="lightParticles" value="..." unit="eV"/>
        ...
      </averageEnergies>
    </decayData>
  </baryon>
```

A second example shows a more complex decay scheme:

```
<chemicalElement symbol="Mg" name="Magnesium" Z="12">
  <isotopes>
    <isotope symbol="Mg32" A="32">
      <nuclides>
        <nuclide id="Mg32">
          <mass>
            <double label="eval" value="31.998975" unit="amu">
              <uncertainty>
                <standard>
                  <double value="1.9e-5"/></standard></uncertainty></double>
          </mass>
          <nucleus id="mg32" index="0">
            <energy>
              <double label="0" value="0" unit="eV"/></energy>
            <spin>
              <fraction label="0" value="0" unit="hbar"/></spin>
            <parity>
              <integer label="0" value="1" unit=""/></parity>
            <halflife>
              <double label="0" value="0.086" unit="s">
                <uncertainty>
                  <standard>
                    <double value="0.005"/></standard></uncertainty></double>
            <halflife>
            <decayData>
              <decayModes>
                <decayMode label="0" mode="beta-">
                  <probability>
                    <double label="BR" value="0.945" unit=""/></probability>
                  <Q>
                    <double label="0" value="10.15" unit="MeV"/></Q>
                  <decayPath>
                    <decay index="0">
                      <products>
                        <product label="e-" pid="e-"/>
                        <product label="nu_e-_anti" pid="nu_e-_anti"/>
                        <product label="Al32" pid="Al32"/></products></decay>
                  </decayPath>
                </decayMode>
                <decayMode index="1" mode="beta-,n">
                  <probability>
                    <double label="0" value="0.055" unit=""/></probability>
                  <Q>
                    <double label="" value="5.92753" unit="MeV"/></Q>
                  <decayPath>
                    <decay index="0">
                      <products>
                        <product label="e-" pid="e-"/>
                        <product label="nu_e-_anti" pid="nu_e-_anti"/>
                        <product label="Al32" pid="Al32"/></products></decay>
                    <decay index="1">
                      <products>
                        <product label="n" pid="n"/>
                        <product label="Al31" pid="Al31"/></products></decay>
                  </decayPath>
                </decayMode>
              </decayModes>
            </decayData>
          </nucleus>
        </nuclide>
        <nuclide id="Mg32_e1">
          <nucleus id="mg32_e1" index="1">
            <energy>
              <double label="0" value="885.3" unit="keV">
                <uncertainty>
                  <standard>
                    <double value="0.1"/></standard></uncertainty></double>
            </energy>
            <spin>
              <fraction label="0" value="2" unit="hbar"/></spin>
            <parity>
              <integer label="0" value="1"/></parity>
            <halflife>
              <double value="11.4" unit="ps">
                <uncertainty>
                  <standard>
                    <double value="2.0"/></standard></uncertainty></double>
            </halflife>
            <decayData>
              <decayMode index="0" mode="gamma">
                <probability>
                  <double value="1"/></probability>
```

```
                <decayPath>
                  <decay index="0">
                    <products>
                      <product label="photon" pid="photon"/>
                      <product label="Mg32" pid="Mg32"/></products></decay>
              </decayMode>
            </decayData>
          </nucleus>
        </nuclide>
        <nuclide id="Mg32_e2" index="2">
        ...
      </nuclides>
    </isotope>
    ...
  </isotopes>
</chemicalElement>
```

12.4 Discussion

This section summarises some questions about the current layout of the particle database.

- What about storing decays that have a positive Q-value but have not been mea--sured? Should the database support decay probability limits like '<0.01%'? Per--haps these as strings could be stored like

```
<probability>
  <string value="allowed but not observed"/></probability>
```

- PoPs supports two main ways of giving decay data: can give every step of the decay explicitly using a `decayPath` node, or can just summarise all the emitted particles using a `spectra` node. Should this document make recommendations about which is preferred? Or is that left up to library managers?
  Examples below show two different ways of storing the same type of decay data (excited state in C12 that can emit two different gammas):

```
<nuclide id="C12_e8">
  ...
  <nucleus id="c12_e8" index="8">
    <decayData>
      <decayModes>
        <decayMode label="0" mode="photon">
          <probability>
            <double value="0.87"></probability>
          <decayPath>
            <decay index="0">
              <products>
                <product label="photon" pid="photon"/>
                <product label="c12" pid="c12"/></products></decay>
          </decayPath>
        </decayMode>
        <decayMode label="1" mode="photon">
          <probability>
            <double value="0.13"/></probability>
          <decayPath>
            <decay index="0">
              <products>
                <product label="photon" pid="photon"/>
                <product label="c12_e1" pid="c12_e1"/>
              </products>
            </decay>
          </decayPath>
        </decayMode>
      </decayModes>
    </decayData>
  </nucleus>
</nuclide>
```

Here the second decay mode leaves the c12 nucleus in an excited state. Decay properties of that state (c12_e1) can be found by looking it up in the database. That particle would contain information about another gamma decay that leads to the ground state and emits a 4.4389 MeV gamma.
Another option for storing this data is to use the `spectra` node:

```
<nuclide id="C12_e8">
  ...
  <nucleus id="c12_e8" index="8">
    <decayData>
      <decayModes>
        <decayMode label="0" mode="photon">
          <probability>
            <double value="1"/></probability>
          <decayPath>
            <products>
              <product label="photon" pid="photon"/>
              <product label="c12" pid="c12"/></products></decayPath>
          <spectra>
            <spectrum label="gamma" pid="photon">
              <discrete label="0">
                <intensity value="0.87"/>
                <energy value="12.71" unit="MeV"/></discrete>
              <discrete label="1">
                <intensity value="0.13"/>
                <energy value="8.271" unit="MeV"/></discrete>
              <discrete label="2">
                <intensity value="0.13"/>
                <energy value="4.4389" unit="MeV"/></discrete>
            </spectrum>
          </spectra>
        </decayMode>
      </decayModes>
    </decayData>
  </nucleus>
```

```
</nuclide>
```

In this case all photons are stored together. This is good option for storing exper--imental decay spectra, since experiments are actually measuring the photons rather than the nuclides or nuclei that emitted them. It is also useful when search--ing through measured decay radiation to find what parent likely emitted the radiation. On the other hand, it obscures the fact that the 8.271 MeV and 4.4389 MeV gammas occur in coincidence. This style also leads to possible redundancy and discrepancies if more than one parent particle can decay through the same intermediate states, since the same gamma products may also be emitted as part of the decay cascade from a higher level.
In both examples above, the product is stored as a nucleus rather than a nuclide. In practice, user codes will need to detect whether the decay parent was a nucleus or a nuclide, and return the same type following the decay.
This rule becomes more complicated when internal conversion is possible as well as gamma decay. Since internal conversion necessarily involves electrons, it cannot be defined inside the bare `nucleus` node. Furthermore, the decay half-life might be longer when all electrons are stripped away. Although gamma decay and internal conversion are closely linked, the database needs to be able to differentiate between them. The simplest option to do that may be to store the same decay under both the `nuclide` and `nucleus`, but with different decay probabilities. For example:

```
<nuclide id="C12_e8">
  ...
  <halflife>
    <double value="..." unit="s"/>
    <!-- fairly short since IC is open -->
  </halflife>
  <decayData>
    <decayModes>
      <decayMode label="0" mode="internalConversion">
        <probability> ... </probability>
        <decayPath>
          <decay index="0">
            <products>
              <product label="e-" pid="e-"/>
              <product label="C12" pid="C12"/>
            </products>
          </decay>
        </decayPath>
      </decayMode>
      <decayMode label="1" mode="internalConversion">
        <probability> ... </probability>
        <decayPath>
          <decay index="0">
            <products>
              <product label="e-" pid="e-"/>
              <product label="C12_e1" pid="C12_e1"/>
            </products>
          </decay>
        </decayPath>
      </decayMode>
    </decayModes>
  </decayData>
  ...
  <nucleus id="c12_e8" index="8">
    <halflife>
      <double value="..." unit="s"/>
      <!-- potentially much longer since IC is suppressed -->
    </halflife>
    <decayData>
      <decayModes>
        <decayMode label="0" mode="photon">
          <probability> ... </probability>
          <decayPath>
            <decay index="0">
              <products>
                <product label="gamma" pid="photon"/>
                <product label="c12" pid="c12"/>
              </products>
            </decay>
          </decayPath>
        </decayMode>
        <decayMode label="1" mode="photon">
          <probability> ... </probability>
          <decayPath>
            <decay index="0">
              <products>
                <product label="gamma" pid="photon"/>
                <product label="c12_e1" pid="c12_e1"/>
              </products>
            </decay>
          </decayPath>
        </decayMode>
      </decayModes>
    </decayData>
  </nucleus>
</nuclide>
```

This allows us to give different half-lives when electrons are missing vs. present. Also like that it makes clear that gamma decay remains open for the bare nucleus (if all decay data were stored inside the nuclide one could use internal conversion coefficients, but then codes trying to decay a bare nucleus would need some complex logic to look at the parent state and see if ICC < 1, etc.).
However, this proposal breaks up the decay into two separate places without an explicit connexion between them. Should there be a link from the internalCon--version decay to the associated gamma decay?

Part IV

REACTION DATA

13. Introduction

This part of the specifications document describes the hierarchy for storing evaluated and processed reaction data. These data include general transport data (Chapters 13 through 18) as well as special topics like resonances (Chapter 19), thermal neutron scattering (Chapter 20), fission fragment data (Chapter 21), fission product yields (Chapter 22) and atomic data (Chapter 24).

For a neutral particle (such as a neutron or photon) with energy E traversing a material, the particle's mean free path (λ_{mfp}) is defined as

$$\lambda_{\text{mfp}} = \frac{1}{\sum_x n_x \sigma_{x,\text{tot}}(E)}, \tag{13.1}$$

where the sum goes over each component x of the material, and $\sigma_{x,\text{tot}}$ and n_x are the total cross section and number density respectively of component x. The mean free path determines the transit distance and time between "hard" nuclear collisions. The total cross section for each component should be tabulated in a `crossSectionSum` element.

For charged particles, the total cross section does not exist because of the Coulomb singularity in, for example, the elastic scattering reaction. Coulomb scattering is a "soft collision" in that the Coulomb force is sufficiently long-ranged to affect charged particles that do not come close enough to undergo "hard" nuclear reactions. To implement Coulomb scattering in practice one divides up scattering events by angle relative to the center-of-mass momentum of the target and projectile. At small angles, one uses condensed history treatments (Seco & Verhaegen, 2016). At large angles, one uses the large-angle Coulomb scattering plus "nuclear plus interference" approximation which treats the reaction as a "hard" inelastic collision. This mixture of treatments means the data cannot be heated.

For a "hard" collision, one can proceed as follows:

1. Determine which material was hit by assigning probabilities proportional to $\sigma_x(E)$
2. For the material, decide what reaction occurred by assigning probabilities proportional to the ratio of each partial cross section to the total cross section for that material
3. Once the reaction is decided, determine what particles will be emitted
4. Loop over emitted particles

   (a) If the multiplicity is not constant, sample the number of emitted particles using the energy-dependent multiplicity which may be given as a distribution (e.g. $P(\nu)$).
   (b) Depending on the kinematics (two-body or uncorrelated), sample the emitted particle's energy and/or angle
   (c) If particle or reaction product decays, follow the decays
5. Where possible, use energy/momentum conservation to determine the recoil energy and momentum

All information necessary to simulate *most* nuclear reactions can be stored inside a GNDS `reactionSuite` file (see Section 14.1). The main exceptions are the particle induced fission yield data which are stored in a `fissionFragmentData` file (Chapter 22). GNDS also defines a `covarianceSuite` file for storing covariances and cross-correlations between different types of reaction data.

Before presenting the organisation of data in these files, one begins by defining some common data types that appear in GNDS reaction data.

13.1 Physical Quantities

13.1.1 Common type: `mass`

Specifications for `mass`

Node name: `mass`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `double`: [**required**, must appear at least one time]

XML Example(s) of `mass`

```
<mass>
    <double>...</double></mass>
```

13.1.2 Common type: `energy`

Specifications for `energy`

Node name: `energy`
Abstract node: `physicalQuantity`

Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `double`: [**required**, must appear at least one time]

XML Example(s) of `energy`

```
<energy>
    <double>...</double></energy>
```

13.1.3 Common type: `temperature`

Specifications for `temperature`

Node name: `temperature`
Abstract node: `physicalQuantity`
Attributes: The list of additional allowed attributes is:
  `unit` [`XMLName`, optional]
  `value` [`Float64`, optional]
Child nodes: This element has no child nodes

XML Example(s) of `temperature`

```
<temperature
        unit="..."
        value="..."></temperature>
```

13.2 External Files

The `externalFiles` node is used to manage links between different GNDS files. The most common use is to establish the link between a `reactionSuite` file and one or more corresponding `covarianceSuite` files. The section stores one or more external files (using the relative path to the other file).

Note that the xPath syntax (used to store links in GNDS) supports linking to data in a different file. This syntax requires encoding the file path in the xPath expression for each link (e.g. `<link xpath:href="./externalFile.xml#/xpath/within/file"/>`), so if the file name changes many links need to be changed as well. GNDS addresses this problem by storing the path to each external file only once, along with a unique label that can be used internally in place of the file path. Thus if the external file name changes, it only needs to be changed in one place. If the label for the external file is `extern` the above link would be `<link href="$extern#/xpath/within/file"/>`

A hashsum (MD5 or SHA1) can be added to the `externalFile`. This can be used to check for any changes to the externalFile since it was generated.

13.2.1 Common type: `externalFiles`

The `externalFiles` node is a suite that stores a list of `externalFile` nodes. An `ex-ternalFile` node establishes the link between a `reactionSuite` file and one or more corresponding `covarianceSuite` files.

Specifications for `externalFiles`

Node name: `externalFiles`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `externalFile`: [**required**, must appear at least one time] A file

XML Example(s) of `externalFiles`

```
<externalFiles>
    <externalFile>...</externalFile></externalFiles>
```

13.2.2 Common type: `externalFile`

Stores a single external file, including a unique label and file path and optionally a checksum to help ensure file integrity.

Specifications for `externalFile`

Node name: `externalFile`
Abstract node: `labelNode`
Attributes: The list of additional allowed attributes is:
 `label` [`XMLName`, **required**] unique string that can be used to identify this external file.
 `path` [`XMLName`, **required**] relative path to the external file, using unix-style path separators to maintain compatability with existing xPath tools.
 `checksum` [`checksum`, optional] hexadecimal checksum for the external file. Can be used to check that whether an external file has been modified.
 `algorithm` [`algorithm`, optional] Algorithm used to compute the ‘checksum’. Al-lowed values are ‘md5’ and ‘sha1’.
Child nodes: This element has no child nodes

XML Example(s) of `externalFile`

```
<externalFile
        label="..."
        path="..."
        checksum="..."
        algorithm="...">
</externalFile>
```

13.3 Probabilities and Cumulative Probabilities

13.3.1 Common type: `probability`

Specifications for `probability`

Node name: `probability`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `double`: [optional, when present, must appear one time]

XML Example(s) of `probability`

```
<probability>
    <double>...</double></probability>
```

14. The main reaction hierarchy

14.1 The `reactionSuite` element

All reactions involving the same combination of projectile and target can be grouped together inside a single evaluation. That evaluation, or `reactionSuite`, should specify what projectile/target combination to which it pertains, the list of allowed `reactions` corresponding to tabulated reactions and suitable documentation.

14.1.1 General transport type: `reactionSuite`

This element stores an evaluation of the reactions involving the combination of a projectile and target.

Specifications for `reactionSuite`

Node name: `reactionSuite`
Root node: This node may be the root node of a GNDS file
Attributes: The list of additional allowed attributes is:

- **`evaluation`** [`XMLName`, **required**] Name of the evaluation, e.g. 'ENDF-VIII.0'
- **`format`** [`XMLName`, **required**] GNDS format version, e.g. '2.1'. See Appendix B for an extended discussion of the GNDS version numbering.
- **`projectile`** [`XMLName`, **required**] Projectile particle id.
- **`projectileFrame`** [`frame`, **required**] Options are 'lab' or 'centerOfMass'
- **`target`** [`XMLName`, **required**] Target particle id.
- **`interaction`** [`interaction`, **required**] Defines the type of interaction for the reac--tion data contained within the reactionSuite. Options are 'nuclear', 'atomic' and 'thermalNeutronScatteringLaw'. For example, a photo-nuclear reaction--Suite must have a value of 'nuclear' while a photo-atomic reactionSuite must have a value of 'atomic'.

Child nodes: The list of additional allowed child nodes is:

- **`externalFiles`:** [optional, when present, must appear one time] stores a list of external files related to this `reactionSuite`. Often used to link a `reactionSuite` to one or more `covarianceSuite` files.

`styles`: [**required**, must appear one time] Element containing a list of styles inside this covarinceSuite. Each style describes information about the evalualated data (e.g. library, version) representation and each processed data represen--tation.

`PoPs`: [**required**, must appear one time] Particle database, as described in doc--ument (NEA, 2016b). The database describes all particles involved in the reactionSuite, including projectile, target and reaction products.

`resonances`: [optional, when present, must appear one time] Describes resonance parameters.

`reactions`: [optional, when present, must appear one time] List of all excusive reaction nodes. The sum of the cross sections from each reaction must sum to the total cross section for neutral incident particles.

`orphanProducts`: [optional, when present, must appear one time] Collection of products not associated with a particlar reaction. In legacy evaluations, this element is used to store gamma-rays that are observed in experiment but whose provenance is unknown.

`sums`: [optional, when present, must appear one time] The `sums` node contains cross sections that are summations over two or more reaction cross sections, and also multiplicities that are summations over two or more reaction product multiplicities.

`fissionComponents`: [optional, when present, must appear one time] List of partial fission cross sections, often used to give first-chance, second-chance, etc. fission without supplying distributions for each chance.

`productions`: [optional, when present, must appear one time] List of production reactions.

`incompleteReactions`: [optional, when present, must appear one time] List of re--actions where only some data are available. May be used to store very small cross sections (i.e. sub-actinide fission) or for sharing evaluations where some reactions are still not complete.

`applicationData`: [optional, when present, must appear one time] List of application-specific data.

XML Example(s) of `reactionSuite`

```
<reactionSuite
        evaluation="..."
        format="..."
        projectile="..."
        projectileFrame="..."
        target="..."
        interaction="...">
    <externalFiles>...</externalFiles>
    <styles>...</styles>
    <PoPs>...</PoPs>
    <resonances>...</resonances>
    <reactions>...</reactions>
    <orphanProducts>...</orphanProducts>
    <sums>...</sums>
    <fissionComponents>...</fissionComponents>
    <productions>...</productions>
    <incompleteReactions>...</incompleteReactions>
    <applicationData>...</applicationData></reactionSuite>
```

Several of the child elements in the `reactionSuite` are described here, others are found in other chapters throughout this document.

14.1.2 Additional comments

According to these specifications, nodes like `reactions`, `sums`, `productions`, etc., are optional. This makes the format more flexible, but individual nuclear data libraries may want to set their own requirements for which nodes are required. For example, a radiation transport library manager may wish to require `reactions` and `sums` for every file in their library, while a medical isotope production library manager may be more interested in requiring `productions` in their library.

14.1.3 General transport type: `interaction`

Type name: `interaction`
Allowed values: [‘nuclear’, ‘atomic’, ‘thermalNeutronScatteringLaw’]

14.2 Reaction lists

14.2.1 General transport type: `reactions`

The `reactions` section collects all of the `reaction` elements. The sum of the cross sections from all `reaction` elements inside `reactions` should be equal to the total cross section (note that the total cross section is *not* stored in `reactions` but rather inside `sums` (see section 14.3.1).

Specifications for `reactions`

Node name: `reactions`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `reaction`: [**required**, must appear at least one time] A `reaction`

XML Example(s) of `reactions`

```
<reactions>
    <reaction>...</reaction></reactions>
```

14.2.2 General transport type: `orphanProducts`

The `orphanProducts` node is similar to the `reactions` node described in section 14.2.1, containing a list of `orphanProduct` nodes each with a `crossSection` and `outputChannel`. However, products inside the `orphanProducts` section are not associated with a particu--lar reaction. This section is most often used to store gamma-rays that are observed during reaction experiments but whose provenance is unknown. The `crossSection` inside an orphan product reaction should be a link to one of the summed cross sections inside the `sums` section (see section 14.3.1).

Specifications for `orphanProducts`

Node name: `orphanProducts`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`orphanProduct`**: [**required**, must appear at least one time] A reaction container to group together groups of correlated 'orphan' products.

XML Example(s) of `orphanProducts`

```
<orphanProducts>
    <orphanProduct>...</orphanProduct></orphanProducts>
```

14.2.3 Fission transport type: `fissionComponents`

Nuclear data evaluations sometimes break a reaction up into multiple components, but only supply a cross section (not outgoing products) for each component. For example, cross sections for first-chance, second-chance, etc. fission may be stored, even though product distributions are only given for total fission. The `fissionComponent` node supports storing this type of incomplete reaction.

Specifications for `fissionComponents`

Node name: `fissionComponents`
Attributes: This element has no attributes

Child nodes: The list of additional allowed child nodes is:
  `fissionComponent`: [optional, when present, must appear at least one time]

XML Example(s) of `fissionComponents`

```
<fissionComponents>
    <fissionComponent>...</fissionComponent></fissionComponents>
```

14.2.4 General transport type: `productions`

Production reactions are used for inventory applications (such as dosimetry, decay heat, and activity) and for calculating the amount of material accreted and/or depleted during radiation exposure.

Specifications for `productions`

Node name: `productions`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `production`: [optional, when present, must appear one time]

XML Example(s) of `productions`

```
<productions>
    <production>...</production></productions>
```

14.2.5 General transport type: `incompleteReactions`

The `incompleteReactions` section allows evaluators to include information about re--actions without providing complete information. For example, some sub-actinide fission reaction cross sections may be included as only a total fission cross section. This allows simulations to account for the reaction while not considering all of the reaction process (e.g. neutron production and fission products) or allows the evaluator to provide other information. It is strongly recommended that this be used only in cases where the reaction cross sections are negligible.

Specifications for `incompleteReactions`

Node name: `incompleteReactions`

Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `reaction`: [optional, when present, must appear one time]

XML Example(s) of `incompleteReactions`

```
<incompleteReactions>
    <reaction>...</reaction></incompleteReactions>
```

14.3 Treating sums of reactions

In principal, an evaluation does not need to store the 'total' cross section since it can be computed by summing together the cross sections for each `reaction` in the `reactions` node. In practice, however there are several inclusive reactions that one encounters in ENDF data:

1. Total cross section (a requirement for neutrons). Note the evaluated total cross section with covariance and may not exactly equal the sum of partial cross sec--tions.
2. ENDF sum rules (e.g. MT103 vs. MT600 – MT649).
3. Lumped cross sections for lumped covariances.

Evaluators can both fit total cross section data and model the total cross section using the optical model with high fidelity. Therefore the evaluation of the total cross section is usually quite solid with high quality covariances. However, there is no guarantee that if one sums up the partial cross sections of an evaluation that one will recover the total cross section. While most evaluations will provide a total cross section (and place it in the `crossSectionSum` element), many processing codes will discard the evaluated total cross section and recompute the total from the sum of partials in order to ensure unitarity.

Evaluations also frequently contain summed multiplicities (especially for outgoing photons). Within a `reactionSuite`, summed cross sections and multiplicities are stored inside the `sums` node.

14.3.1 General transport type: `sums`

The `sums` node contains two child nodes: cross section nodes, which are summations over two or more reaction cross sections, and multiplicities, which are summations over two or more reaction product multiplicities.

Specifications for `sums`

Node name: `sums`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`crossSectionSums`:** [**required**, must appear one time] Contains one or more sums of partial cross sections.
- **`multiplicitySums`:** [optional, when present, must appear one time] Contains one or more sums of product multiplicites. Each sum should only include multi--plicities for a single type of particle. Most often used for outgoing photons and fission neutrons (e.g. to store 'total nubar' as the sum of prompt + delayed nubar).

XML Example(s) of `sums`

```
<sums>
    <crossSectionSums>...</crossSectionSums>
    <multiplicitySums>...</multiplicitySums></sums>
```

14.3.2 General transport type: `crossSectionSums`

Specifications for `crossSectionSums`

Node name: `crossSectionSums`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`crossSectionSum`:** [optional, when present, must appear at least one time] The `crossSectionSum` is similar to the `crossSection`, but is a sum of the cross section for multiple partial reactions.

XML Example(s) of `crossSectionSums`

```
<crossSectionSums>
    <crossSectionSum>...</crossSectionSum></crossSectionSums>
```

14.3.3 General transport type: `crossSectionSum`

The `crossSectionSum` is similar to the `crossSection`, but is a sum of the cross section for multiple partial reactions.

Specifications for `crossSectionSum`

Node name: `crossSectionSum`
Abstract node: `labelNode`
Attributes: The list of additional allowed attributes is:

- **`ENDF_MT`** [`Integer32`, optional] The corresponding ENDF MT number.
- **`label`** [`XMLName`, **required**] An arbitrary, but unique identifier for the summed cross section.

Child nodes: The list of additional allowed child nodes is:

- **`Q`**: [**required**, must appear one time] The Q-value of the summed reaction.
- **`crossSection`**: [**required**, must appear one time] The summed cross section values.
- **`summands`**: [**required**, must appear one time] A list of links to the partial cross sections that are the summands (hence the name of the element) of the summed reaction.

XML Example(s) of `crossSectionSum`

```
<crossSectionSum
        ENDF_MT="..."
        label="...">
    <Q>...</Q>
    <crossSection>...</crossSection>
    <summands>...</summands></crossSectionSum>
```

14.3.4 General transport type: `multiplicitySums`

Specifications for `multiplicitySums`

Node name: `multiplicitySums`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`multiplicitySum`**: [optional, when present, must appear at least one time] Sums over the multiplicity of multiple products of the same type of particle.

XML Example(s) of `multiplicitySums`

```
<multiplicitySums>
    <multiplicitySum>...</multiplicitySum></multiplicitySums>
```

14.3.5 General transport type: `multiplicitySum`

The `multiplicitySum` sums over the multiplicity of multiple products of the same type of particle.

Specifications for `multiplicitySum`

Node name: `multiplicitySum`
Abstract node: `labelNode`
Attributes: The list of additional allowed attributes is:

- **`ENDF_MT`** [`Integer32`, optional] The corresponding ENDF MT number.
- **`label`** [`XMLName`, **required**] An arbitrary, but unique identifier for the summed cross section.

Child nodes: The list of additional allowed child nodes is:

- **`multiplicity`:** [**required**, must appear one time] The summed multiplicity
- **`summands`:** [**required**, must appear one time] A list of links to the partial cross sections that are the summands (hence the name of the element) of the summed reaction.

XML Example(s) of `multiplicitySum`

```
<multiplicitySum
        ENDF_MT="..."
        label="...">
    <multiplicity>...</multiplicity>
    <summands>...</summands></multiplicitySum>
```

A `summands` node appears inside both the `crossSectionSum` and `multiplicitySum`. It contains a list of `add` nodes with links pointing to all terms included in the sum. For a `crossSectionSum`, each link points to a `crossSection`, for the `multiplicitySum` each link points to a product's `multiplicity`

14.3.6 General transport type: `summands`

Specifications for `summands`

Node name: `summands`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`add`:** [optional, when present, must appear at least one time]

XML Example(s) of `summands`

```
<summands>
    <add>...</add></summands>
```

14.3.7 General transport type: `add`

Specifications for `add`

Node name: `add`
Attributes: The list of additional allowed attributes is:
    `href` [`bodyText`, optional]
Child nodes: This element has no child nodes

XML Example(s) of `add`

```
<add
        href="..."></add>
```

15. Collecting information from one reaction

15.1 The `reaction` node

15.1.1 General transport type: `reaction`

A nuclear reaction is a process in which nucleus or nuclear particles changes by producing a set of products, either through spontaneous decay or particle collision. The `reaction` element collects the information pertaining to the final state of the reaction.

Specifications for `reaction`

Node name: `reaction`
Attributes: The list of additional allowed attributes is:

- **`ENDF_MT`** [`Integer32`, **required**] The ENDF MT number for this reaction. This at--tribute is currently required, but should be considered deprecated. Future evaluations may include reactions with no MT equivalent, so codes should not rely on the `ENDF_MT`.
- **`fissionGenre`** [`XMLName`, optional] Identifies the type of fission. Must be one of 'total', 'firstChance', 'secondChance', etc.
- **`label`** [`XMLName`, **required**] Arbitrary string identifier. It must be unique among all other `reaction` elements.

Child nodes: The list of additional allowed child nodes is:

- **`doubleDifferentialCrossSection`:** [optional, when present, must appear one time] Differential cross section $d\sigma(E)/dE'd\Omega$ or $d\sigma(E)/d\Omega$ if using two-body kine--matics.
- **`crossSection`:** [**required**, must appear one time]
- **`outputChannel`:** [**required**, must appear one time] Describes the secondary particle emissions from the reaction

XML Example(s) of `reaction`

```
<reaction
        ENDF_MT="..."
        fissionGenre="..."
        label="...">
    <doubleDifferentialCrossSection>...</doubleDifferentialCrossSection>
    <crossSection>...</crossSection>
    <outputChannel>...</outputChannel></reaction>
```

15.2 Total available energy/momentum

In processed files, a `reaction` may contain `availableEnergy` and `availableMomentum` nodes. The available energy is equal to the projectile energy plus the reaction Q-value. It is useful for computing quantities such as the KERMA (Kinetic Energy Release to the Material).

15.2.1 Processed type: `availableEnergy`

For a reaction this is the total available energy for all outgoing particle. It is calculated as the energy of the projectile plus the final Q-value.

Specifications for `availableEnergy`

Node name: `availableEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `XYs1d`: [optional, when present, must appear one time]
  `gridded1d`: [optional, when present, must appear one time]

XML Example(s) of `availableEnergy`

```
<availableEnergy>
    <XYs1d>...</XYs1d>
    <gridded1d>...</gridded1d></availableEnergy>
```

15.2.2 Processed type: `availableMomentum`

For a reaction this is the total available momentum for all outgoing particle along the direction of the projectile. This is the momentum of the projectile.

Specifications for `availableMomentum`

Node name: `availableMomentum`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`XYs1d`:** [optional, when present, must appear one time]
- **`gridded1d`:** [optional, when present, must appear one time]

XML Example(s) of `availableMomentum`

```
<availableMomentum>
    <XYs1d>...</XYs1d>
    <gridded1d>...</gridded1d></availableMomentum>
```

15.3 Other reaction-like nodes

Every child of the `reactions` node is a `reaction` as described in the previous chapter. The `reactionSuite` may also contain lists of reaction-like nodes that contain a cross section and list of outgoing products. These include the `orphanProduct`, `fissionComponent` and `production` nodes.

15.3.1 General transport type: `orphanProduct`

An orphanProduct is observed to be emitted during the interaction between a given projectile/target, but cannot be associated with a particular reaction.

Specifications for `orphanProduct`

Node name: `orphanProduct`
Attributes: The list of additional allowed attributes is:
- **`ENDF_MT`** [`Integer32`, **required**] The ENDF MT number for this orphanProduct. This attribute is currently required, but should be considered deprecated. Future evaluations may include reactions with no MT equivalent, so codes should not rely on the `ENDF_MT`.
- **`label`** [`XMLName`, **required**] Arbitrary string identifier. It must be unique among all other `orphanProduct` elements.

Child nodes: The list of additional allowed child nodes is:
- **`crossSection`:** [**required**, must appear one time] Cross section for emission of this orphan product. The cross section is typically a reference to one of the summed cross sections defined in the `sums` section of the `reactionSuite`.

outputChannel: [**required**, must appear one time] Describes the outgoing energy/an--gular distributions for this orphanProduct.

XML Example(s) of `orphanProduct`

```
<orphanProduct
        ENDF_MT="..."
        label="...">
    <crossSection>...</crossSection>
    <outputChannel>...</outputChannel></orphanProduct>
```

15.3.2 Fission transport type: `fissionComponent`

Fission without pre-fission particle emission is known as first chance fission, while the emission of one (n,nf) or two neutrons (n,2nf) are known as second and third chance fission, respectively. It can be advantageous to separate out these as distinct components. The `fissionComponent` supports storing each of these fission chances. The `fissionComponent` node is similar to a `reaction`: it contains a cross section and an outputChannel along with a Q-value. The list of reaction products is generally empty, however.

Specifications for `fissionComponent`

Node name: `fissionComponent`
Attributes: The list of additional allowed attributes is:
ENDF_MT [`Integer32`, optional] integer MT number (e.g. '20' for 2^{nd}-chance fission).
fissionGenre [`XMLName`, **required**]
label [`XMLName`, **required**]
Child nodes: The list of additional allowed child nodes is:
crossSection: [optional, when present, must appear one time]
outputChannel: [optional, when present, must appear one time] Describes the secondary particle emissions from the reaction

XML Example(s) of `fissionComponent`

```
<fissionComponent
        ENDF_MT="..."
        fissionGenre="..."
        label="...">
    <crossSection>...</crossSection>
    <outputChannel>...</outputChannel></fissionComponent>
```

15.3.3 General transport type: `production`

A `production` node defines a single production reaction. Like the `reaction` it contains a cross section and an outputChannel. Unlike a `reaction`, the `production` does not force the evaluator to differentiate between different ways of getting to the same residual. For example, the ‘n,d’ ‘n,n+p’ and ‘n,p+n’ reactions all end up at the same residual. Inside the `reactions` section, evaluators may choose to store each one separately. In the `productions` section the three methods can be combined into a single reaction that produces the final residual.

Specifications for `production`

Node name: `production`
Attributes: The list of additional allowed attributes is:
- **`ENDF_MT`** [`Integer32`, optional] An integer MT number (e.g. ‘3’ for the non-elastic cross section which may produce gamma products).
- **`label`** [`XMLName`, optional]

Child nodes: The list of additional allowed child nodes is:
- **`crossSection`**: [optional, when present, must appear one time]
- **`outputChannel`**: [optional, when present, must appear one time] Describes the secondary particle emissions from the reaction

XML Example(s) of `production`

```
<production
        ENDF_MT="..."
        label="...">
    <crossSection>...</crossSection>
    <outputChannel>...</outputChannel></production>
```

16. Cross sections

16.1 Cross sections

This subsection details the specifications for the two cross-section nodes. They are:

- `crossSection`: The `crossSection` node is the most basic and most common mark up for cross sections. This section details cross sections that are a function only of incident energy: $\sigma(E)$. These are most often found in `reaction` elements (see 15.1.1) and `crossSectionSum` nodes (see 14.3.1), but also may be found as background cross sections in the resonance region (see Chapter 19).
- `doubleDifferentialCrossSection`: Occasionally there is a need to store the differ--ential cross section with respect to both incident energy and outgoing product angle and/or outgoing product energy. Such is the case for charged particle scattering (see Chapter 23), thermal neutron scattering (see Chapter 20) and photo-atomic scattering (see Chapter 24). In these cases the data is stored in physics-specific parametric forms.

For incident neutrons, it is convienient to define a differential cross section in terms of the angle integrated cross section and the outgoing energy-angle distributions of the particle(s) of interest:

$$\frac{\sigma(E)}{d\mu d\phi dE'} = \sigma(E)P(\mu,\phi,E'|E). \tag{16.1}$$

In practice assume azimuthal symmetry simplifying Eq. (16.1) to

$$\frac{\sigma(E)}{d\mu dE'} = \sigma(E)P(\mu,E'|E). \tag{16.2}$$

Fully correlated distributions $P(\mu, E'|E)$ are described in sections 18.3, 18.4 and 18.5. The integrated cross section is described below using the `crossSection` element. In the case for charged particle scattering, the Coulomb singularity prevents may prevent this representation (the angular distribution of the cross section may be divergent), so one must use the `doubleDifferentialCrossSection`.

Each cross section is assumed to be constructed from one or more interpolation regions (or given by the resonance data). Thus, the format for either type of cross section contains the list of interpolation regions, each given in terms of XYs tables and or a hyperlink. It is up to the evaluator and the processing code to ensure there is no overlap between the energy ranges of each region.

16.1.1 General transport type: `crossSection`

The effective target area for a particular reaction or reaction-like quantity on a given target, $\sigma(E)$. Child elements of a `crossSection` correspond to various representations of the cross section. At least one element that describes the dependance of the cross section as a function of energy is required. For evaluated files, one element must contain the `label="eval"` attribute.

Specifications for `crossSection`

Node name: `crossSection`
Abstract node: `label`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

- **`XYs1d`:** [optional, when present, * one of marked children must appear one time] For pointwise data using a single interpolation everywhere.
- **`regions1d`:** [optional, when present, * one of marked children must appear one time] Cross section stored in multiple regions, with different interpolations and/or discontinuities.
- **`resonancesWithBackground`:** [optional, when present, * one of marked children must appear one time] Indicates that resonance parameters need to be re--constructed and added to the background to obtain the cross section.
- **`CoulombPlusNuclearElastic`:** [optional, when present, * one of marked children must appear one time] This is either a container holding the differential charged particle scattering cross section or a reference to said container.
- **`thermalNeutronScatteringLaw1d`:** [optional, when present, * one of marked chil--dren must appear one time] This contains a reference to the double-differential thermal neutron scattering law cross section.
- **`reference`:** [optional, when present, * one of marked children must appear one time] Link to another cross section form, e.g. in another reaction.
- **`gridded1d`:** [optional, when present, * one of marked children must appear one time] Grouped (and possibly flux-weighted) cross section. Appears in pro--cessed files only.
- **`Ys1d`:** [optional, when present, * one of marked children must appear one time] Y-values, used when a common grid is used for multiple reaction cross sections for more efficient Monte Carlo sampling. Appears in processed files only.
- **`URR_probabilityTables1d`:** [optional, when present, * one of marked children must appear one time] Stores cross section probability tables $P(\sigma|E)$ to better cap--ture the rapid cross section fluctuations possible in the unresolved resonance region. This element appears in processed files only.

XML Example(s) of `crossSection`

Here an example of a file whose evaluated data consists of resonances and a fast region is presented. This data is processed in two ways: 1) into a pointwise table with a URR probability table and 2) into a grouped cross section.

```
<crossSection>
  <resonancesWithBackground label="eval">
    <resonances href="/reactionSuite/resonances"/>
    <background>
      <resolvedRegion>
        <XYs1d interpolation="lin,lin">...</XYs1d></resolvedRegion>
      <unresolvedRegion>
        <regionsd1d>...</regions1d></unresolvedRegion>
      <fastRegion>
        <XYs1d>...</XYs1d></fastRegion></background>
  </resonancesWithBackground>
  <XYs1d label="recon">...</XYs1d>
  <URR_probabilityTables1d label="URR_tables">...</URR_probabilityTables1d>
  <gridded1d label="MultiGroup_000">...</gridded1d>
</crossSection>
```

16.1.2 General transport type: `doubleDifferentialCrossSection`

This node represents multi-dimensional differential cross section data, e.g. $d\sigma(E)/d\mu$ or $d\sigma(E)/d\mu dE'$. In most cases, such as in charged particle, thermal neutron or (in)coherent x-ray scattering, a parameterised form of the data exists to simplify the representation of the tabulated data.

Specifications for `doubleDifferentialCrossSection`

Node name: `doubleDifferentialCrossSection`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`regions3d`:** [optional, when present, must appear one time] For a differential cross section broken up into incident energy regions. Regions may contain different interpolations, or different representations (i.e. a reference for one incident energy range followed by a CoulombPlusNuclearElastic in another energy range).
- **`XYs3d`:** [optional, when present, must appear one time] A pointwise table of $\sigma(\mu, E'|E)$ or $\sigma(E', \mu|E)$ values.
- **`CoulombPlusNuclearElastic`:** [optional, when present, must appear one time] This is either a container holding the differential charged particle scattering cross section or a reference to said container.
- **`reference`:** [optional, when present, must appear one time] A link to resonance data, from which the differential cross section should be derived.

coherentPhotonScattering: [optional, when present, must appear one time] This is either a container holding the coherent photon scattering cross section or a reference to said container.

incoherentPhotonScattering: [optional, when present, must appear one time] This is either a container holding the incoherent photon scattering cross section or a reference to said container.

thermalNeutronScatteringLaw_coherentElastic: [optional, when present, must ap--pear one time] Contains the coherent elastic double-differential cross section for thermal neutrons.

thermalNeutronScatteringLaw_incoherentElastic: [optional, when present, must appear one time] Contains the incoherent elastic double-differential cross section for thermal neutrons.

thermalNeutronScatteringLaw_incoherentInelastic: [optional, when present, must appear one time] Contains the incoherent inelastic double-differential cross section for thermal neutrons.

XML Example(s) of `doubleDifferentialCrossSection`

Here a charged particle reaction given as an `CoulombPlusNuclearElastic`, namely elastic scattering of a deuteron off of ^{3}H, is shown. Along with the `doubleDifferentialCrossSec--tion` containing the `CoulombPlusNuclearElastic` (labelled with the 'eval' style), there is a `crossSection` and `outputChannel`. Neither the `crossSection` and `outputChannel` contain actual data, rather they contain `CoulombPlusNuclearElastic` elements that link back to the evaluated version in the `doubleDifferentialCrossSection` element. Similarly, the recoil distribution for the ^{3}H also links back to the evaluated `doubleDifferentialCross--Section` element

```xml
<doubleDifferentialCrossSection>
    <CoulombPlusNuclearElastic label="eval" pid="H2" productFrame="centerOfMass">
        <RutherfordScattering/>
        <nuclearPlusInterference muCutoff="0.94">
            <crossSection>
                <XYs1d>...</XYs1d>
            </crossSection>
            <distribution>
                <XYs2d>...</XYs2d>
            </distribution>
        </nuclearPlusInterference>
</doubleDifferentialCrossSection>
<crossSection>
     <CoulombPlusNuclearElastic
         label="eval"
         href="../../doubleDifferentialCrossSection/CoulombPlusNuclearElastic[@label='eval']"
         ↪  />
</crossSection>
<outputChannel genre="twoBody">
    <Q>...</Q>
    <products>
        <product pid="H2" label="H2">
            <multiplicity>...</multiplicity>
            <distribution>
                <CoulombPlusNuclearElastic
                        label="eval"
                        href="../../../../../doubleDifferentialCrossSection/
                             CoulombPlusNuclearElastic[@label='eval']"/>
            </distribution>
        </product>
        <product pid="H3" label="H3">
            <multiplicity>...</multiplicity>
            <distribution>
                <angularTwoBody label="eval" productFrame="centerOfMass">
                    <recoil
                        href="../../../../product[@label='H2']/distribution/
                             CoulombPlusNuclearElastic[@label='eval']"/>
                </angularTwoBody>
            </distribution>
        </product>
    </products>
</outputChannel>
```

16.2 Cross sections for resonance reactions

Many nuclear data evaluations use resonance parameters to express the fluctuating cross section in the resolved and unresolved resonance region. Cross sections for several different reactions can be derived from these resonance parameters, so a special `resonancesWithBackground` cross section form is defined to indicate that resonance reconstruction is needed. Currently this shows up most often in neutron-induced reactions, but in principal resonance parameters can also apply to incident charged particle reactions.

16.2.1 General transport type: `resonancesWithBackground`

The `resonancesWithBackground` node describes the cross sections as a set of resonance parameters with associated background cross sections.

Specifications for `resonancesWithBackground`

Node name: `resonancesWithBackground`
Attributes: The list of additional allowed attributes is:
- `label` [`XMLName`, **required**] Name of the style in the `styles` node associated with these data.

Child nodes: The list of additional allowed child nodes is:
- `resonances`: [**required**, must appear one time] Link to the resonance parameters that have to be combined with the background.
- `background`: [**required**, must appear one time] The background cross section to be applied.

`uncertainty`: [optional, when present, must appear one time]

XML Example(s) of `resonancesWithBackground`

```
<resonancesWithBackground
        label="...">
    <resonances>...</resonances>
    <background>...</background>
    <uncertainty>...</uncertainty></resonancesWithBackground>
```

16.2.2 General transport type: `resonances`

Links to the `resonances` section defined under the `reactionSuite`.

Specifications for `resonances`

Node name: `resonances`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, optional] Label for this link.
  `href` [`bodyText`, **required**] xPath link to the `resonances` section.
Child nodes: This element has no child nodes

XML Example(s) of `resonances`

```
<resonances
        label="..."
        href="..."></resonances>
```

16.2.3 General transport type: `background`

The `background` gives the background cross sections that have to be added to the resonance parameters to obtain the full set of cross sections.

Specifications for `background`

Node name: `background`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

resolvedRegion: [optional, when present, must appear one time] The background cross section in the resolved resonance region.
unresolvedRegion: [optional, when present, must appear one time] The back--ground cross section in the unresolved resonance region.
fastRegion: [optional, when present, must appear one time] The cross section in the fast region beyond the resolved and unresolved resonance region.

XML Example(s) of `background`

```
<background>
    <resolvedRegion>...</resolvedRegion>
    <unresolvedRegion>...</unresolvedRegion>
    <fastRegion>...</fastRegion></background>
```

16.2.4 General transport type: `resolvedRegion`

The `resolvedRegion` gives the background cross section in the resolved resonance region.

Specifications for `resolvedRegion`

Node name: `resolvedRegion`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of them is allowed):
XYs1d: [optional, when present, * one of marked children must appear one time] The background cross section is an energy dependent function with a single interpolation range.
regions1d: [optional, when present, * one of marked children must appear one time] The background cross section is an energy dependent function with multiple interpolation ranges.

XML Example(s) of `resolvedRegion`

```
<resolvedRegion>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></resolvedRegion>
```

16.2.5 General transport type: `unresolvedRegion`

The `unresolvedRegion` gives the background cross section in the unresolved resonance region.

Specifications for `unresolvedRegion`

Node name: `unresolvedRegion`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of them is allowed):
- **`XYs1d`:** [optional, when present, * one of marked children must appear one time] The background cross section is an energy dependent function with a single interpolation range.
- **`regions1d`:** [optional, when present, * one of marked children must appear one time] The background cross section is an energy dependent function with multiple interpolation ranges.

XML Example(s) of `unresolvedRegion`

```
<unresolvedRegion>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></unresolvedRegion>
```

16.2.6 General transport type: `fastRegion`

The `fastRegion` gives the entire cross section in the fast region, beyond the resolved and unresolved resonance region.

Specifications for `fastRegion`

Node name: `fastRegion`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of them is allowed):
- **`XYs1d`:** [optional, when present, * one of marked children must appear one time] The cross section as an energy dependent function with a single interpolation range.
- **`regions1d`:** [optional, when present, * one of marked children must appear one time] The cross section as an energy dependent function with multiple inter--polation ranges.

XML Example(s) of `fastRegion`

```
<fastRegion>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></fastRegion>
```

16.2.7 General transport type: `URR_probabilityTables1d`

The `URR_probabilityTables1d` gives probability distributions for the cross section as a function of incident energy, $P(\sigma|E)$. It is derived from the average widths and level spacings in the `unresolved` resonance section, and is used in in Monte Carlo transport codes. The child `XYs2d` or `regions2d` should generally use an `interpolationQualifier` like 'unitBase' since the cross section domain varies with incident energy.

Specifications for `URR_probabilityTables1d`

Node name: `URR_probabilityTables1d`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**] Name of the style in the `styles` node associated with these data.
Child nodes: The list of additional allowed child nodes is:
  `XYs2d`: [optional, when present, * one of marked children must appear one time] Probability distribution stored as an `XYs2d`.
  `regions2d`: [optional, when present, * one of marked children must appear one time] Probability distribution stored as a `regions2d`.

XML Example(s) of `URR_probabilityTables1d`

Here an example of a URR probability table is shown. The probability table consists of a set of cross section pdfs at various incident energies. The cdf is pre-computed for more efficient Monte Carlo sampling.

```
<URR_probabilityTables1d label="URR_tables">
  <XYs2d interpolationQualifier="correspondingPoints">
    <axes>
      <axis index="2" label="energy_in" unit="eV"/>
      <axis index="1" label="crossSection" unit="b"/>
      <axis index="0" label="P(crossSection|energy_in)" unit="1/b"/></axes>
    <xs_pdf_cdf1d value="1e3">...</xs_pdf_cdf1d>
    ...
    <xs_pdf_cdf1d value="2e4">...</xs_pdf_cdf1d>
  </XYs2d>
</URR_probabilityTables1d>
```

17. Reaction outputs

17.1 The `outputChannel` element

A `reaction` is composed of a `crossSection` and an output channel which includes the reaction Q-value and a list of products. Furthermore, some reaction products are subject to breakup or decay. These processes also have an associated Q-value and list of products. Whether the output channel comes directly from a reaction or from a subsequent decay, it is stored as an `outputChannel` in GNDS.

17.1.1 General transport type: `outputChannel`

The `outputChannel` node contains all the data for the results of a reaction, including the Q-value and all outgoing products.

Specifications for `outputChannel`

Node name: `outputChannel`
Attributes: The list of additional allowed attributes is:
- **`genre`** [`XMLName`, optional] This can have a value of either `twoBody` or `NBody`
- **`process`** [`XMLName`, optional]

Child nodes: The list of additional allowed child nodes is:
- **`Q`:** [optional, when present, must appear one time] The Q-value of the reaction. Fission reactions have a different format than other reactions.
- **`products`:** [optional, when present, must appear one time] A list of secondary products from the reaction
- **`fissionFragmentData`:** [optional, when present, must appear one time] For use in fission reactions only. Contains information about delayed neutrons, fission product yields and energy partition.

XML Example(s) of `outputChannel`

```
<outputChannel
        genre="..."
        process="...">
    <Q>...</Q>
    <products>...</products>
    <fissionFragmentData>...</fissionFragmentData></outputChannel>
```

17.1.2 Common type: `Q`

Specifications for `Q`

Node name: `Q`
Abstract node: `physicalQuantity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

- **`constant1d`:** [optional, when present, * one of marked children must appear one time]
- **`XYs1d`:** [optional, when present, * one of marked children must appear one time]
- **`regions1d`:** [optional, when present, * one of marked children must appear one time]
- **`polynomial1d`:** [optional, when present, * one of marked children must appear one time]
- **`gridded1d`:** [optional, when present, * one of marked children must appear one time] Grouped Q-value, weighted by the cross-section and optionally by the flux. Appears in processed files only.

XML Example(s) of `Q`

```
<Q>
    <constant1d>...</constant1d>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d>
    <polynomial1d>...</polynomial1d>
    <gridded1d>...</gridded1d></Q>
```

17.1.3 Common type: `products`

There are three “types” of product lists:

- reaction

- orphaned
- decay

Each type has the same structure; a constant for the Q-value, one or more child `product` elements, and optional `documentations` child element. Some of the child `product` ele--ments may be unstable and decay further. The decay products of the unstable product are not given here, but are given explicitly in the particle database.

Specifications for `products`

Node name: `products`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `product`: [optional, when present, must appear at least one time] One or more `product` child element.

XML Example(s) of `products`

```
<products>
    <product>...</product></products>
```

17.2 A product (of a reaction or decay)

Reaction products (and decay / breakup products) are stored in the GNDS `product` node. Each product has a `multiplicity` and `distribution`, and may also contain information about the average product energy / momentum (commonly contained in processed files).

17.2.1 Common type: `product`

A product is a particle that is the result of a reaction or decay. More than one particle of the same type can be released in a single reaction (e.g. (n, 2n)). A multiplicity (how many are produced) must be defined for each product. A distribution describing the outgoing energy and angle of the product is also required, although it may have a value of `"unspecified"`.

Specifications for `product`

Node name: `product`
Attributes: The list of additional allowed attributes is:

`label` [`XMLName`, **required**] Product label. Must be unique within the list of `products`
`pid` [`XMLName`, **required**] Particle id, must correspond to the id of a particle in the particle database.

Child nodes: The list of additional allowed child nodes is:

`multiplicity`: [**required**, must appear one time] Component containing the mul- -tiplicity of the product.
`distribution`: [**required**, must appear one time] Component containing the en- -ergy/angle distribution of the product.
`outputChannel`: [optional, when present, must appear one time] Used if the prod- -uct breaks up into fragments (e.g. `"Be8 -> He4 + He4"`)

XML Example(s) of `product`

```
<product
        label="..."
        pid="...">
    <multiplicity>...</multiplicity>
    <distribution>...</distribution>
    <outputChannel>...</outputChannel></product>
```

17.3 Multiplicities

The `multiplicity` node tells how many of a particular product are emitted by a reaction. For most reactions a product's multiplicity is a constant, e.g. '1'. For more complex reactions such as fission, however, the multiplicity is typically given as an energy-dependent average rather than as a constant.

17.3.1 General transport type: `multiplicity`

Specifications for `multiplicity`

Node name: `multiplicity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

`XYs1d`: [optional, when present, * one of marked children must appear one time]
`constant1d`: [optional, when present, * one of marked children must appear one time]
`polynomial1d`: [optional, when present, * one of marked children must appear one time]
`branching1d`: [optional, when present, * one of marked children must appear one time] Multiplicity is computed from decay branching ratios listed in the `PoPs` database.

reference: [optional, when present, * one of marked children must appear one time]
gridded1d: [optional, when present, * one of marked children must appear one time] Grouped multiplicy, weighted by the cross-section and optionally by the flux. Appears in processed files only.
regions1d: [optional, when present, * one of marked children must appear one time]

XML Example(s) of `multiplicity`

```
<multiplicity>
    <XYs1d>...</XYs1d>
    <constant1d>...</constant1d>
    <polynomial1d>...</polynomial1d>
    <branching1d>...</branching1d>
    <reference>...</reference>
    <gridded1d>...</gridded1d>
    <regions1d>...</regions1d></multiplicity>
```

17.3.2 General transport type: `reference`

A multiplicity may consist of a link to another multiplicity in the same file. The primary purpose for this option is to associate fission neutron multiplicity with a fission `reaction`.

Specifications for `reference`

Node name: `reference`
Abstract node: `labelNode`
Attributes: The list of additional allowed attributes is:
href [`bodyText`, optional] Link (using xPath syntax) to another multiplicity in the file. Link may be absolute or relative.
label [`XMLName`, optional]
Child nodes: This element has no child nodes

XML Example(s) of `reference`

```
<reference
        href="..."
        label="..."></reference>
```

17.3.3 General transport type: `branching1d`

Indicates that the multiplicity can be computed from decay information in the `PoPs` database. Mainly used for photons emitted from discrete excited states. Also see Section 18.4.

Specifications for `branching1d`

Node name: `branching1d`
Abstract node: `labelNode`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**] Unique label for the multiplicity form.
Child nodes: This element has no child nodes

XML Example(s) of `branching1d`

```
<branching1d
        label="..."></branching1d>
```

17.4 Average product data

Processed files sometimes contain the average cross-section-weighted outgoing product energy and momentum. These can be used to compute quantities such as the KERMA (Kinetic Energy Release to the Material), useful for computing heating and material damage.

17.4.1 Processed type: `averageProductEnergy`

A component function1d representing the lab-frame, average energy for an outgoing particle integrated over outgoing μ and energy E'. This is needed to, for example, calculate KERMA.

Specifications for `averageProductEnergy`

Node name: `averageProductEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `XYs1d`: [**required**, must appear one time] A function1d representing the lab-frame, average energy for an outgoing particle integrated over outgoing μ and energy E'.

gridded1d: [optional, when present, must appear one time]
regions1d: [optional, when present, must appear one time]

XML Example(s) of `averageProductEnergy`

```
<averageProductEnergy>
    <XYs1d>...</XYs1d>
    <gridded1d>...</gridded1d>
    <regions1d>...</regions1d></averageProductEnergy>
```

17.4.2 Processed type: `averageProductMomentum`

A component function1d representing the lab-frame, average momentum for an outgo--ing particle integrated over outgoing μ and energy E'.

Specifications for `averageProductMomentum`

Node name: `averageProductMomentum`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
XYs1d: [optional, when present, must appear one time]
gridded1d: [optional, when present, must appear one time]
regions1d: [optional, when present, must appear one time]

XML Example(s) of `averageProductMomentum`

```
<averageProductMomentum>
    <XYs1d>...</XYs1d>
    <gridded1d>...</gridded1d>
    <regions1d>...</regions1d></averageProductMomentum>
```

18. Product distributions

18.1 The `distributions` element

When a moving particle, called a projectile, hits another particle at rest, called a target, two or more outgoing particles, called products, are produced. A distribution describes the probability $P(\mu, E'|E)$ for an product to be emitted with outgoing angular cosine $\mu = \cos(\theta)$ and energy E' when the projectile has energy E. Here θ is the angle between the velocity vectors of the projectile and product. The projectile's energy can be given in the laboratory (lab) frame or the center-of-mass (com) frame. Since the projectile's frame is specified as the `projectileFrame` attribute in the `reactionSuite` element, it does not need to be stored in the product's distribution data. The product quantities (E', μ and $P(\mu, E'|E)$) can also be given in the laboratory or center-of-mass frame. The frames for these three product quantities are the same (i.e. they are whatever frame the product is being expressed in). In addition, the frame can depend on the representation of the distribution. For example, in two-body reactions (see Section 18.2.1) the evaluator should only use a $P(\mu|E)$ in the center-of-mass frame for one of the products. When these data are processed by a deterministic Pn code, they generally produce multi-group data in the lab frame. Therefore, the product's frame must be attached to each distribution representation.

A product's distribution is specified by the `distribution` node. A `distribution` can contain various representations of the distribution.

18.1.1 General transport type: `distribution`

Container for all (un)correlated energy-angle distributions.

Specifications for `distribution`

Node name: `distribution`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

angularTwoBody: [optional, when present, * one of marked children must appear one time] This format is used to describe the distribution in energy and angle of particles described by two-body kinematics. Since the energy of a particle emitted with a particular scattering cosine μ is determined by kinematics, it is only necessary to give $P(\mu|E)$.

uncorrelated: [optional, when present, * one of marked children must appear one time] The outgoing energy and angular data are uncorrelated with respect to the projectile's energy and can thus be written as a product of the outgoing energy probability times the outgoing angular probability as $P(\mu, E'|E) = P(E'|E)\,P(\mu|E)$. Both $P(\mu, E'|E)$ and $P(E'|E)$ must be normalised to 1.

angularEnergy: [optional, when present, * one of marked children must appear one time] The distribution is a hierarchy in E, then μ and finally E', P. That is, the data are given at a list of E values. For each E, there is an associated $P(\mu, E')$. Each $P(\mu, E')$ is given as a list μ values. For each μ, there is an associated $f(E')$. The $f(E')$ can be an `XYs1d` or `regions1d` functional data container.

energyAngular: [optional, when present, * one of marked children must appear one time] The distribution is stored as a hierarchy in E, then E' and finally μ, P. That is, the data are given at a list of E values. For each E, there is an associated $P(E', \mu)$. Each $P(E', \mu)$ is given as a list E' values. For each E', there is an associated $f(\mu)$. The $f(\mu)$ can be an `XYs1d`, `regions1d` or `Legendre` functional data container.

KalbachMann: [optional, when present, * one of marked children must appear one time] The distribution is defined using Kalbach-Mann systematics.

reference: [optional, when present, * one of marked children must appear one time] Contains a link to another `distribution` form.

branching3d: [optional, when present, * one of marked children must appear one time] Indicates that the distribution can be computed from the decay information listed in the `PoPs` database. Currently only used for photon distributions from the decay of discrete excited states (e.g. after inelastic scattering).

CoulombPlusNuclearElastic: [optional, when present, * one of marked children must appear one time] This is either a container holding the differential charged particle scattering cross section or a reference to said container.

thermalNeutronScatteringLaw: [optional, when present, * one of marked children must appear one time] This contains a reference to the double-differential thermal neutron scattering law cross section.

coherentPhotonScattering: [optional, when present, * one of marked children must appear one time] Describes coherent photon scattering off of an atom.

incoherentPhotonScattering: [optional, when present, * one of marked children must appear one time] Describes incoherent photon scattering off of an atom.

unspecified: [optional, when present, * one of marked children must appear one time] The distribution is not specified[17].

multiGroup3d: [optional, when present, * one of marked children must appear one time] (appears only in processed data files). This is a multi-group Legen-

[17]This often happens for reactions with a heavy product, often called the residual, where the evaluator does not give the heavy product's distribution information. It also happens in legacy ENDF files where only the neutron and maybe the gamma distributions are given by the evaluator.

-dre expanded representation with grouped integration over E and E'. The representation is used by deterministic transport codes.

angularEnergyMC: [optional, when present, * one of marked children must appear one time] (appears only in processed data files). The distribution is the product of $P(\mu, E) \times P(E'|E, \mu)$. This representation is used by Monte Carlo transport codes.

energyAngularMC: [optional, when present, * one of marked children must appear one time] (appears only in processed data files). The distribution is the product of $P(E', E) \times P(\mu|E, E')$. This representation is used by Monte Carlo transport codes.

XML Example(s) of `distribution`

```
<distribution>
    <angularTwoBody>...</angularTwoBody>
    <uncorrelated>...</uncorrelated>
    <angularEnergy>...</angularEnergy>
    <energyAngular>...</energyAngular>
    <KalbachMann>...</KalbachMann>
    <reference>...</reference>
    <branching3d>...</branching3d>
    <CoulombPlusNuclearElastic>...</CoulombPlusNuclearElastic>
    <thermalNeutronScatteringLaw>...</thermalNeutronScatteringLaw>
    <coherentPhotonScattering>...</coherentPhotonScattering>
    <incoherentPhotonScattering>...</incoherentPhotonScattering>
    <unspecified>...</unspecified>
    <multiGroup3d>...</multiGroup3d>
    <angularEnergyMC>...</angularEnergyMC>
    <energyAngularMC>...</energyAngularMC></distribution>
```

Each `distribution` element can have multiple representations of the product's distribution. Each representation of a product's distribution can be in a different frame; therefore, the product's frame must be specified with each representation using the attribute `productFrame`.

The distribution forms listed under the `distribution` node fall into a few basic types. For *evaluated* nuclear data, distributions may be two-body, 'N-body' (also known as 'un--correlated bodies'[18]), or they may be derived from a `doubleDifferentialCrossSection`.

[18]Uncorrelated bodies and uncorrelated distributions are not the same thing. For a reaction containing uncorrelated bodies, each product's distribution is listed independent of all the other products; hence when sampling the products in a typical Monte Carlo code, energy and momentum cannot be conserved per reaction as each product's distribition is sampled independent of the other product's distributions. They are only conserved when statistically averaged. An uncorrelated distribution, on the other hand, implies that for a single product the outgoing energy E' and angle θ are uncorrelated. A reaction can contain both correlated and uncorrelated products. For example, a reaction may start as two-body but then one of the products breaks up into more products. For the initial two-body reaction the energy and momentum of the two products are correlated, but the products listed in the breakup may be two-body emission or uncorrelated bodies.

18.2 Two-body distribution representations

For two-body, only two products are emitted[19] and only the angular distribution in the center-of-mass frame for one of the products is needed. The outgoing energy in the center-of-mass and laboratory frames, and the outgoing angle in the laboratory frame are easily derived from kinematics. In the center-of-mass the distribution[20] $P_{\text{com}}(\mu, E'|E)$ can be written as

$$P_{\text{com}}(\mu, E'|E) = P_{\text{com}}(\mu|E)\,\delta(E' - E'(E)) \tag{18.1}$$

where $\delta(E' - E'(E,\mu))$ is the Dirac delta function and $E'(E,\mu)$ is the product's outgoing energy calculated from kinematics[21].

A two-body output channel is contained in an `angularTwoBody` node.

18.2.1 General transport type: `angularTwoBody`

This form is used to describe the distribution in energy and angle of particles described by two-body kinematics. It corresponds to ENDF's File 6, LAW=2 and is very similar to ENDF's File 4. Since the energy of a particle emitted with a particular scattering cosine μ is determined by kinematics, it is only necessary to specify the angular distribution which is easiest to do using Legendre polynomials:

$$\begin{aligned} p_i(\mu, E) &= \int dE'\; f_i(\mu, E, E') \\ &= \frac{1}{2} + \sum_{l=1}^{\text{NL}} \frac{2l+1}{2}\; a_l(E)\; P_l(\mu), \end{aligned} \tag{18.2}$$

where the P_l are the Legendre polynomials with maximum, order NL. Note that the angular distribution p_i is normalised.

Specifications for `angularTwoBody`

Node name: `angularTwoBody`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:

`label` [`XMLName`, optional] Name of the style in the `styles` element associated with these data.

[19]Either of the intial two outgoing products may subsequently break up. In this case, the first step is two-body and subsequent steps may be a two-body and/or and uncorrelated bodies break ups of the parent product.

[20]Since the product quantities E', μ and $P(\mu|E)$ all have the same frame, in this document the frame will be designated on the dependent quantity P.

[21]$E'(E,\mu)$ also depends on other variables, like the masses of the ingoing and outgoing particles, which are not explicitly shown.

`productFrame` [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- **`XYs2d`:** [optional, when present, must appear one time] $P(\mu|E)$ given as a 2d inter--polation table.
- **`regions2d`:** [optional, when present, must appear one time] The angular distribu--tion is composed of multiple regions contained within this container.
- **`isotropic2d`:** [optional, when present, must appear one time] The angular distri--bution is isotropic so $P(\mu|E) = 1/2$.
- **`recoil`:** [optional, when present, must appear one time] The particle in question is recoiling from the rest of the system and it's angular distribution can be determined by kinematics.

XML Example(s) of `angularTwoBody`

```
<angularTwoBody
        label="..."
        productFrame="...">
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d>
    <isotropic2d>...</isotropic2d>
    <recoil>...</recoil></angularTwoBody>
```

If the `angularTwoBody` contains `XYs2d` or `regions2d` functional data, they must repre--sents a valid $f(E, \mu)$ or $P(\mu|E)$ with the proper normalisation. One cannot tell from the top element of the functional angular representation whether the lower level angular data $f(\mu)$ are represented with an `XYs1d` and/or `Legendre` functional data containers. Each `XYs2d` can only contain either a list of `XYs1d` or a list of `Legendre`. To have a `Legendre` in one region and a `XYs1d` in another, one must use a `regions2d` containing two `XYs2d`: one containing only the `Legendre` containers and the other containing only the `XYs1d` containers.

18.2.2 General transport type: `isotropic2d`

When the scattering is isotropic and independent of energy E the `isotropic` angular distribution can be used. Normalisation is always 1; hence, $f(E, \mu) = P(\mu, E) = 1/2$.

Specifications for `isotropic2d`

Node name: `isotropic2d`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `isotropic2d`

```
<isotropic2d/>
```

18.2.3 General transport type: `recoil`

In `recoil` scattering, the angular distribution is the reflection in all axes of the other emitted particle's two-body product. Hence, $f(E,\mu) = f_{\text{other}}(E,-\mu)$ where $f_{\text{other}}(E,\mu)$ is the other product's angular distribution.

Specifications for `recoil`

Node name: `recoil`
Abstract node: `link`
Attributes: The list of additional allowed attributes is:
 `href` [`bodyText`, **required**] A link to the other product's distribution representation in a two-body channel.
Child nodes: This element has no child nodes

XML Example(s) of `recoil`

```
<recoil
       href="..."></recoil>
```

18.3 N-body distribution representations

For uncorrelated bodies, the full $P(\mu, E'|E)$ must be given for each product. There are various way to represent $P(\mu, E'|E)$ in the legacy ENDF-6 and all are supported in GNDS as described in the following subsections.

18.3.1 General transport type: `angularEnergy`

A valid $P(\mu, E'|E)$ represented with either a `XYs3d` or `regions3d` functional data con- -tainer. A `regions3d` must contain one or more `XYs3d`. Each `XYs3d` must contain a list of `XYs2d` and/or `regions2d` functional data container. A `regions2d` must contain one or more `XYs2d`. Each `XYs2d` must contain a list of `XYs1d` and/or `regions1d` functional data container. Each `regions1d` must contain one or more `XYs1d` functional data container.

Specifications for `angularEnergy`

Node name: `angularEnergy`
Attributes: The list of additional allowed attributes is:

- `label` [`XMLName`, **required**] Name of the style in the `styles` element associated with these data.
- `productFrame` [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- `XYs3d`: [**required**, must appear one time] Distribution given as $P(\mu, E'|E)$ using a μ major ordering.

XML Example(s) of `angularEnergy`

```
<angularEnergy
        label="..."
        productFrame="...">
    <XYs3d>...</XYs3d></angularEnergy>
```

18.3.2 General transport type: `energyAngular`

Specifications for `energyAngular`

Node name: `energyAngular`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:

- `label` [`XMLName`, optional] Name of the style in the `styles` element associated with these data.
- `productFrame` [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- `XYs3d`: [**required**, must appear one time] Distribution given as $P(E', \mu|E)$ using a E' major ordering.

XML Example(s) of `energyAngular`

```
<energyAngular
        label="..."
        productFrame="...">
    <XYs3d>...</XYs3d></energyAngular>
```

18.3.3 General transport type: `unspecified`

The distribution is not specified[22].

Specifications for `unspecified`

Node name: `unspecified`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, optional] Name of the style in the `styles` element associated with these data.
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in.

Child nodes: This element has no child nodes

XML Example(s) of `unspecified`

```
<unspecified
        label="..."
        productFrame="..."></unspecified>
```

18.4 Distributions computed from decay branching ratios

Branching multiplicity (`branching1d`) and distribution (`branching3d`) nodes were added to support photon decay of the residual in (n,n'), (n,p') and similar reactions where the outgoing particles are a photon (i.e. gamma) and a residual in an excited state. And, where the decay of the residual is specified in the `PoPs` node as branching ratios. It is possible that a decay may lead to a meta-stable and where the decay should stop at the meta-stable (depending on the need of the user). For example, consider a neutron hitting Am242 which can have, among its many reactions, the two output channels "n + Am242", "n + Am242_e2". If the user wants to track Am242_m1 which is Am242_e2, then all decays leading to Am242_e2 should stop there. To determine the decay paths and photons leading to Am242_e2, one must stop when Am242_e2 is reached and ignore the other decays that lead to Am242. Similarly, to calcuate the photons where the decay ends with Am242, one must ignore all paths reaching Am242_e2. In general, if isotope with symbol S and nucleon number A (SA) has the listed meta-stables SA_m1, SA_m2 and SA_m3, then for all photons leading to SA_m2, one must stop all decays when they reach SA_m2 and ignore all decays that reach SA, SA_m1 and SA_m3. That is, form the list of the ground state and all meta-stables, then ignore all decays that lead to the particles in the list that are not being considered and stop when the one being considered is reached.

[22]This often happens for reactions with a heavy product, (the heavy product is often called the residual) where the evaluator does not give the heavy product's distribution information. It also happens in legacy ENDF files where only the neutron and maybe the gamma distributions are given by the evaluator.

18.4.1 General transport type: `branching3d`

Used when a distribution (typically for photons emitted from a discrete excited state) can be computed from decay branching ratios listed in `PoPs`.

Specifications for `branching3d`

Node name: `branching3d`
Attributes: The list of additional allowed attributes is:

- `label` [`XMLName`, **required**] Name of the style in the `styles` element associated with these data.
- `productFrame` [`frame`, **required**] The frame that the product data are defined in.

Child nodes: This element has no child nodes

XML Example(s) of `branching3d`

```
<branching3d
        label="..."
        productFrame="..."></branching3d>
```

18.5 The Kalbach-Mann formulation for correlated energy-angular distributions

Kalbach-Mann systematics are a powerful representation for outgoing neutron and charged particle energy-angle distributions in high energy (usually pre-equilibrium) reactions (Kalbach & Mann, 1981). The distributions described here are represented by using the systematics in the extended form developed by Kalbach (Kalbach, 1988) and Chadwick, Young and Chiba (Chadwick et al., 1995). The distributions are given in terms of the parameters r and a, which are described below, and the parameter f_0. The parameter f_0 is the total emission probability.

Note, it is ***required*** that the f_0, a, b and E' be specified in the *center-of-mass system*.

The Kalbach-Mann formulation addresses reactions of the form

A + a → C → B + b,

where:

A is the target,
a is the incident projectile,
C is the compound nucleus,

b is the emitted particle,
B is the residual nucleus.

The following quantities are defined:

E_{a}	energy of the incident projectile a in the *laboratory system*. Usually this is denoted E, but here the "a" subscript adds clarity.
ϵ_{a}	entrance channel energy, the kinetic energy of the incident projectile a and the target particle A in the *center-of-mass system*, defined by: $$\epsilon_{\mathsf{a}} = E_{\mathsf{a}} \times \frac{\mathrm{AWR}_{\mathsf{A}}}{\mathrm{AWR}_{\mathsf{A}} + \mathrm{AWR}_{\mathsf{a}}}$$
$E_{\mathsf{b},cm}$	energy of the emitted particle in the *center-of-mass system*. Usually this is denoted E', but here the "b" and "cm" subscripts add clarity.
ϵ_{b}	emission channel energy, the sum of the kinetic energies of the emitted particle b and the residual nucleus B in the *center-of-mass system*, defined by: $$\epsilon_{\mathsf{b}} = E_{\mathsf{b},cm} \times \frac{\mathrm{AWR}_{\mathsf{B}} + \mathrm{AWR}_{\mathsf{b}}}{\mathrm{AWR}_{\mathsf{B}}}$$
$\mu_{\mathsf{b},cm}$	cosine of the scattering angle of the emitted particle b in the *center-of-mass system*. Usually this is denoted μ, but here the "b" and "cm" subscripts add clarity.

18.5.1 For projectiles other than photons

The double differential distribution is represented by (Kalbach, 1988):

$$f(\mu_{\mathsf{b},cm}, E_{\mathsf{a}}, E_{\mathsf{b},cm}) = \frac{a f_0}{2\sinh(a)}\left[\cosh\left(a\mu_{\mathsf{b},cm}\right) + r\sinh\left(a\mu_{\mathsf{b},cm}\right)\right] \tag{18.3}$$

where $r = r(E_{\mathsf{a}}, E_{\mathsf{b},cm})$ is the pre-compound fraction as given by the evaluator, $a = a(E_{\mathsf{a}}, E_{\mathsf{b},cm})$ is a simple parameterised function that depends mostly on the center-of-mass emission energy $E_{\mathsf{b},cm}$, but also depends slightly on particle type and the incident energy at higher values of E_{a}, and $f_0 = f_0(E_{\mathsf{a}}, E_{\mathsf{b},cm})$ is the total emission probability. The pre-compound fraction r ranges from 0.0 to 1.0 and is usually computed by a model code, although it can be chosen to fit experimental data.

The formula[23] for calculating the slope value $a = a(E_{\mathsf{a}}, E_{\mathsf{b},cm})$ is:

[23]Equation 10 of (Kalbach, 1988).

$$a = C_1 X_1 + C_2 X_1^3 + C_3 M_{\mathrm{a}} m_{\mathrm{b}} X_3^4$$

where:

$e_{\mathrm{a}} = \epsilon_{\mathrm{a}} + S_{\mathrm{a}}$	$e_{\mathrm{b}} = \epsilon_{\mathrm{b}} + S_{\mathrm{b}}$
R_1 = minimum(e_{a}, E_{t1})	R_3 = minimum(e_{a}, E_{t3})
$X_1 = R_1 e_{\mathrm{b}}/e_{\mathrm{a}}$	$X_3 = R_3 e_{\mathrm{b}}/e_{\mathrm{a}}$

The parameter values for light particle induced reactions as given in Ref. 2[24] are:

C_1 = 0.04/MeV	C_2 = 1.8$\times 10^{-6}$/(MeV)^3
C_3 = 6.7$\times 10^{-7}$/(MeV)^4	
E_{t1} = 130 MeV	E_{t3} = 41 MeV
M_n = 1	M_p = 1
M_d = 1	M_α = 0
m_n = 1/2	m_p = 1
m_d = 1	m_t = 1
m_{3He} = 1	m_α = 2

Note that Kalbach never specified M_t or M_{3He} in (Kalbach, 1988) as the systematics have not been extended to high energy t and ^{3}He projectiles, where the X_3^4 term contributes.

S_{a} and S_{b} are the separation energies for the incident and emitted particles, re--spectively, for the reaction A + a $\rightarrow$ C $\rightarrow$ B + b. As the particle emission is occuring for relatively high energies, pairing and shell effects in the compound nucleus are muted and must be neglected in the calculation of separation energies. Therefore, one should use the following formulae for the separation energies[25] in MeV:

$$\begin{aligned} S_{\mathrm{a}} &= 15.68\left[A_{\mathrm{C}} - A_{\mathrm{A}}\right] - 28.07\left[\frac{(N_{\mathrm{C}}-Z_{\mathrm{C}})^2}{A_{\mathrm{C}}} - \frac{(N_{\mathrm{A}}-Z_{\mathrm{A}})^2}{A_{\mathrm{A}}}\right] \\ &- 18.56\left[A_{\mathrm{C}}^{2/3} - A_{\mathrm{A}}^{2/3}\right] + 33.22\left[\frac{(N_{\mathrm{C}}-Z_{\mathrm{C}})^2}{A_{\mathrm{C}}^{4/3}} - \frac{(N_{\mathrm{A}}-Z_{\mathrm{A}})^2}{A_{\mathrm{A}}^{4/3}}\right] \\ &- 0.717\left[\frac{Z_{\mathrm{C}}^2}{A_{\mathrm{C}}^{1/3}} - \frac{Z_{\mathrm{A}}^2}{A_{\mathrm{A}}^{1/3}}\right] + 1.211\left[\frac{Z_{\mathrm{C}}^2}{A_{\mathrm{C}}} - \frac{Z_{\mathrm{A}}^2}{A_{\mathrm{A}}}\right] - I_{\mathrm{a}} \end{aligned}$$

and

$$S_{\mathrm{b}} = 15.68\left[A_{\mathrm{C}} - A_{\mathrm{B}}\right] - 28.07\left[\frac{(N_{\mathrm{C}}-Z_{\mathrm{C}})^2}{A_{\mathrm{C}}} - \frac{(N_{\mathrm{B}}-Z_{\mathrm{B}})^2}{A_{\mathrm{B}}}\right]$$

[24]Table V of (Kalbach, 1988).

[25]Equation 4 of (Kalbach, 1988).

$$- \quad 18.56 \left[A_{\mathrm{C}}^{2/3} \quad - A_{\mathrm{B}}^{2/3}\right] + 33.22 \left[\frac{(N_{\mathrm{C}} - Z_{\mathrm{C}})^2}{A_{\mathrm{C}}^{4/3}} - \frac{(N_{\mathrm{B}} - Z_{\mathrm{B}})^2}{A_{\mathrm{B}}^{4/3}}\right]$$
$$- \quad 0.717 \left[\frac{Z_{\mathrm{C}}^2}{A_{\mathrm{C}}^{1/3}} - \frac{Z_{\mathrm{B}}^2}{A_{\mathrm{B}}^{1/3}}\right] + 1.211 \left[\frac{Z_{\mathrm{C}}^2}{A_{\mathrm{C}}} - \frac{Z_{\mathrm{B}}^2}{A_{\mathrm{B}}}\right] - I_{\mathrm{b}}$$

where:

A, B, C	subscripts refer to the target nucleus, the residual nucleus, and the compound nucleus, as before,
N, Z, A	are the neutron, proton, and mass numbers of each nuclei,
I_{a}, I_{b}	are the energies required to separate the incident and emitted particles into their constituent nucleons (see Appendix H for values used for given particles).

The Kalbach formulation uses a single-particle-emission concept; it is assumed that each and every secondary particle is emitted from the same original compound nucleus C. For example, in a reaction induced by particle z and forming compound nucleus C, all of the neutrons emitted in the following reactions use the same S_{n}: (z,nα), (z,n3α), (z,2nα), (z,np), (z,2n2α), and (z,nt2α). As an additional consequence, when the incident projectile a and the emitted particle b are the same, $S_{\mathrm{a}} = S_{\mathrm{b}}$ regardless of the reaction.

18.5.2 For photons as projectiles

In (Chadwick et al., 1995), Chadwick, Young and Chiba extended the Kalbach-Mann systematics to the case of incident gammas in the quasi-deuteron regime. These systematics are used in several libraries, including ENDF/B-VII.0 (Chadwick et al., 2006), BROND-3 (Blokhin et al., 2016) and the IAEA Photonuclear Library (IAEA, 2000). E_γ is incident gamma energy in laboratory system. The extension to incident gammas requires one to plug E_γ into the spot where one would use the incident neutron energy when computing corresponding a for neutrons to obtain a_n. One then multiplies by a factor which accounts for the lower momentum carried by the incident photon:

$$a_\gamma(E_\gamma, E_{\mathrm{b},cm}) = a_n(E_\gamma, E_{\mathrm{b},cm})\sqrt{\frac{E_\gamma}{2m_n}}\,\mathrm{min}(4, \mathrm{max}(1, 9.3/\sqrt{E_{\mathrm{b},cm}})) \tag{18.4}$$

where both emitted particle energy, $E_{\mathrm{b},cm}$, and the neutron mass, m_n, are given in MeV. With this, the double differential distribution is modified as follows:

$$f(\mu_{\mathrm{b},cm}, E_\gamma, E_{\mathrm{b},cm}) = \frac{f_0}{2}\left[(1-r) + r\frac{a}{\sinh(a)}\exp(a\mu_{\mathrm{b},cm})\right] \tag{18.5}$$

Here f_0 and r are a function of E_γ and $E_{\mathrm{b},cm}$ and have the same interpretation as for all of the other projectile types.

18.5.3 General transport type: `KalbachMann`

Kalbach-Mann distribution for outgoing particles.

Specifications for `KalbachMann`

Node name: `KalbachMann`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, optional] Name of the style in the `styles` element associated with these data.
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in. Must always be `"centerOfMass"`.

Child nodes: The list of additional allowed child nodes is:
- **`f`:** [**required**, must appear one time] Kalbach-Mann f parameter.
- **`r`:** [**required**, must appear one time] Kalbach-Mann r parameter.
- **`a`:** [optional, when present, must appear one time] Kalbach-Mann a parameter.

XML Example(s) of `KalbachMann`

```
<KalbachMann
        label="..."
        productFrame="...">
    <f>...</f>
    <r>...</r>
    <a>...</a></KalbachMann>
```

18.5.4 General transport type: `f`

Kalbach-Mann $f(E, E')$ parameter, stored either as an `XYs2d` or `regions2d` functional data container.

Specifications for `f`

Node name: `f`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of them is allowed):
- **`XYs2d`:** [optional, when present, * one of marked children must appear one time]
- **`regions2d`:** [optional, when present, * one of marked children must appear one time]

XML Example(s) of `f`

```
<f>
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d></f>
```

18.5.5 General transport type: `r`

Kalbach-Mann $r(E, E')$ parameter, stored either as an `XYs2d` or `regions2d` functional data container.

Specifications for `r`

Node name: `r`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of them is allowed):
- **`XYs2d`:** [optional, when present, * one of marked children must appear one time]
- **`regions2d`:** [optional, when present, * one of marked children must appear one time]

XML Example(s) of `r`

```
<r>
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d></r>
```

18.5.6 General transport type: `a`

Kalbach-Mann $a(E, E')$ parameter, stored either as an `XYs2d` or `regions2d` functional data container.

Specifications for `a`

Node name: `a`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of them is allowed):
- **`XYs2d`:** [optional, when present, * one of marked children must appear one time]
- **`regions2d`:** [optional, when present, * one of marked children must appear one time]

XML Example(s) of `a`

```
<a>
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d></a>
```

18.6 Uncorrelated energy-angular distributions

18.6.1 General transport type: `uncorrelated`

The outgoing energy and angular data are uncorrelated with respect to the projectile's energy and can thus be written as a product of the outgoing energy probability times the outgoing angular probability as $P(\mu, E'|E) = P(E'|E)\, P(\mu|E)$. Note, both $P(\mu, E'|E)$ and $P(E'|E)$ must be normalised to 1.

Specifications for `uncorrelated`

Node name: `uncorrelated`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:

- `label` [`XMLName`, optional] Name of the style in the `styles` element associated with these data.
- `productFrame` [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- `angular`: [**required**, must appear one time] Any valid $P(\mu|E)$ as defined in Sec--tion 18.6.2.
- `energy`: [**required**, must appear one time] Any valid $P(E'|E)$ as defined in Sec--tions 18.7, 18.8, 18.10, or 18.9.

XML Example(s) of `uncorrelated`

```
<uncorrelated
        label="..."
        productFrame="...">
    <angular>...</angular>
    <energy>...</energy></uncorrelated>
```

18.6.2 General transport type: `angular`

Specifications for `angular`

Node name: `angular`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `XYs2d`: [optional, when present, must appear one time] $P(\mu|E)$ given as a 2d inter--polation table.
- `isotropic2d`: [optional, when present, must appear one time] The angular distri--bution is isotropic so $P(\mu|E) = 1/2$.
- `forward`: [optional, when present, must appear one time] The angular distribution is pure forward-peaked so $P(\mu|E) = \delta(\mu - 1)$.

XML Example(s) of `angular`

```
<angular>
    <XYs2d>...</XYs2d>
    <isotropic2d>...</isotropic2d>
    <forward>...</forward></angular>
```

18.6.3 General transport type: `forward`

In forward scattering, the product goes in the same direction as the projectile. As the distributions is always normalised to 1, $f(E, \mu) = P(\mu|E) = \delta(\mu - 1)$. This is needed per Section 26.1, paragraph 3 of the ENDF manual (Cross Section Evaluation Working Group, 2018).

Specifications for `forward`

Node name: `forward`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `forward`

```
<forward/>
```

18.6.4 General transport type: `energy`

Specifications for `energy`

Node name: `energy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `XYs2d`: [optional, when present, must appear one time] $P(E'|E)$ given as a 2d interpolation table.
- `regions2d`: [optional, when present, must appear one time] $P(E'|E)$ given in mul--tiple regions.
- `generalEvaporation`: [optional, when present, must appear one time] $P(E'|E)$ stored as a general evapouration spectrum.
- `discreteGamma`: [optional, when present, must appear one time] $P(E'|E) = \delta(E_\gamma)$.
- `primaryGamma`: [optional, when present, must appear one time] Energy distribution for a 'primary capture gamma'. Unlike discrete gammas, the primary gamma energy varies with incident energy E.
- `NBodyPhaseSpace`: [optional, when present, must appear one time] $P(E'|E)$ repre--sented using N-body phase space (must be in the centre of mass).
- `evaporation`: [optional, when present, must appear one time] $P(E'|E)$ stored as an evapouration spectrum.
- `weightedFunctionals`: [optional, when present, must appear one time] $P(E'|E)$ stored as the weighted sum of 2 or more other energy spectra.
- `simpleMaxwellianFission`: [optional, when present, must appear one time] Maxwellian representation for $P(E'|E)$, primarily for fission neutrons.
- `Watt`: [optional, when present, must appear one time] Watt spectrum representa--tion for $P(E'|E)$, primarily for fission neutrons.
- `MadlandNix`: [optional, when present, must appear one time] Madland-Nix param--eterisation for $P(E'|E)$, primarily for fission neutrons.

XML Example(s) of `energy`

```
<energy>
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d>
    <generalEvaporation>...</generalEvaporation>
    <discreteGamma>...</discreteGamma>
    <primaryGamma>...</primaryGamma>
    <NBodyPhaseSpace>...</NBodyPhaseSpace>
    <evaporation>...</evaporation>
    <weightedFunctionals>...</weightedFunctionals>
    <simpleMaxwellianFission>...</simpleMaxwellianFission>
    <Watt>...</Watt>
    <MadlandNix>...</MadlandNix></energy>
```

18.7 Energy distribution representations

Multiple options for storing the outgoing energy spectra are allowed within the `uncor--related` distribution form. An energy distribution gives the probability of finding an outgoing particle with energy E' given a reaction induced by a particle with energy E: $P(E'|E)$. The $P(E'|E)$ are normalised such that

$$\int_0^{E'_{max}} P(E'|E)dE' = 1 \tag{18.6}$$

where E'_{max} is the maximum possible secondary particle energy and its value depends on the incoming particle energy E and the analytic representation of $P(E'|E)$. The secondary particle energy E' is always expressed in the *laboratory system*.

The differential cross section is obtained from

$$\frac{d\sigma(E)}{dE'} = M(E)\ \sigma(E)\ P(E'|E) \tag{18.7}$$

where $\sigma(E)$ is the cross section $M(E)$ is the multiplicity for this reaction for the particle in question.

Energy distributions for fission are included in the discussion of fission transport data in section 18.9.

18.7.1 General transport type: `weightedFunctionals`

The energy distributions $P(E'|E)$ can be broken down into partial energy distributions $f_k(E \to E')$, where each of the partial distributions can be described by a different analytic representation;

$$P(E'|E) = \sum_{k=1}^{\mathrm{NK}} p_k(E) f_k(E \to E') \tag{18.8}$$

and at a particular incident energy E,

$$\sum_{k=1}^{\mathrm{NK}} p_k(E) = 1 \tag{18.9}$$

where $p_k(E)$ is the fractional probability that the distribution $f_k(E \to E')$ can be used at energy E. The partial energy distributions $f_k(E \to E')$ are represented by various analytical formulations. Each formulation is also an energy distribution.

Specifications for `weightedFunctionals`

Node name: `weightedFunctionals`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`weighted`:** [**required**, must appear at least one time] The partial energy distribu--tions $f_k(E \rightarrow E')$ are represented by various analytical formulations. Each formulation is also an energy distribution.

XML Example(s) of `weightedFunctionals`

```
<weightedFunctionals>
    <weighted>...</weighted></weightedFunctionals>
```

18.7.2 General transport type: `weighted`

The partial energy distributions $f_k(E \rightarrow E')$ are represented by various analytical formulations. Each formulation is itself an energy distribution.

Specifications for `weighted`

Node name: `weighted`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`:** [**required**, must appear one time] The weighting function $p_k(E)$ given as an XYs1d.
- **`XYs2d`:** [optional, when present, must appear one time] The energy distribution $f_k(E \rightarrow E')$ given as an `XYs2d`.
- **`evaporation`:** [optional, when present, must appear one time] The energy distri--bution $f_k(E \rightarrow E')$ given as an `evaporation`.
- **`generalEvaporation`:** [optional, when present, must appear one time] The energy distribution $f_k(E \rightarrow E')$ given as an `generalEvaporation`.
- **`simpleMaxwellianFission`:** [optional, when present, must appear one time] Maxwellian representation for $P(E'|E)$, primarily for fission neutrons.
- **`Watt`:** [optional, when present, must appear one time] Watt spectrum representa--tion for $P(E'|E)$, primarily for fission neutrons.
- **`MadlandNix`:** [optional, when present, must appear one time] Madland-Nix param--eterisation for $P(E'|E)$, primarily for fission neutrons.

XML Example(s) of `weighted`

```
<weighted>
    <XYs1d>...</XYs1d>
    <XYs2d>...</XYs2d>
    <evaporation>...</evaporation>
    <generalEvaporation>...</generalEvaporation>
    <simpleMaxwellianFission>...</simpleMaxwellianFission>
    <Watt>...</Watt>
    <MadlandNix>...</MadlandNix></weighted>
```

18.8 Energy distribution given by ENDF's evaporation models

The ENDF-6 format provides two similar methods to represent outgoing particle energy distributions as thermal evaporation spectra. These two methods differ slightly in flexibility and both are included in GNDS.

18.8.1 General transport type: `evaporation`

This element allows one to store a "fixed temperature" thermal spectrum and is equiv--alent to ENDF-6's MF=6, LF=9 format.

$$P(E'|E) = \frac{E'}{I} e^{-E'/\theta(E)}$$

where I is the normalisation constant:

$$I = \theta^2 \left[1 - e^{-(E-U)/\theta} \left(1 + \frac{E-U}{\theta} \right) \right]$$

Specifications for `evaporation`

Node name: `evaporation`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`U`:** [optional, when present, must appear one time] The U value.
- **`theta`:** [optional, when present, must appear one time] Data container that repre--sents a valid $\theta(E)$

XML Example(s) of `evaporation`

```
<evaporation>
    <U>...</U>
    <theta>...</theta></evaporation>
```

18.8.2 General transport type: `theta`

θ is an effective emission temperature.

Specifications for `theta`

Node name: `theta`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`:** [optional, when present, must appear one time] The $\theta(E)$ given as an interpolation table.
- **`regions1d`:** [optional, when present, must appear one time] The $\theta(E)$ given as a few interpolation tables.

XML Example(s) of `theta`

```
<theta>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></theta>
```

18.8.3 General transport type: `U`

U is a constant introduced to define the proper upper limit for the final particle energy such that $0 \leq E' \leq (E - U)$.

Specifications for `U`

Node name: `U`
Attributes: The list of additional allowed attributes is:

- **`unit`** [`XMLName`, optional] A valid energy unit for U
- **`value`** [`Float64`, optional] The value of U given as a Float64

Child nodes: This element has no child nodes

XML Example(s) of `U`

```
<U
        unit="..."
        value="..."></U>
```

18.8.4 General transport type: `generalEvaporation`

This defines the energy distribution for

$$P(E'|E) = g\left(E'/\theta(E)\right)$$

and is equivalent to ENDF-6's format's $f(E \rightarrow E')$ in MF=5, LF=5.

Specifications for `generalEvaporation`

Node name: `generalEvaporation`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`U`**: [optional, when present, must appear one time] The U value.
- **`g`**: [optional, when present, must appear one time] Container that represents a valid $g(x)$ where $x = E'/\theta(E)$
- **`theta`**: [optional, when present, must appear one time] Data container that repre--sents a valid $\theta(E)$

XML Example(s) of `generalEvaporation`

```
<generalEvaporation>
    <U>...</U>
    <g>...</g>
    <theta>...</theta></generalEvaporation>
```

18.8.5 General transport type: `g`

An arbitrary energy dependent weighting, tabulated as a function of x, and $x = E'/\theta(E)$.

Specifications for `g`

Node name: `g`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`XYs1d`**: [optional, when present, must appear one time] The $g(x)$ given as an inter--polation table.
- **`regions1d`**: [optional, when present, must appear one time] The $g(x)$ given as a few interpolation tables.

XML Example(s) of g

```
<g>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></g>
```

18.9 Energy distributions for fission neutrons

There are a variety of parameterisations of the fission neutron spectrum of varying de--grees of complexity. Many of these are no longer in general use (e.g. simple Maxwellian and Watt spectra), but are retained to support older evaluations.)

18.9.1 Fission transport type: `simpleMaxwellianFission`

This element is the format for ENDF-6's Simple Maxwellian Fission Spectrum (MF=5, LF=7).

$$f(E \to E') = \frac{\sqrt{E'}}{I} e^{-E'/\theta(E)}$$

where the parameters U and θ are given in sections 18.8.3 and 18.8.2 respectively and I is given by

$$I = \theta^{3/2} \left[\frac{\sqrt{\pi}}{2} \text{erf} \left(\sqrt{(E-U)/\theta} \right) - \sqrt{(E-U)/\theta}\; e^{-(E-U)/\theta} \right]$$

Specifications for `simpleMaxwellianFission`

Node name: `simpleMaxwellianFission`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- `U`: [optional, when present, must appear one time] A constant introduced to define the proper upper limit for the final particle energy such that $0 \leq E' \leq (E - U)$.
- `theta`: [optional, when present, must appear one time] The effective emission temperature.

XML Example(s) of `simpleMaxwellianFission`

```
<simpleMaxwellianFission>
    <U>...</U>
    <theta>...</theta></simpleMaxwellianFission>
```

18.9.2 Fission transport type: `Watt`

This subsection provides the specification for the `Watt` energy representation (equiva--lent to the ENDF-6 format's MF=5, LF=11 format). This is essentially a Lorentz-boosted Maxwellian.

$$P(E'|E) = \frac{e^{-E'/a}}{I} \sinh\left(\sqrt{bE'}\right)$$

where

I is the normalisation constant:

$$\begin{aligned} I &= \frac{1}{2}\sqrt{\frac{\pi a^3 b}{4}} \exp\left(\frac{ab}{4}\right)\left[\mathrm{erf}\left(\sqrt{\frac{E-U}{a}} - \sqrt{\frac{ab}{4}}\right) + \mathrm{erf}\left(\sqrt{\frac{E-U}{a}} + \sqrt{\frac{ab}{4}}\right)\right] \\ &- a \exp\left[-\left(\frac{E-U}{a}\right)\right] \sinh\sqrt{b(E-U)} \end{aligned}$$

a and b are tabulated energy-dependent parameters;

U is a constant introduced to define the proper upper limit for the final particle energy such that $0 \leq E' \leq (E-U)$.

Specifications for `Watt`

Node name: `Watt`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `U`: [**required**, must appear one time] A constant introduced to define the proper upper limit for the final particle energy such that $0 \leq E' \leq (E-U)$., discussed in section 18.8.3
- `a`: [**required**, must appear one time] Contains one `XYs1d` or `regions1d` functional data container that represents a valid $a(E)$.
- `b`: [**required**, must appear one time] Contains one `XYs1d` or `regions1d` functional data container that represents a valid $b(E)$.

XML Example(s) of `Watt`

```
<Watt>
    <U>...</U>
    <a>...</a>
    <b>...</b></Watt>
```

18.9.3 Fission transport type: `a`

a for Watt spectrum

Specifications for `a`

Node name: `a`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `a`

```
<a/>
```

18.9.4 Fission transport type: `b`

b for Watt spectrum

Specifications for `b`

Node name: `b`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `b`

```
<b/>
```

18.9.5 Fission transport type: `MadlandNix`

The `MadlandNix` element encodes the energy-dependent fission neutron spectrum using Madland and Nix's parameterisation. This format is equivalent to ENDF-6's MF=5, LF=12. The distribution is parametrically given by,

$$\begin{aligned} P(E'|E) &= \frac{1}{2}\left[g(E', E_F(L)) + g(E', E_F(H))\right] \\ g(E', E_F) &= \frac{1}{3\sqrt{(E_F\, T_M)}}\left[u_2^{3/2}\mathrm{E}_1(u_2) - u_1^{3/2}\mathrm{E}_1(u_1) + \gamma\left(\frac{3}{2}, u_2\right) - \gamma\left(\frac{3}{2}, u_1\right)\right] \end{aligned}$$

$$u_1 = \left(\sqrt{E'} - \sqrt{E_F}\right)^2 / T_M$$
$$u_2 = \left(\sqrt{E'} + \sqrt{E_F}\right)^2 / T_M$$

where:

$E_F(X)$	are constant, which represent the average kinetic energy per nucleon of the fission fragment; arguments L and H refer to the average light fragment (given by the parameter EFL in the file) and the average heavy fragment (given by the parameter EFH in the file), respectively.
T_M	parameter tabulated as a function of incident neutron energy
$\mathrm{E}_1(x)$	is the exponential integral
$\gamma(a, x)$	is the incomplete gamma function. The integral of this spectrum between zero and infinity is one.

Note the range of E' is such that $0 \leq E' \leq \infty$. Physically, energy conservation cuts off the integral at finite E'_{max} but the dependence of $g(E', E_F)$ on E' ensures that the contribution from large values of E' is negligible.

Specifications for `MadlandNix`

Node name: `MadlandNix`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`EFH`:** [**required**, must appear one time] The $E_F(H)$ value.
- **`EFL`:** [**required**, must appear one time] The $E_F(L)$ value.
- **`T_M`:** [**required**, must appear one time] Temperature like parameter tabulated as a function of incident neutron energy, given as an `XYs1d`. The y-axis unit must be an energy unit.

XML Example(s) of `MadlandNix`

```
<MadlandNix>
    <EFH>...</EFH>
    <EFL>...</EFL>
    <T_M>...</T_M></MadlandNix>
```

18.9.6 Fission transport type: `EFH`

The $E_F(H)$ value.

Specifications for `EFH`

Node name: `EFH`
Attributes: The list of additional allowed attributes is:
  `unit` [`XMLName`, **required**] An energy unit.
  `value` [`Float64`, **required**] The value of $E_F(H)$ itself.
Child nodes: This element has no child nodes

XML Example(s) of `EFH`

```
<EFH
        unit="..."
        value="..."></EFH>
```

18.9.7 Fission transport type: `EFL`

The $E_F(L)$ value.

Specifications for `EFL`

Node name: `EFL`
Attributes: The list of additional allowed attributes is:
  `unit` [`XMLName`, **required**] An energy unit.
  `value` [`Float64`, **required**] The value of $E_F(L)$ itself.
Child nodes: This element has no child nodes

XML Example(s) of `EFL`

```
<EFL
        unit="..."
        value="..."></EFL>
```

18.9.8 Fission transport type: `T_M`

Temperature-like parameter tabulated as a function of incident neutron energy, given as an `XYs1d`.

Specifications for `T_M`

Node name: `T_M`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`XYs1d`**: [**required**, must appear one time] The table of `T_M` values vs. incident energy.

XML Example(s) of `T_M`

```
<T_M>
    <XYs1d>...</XYs1d></T_M>
```

18.10 Energy distributions for discrete and primary gammas

This sections covers elements specific to photon production and transport in nuclear reactions. In particular the data needed to describe photons emitted from bound-bound and unbound-bound nuclear transitions.

Note that the descriptions of primary and secondary gammas in both MF=12 and MF=6 of the legacy ENDF-6 formats manual (Cross Section Evaluation Working Group, 2018) ignore the recoil of the emitting residual nucleus and relativistic corrections.

18.10.1 General transport type: `discreteGamma`

This format is to store the discrete energy of a photon from a bound level-bound level transition in which either the initial or final level is unspecified. The photon energy is simply E'_γ.

Specifications for `discreteGamma`

Node name: `discreteGamma`
Attributes: The list of additional allowed attributes is:
- **`domainMax`** [`Float64`, optional]
- **`domainMin`** [`Float64`, optional]
- **`value`** [`Float64`, optional] E_γ, the outgoing gammma energy (in eV)

Child nodes: The list of additional allowed child nodes is:
- **`axes`**: [optional, when present, must appear one time]

XML Example(s) of `discreteGamma`

```
<discreteGamma
        domainMax="..."
        domainMin="..."
        value="...">
    <axes>...</axes></discreteGamma>
```

18.10.2 General transport type: `primaryGamma`

This format is to store the discrete energy of a photon from an unbound level-bound level transition. The unbound level is above the target+projectile separation energy. The photon energy depends on the incident projectile energy E and an optional parameter E^0_γ. Ignoring the recoil of the residual nucleus,

$$E'_\gamma = E^0_\gamma + E\,\frac{M_\text{target}}{M_\text{target} + M_\text{projectile}}.$$

Including the recoil, one finds

$$E'_\gamma = E^0_\gamma + E\,\frac{M_\text{target}}{M_\text{target} + M_\text{projectile}}\left(1 - \frac{Q}{(M_\text{target} + M_\text{projectile})c^2}\right).$$

where $Q = M_\text{target} + M_\text{projectile} - M_\text{residual}$. The optional parameter E^0_γ is

$$E^0_\gamma = Q\left(1 - \frac{Q}{2(M_\text{target} + M_\text{projectile})c^2}\right).$$

Thus, if the final state of the residual nucleus is known, then M_residual is known, enabling both the recoil corrections and complete specification of E^0_γ. Furthermore, one has the additional information needed to continue the secondary cascade induced in the residual nucleus by the present primary gamma.

Specifications for `primaryGamma`

Node name: `primaryGamma`
Attributes: The list of additional allowed attributes is:
- **`domainMax`** [`Float64`, optional]
- **`domainMin`** [`Float64`, optional]
- **`value`** [`Float64`, optional] E^0_γ, the parameter needed to define the outgoing gam- -mma energy as a function of the incident particle energy (in eV). If not given, then a `finalState` must be specified.
- **`finalState`** [`XMLName`, optional] The pid of the final state for this primary gamma ray transition. This should be a valid `nuclide`'s `id`. If known, then modeling the coincidence gamma emission following this primary gamma can be done.

Child nodes: The list of additional allowed child nodes is:
- **`axes`:** [optional, when present, must appear one time]

XML Example(s) of `primaryGamma`

```
<primaryGamma value="7790329.0" domainMin="1e-05"
            domainMax="20000000.0" finalState="Cl36_e1">
      <axes>
        <axis index="2" label="energy_in" unit="eV"/>
        <axis index="1" label="energy_out" unit="eV"/>
        <axis index="0" label="P(energy_out|energy_in)"
            unit="1/eV"/></axes></primaryGamma>
```

18.11 Center-of-mass energy distributions given using the N-body phase space formulation

In the absence of detailed information, it is worthwhile to use N-body phase-space distributions for the particles emitted from neutron and charged-particle reactions. These distributions are isotropic in the center-of-mass frame. They conserve energy and momentum, and they provide reasonable kinematic limits for secondary energy and angle in the LAB system. This is meant for situations where the number of emitted particles is ≥ 3. This is equivalent to ENDF-6's MF=6, LAW=6.

In the center of mass (COM) system, the phase-space distribution for particle i is

$$P_i^{\text{com}}(\mu, E'|E) = C_n\sqrt{E'}(E_i^{\text{max}} - E')^{(3n/2)-4} \tag{18.10}$$

where

E_i^{max}	is the maximum possible center-of-mass energy for particle i,
μ, E'	are cosine of the scattering angle and outgoing particle energy in the CM system, and
C_n	are normalisation constants:

$$C_3 = \frac{4}{\pi(E_i^{\text{max}})^2} \tag{18.11}$$

$$C_4 = \frac{105}{32(E_i^{\text{max}})^{7/2}} \tag{18.12}$$

$$C_5 = \frac{256}{14\pi(E_i^{\text{max}})^5} \tag{18.13}$$

Note that in the COM frame, $E' < E_i^{\text{max}}$ in order that the probability distribution $P_i^{\text{com}}(\mu, E, E')$ remain real.

In the laboratory system, the distributions become

$$P_i^{\text{lab}}(\mu, E, E') = C_n\sqrt{E'}\left[E_i^{\text{max}} - \left(E^* + E' - 2\mu\sqrt{E^*E'}\right)\right]^{(3n/2)-4} \tag{18.14}$$

where μ and E' are in the laboratory system and E^* is given by:

$$E^* = E\frac{A^{\text{incident}}A^{\text{exit}}}{(\text{AWR} + A^{\text{incident}})^2} \tag{18.15}$$

and A^{incident} and A^{exit} are the mass ratios to the neutron mass of the incident and exit particles, respectively. In the laboratory frame in the general case, the range of both E' and μ is limited by the condition that the quantity in square brackets remains non-negative.

The value of E_i^{max} is a fraction of the energy available in COM,

$$E_i^{\text{max}} = \frac{M - m_i}{M}E_a \tag{18.16}$$

where M is the total mass of the n particles being treated by this law. Note that M may be less than the total mass of the products for reactions such as:

α + ^{9}Be $\rightarrow$ n + 3α
where the neutron can be treated as a two-body event and the alphas by a 3-body phase-space law.

The energy available in COM for one-step reactions is

$$E_a = \frac{m_T}{m_p + m_T}E + Q \tag{18.17}$$

where

m_T	is the target mass,
m_p	is the projectile mass,
E	is the energy in the LAB system, and
Q	is the reaction Q value (equivalent to the QI from File 3 in the corre--sponding ENDF-6 formatted file).

For two-step reactions such as the one discussed above, E_a is just the recoil energy from the first step.

18.11.1 General transport type: `NBodyPhaseSpace`

Outgoing particles using the n-body phase-space formalism

Specifications for `NBodyPhaseSpace`

Node name: `NBodyPhaseSpace`
Attributes: The list of additional allowed attributes is:

- **`numberOfProducts`** [`Integer32`, optional] The number of products being treated by this distribution.

Child nodes: The list of additional allowed child nodes is:

- **`mass`:** [optional, when present, must appear one time] The total mass of the prod--ucts being treated by this distribution.

XML Example(s) of `NBodyPhaseSpace`

```
<NBodyPhaseSpace
        numberOfProducts="...">
    <mass>...</mass></NBodyPhaseSpace>
```

18.12 Processed distributions for Monte Carlo transport

18.12.1 Processed type: `angularEnergyMC`

This distribution is processed for Monte Carlo transport. The outgoing particle distribu--tion $P(\mu, E'|E)$ is the projectile energy E stored as $P(\mu|E) \times P(E'|E, \mu)$. Each function1d must be a `xs_pdf_cdf1d` node.

Specifications for `angularEnergyMC`

Node name: `angularEnergyMC`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] Name of the style in the <styles> element associated with these data.
- **`productFrame`** [`frame`, **required**] The frame in which the product data are defined.

Child nodes: The list of additional allowed child nodes is:

- **`angular`:** [**required**, must appear one time] The outgoing particle μ distribution $P(\mu|E)$ give the projectile energy using a `XYs2d` or `regions2d` is a list of `xs_-pdf_cdf1d` for its function1d's.
- **`angularEnergy`:** [**required**, must appear one time] The outgoing particle energy distribution $P(E'|E, \mu)$ give the projectile energy and outgoing particle μ using a `XYs3d` or `regions3d` with a list of `XYs2d` for its function2d's. Each `XYs2d` is a list `xs_pdf_cdf1d` for its function1d's.

XML Example(s) of `angularEnergyMC`

```
<angularEnergyMC
        label="..."
        productFrame="...">
    <angular>...</angular>
    <angularEnergy>...</angularEnergy></angularEnergyMC>
```

18.12.2 Processed type: `energyAngularMC`

This distribution is processed for Monte Carlo transport. The outgoing particle dis--tribution $P(\mu, E'|E)$ given the projectile energy E stored as $P(E'|E) \times P(\mu|E, E')$. Each function1d must be a `xs_pdf_cdf1d` node.

Specifications for `energyAngularMC`

Node name: `energyAngularMC`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] Name of the style in the <styles> element associated with these data.
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- **`energy`:** [**required**, must appear one time] The outgoing particle E' distribution $P(E'|E)$ give the projectile energy using a `XYs2d` or `regions2d` with a list of `xs_pdf_cdf1d` for its function1d's.
- **`energyAngular`:** [**required**, must appear one time] The outgoing particle μ distri--bution $P(\mu|E, E')$ give the projectile energy and outgoing particle E' using a `XYs3d` or `regions3d` with a list of `XYs2d` for its function2d's. Each `XYs2d` as a list `xs_pdf_cdf1d` for its function1d's.

XML Example(s) of `energyAngularMC`

```
<energyAngularMC
        label="..."
        productFrame="...">
    <energy>...</energy>
    <energyAngular>...</energyAngular></energyAngularMC>
```

18.13 Processed distributions for deterministic transport

The `multiGroup3d` node represent the multi-grouped, Legendre expanded scattering matrix[26] $\mathrm{SM}_{g,g',l}$. This matrix is defined as,

$$\mathrm{SM}_{g,g',l} = \frac{\int_{E_g}^{E_{g+1}} dE\, m(E)\, \sigma(E)\, f(E)\ \int_{-1}^{1} d\mu\, P_l(\mu)\ \int_{E'_{g'}}^{E'_{g'+1}} dE'\ (E')^w P(\mu, E'|E)}{N_g} \tag{18.18}$$

where $f(E)$ is a weight function, w is the power for weighting E' and N_g is a multi-group norm. The projectile and product multi-group boundaries are defined by the points E_g and $E'_{g'}$, respectively.

18.13.1 Processed type: `multiGroup3d`

A multi-group transfer array.

Specifications for `multiGroup3d`

Node name: `multiGroup3d`
Abstract node: `distributionNode`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] A string identifier for the style and all data assoicated with it. The identifier must be unique among all the styles.
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- **`gridded3d`:** [**required**, must appear one time] Stores the multi-group transfer array with the first axis being the projectile energy, the second axis being the outgoing particle's energy and the thirds axis the Legendre order.

XML Example(s) of `multiGroup3d`

```
<multiGroup3d
        label="..."
        productFrame="...">
    <gridded3d>...</gridded3d></multiGroup3d>
```

[26]Here the historical name "scattering matrix" is used. However, the data are an array of dimension 3 (independent variables being l, g and g') rather than a 2-dimensional matrix. Thus, a more appropriate name would be a scattering array.

19. Resonance data

This section stores resonance parameters, from which cross sections, cross section probability tables and potentially angular distributions can be derived.

19.1 Reaction channel and general reaction information

The following give information about the reaction channels used in the resonance parameters (both resolved and unresolved):

- channel radii to be used for the calculation of the the penetrability P, the shift function S and the phase shift ϕ
- the reaction channels for which resonance parameters are given
- the reactions to which the channels contribute

They can appear in multiple places and are used for both resolved an unresolved resonance parameter formats.

19.1.1 Resonance type: `scatteringRadius`

The `scatteringRadius` node provides the scattering radius for potential scattering. This scattering radius is also to be used as the default channel radius for all channels in the resolved and unresolved resonance range when calculating the penetrability P, the shift function S and the phase shift ϕ. The scattering radius should be given in length units (the recommended unit is fm) and can be given as either a constant value or an energy dependent function of the same style. The use of an energy dependent function for the scattering radius is deprecated. For a given evaluation label there can only be one `XYs1d`, `regions1d` or `constant1d`

Specifications for `scatteringRadius`

Node name: `scatteringRadius`
Attributes: This element has no attributes

Child nodes: The list of additional allowed child nodes is (only one of the children marked with * for each unique style label is allowed. However, the style label can be omitted and empty, in which case only one of the marked children is allowed):

`constant1d`: [optional, when present, * one of marked children must appear one time] The scattering radius is a constant value.

`XYs1d`: [optional, when present, * one of marked children must appear one time] The scattering radius is an energy dependent function with a single interpo--lation range.

`regions1d`: [optional, when present, * one of marked children must appear one time] The scattering radius is an energy dependent function with multiple interpolation range.

XML Example(s) of `scatteringRadius`

```
<scatteringRadius>
    <constant1d>...</constant1d>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></scatteringRadius>
```

19.1.2 Resonance type: `hardSphereRadius`

The `hardSphereRadius` node provides the channel radius to be used in the calculation of the phase shift instead of the radius given in a `scatteringRadius` node. The `hardSpher--eRadius` is deprecated and should not be used in new evaluations. Evaluators wishing to modify the phase shift should consider using a background R-Matrix term instead.

Specifications for `hardSphereRadius`

Node name: `hardSphereRadius`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`constant1d`: [**required**, must appear one time] The hard-sphere radius.

XML Example(s) of `hardSphereRadius`

```
<hardSphereRadius>
    <constant1d>...</constant1d></hardSphereRadius>
```

19.1.3 Resonance type: `resonanceReactions`

The `resonanceReactions` node provides data on all reactions for which resonance data is provided. Each reaction that is used in the description of the resonance parameters must be described here.

Specifications for `resonanceReactions`

Node name: `resonanceReactions`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`resonanceReaction`**: [**required**, must appear at least one time] A single reaction for which resonances are specified.

XML Example(s) of `resonanceReactions`

```
<resonanceReactions>
    <resonanceReaction>...</resonanceReaction></resonanceReactions>
```

19.1.4 Resonance type: `resonanceReaction`

The `resonanceReaction` node gives information on a single reaction for which reso--nances will be given. It contains a link to the corresponding `reaction` node. It also defines some additional information for use when reconstructing cross sections. This includes override values for the channel radii to be used for the calculation of the penetrability P, the shift function S and the phase shift ϕ and whether or not the reaction is eliminated (as would be the case for neutron capture in the Reich-Moore approximation).

Specifications for `resonanceReaction`

Node name: `resonanceReaction`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**]
- **`ejectile`** [`XMLName`, **required**] A string identifying the ejectile of the reaction (e.g. n). This attribute is required because the sign on the channel spin depends on which outgoing particle is the ejectile, and which the residual.
- **`boundaryConditionValue`** [`Float64`, optional] A value of the boundary condition B in $L = S - B + iP$ to be used for every channel that points to this resonance reaction. This value overrides the boundary condition option and/or value given in the `RMatrix` node. This attribute cannot be used if the boundary

condition option is set to `Brune` or `EliminateShiftFunction` on the `RMatrix` node. The boundary condition value given here can be overriden in a `channel` node that points to this resonance reaction.

`eliminated` [`Boolean`, optional, default is "false"] A flag to indicate whether or not a reaction is eliminated. This is used for the Reich-Moore approximation for the neutron radiative capture reaction. At most one reaction can be treated as eliminated.

Child nodes: The list of additional allowed child nodes is:

`Q`: [optional, when present, must appear one time] An optional override for the Q value of the reaction given in the `reaction` node.

`scatteringRadius`: [optional, when present, must appear one time] An optional override for the scattering radius for this reaction.

`hardSphereRadius`: [optional, when present, must appear one time] An optional override for the channel radius used in the calculation of the phase shift ϕ for this reaction.

`link`: [**required**, must appear one time] A link to the corresponding `reaction` node.

XML Example(s) of `resonanceReaction`

```
<resonanceReaction
        label="..."
        ejectile="..."
        boundaryConditionValue="..."
        eliminated="...">
    <Q>...</Q>
    <scatteringRadius>...</scatteringRadius>
    <hardSphereRadius>...</hardSphereRadius>
    <link>...</link></resonanceReaction>
```

19.1.5 Resonance type: `channels`

The `channels` section provides data on all reaction channels for which resonance data is provided in the current spin group. If no resonance parameters are given (i.e. the `resonanceParameters` node contains an empty table), the contribution of the potential scattering is still present in the elastic channel. In this case, only the elastic channel must be specified in the `channels` node.

Specifications for `channels`

Node name: `channels`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`channel`: [**required**, must appear at least one time] A node used to identify the channel and the resonance reaction to which the channel contributes as well as the override values to be used for channel related data.

XML Example(s) of `channels`

```
<channels>
    <channel>...</channel></channels>
```

19.1.6 Resonance type: `channel`

The `channel` node identifies the reaction to which the channel contributes and can be used to provide channel dependendent override values for the `scatteringRadius` and `hardSphereRadius` nodes. The `channel` node can also be used to set a boundary condition value. This boundary condition value will then override any previously defined value (defined on the `RMatrix` or `resonanceReaction` node) and will even override the `boundaryCondition` option. For example: the `boundaryCondition` option can be set to `NegativeOrbitalMomentum` which means that the boundary condition value is set to be equal to minus l for each channel. Setting a `boundaryConditionValue` on a given channel overrides that option.

Specifications for `channel`

Node name: `channel`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] A unique string identifying the channel in the spin group.
- **`resonanceReaction`** [`bodyText`, **required**] The label of the resonance reaction (as defined in the `resonanceReaction` nodes) that the channel contributes to.
- **`L`** [`Integer32`, **required**] The value of the orbital momentum quantum number ℓ in units of hbar.
- **`channelSpin`** [`Fraction32`, **required**] The value of the channel spin s in units of hbar, given as a fraction, e.g. `0`, `1/2`, `1`, `3/2`, etc..
- **`boundaryConditionValue`** [`Float64`, optional] A value of the boundary condition B in $L = S - B + iP$ to be used for this channel. This value overrides the boundary condition option and/or value given in the `RMatrix` node or the boundary condition value in the `resonanceReaction` node. This attribute cannot be used if the boundary condition option is set to `Brune` or `EliminateShiftFunction` on the `RMatrix` node.
- **`columnIndex`** [`Integer32`, **required**] The index of the column of the table in the `resonanceParameters` that contains the resonance widths for this channel.

Child nodes: The list of additional allowed child nodes is:

- **`externalRMatrix`:** [optional, when present, must appear one time] An optional external R-matrix element that needs to be added to the Reich-Moore R matrix.
- **`scatteringRadius`:** [optional, when present, must appear one time] An optional override for the scattering radius for this channel in the current spin group.

hardSphereRadius: [optional, when present, must appear one time] An optional override for the channel radius used in the calculation of the phase shift ϕ for this channel in the current spin group.

XML Example(s) of `channel`

```
<channel
        label="..."
        resonanceReaction="..."
        L="..."
        channelSpin="..."
        boundaryConditionValue="..."
        columnIndex="...">
    <externalRMatrix>...</externalRMatrix>
    <scatteringRadius>...</scatteringRadius>
    <hardSphereRadius>...</hardSphereRadius></channel>
```

19.1.7 Resonance type: `externalRMatrix`

The statistical approximation for the resonances external to the observed R-matrix resonances using one of the following formalisms: `Froehner`, `SAMMY`. For the Froehner formalism, a `double` should be provided for each of the following terms, with the `label` attribute specified below:

averageRadiationWidth: Average Radiation Width. Recommended unit: ‘eV’, ‘MeV’, etc.
constantExternalR: Constant term for the external contribution to the R-matrix. Recommended unit: unitless
poleStrength: Pole strength as defined by Fröhner. Recommended unit: unitless
singularityEnergyBelow: Energy below the energy range of observed resonances at which the Froehner and SAMMY R-external equations produce infinite terms. Recommended unit: ‘eV’, ‘MeV’, etc.
singularityEnergyAbove: Energy above the energy range of observed resonances at which the Froehner and SAMMY R-external equations produce infinite terms. Recommended unit: ‘eV’, ‘MeV’, etc.

For the SAMMY formalism, a `double` should be provided for each of the following terms, with the `label` attribute specified below:

constantExternalR: Constant term for the external contribution to the R-matrix. Recommended unit: unitless
linearExternalR: Linear term for the external contribution to the R-matrix. Rec--ommended unit: ‘eV^{-1}’
quadraticExternalR: Quadratic term for the external contribution to the R-matrix. Recommended unit: ‘eV^{-2}’

constantLogarithmicCoefficient: Constant multiplicative coefficient on the log--arithmic term in the SAMMY equation. Recommended unit: unitless

linearLogarithmicCoefficient: Constant multiplicative coefficient on the loga--rithmic term in the SAMMY equation. Recommended unit: 'eV^{-1}'

singularityEnergyBelow: Energy below the energy range of observed resonances at which the Froehner and SAMMY R-external equations produce infinite terms. Recommended unit: 'eV', 'MeV', etc.

singularityEnergyAbove: Energy above the energy range of observed resonances at which the Froehner and SAMMY R-external equations produce infinite terms. Recommended unit: 'eV', 'MeV', etc.

Specifications for `externalRMatrix`

Node name: `externalRMatrix`
Attributes: The list of additional allowed attributes is:
type [`XMLName`, **required**, default is "Froehner"] The statistical approximation for the resonances external to the observed R-matrix resonances using one of the following formalisms: `Froehner`, `SAMMY`.
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):
double: [**required**, must appear at least one time] Each of the terms in the Froehner or SAMMY formalisms

XML Example(s) of `externalRMatrix`

```
<externalRMatrix type="Froehner">
    <double label="averageRadiationWidth" value="0.064" unit="eV" />
    <double>...</double>
</externalRMatrix>
```

Remarks:

F. Fröhner's derivation (Fröhner & Bouland, 2001) for average contributions from external resonances to the cross section observed in the internal region (the observed resolved resonance region) leads to the following formula for the external R-matrix:

$$R^{ext}_{cc'}(E) = \left[R^{\infty}_{c} + 2s_c \arctan\left(\frac{E-\overline{E}}{I/2}\right) + \frac{i\overline{\Gamma_\gamma} I/4}{I^2/4 - (E-\overline{E})^2}\right]\delta_{cc'}, \tag{19.1}$$

where R^{∞}_{c} is a constant term, s_c is the pole strength, $\overline{E}$ and I are, respectively, the midpoint and length of the energy interval given by the singularity energies above and below the internal region. $\overline{\Gamma_\gamma}$ is the average radiation width of the internal levels.

The SAMMY cross section evaluation program (Larson, 2008) uses the equation

$$R^{ext}_{cc'}(E) = \left[R_{con} + R_{lin}E + R_qE^2 - s_{lin}\left(E^{up} - E^{down}\right) - (s_{con} + s_{lin}E)\ln\left(\frac{E^{up} - E}{E - E^{down}}\right)\right], \tag{19.2}$$

to model the contributions of resonances external to the observed resolved resonance region to the internal region cross section. R_{con}, R_{lin}, R_q are the constant, linear, and quadratic terms. s_{con} and s_{lin} are the constant and linear coefficients on the logarith--mic term. E_{up} and E_{down} are the energies at which singularities are produced by the logarithmic term.

19.2 Resolved and unresolved resonance regions

19.2.1 Resonance type: `resonances`

The `resonances` node contains the scattering radius for the potential scattering, along with resolved and/or unresolved resonance parameter data (if they are given).

Specifications for `resonances`

Node name: `resonances`
Attributes: The list of additional allowed attributes is:
`href` [`XMLName`, optional]
Child nodes: The list of additional allowed child nodes is:
`scatteringRadius`: [**required**, must appear one time] The scattering radius to be used by default.
`hardSphereRadius`: [optional, when present, must appear one time] The channel radius to be used for calculating the phase shift.
`resolved`: [optional, when present, may appear any number of times] The resolved resonance parameters. The use of more than one `resolved` node is deprecated. It is allowed for historic reasons but should not be used for new evaluations.
`unresolved`: [optional, when present, may appear any number of times] The unre--solved resonance parameters.

XML Example(s) of `resonances`

```
<resonances
        href="...">
    <scatteringRadius>...</scatteringRadius>
    <hardSphereRadius>...</hardSphereRadius>
    <resolved>...</resolved>
    <unresolved>...</unresolved></resonances>
```

19.2.2 Resonance type: `resolved`

The `resolved` node provides a resolved resonance region over a given energy range. The energy domain of this resolved resonance region should not overlap with any of the other resonance regions defined in the `resonances` node. The `resolved` node defines the following formalisms for the resonance reconstruction:

> **`RMatrix`:** the R-matrix formalism using the single-level Breit-Wigner (SLBW), the multi-level Breit-Wigner (MLBW), the Reich-Moore approximation or the full R-matrix formalism.
> **`BreitWigner`:** the deprecated ENDF Breit-Wigner formalisms (LRF=1 and 2).

Specifications for `resolved`

Node name: `resolved`
Attributes: The list of additional allowed attributes is:
- **`domainMin`** [`Float64`, **required**] The lower energy limit of the resolved resonance range.
- **`domainMax`** [`Float64`, **required**] The upper energy limit of the resolved resonance range.
- **`domainUnit`** [`XMLName`, **required**] The energy unit used for the lower and upper energy limits.

Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):
- **`RMatrix`:** [optional, when present, * one of marked children must appear one time] The resonance parameters are given using the R-matrix formalism. More than one is only allowed if they are different in the style label
- **`BreitWigner`:** [optional, when present, * one of marked children must appear one time] The resonance parameters are given using the deprecated ENDF Breit-Wigner formalism. More than one is only allowed if they are different in the style label
- **`energyIntervals`:** [optional, when present, * one of marked children must appear one time] Resonance parameters are broken up into several sections, with interference between sections ignored. This option is deprecated but appears in older evaluations.

XML Example(s) of `resolved`

```
<resolved
        domainMin="..."
        domainMax="..."
        domainUnit="...">
    <RMatrix>...</RMatrix>
    <BreitWigner>...</BreitWigner>
    <energyIntervals>...</energyIntervals></resolved>
```

19.2.3 Resonance type: `unresolved`

The `unresolved` node provides an unresolved resonance region over a given energy range. The energy domain of this unresolved resonance region should not overlap with any of the other resonance regions defined in the `resonances` node.

Specifications for `unresolved`

Node name: `unresolved`
Attributes: The list of additional allowed attributes is:

- **`domainMin`** [`Float64`, **required**] The lower energy limit of the resolved resonance range.
- **`domainMax`** [`Float64`, **required**] The upper energy limit of the resolved resonance range.
- **`domainUnit`** [`XMLName`, **required**] The energy unit used for the lower and upper energy limits.

Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

- **`tabulatedWidths`:** [optional, when present, * one of marked children must appear one time] The unresolved resonance parameters. Only one instance per evaluation label is allowed.

XML Example(s) of `unresolved`

```
<unresolved
        domainMin="..."
        domainMax="..."
        domainUnit="...">
    <tabulatedWidths>...</tabulatedWidths></unresolved>
```

19.3 Resolved resonance parameters

19.3.1 Resonance type: `RMatrix`

The `RMatrix` node is used to specify resonance parameters using the R-matrix formal- -ism, which can be given in multiple approximations. The following approximations for the resonance reconstruction using the R-matrix formalism are currently defined:

> **`SingleLevelBreitWigner`** : the Single-level Breit-Wigner approximation
> **`MultiLevelBreitWigner`** : the Multi-level Breit-Wigner approximation
> **`ReichMoore`** : the Reich-Moore approximation
> **`FullRMatrix`** : full R-matrix formalism

Specifications for `RMatrix`

Node name: `RMatrix`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**]
- **`approximation`** [`XMLName`, **required**] The approximation to the R-matrix resonance formalism to be used for the resonance reconstruction (one of `SingleLevel-BreitWigner`, `MultiLevelBreitWigner`, `ReichMoore` or `FullRMatrix`).
- **`boundaryCondition`** [`boundaryCondition`, optional, default is "EliminateShiftFunction"] The boundary condition option to be used.
- **`boundaryConditionValue`** [`Float64`, optional] A global default value of the boundary condition B in $L = S - B + iP$ when the boundary condition option selected on the RMatrix node is `"Given"`. This value can be overridden in the `resonanceReaction` or `channel` nodes.
- **`calculateChannelRadius`** [`Boolean`, optional, default is "false"] An optional flag to indicate whether or not the channel radius should be calculated for the calculation of the penetrability P and shift function S.
- **`calculatePenetrability`** [`Boolean`, optional, default is "true"] A flag to indicate whether or not the penetrability should be calculated. If false, the penetrability is assumed to be 1.
- **`useForSelfShieldingOnly`** [`Boolean`, optional, default is "false"] Flag indicating that resonance parameters are only meant for use in computing self-shielding, not for calculating the cross section.
- **`supportsAngularReconstruction`** [`Boolean`, optional, default is "false"] A flag to indicate whether or not the resonance parameters may be used to compute angular distributions for the reactions.

Child nodes: The list of additional allowed child nodes is:

- **`PoPs`**: [optional, when present, must appear one time] Particle database to be used for the resonance reconstruction. This element only needs to be given if mass or spin values different from the particle database used in the parent `reactionSuite` are to be used in the calculations. The use of this element is deprecated.
- **`resonanceReactions`**: [**required**, must appear one time] The reactions for which resonance parameters will be provided.
- **`spinGroups`**: [**required**, must appear one time] The spin groups with resonance parameters.

XML Example(s) of `RMatrix`

```
<RMatrix
        label="..."
        approximation="..."
        boundaryCondition="..."
        boundaryConditionValue="..."
        calculateChannelRadius="..."
        calculatePenetrability="..."
        useForSelfShieldingOnly="..."
        supportsAngularReconstruction="...">
    <PoPs>...</PoPs>
    <resonanceReactions>...</resonanceReactions>
    <spinGroups>...</spinGroups></RMatrix>
```

19.3.2 Resonance type: `boundaryCondition`

Allowed values for the `boundaryCondition` attribute. The R-matrix formalism requires a choice in the boundary condition used in the calculation of the scattering matrix. This boundary condition is used in the term $S - B + iP$ for each channel in each spin group (in which S is the shift function, B is the boundary condition and P is the penetrability). The following options for the boundary condition are currently defined:

> **`EliminateShiftFunction`** : the boundary condition cancels out the shift factor ($S - B = 0$) so that the shift factor never has to be calculated.
> **`NegativeOrbitalMomentum`** : the boundary condition is set to be equal to the minus the orbital momentum L.
> **`Brune`** : the boundary condition follows the Brune parametrisation.
> **`Given`** : the boundary condition is given for each channel.

When the boundary condition option is `Brune`, the resonance energies and widths given will be the Brune parameters, i.e. not the regular energies and widths for the traditional R-matrix defined by Lane and Thomas but the transformed ones defined by the Barker transform. Physical observables such as cross sections can be calculated from the Brune parameters using the level matrix, or by converting the parameters back to the the traditional R-matrix. When the `Given` option is selected, boundary condition values must be given for all channels, either by providing a global default value on the `RMatrix` node or by providing a value for each individual resonance reaction or channel. The `boundaryConditionValue` can also be used to override a `NegativeOrbitalMomentum` boundary condition for specific resonance reactions, but cannot be used when the `Brune` or `EliminateShiftFunction` option is chosen.

Type name: `boundaryCondition`
Allowed values: [‘EliminateShiftFunction’, ‘NegativeOrbitalMomentum’, ‘Brune’, ‘Given’]

19.3.3 Resonance type: `spinGroups`

The `spinGroups` node provides data on all spin groups for which resonance data is provided.

Specifications for `spinGroups`

Node name: `spinGroups`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`spinGroup`:** [**required**, must appear at least one time] The resolved resonance data for a given value of the total angular momentum and associated parity J^{π} as well as the contributing channels.

XML Example(s) of `spinGroups`

```
<spinGroups>
    <spinGroup>...</spinGroup></spinGroups>
```

19.3.4 Resonance type: `spinGroup`

The `spinGroup` node contains the resolved resonance data for a given value of the total angular momentum and associated parity J^{π}.

Specifications for `spinGroup`

Node name: `spinGroup`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] A label string identifying the spin group.
- **`spin`** [`Fraction32`, **required**] The value of the total angular momentum J for the spin group in units of hbar, given as a fraction, e.g. `0`, `1/2`, `1`, `3/2`, etc.
- **`parity`** [`Integer32`, **required**, default is “1”] The parity for the spin group, either `1` or `-1`.

Child nodes: The list of additional allowed child nodes is:

- **`channels`:** [**required**, must appear one time] The channels that are available in this spin group.
- **`resonanceParameters`:** [**required**, must appear one time] The resonance parameter table for this spin group.

XML Example(s) of `spinGroup`

```
<spinGroup
        label="..."
        spin="..."
        parity="...">
    <channels>...</channels>
    <resonanceParameters>...</resonanceParameters></spinGroup>
```

19.3.5 Resonance type: `resonanceParameters`

The `resonanceParameters` node provides a table of resonance parameters for the current spin group. The number of columns of the table is equal to the number of channels that have at least one non zero width in the current spin group plus one column for the energy values (column index 0). The number of rows in the table is equal to the number of resonances that are defined for this spin group. The column indices used in this table must be given in the corresponding `channel` node in the `channels` node of the current spin group. The table may be empty to indicate that the spin group only contributes to potential scattering. The `BreitWigner` node also has a `resonanceParameters` but the structure of the table is different.

Specifications for `resonanceParameters`

Node name: `resonanceParameters`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
    `table`: [**required**, must appear one time] A table of resonance widths.

XML Example(s) of `resonanceParameters`

```
<resonanceParameters>
    <table>...</table></resonanceParameters>
```

19.3.6 Resonance type: `BreitWigner`

The `BreitWigner` node is used to specify the resonance parameters in the Single-Level or Multi-Level Breit-Wigner approximation. The use of this element is deprecated in favour of using an `RMatrix` node with the appropriate approximation selected.

Specifications for `BreitWigner`

Node name: `BreitWigner`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**]
- **`approximation`** [`XMLName`, **required**] The approximation to the R-matrix resonance formalism to be used for the resonance reconstruction (either `SingleLevel` or `MultiLevel`).
- **`calculateChannelRadius`** [`Boolean`, optional, default is “false”] An optional flag to indicate whether or not the channel radius should be calculated for the calculation of the penetrability P and shift function S.
- **`useForSelfShieldingOnly`** [`Boolean`, optional, default is “false”] An optional flag to indicate that resonance parameters are only intended for self-shielding, not for reconstructing a cross section.

Child nodes: The list of additional allowed child nodes is:

- **`PoPs`**: [optional, when present, must appear one time] Particle database to be used for the resonance reconstruction. This element only needs to be given if mass or spin values different from the particle database used in the parent `reactionSuite` are to be used in the calculations. The use of this element is deprecated.
- **`scatteringRadius`**: [optional, when present, must appear one time] An optional override for the scattering radius for the resolved resonance region.
- **`hardSphereRadius`**: [optional, when present, must appear one time] An optional override for the channel radius used in the calculation of the phase shift ϕ for the resolved resonance region.
- **`resonanceParameters`**: [optional, when present, must appear one time] The reso--nance parameter table.

XML Example(s) of `BreitWigner`

```
<BreitWigner
        label="..."
        approximation="..."
        calculateChannelRadius="..."
        useForSelfShieldingOnly="...">
    <PoPs>...</PoPs>
    <scatteringRadius>...</scatteringRadius>
    <hardSphereRadius>...</hardSphereRadius>
    <resonanceParameters>...</resonanceParameters></BreitWigner>
```

Additional information

The `resonanceParameters` node inside this `BreitWigner` node contains a table of res--onance parameters. While the name of the node is the same as the node inside a `spinGroup`, the structure of the actual table is different than what is indicated above.

The `table` node inside `BreitWigner` contains the following columns, uniquely defined by the name attribute:

> **`energy:`** The column containing the resonance energy. An additional unit attribute is required.
> **`L:`** The value of l (orbital angular momentum of the incoming neutron) for the resonance.
> **`J:`** The value of J (the spin, or total angular momentum) for the resonance.
> **`totalWidth:`** Resonance total width evaluated at the resonance energy in units of energy. An additional unit attribute is required.
> **`neutronWidth:`** Neutron width evaluated at the resonance energy in units of energy. An additional unit attribute is required.
> **`captureWidth:`** Radiation width in units of energy. An additional unit attribute is required.
> **`fissionWidth (optional)`** Fission width in units of energy. An additional unit attribute is required.

19.3.7 Resonance type: `energyIntervals`

Used to break the resolved resonance region into multiple independant sub-regions. This was done in some older evaluations (notably the ENDF-VII.1 n + ^{239}Pu evaluation) to reduce the computational load of summing over all resonances when reconstructing the cross section. This option is deprecated and should not be used for new evaluations.

Specifications for `energyIntervals`

Node name: `energyIntervals`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, **required**] Identifier label. Must be unique within the parent `resolved` node.

Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):
- **`energyInterval`:** [**required**, must appear at least two times] A single energy interval including domain limits and resonance parameterisation.

XML Example(s) of `energyIntervals`

```xml
<energyIntervals
        label="...">
    <energyInterval>...</energyInterval></energyIntervals>
```

19.3.8 Resonance type: `energyInterval`

A single energy interval, including domain limits and a resonance parameterisation.

Specifications for `energyInterval`

Node name: `energyInterval`
Attributes: The list of additional allowed attributes is:

- **`index`** [`Integer32`, **required**] Index of this `energyInterval` within the parent `ener-gyIntervals`.
- **`domainMin`** [`Float64`, **required**] The lower energy limit of the resolved resonance range.
- **`domainMax`** [`Float64`, **required**] The upper energy limit of the resolved resonance range.
- **`domainUnit`** [`XMLName`, **required**] The energy unit used for the lower and upper energy limits.

Child nodes: The list of additional allowed child nodes is:

- **`RMatrix`**: [optional, when present, * one of marked children must appear one time] The resonance parameters in this interval are given using the R-matrix formalism.
- **`BreitWigner`**: [optional, when present, * one of marked children must appear one time] The resonance parameters in this interval are given using the deprecated ENDF Breit-Wigner formalism.

XML Example(s) of `energyInterval`

```
<energyInterval
        index="..."
        domainMin="..."
        domainMax="..."
        domainUnit="...">
    <RMatrix>...</RMatrix>
    <BreitWigner>...</BreitWigner></energyInterval>
```

19.4 Unresolved resonance parameters

19.4.1 Resonance type: `tabulatedWidths`

The `tabulatedWidths` node is used to describe average parameters in the unresolved range. Currently parameters can only be given in the single-level Breit-Wigner approxi-mation.

Specifications for `tabulatedWidths`

Node name: `tabulatedWidths`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**]
- **`approximation`** [`XMLName`, **required**] The approximation to the R-matrix resonance formalism for resonance reconstruction (currently only `"SingleLevelBre--itWigner"` is supported by ENDF-6 and hence GNDS).
- **`useForSelfShieldingOnly`** [`Boolean`, optional, default is “false”] An optional flag to indicate whether or not the resonance parameters are provided for self-shielding or scattering radius purposes only.

Child nodes: The list of additional allowed child nodes is:

- **`PoPs`**: [optional, when present, must appear one time] Particle database to be used for the resonance reconstruction. This element only needs to be given if mass or spin values different from the particle database used in the parent `reactionSuite` are to be used in the calculations. The use of this element is deprecated.
- **`scatteringRadius`**: [optional, when present, must appear one time] An optional override for the scattering radius for the unresolved resonance region.
- **`hardSphereRadius`**: [optional, when present, must appear one time] An optional override for the channel radius used in the calculation of the phase shift ϕ for the unresolved resonance region.
- **`resonanceReactions`**: [**required**, must appear one time] The reactions for which resonance parameters will be provided.
- **`Ls`**: [**required**, must appear one time] The average unresolved resonance parame--ters for all values of the orbital angular momentum ℓ.

XML Example(s) of `tabulatedWidths`

```
<tabulatedWidths
        label="..."
        approximation="..."
        useForSelfShieldingOnly="...">
    <PoPs>...</PoPs>
    <scatteringRadius>...</scatteringRadius>
    <hardSphereRadius>...</hardSphereRadius>
    <resonanceReactions>...</resonanceReactions>
    <Ls>...</Ls></tabulatedWidths>
```

19.4.2 Resonance type: `Ls`

The `Ls` node is gives the average unresolved resonance parameters for the values of the orbital angular momentum ℓ.

Specifications for `Ls`

Node name: `Ls`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `L`: [**required**, must appear at least one time] The average unresolved resonance parameters for a given value of the orbital angular momentum ℓ.

XML Example(s) of `Ls`

```
<Ls>
    <L>...</L></Ls>
```

19.4.3 Resonance type: `L`

The `L` node is gives the average unresolved resonance for a given value of the orbital angular momentum ℓ. It contains data for all associated values of the total angular momentum J

Specifications for `L`

Node name: `L`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**]
  `value` [`Integer32`, **required**] The value of the orbital angular momentum ℓ in units of hbar.
Child nodes: The list of additional allowed child nodes is:
  `Js`: [**required**, must appear one time] The average unresolved resonance parame--ters for all values of the total angular momentum J allowed by the current orbital angular momentum ℓ.

XML Example(s) of `L`

```
<L
        label="..."
        value="...">
    <Js>...</Js></L>
```

19.4.4 Resonance type: `Js`

The `Js` node gives the average unresolved resonance parameters for all values of the total angular momentum J allowed by the current orbital angular momentum ℓ.

Specifications for `Js`

Node name: `Js`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`J`: [**required**, must appear at least one time] The average unresolved resonance parameters for a given value of the total angular momentum J allowed by the current orbital angular momentum ℓ.

XML Example(s) of `Js`

```
<Js>
    <J>...</J></Js>
```

19.4.5 Resonance type: `J`

The `J` node gives the unresolved resonance data for the current orbital angular momen--tum ℓ and the current total angular momentum J. It contains level spacing information and resonance widths.

Specifications for `J`

Node name: `J`
Attributes: The list of additional allowed attributes is:

`label` [`XMLName`, **required**]
`value` [`Fraction32`, **required**] The value of the total angular momentum J for the spin group in units of hbar, given as a fraction, e.g. `0`, `1/2`, `1`, `3/2`, etc.

Child nodes: The list of additional allowed child nodes is:

`levelSpacing`: [**required**, must appear one time] The average level spacing for the current orbital angular momentum ℓ and the current total angular momen--tum J.
`widths`: [**required**, must appear one time] The average resonance parameters for the current orbital angular momentum ℓ and the current total angular momentum J.

XML Example(s) of `J`

```
<J
        label="..."
        value="...">
    <levelSpacing>...</levelSpacing>
    <widths>...</widths></J>
```

19.4.6 Resonance type: `levelSpacing`

The `levelSpacing` node gives the average level spacing D for the current orbital angular momentum ℓ and the current total angular momentum J.

Specifications for `levelSpacing`

Node name: `levelSpacing`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`constant1d`:** [optional, when present, must appear one time] The level spacing is a constant value.
- **`XYs1d`:** [optional, when present, must appear one time] The level spacing is an energy dependent function with a single interpolation range.
- **`regions1d`:** [optional, when present, must appear one time] The level spacing is an energy dependent function with multiple interpolation range.

XML Example(s) of `levelSpacing`

```
<levelSpacing>
    <constant1d>...</constant1d>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></levelSpacing>
```

19.4.7 Resonance type: `widths`

The `widths` node gives the average resonance widths for each channel listed in the `resonanceReactions` node for the current orbital angular momentum ℓ and the current total angular momentum J. If the single Level Breit Wigner aproximation is used, an optional competitive width can also be given. In this case, the associated resonance reaction should have its label set to `competitive`.

Specifications for `widths`

Node name: `widths`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`width`:** [**required**, must appear at least one time] The average resonance width for a given channel.

XML Example(s) of `widths`

```
<widths>
    <width>...</width></widths>
```

19.4.8 Resonance type: `width`

The `width` node gives the average resonance width for a given channel.

Specifications for `width`

Node name: `width`
Attributes: The list of additional allowed attributes is:

- **`resonanceReaction`** [`bodyText`, **required**] The label of the resonance reaction (as defined in the `resonanceReaction` nodes) that the channel contributes to.
- **`degreesOfFreedom`** [`Float64`, **required**] The number of degrees of freedom to be used to sample the average width.

Child nodes: The list of additional allowed child nodes is:

- **`constant1d`:** [optional, when present, must appear one time] The average width is a constant value.
- **`XYs1d`:** [optional, when present, must appear one time] The average width is an energy dependent function with a single interpolation range.
- **`regions1d`:** [optional, when present, must appear one time] The average width is an energy dependent function with multiple interpolation range.

XML Example(s) of `width`

```
<width
        resonanceReaction="..."
        degreesOfFreedom="...">
    <constant1d>...</constant1d>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></width>
```

20. Thermal neutron scattering

When an incident particle energy is low enough, its de Broglie wavelength grows so large that the particle can no longer resolve individual scattering centers. For neutron projectiles, this regime is known as the thermal neutron scattering regime. In this regime, the elastic or inelastic neutron scattering cross sections can be written as an incoherent sum of contributions from individual scatterers:

$$\frac{d^2\sigma(E \to E', \mu, T)}{dE'd\Omega} = \sum_{i=0}^{NS-1} \frac{d^2\sigma_i(E \to E', \mu, T)}{dE'd\Omega} \tag{20.1}$$

where the sum over i loops over the NS scatterers. One may presume that interfer--ence between scattering on different scatterers is possible but it apparently is not a significant contribution to the overall cross-section and is ignored in practice. In this section the various approximations and ways of representing the elastic and inelastic scattering cross sections of neutrons off the scattering centres are explained.

As a scattering neutron cannot resolve individual scattering centers, it sees large collections of molecules. For this reason, all energies are given in lab frame[27]

There are a variety of approximations in use to describe thermal neutron scattering data. The legacy ENDF-6 format as well as this format define the following (note indentation denotes further levels of approximation):

- Elastic scattering ($\omega \to 0$)
  - Coherent elastic scattering: §20.1.
  - Incoherent elastic scattering ($\mathcal{S}_{\text{dis}} \to 0$): §20.2.
- Inelastic scattering (any ω is allowed, so confusingly elastic can be a limit of inelastic scattering): §20.3.
  - Incoherent inelastic scattering($\mathcal{S}_{\text{dis}} \to 0$): §20.3, with `incoherentApproxima--tion="true"`.
    - Gaussian approximation (the “LEAPR” approximation): §20.4.
      - Short collision time approximation (phonon expansion): §20.5.

In GNDS 1.9, thermal neutron scattering law (TNSL) data were stored in a separate

[27]In a way this is obvious as one scatters off not just one molecule, but many many molecules. Their collective mass dwarfs the mass of the neutron so the mass of the target is essentially infinite. This means all masses *must* be given in the lab frame as no other frame is mathematically well defined.

type of document, under the root node `thermalScattering`. However, fundamentally these data are still descriptions of different types of reactions between a neutron and a target material. These data have therefore been merged into a `reactionSuite`. The ther--mal neutron scattering law parameters describe the double-differential cross section $d\sigma(E \to E', \mu, T)/dE'd\Omega$ and can be stored inside the `doubleDifferentialCrossSection` (see Section 16.1.2) for a `reaction` element. The `crossSection` and `distribution` can both be derived from the double-differential scattering law, so they simply link to that section. An explanation of this arrangement is provided in Section 20.8.

The TNSL format also allows one to support storing all thermal neutron law scat--tering contributions for a molecule in a single file. In the legacy ENDF-6 format, if more than one type of scattering atom plays a significant role in neutron scattering the evaluator must divide the evaluation into multiple files, e.g. 'tsl-HinH2O' and 'tsl-OinH2O'. While one must continue to support these legacy evaluations, GNDS should support generating a single 'H2O' evaluation with all information required to describe all scattering atoms and possible correlations between them.

For the discussions below, common parameters are defined:

T	Target material temperature
E	Incident (neutron) energy
E'	Secondary (neutron) energy
k_B	Boltzmann's constant
μ	Cosine of the scattering (neutron) angle (in the lab system)

20.1 Coherent elastic scattering

20.1.1 Thermal scattering law type: `thermalNeutronScatteringLaw_coherentElastic`

Neutrons can elastically scatter coherently off of regular substances such as crystals, including powdered crystalline material. The differential cross section for such scat--tering off a single scatterer can be written (Cross Section Evaluation Working Group, 2018)

$$\frac{d^2\sigma}{dE'\,d\Omega}(E \to E', \mu, T) = \frac{1}{2\pi E} \sum_{j=1}^{E_j(T)<E} s_j(T)\,\delta(\mu - \mu_j(T))\,\delta(E - E') \tag{20.2}$$

where:

$$\mu_j(T) = 1 - \frac{2E_j(T)}{E} \tag{20.3}$$

The sum in (20.2) extends over all Bragg edges with energy less than E.

In other sources (e.g. (Cacuci, 2010)), the structure factors $s_j(T)$ for the i^{th} scatterer are written as

$$s_j(T) = 2\pi\sigma_{\text{coh},i} f_j(T) \exp(-4WE_j(T)). \tag{20.4}$$

Here $f_j(T)$ is material dependent and plays the role of the crystallographic structure factors and W is the Debye Waller factor.

Two types of data containers can be used to store the energy- and temperature-dependent coherent elastic scattering factor:

- Using an ‘S_table’ that stores the cumulative sum of the structure factor
$$S(E,T) = \sum_{i=1}^{E_i<E} s_i(T)$$
$S(E,T)$ uses histogram interpolation along incident energy E, i.e. the cumulative structure factor is flat until the next Bragg edge is encountered. This is the same method used to store the coherent elastic scattering factor in ENDF-6.
- Storing a list of Bragg edges which include the temperature-dependent structure factor $s_j(T)$ and Bragg energy $E_j(T)$ for each Bragg edge.

Parameterised coherent and incoherent elastic scattering were added to the ENDF format in the early 1990’s.

Specifications for `thermalNeutronScatteringLaw_coherentElastic`

Node name: `thermalNeutronScatteringLaw_coherentElastic`
Attributes: The list of additional allowed attributes is:
- `label` [`XMLName`, **required**] Style label for this element.
- `pid` [`XMLName`, **required**, default is “n”] The id of the scattered particle (should always be ‘n’).
- `productFrame` [`frame`, **required**, default is “lab”] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:
- `S_table`: [optional, when present, * one of marked children must appear one time] Cumulative structure factor as a function of incident energy and temperature.
- `BraggEdges`: [optional, when present, * one of marked children must appear one time] Stores a list of Bragg edges along with energies and structure factors.

XML Example(s) of `thermalNeutronScatteringLaw_coherentElastic`

```
<thermalNeutronScatteringLaw_coherentElastic
        label="..."
        pid="..."
        productFrame="...">
    <S_table>...</S_table>
    <BraggEdges>...</BraggEdges></thermalNeutronScatteringLaw_coherentElastic>
```

20.1.2 Thermal scattering law type: `S_table`

Stores the cumulative structure factor as a function of incident energy and temperature.

Specifications for `S_table`

Node name: `S_table`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
`gridded2d`: [**required**, must appear one time] Cumulative structure factors,

$$S(E,T) = \sum_{i=1}^{E_i<E} s_i(T)$$

The cumulative structure factor uses histogram interpolation along energy, with steps at each Bragg edge.

XML Example(s) of `S_table`

```
<S_table>
    <gridded2d>...</gridded2d></S_table>
```

20.1.3 Thermal scattering law type: `BraggEdges`

This is the container for descriptions of the Bragg edges.

Specifications for `BraggEdges`

Node name: `BraggEdges`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
`BraggEdge`: [**required**, must appear at least one time] One Bragg edge's container. See section 20.1.4.

XML Example(s) of `BraggEdges`

```
<BraggEdges>
    <BraggEdge>...</BraggEdge></BraggEdges>
```

20.1.4 Thermal scattering law type: `BraggEdge`

Each Bragg edge has a (temperature-dependent) energy and intensity. The evaluator must define a scheme for interpolating both the energy and structure factor in tem--perature. Note that ENDF-6 does not provide a simple way to denote the temperature dependence of Bragg energies, leaving them to processing codes to determine.

Specifications for `BraggEdge`

Node name: `BraggEdge`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**]
Child nodes: The list of additional allowed child nodes is:
  `BraggEnergy`: [**required**, must appear one time] An `XYs1d` specifying the tempera--ture dependence of the Bragg edge energy $E_j(T)$ in Eqs. (20.2) and (20.3).
  `structureFactor`: [**required**, must appear one time] An `XYs1d` specifying the struc--ture factor for this Bragg edge, $s_j(T)$, for use in Eq. (20.2).

XML Example(s) of `BraggEdge`

```
<BraggEdge
        label="...">
    <BraggEnergy>...</BraggEnergy>
    <structureFactor>...</structureFactor></BraggEdge>
```

20.1.5 Thermal scattering law type: `BraggEnergy`

An `XYs1d` specifying the temperature dependence of the Bragg edge energy $E_j(T)$ for use in Eqs. (20.2) and (20.3)

Specifications for `BraggEnergy`

Node name: `BraggEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `XYs1d`: [**required**, must appear one time] The temperature dependence of this Bragg edge's energy $E_j(T)$.

XML Example(s) of `BraggEnergy`

```
<BraggEnergy>
    <XYs1d>...</XYs1d></BraggEnergy>
```

20.1.6 Thermal scattering law type: `structureFactor`

An `XYs1d` specifying the structure factor for this Bragg edge, $s_j(T)$, for use in Eq. (20.2).

Specifications for `structureFactor`

Node name: `structureFactor`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `XYs1d`: [**required**, must appear one time] The structure factors for this Bragg edge, $s_j(T) = S(E_j, T)$.

XML Example(s) of `structureFactor`

```
<structureFactor>
    <XYs1d>...</XYs1d></structureFactor>
```

20.2 Incoherent elastic scattering

20.2.1 Thermal scattering law type: `thermalNeutronScatteringLaw_incoherentElastic`

Elastic scattering can be treated in the incoherent approximation for partially ordered systems such as ZrH_x and polyethylene. The differential cross section is given by:

$$\frac{d^2\sigma}{dE'\,d\Omega}(E \to E', \mu, T) = \frac{\sigma_{\text{bound}}}{4\pi}\, e^{-2EW'(T)(1-\mu)}\ \ \delta(E - E') \tag{20.5}$$

where:

σ_{bound}	is the characteristic bound cross section (barns)
W'	is the DebyeWaller integral divided by the atomic mass (eV^{-1})

and all the other symbols have their previous meanings. The integrated cross section is easily obtained:

$$\sigma(E) = \frac{\sigma_{\text{bound}}}{2}\left(\frac{1 - e^{-4EW'}}{2EW'}\right) \tag{20.6}$$

Note that the limit of σ for small E is σ_{bound}.

This formalism can be used for energies up to 5 eV.

For some moderator materials containing more than one kind of atom, the incoherent elastic cross section is computed as the sum of contributions from two different materials. As an example in the ENDF-6 format, H in ZrH$_x$ is given in MAT 0007, and Zr in ZrH$_x$ is given in MAT 0058.

Specifications for `thermalNeutronScatteringLaw_incoherentElastic`

Node name: `thermalNeutronScatteringLaw_incoherentElastic`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, **required**] Style label for this element.
- **`pid`** [`XMLName`, **required**] The id of the scattered particle (should always be ‘n’).
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:
- **`boundAtomCrossSection`:** [**required**, must appear one time] The $\sigma_{\text{bound},i}$ cross section for a single bound atom (of the principal scattering type).
- **`DebyeWallerIntegral`:** [**required**, must appear one time] The Debye-Waller integral, divided by the atomic mass.

XML Example(s) of `thermalNeutronScatteringLaw_incoherentElastic`

```
<thermalNeutronScatteringLaw_incoherentElastic
        label="..."
        pid="..."
        productFrame="...">
    <boundAtomCrossSection>...</boundAtomCrossSection>
    <DebyeWallerIntegral>...</DebyeWallerIntegral>
</thermalNeutronScatteringLaw_incoherentElastic>
```

20.2.2 Thermal scattering law type: `DebyeWallerIntegral`

The Debye-Waller integral, divided by the atomic mass, in units of inverse energy, $W'(T)$

Specifications for `DebyeWallerIntegral`

Node name: `DebyeWallerIntegral`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`:** [**required**, must appear one time] Data container storing the temperature dependence of the Debye-Waller integral. Axes have units of temperature (e.g. ‘K’) and inverse energy (e.g. ‘1/eV’).

XML Example(s) of `DebyeWallerIntegral`

```
<DebyeWallerIntegral>
    <XYs1d>...</XYs1d></DebyeWallerIntegral>
```

20.3 Inelastic scattering

20.3.1 Thermal scattering law type: `thermalNeutronScatteringLaw_incoherentInelastic`

For inelastic scattering, the cross section can be computed by summing together contributions from each type of atom in the molecule. For the ith scattering atom, the contribution to the total cross section is given by

$$\frac{d^2\sigma_i(E \to E', \mu, T)}{dE'd\Omega} = \frac{M_i}{4\pi k_B T}\sqrt{\frac{E'}{E}}\left[\sigma_{\text{bound},i}\mathcal{S}_{\text{self},i}(\alpha,\beta,T) + \sigma_{\text{coh},i}\mathcal{S}_{\text{dis},i}(\alpha,\beta,T)\right] \tag{20.7}$$

Here one has

$\mathcal{S}_{\text{self},i}(\alpha,\beta,T)$	Is the self term of the scattering kernel of the i^{th} type scatterer,
$\mathcal{S}_{\text{dis},i}(\alpha,\beta,T)$	Is the distinct term of the scattering kernel of the i^{th} type scatterer
M_i	Number of atoms of the i^{th} type in the molecule or unit cell
β	Energy transfer, $\beta = (E' - E)/k_B T$
α	Momentum transfer, $\alpha = \left[E' + E - 2\mu\sqrt{EE'}\right] / (m_0 k_B T/m_n)$
m_0	the mass of the principal scattering atom in the molecule
m_n	the mass of the projectile, presumed to be a neutron (as this is thermal *neutron* scattering data)
$\sigma_{\text{coh},i}$	Coherent atom scattering cross section of the i^{th} type scatterer
$\sigma_{\text{inc},i}$	Incoherent atom scattering cross section of the i^{th} type scatterer

$\sigma_{\text{bound},i}$ Bound atom scattering cross section of the i^{th} type scatterer in the molecule in question, note

$$\sigma_{\text{bound},i} = \sigma_{\text{coh},i} + \sigma_{\text{inc},i} \tag{20.8}$$

$\sigma_{\text{free},i}$ Free scattering cross section of the i^{th} type scatterer. In principal this should match the zero energy intercept of the equivalent fast region cross section. In practice it won't match because the fast region evaluation neglects the equation of state of the atom in the molecule in question so is usually treated as a free-gas equation of state. Note, the bound and free cross sections are related through

$$\sigma_{\text{bound},i} = \sigma_{\text{free},i} \left(\frac{m_i + m_n}{m_i} \right)^2 \tag{20.9}$$

Note that GNDS extends the approximations allowed as in the legacy ENDF-6 format, only the incoherent approximation is used.

Specifications for `thermalNeutronScatteringLaw_incoherentInelastic`

Node name: `thermalNeutronScatteringLaw_incoherentInelastic`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] Style label for this element.
- **`pid`** [`XMLName`, **required**, default is "n"] The id of the scattered particle (should always be 'n').
- **`productFrame`** [`frame`, **required**, default is "lab"] The frame that the product data are defined in.
- **`calculatedAtThermal`** [`Boolean`, optional, default is "false"] Indicates that α and β were calculated using kT=0.0253 eV for ALL temperatures. If true, α and β must be multiplied by a temperature conversion factor before looking up values in the $S(\alpha, \beta)$ array. This option is equivalent to the 'LAT=1' option in ENDF-6.
- **`incoherentApproximation`** [`Boolean`, optional, default is "true"] The incoherent ap--proximation, where the "distinct" part of the scattering law is neglected (i.e. setting $\mathcal{S}_{\text{dis},i}(\alpha, \beta, T) = 0$ and $\sigma_{\text{inc},i} = 0$).
- **`primaryScatterer`** [`XMLName`, **required**] the particle id of the primary scatterer. This value must match the `pid` of one of the `scatteringAtom` elements contained in this element. The primary scatterer's mass is used to define α as in ENDF-6.

Child nodes: The list of additional allowed child nodes is:

- **`scatteringAtoms`:** [**required**, must appear one time] The container element for all thermal scattering law scatterers

XML Example(s) of `thermalNeutronScatteringLaw_incoherentInelastic`

```
<thermalNeutronScatteringLaw_incoherentInelastic
        label="..."
        pid="..."
        productFrame="..."
        calculatedAtThermal="..."
        incoherentApproximation="..."
        primaryScatterer="...">
    <scatteringAtoms>...</scatteringAtoms>
</thermalNeutronScatteringLaw_incoherentInelastic>
```

Note: One can easily recover the ENDF-6 presecription for the scattering kernel using the incoherent approximation (`incoherentApproximation=true`) so $\mathcal{S}_{\text{dis},i}(\alpha,\beta,T)=0$ and $\sigma_{\text{inc},i}=0$. With this,

$$S_i(\alpha,\beta,T)=e^{\beta/2}\mathcal{S}_{\text{self},i}(\alpha,\beta,T) \tag{20.10}$$

so Eq. (20.7) here reduced to Eq. (7.6) in the ENDF-6 format manual. Note that even this factor of $e^{\beta/2}$ may be absorbed into the scattering kernel if one is using the `symmetric` flag when specifying the `selfScatteringKernel` below.

20.3.2 Thermal scattering law type: `scatteringAtoms`

To collect the data for all the scatterers together, one defines the following container element for all thermal scattering law scatterers.

Specifications for `scatteringAtoms`

Node name: `scatteringAtoms`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`scatteringAtom`: [**required**, must appear at least one time] Each scatterer has its own contribution to the total thermal scattering law cross section, so all the information needed to compute this contribution is contained in this child element.

XML Example(s) of `scatteringAtoms`

```
<scatteringAtoms>
    <scatteringAtom>...</scatteringAtom></scatteringAtoms>
```

20.3.3 Thermal scattering law type: `scatteringAtom`

Each scatterer has its own contribution to the total thermal scattering law cross section. In the legacy ENDF-6 format, each scatterer is stored in its own file, leading to possible bookkeeping errors. For example, the thermal scatter law data for H_2O is broken into two files, `tsl-HinH2O.endf` and `tsl-OinH2O.endf`. In the `tsl-HinH2O.endf` file, both the Hydrogen and Oxygen scatterers are listed, but only the data for Hydrogen is provided.

In this format specification two possibilities are allowed:

- The legacy, multiple file, arrangement where the thermal scattering law data is split across several files
- A unified file where all the scatterers for a molecule are collected in one file,

Either way one must provide enough data to support legacy data processing where each scattering center's data is separated so that it may be "pasted" on to the equivalent high energy evaluation. The free atom cross section is also provided in order to facilitate this "pasting".

Specifications for `scatteringAtom`

Node name: `scatteringAtom`
Attributes: The list of additional allowed attributes is:
- **`pid`** [`XMLName`, **required**] Particle id for this scatterer. Note that the pid will often be an element (i.e. 'Zr') rather than a nuclide.
- **`primaryScatterer`** [`Boolean`, **required**, default is "false"] Denotes that the current scattering atom is the primary one.
- **`numberPerMolecule`** [`Integer32`, **required**] The number of this scattering atom per 'molecule' of scatterers.

Child nodes: The list of additional allowed child nodes is:
- **`mass`:** [**required**, must appear one time] Mass of this scattering atom. Note that unlike ENDF-6, the mass is for a single atom (not multiplied by the number of atoms/molecule).
- **`e_critical`:** [optional, when present, must appear one time] Energy (E/k_BT) above which the static model of elastic scattering is adequate. (total scattering properties may be obtained from fast region elastic and summed inelastic for the appropriate materials). This is equivalent to ENDF-6's B(2) value.
- **`e_max`:** [**required**, must appear one time] The upper energy limit for which $S_{\alpha,\beta}$ may be used.
- **`boundAtomCrossSection`:** [**required**, must appear one time] The bound atom cross section for the principal scatterer.
- **`coherentAtomCrossSection`:** [optional, when present, must appear one time] The coherent cross section, $\sigma_{\text{coh},i}$. In the incoherent approximation, this cross section is equivalent to the bound cross section, so is redundant.
- **`distinctScatteringKernel`:** [optional, when present, must appear one time] The distinct part of the scattering kernel, $\mathcal{S}_{\text{dis},i}(\alpha,\beta,T)$
- **`selfScatteringKernel`:** [**required**, must appear one time] The self part of the scat- -tering kernel, $\mathcal{S}_{\text{self},i}(\alpha,\beta,T)$
- **`T_effective`:** [optional, when present, must appear one time] Effective tempera- -ture to use for this scattering atom in the short-collision-time approximation.

XML Example(s) of `scatteringAtom`

```
<scatteringAtom
        pid="..."
        primaryScatterer="..."
        numberPerMolecule="...">
    <mass>...</mass>
    <e_critical>...</e_critical>
    <e_max>...</e_max>
    <boundAtomCrossSection>...</boundAtomCrossSection>
    <coherentAtomCrossSection>...</coherentAtomCrossSection>
    <distinctScatteringKernel>...</distinctScatteringKernel>
    <selfScatteringKernel>...</selfScatteringKernel>
    <T_effective>...</T_effective></scatteringAtom>
```

20.3.4 Thermal scattering law type: `e_critical`

Energy above which the static model of elastic scattering is adequate. Above this energy, total scattering properties may be obtained from MT=2 as givin in File 2 or 3 and File 4 of the appropriate materials. This is equivalent to ENDF-6's `B(2)` field.

Specifications for `e_critical`

Node name: `e_critical`
Attributes: The list of additional allowed attributes is:
  `unit` [`XMLName`, optional] Unit of energy. "eV" is a good choice.
  `value` [`Float64`, optional] Value of the energy.
Child nodes: This element has no child nodes

XML Example(s) of `e_critical`

```
<e_critical
        unit="..."
        value="..."></e_critical>
```

20.3.5 Thermal scattering law type: `e_max`

The upper energy limit for which $S_{\alpha,\beta}$ may be used. This is equivalent to ENDF-6's `B(4)` field.

Specifications for `e_max`

Node name: `e_max`
Attributes: The list of additional allowed attributes is:
  `unit` [`XMLName`, optional] Unit of energy. “eV” is a good choice.
  `value` [`Float64`, optional] Value of the energy.
Child nodes: This element has no child nodes

XML Example(s) of `e_max`

```
<e_max
        unit="..."
        value="..."></e_max>
```

20.3.6 Thermal scattering law type: `coherentAtomCrossSection`

The coherent cross section, $\sigma_{\mathrm{coh},i}$. In the incoherent approximation, this cross section is equivalent to the bound cross section, so is redundant.

Specifications for `coherentAtomCrossSection`

Node name: `coherentAtomCrossSection`
Attributes: The list of additional allowed attributes is:
  `unit` [`XMLName`, **required**] Unit of area, such as “b”.
  `value` [`Float64`, **required**] Value of the cross section.
Child nodes: This element has no child nodes

XML Example(s) of `coherentAtomCrossSection`

```
<coherentAtomCrossSection
        unit="..."
        value="...">
</coherentAtomCrossSection>
```

20.3.7 Thermal scattering law type: `boundAtomCrossSection`

The $\sigma_{\mathrm{bound},i}$ cross section.

Specifications for `boundAtomCrossSection`

Node name: `boundAtomCrossSection`
Attributes: The list of additional allowed attributes is:
- **`unit`** [`XMLName`, **required**] Any valid unit of area is acceptable however "b" is most appropriate.
- **`value`** [`Float64`, **required**] Value of the cross section.

Child nodes: This element has no child nodes

XML Example(s) of `boundAtomCrossSection`

```
<boundAtomCrossSection
        unit="..."
        value="...">
</boundAtomCrossSection>
```

20.3.8 Thermal scattering law type: `selfScatteringKernel`

The self-part of the scattering kernel, denoted as $\mathcal{S}_{\text{self},i}(\alpha,\beta,T)$. If the kernel is symmet- -ric in β, then it is useful to define and provide $S_{\text{self},i}(\alpha,\beta,T) = e^{\beta/2}\mathcal{S}_{\text{self},i}(\alpha,\beta,T)$. With this, $\mathcal{S}_{\text{self},i}(\alpha,\beta,T) = e^{-\beta}\mathcal{S}_{\text{self},i}(\alpha,-\beta,T)$ becomes $S_{\text{self},i}(\alpha,\beta,T) = S_{\text{self},i}(\alpha,-\beta,T)$.

Note that the rectangular $\alpha-\beta$ grid covered by the the interpolation tables here, when mapped back to $E-\mu$ space, does not cover the full range of E and μ. In the legacy ENDF format the $\alpha-\beta$ grid is padded with zeros. When one encounters the zeros, one is supposed to switch from the full $\mathcal{S}_{\text{self},i}(\alpha,\beta,T)$ to the short collision time approximation. How this transition is effected in practice is defined by the processing codes.

In ENDF, there is an option for storing $\log(\mathcal{S}_{\text{self},i}(\alpha,\beta,T))$, rather than $\mathcal{S}_{\text{self},i}(\alpha,\beta,T)$. This allows for a larger dynamic range of values of $\mathcal{S}_{\text{self},i}(\alpha,\beta,T)$, given ENDF's limited precision. GNDS does not suffer from the same space limitations of ENDF so has no need for this flag.

Specifications for `selfScatteringKernel`

Node name: `selfScatteringKernel`
Attributes: The list of additional allowed attributes is:
- **`symmetric`** [`Boolean`, optional, default is "false"] Indicates whether the asymmetric version of the scattering kernel is used. Equivalent to the ENDF-6 `LASYM` flag.

Child nodes: The list of additional allowed child nodes is:
- **`gridded3d`:** [optional, when present, * one of marked children must appear one time] The self part of the scattering kernel using a `gridded3d` container with axes α, β and T

GaussianApproximation: [optional, when present, * one of marked children must appear one time] The self part of the scattering kernel using the Gaussian approximation.

SCTApproximation: [optional, when present, * one of marked children must appear one time] The self part of the scattering kernel using the short collision time approximation.

freeGasApproximation: [optional, when present, * one of marked children must appear one time] The self part of the scattering kernel using the free gas approximation.

XML Example(s) of `selfScatteringKernel`

```
<selfScatteringKernel
        symmetric="...">
    <gridded3d>...</gridded3d>
    <GaussianApproximation>...</GaussianApproximation>
    <SCTApproximation>...</SCTApproximation>
    <freeGasApproximation>...</freeGasApproximation></selfScatteringKernel>
```

20.3.9 Thermal scattering law type: `distinctScatteringKernel`

The distinct part of the scattering kernel, $\mathcal{S}_{\mathrm{dis},i}(\alpha, \beta, T)$

Specifications for `distinctScatteringKernel`

Node name: `distinctScatteringKernel`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

gridded3d: [optional, when present, must appear one time] The distinct part of the scattering kernel using a `gridded3d` container with axes α, β and T

XML Example(s) of `distinctScatteringKernel`

```
<distinctScatteringKernel>
    <gridded3d>...</gridded3d></distinctScatteringKernel>
```

20.3.10 Thermal scattering law type: `T_effective`

Effective temperatures for use in the short collision time approximation. Note that the short-collision-time approximation may be required for more than one type of scattering kernel: if the scattering kernel is 'SCTapproximation' then the SCT is used everywhere, but it is also used with scattering kernel 'gridded3d' for values of α and β outside the tabulated array.

Specifications for `T_effective`

Node name: `T_effective`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- `XYs1d`: [**required**, must appear one time] Function giving the correspondence between actual and effective temperature. Both axes are in units of temperature, e.g. 'K'.

XML Example(s) of `T_effective`

```
<T_effective>
    <XYs1d>...</XYs1d></T_effective>
```

20.4 Self part of the scattering kernel in the Gaussian approximation

A further approximation of the incoherent approximation is the Gaussian approximation, where the "self" part of the intermediate scattering function may be written in terms of the material's phonon frequency $\rho(\omega)$.

20.4.1 Thermal scattering law type: `GaussianApproximation`

By making the so-called Gaussian approximation to the intermediate scattering function (Hehr, 2010; Hove, 1954), the self part of the scattering kernel can be written in terms of the material phonon spectrum $\rho(\omega)$:

$$\mathcal{S}_{\text{self}}(\alpha,\beta) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dt\, e^{i\beta t} e^{-\gamma(t)} \tag{20.11}$$

where

$$\gamma(t) = \alpha \int_{-\infty}^{\infty} d\omega \rho(\omega)\left(1 - e^{-i\omega t}\right) \frac{e^{-\omega/2k_BT}}{(2\omega/k_BT)\sinh(\omega/2k_BT)} \tag{20.12}$$

This Fourier transform is coded in R.E. MacFarlane's LEAPR module of NJOY (MacFarlane et al., 2017) and can be used to generate the inelastic scattering kernel.

Note that if the full scattering kernel is well approximated by only the self term and in this Gaussian approximation, the entire scattering kernel can be specified with the phonon spectrum. This spectrum may have a discrete portion and/or a continuous portion. Nevertheless, this offers one a compact way to encapsulate the scattering kernel and it provides one with a two-dimensional object that one can use to specify covariance.

Specifications for `GaussianApproximation`

Node name: `GaussianApproximation`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`phononSpectrum`**: [optional, when present, must appear one time] The phonon spectrum $\rho(\omega)$.

XML Example(s) of `GaussianApproximation`

```
<GaussianApproximation>
    <phononSpectrum>...</phononSpectrum></GaussianApproximation>
```

20.4.2 Thermal scattering law type: `phononSpectrum`

The phonon spectrum $\rho(\omega)$

Specifications for `phononSpectrum`

Node name: `phononSpectrum`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`**: [**required**, must appear one time] The actual interpolation table for the phonon spectrum $\rho(\omega)$

XML Example(s) of `phononSpectrum`

```
<phononSpectrum>
    <XYs1d>...</XYs1d></phononSpectrum>
```

20.5 Self part of the scattering kernel in the short collision time approximation

The Gaussian approximation to the "self" part of the intermediate scattering function may be further approximated. Following (Cacuci, 2010), one can expand the time-dependent part of the scattering kernel to arrive at the phonon expansion:

$$e^{-\gamma(t)} = -\alpha\lambda_s \sum_{n=0}^{\infty} \frac{\alpha^n}{n!} \left[\int_{-\infty}^{\infty} d\omega P_s(\omega) e^{-\omega/2k_BT} e^{-i\omega t} \right]^n, \tag{20.13}$$

where $P_s(\omega) = \rho(\omega)/\left[(2\omega/k_BT)\sinh(\omega/2k_BT)\right]$ and the Debye-Waller coefficient is

$$\lambda_s = \int_{-\infty}^{\infty} d\omega P_s(\omega) e^{-\omega/2k_BT} \tag{20.14}$$

The n^{th} term in this expansion is identified with the number of phonons involved in the collision (Cacuci, 2010). This expansion allows one to arrive at two approximations given in the ENDF format manual that must be grandfathered into the new data structure: elastic coherent, elastic incoherent and the short collision time approximations. In the case of the elastic (in)coherent data, one sets $E = E'$ which then forces $\beta \to 0$ and simplifies the phonon expansion to the first term, the zero phonon limit. Alternatively, in the limit of large n, one arrives at the short collision time approximation (called this because the large number of collisions implies a short time for each individual collision).

20.5.1 Thermal scattering law type: `SCTApproximation`

Indicates that the short-collision-time (SCT) approximation should be used for this scattering atom for all values of α and β, not just for values outside the tabulated grid.[28]

$$S_{\text{self},i}(\alpha,\beta,T) \approx \frac{\exp\left[-\frac{(\alpha-|\beta|)^2 T}{4\alpha T_{\text{eff},i}(T)} - \frac{|\beta|}{2}\right]}{\sqrt{4\pi\alpha\frac{T_{\text{eff},i}(T)}{T}}} \tag{20.15}$$

The legacy ENDF-6 format signals this with the B(7)=0.0 flag

Specifications for `SCTApproximation`

Node name: `SCTApproximation`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

[28]The SCT equation is given correctly in the General Atomics report GA-9950, UC-80, Reactor Technology (1970), but is misprinted in LA-9303-M VOL II (ENDF-324) 1982, and in BNL-NCS-44945-05-Revised June 2005 (ENDF-102).

XML Example(s) of `SCTApproximation`

```
<SCTApproximation/>
```

20.6 Self part of the scattering kernel in the free gas approxima--tion

Functional form of the self-scattering kernel in the form of a free gas approximation.

20.6.1 Thermal scattering law type: `freeGasApproximation`

Indicates that a free-gas approximation should be used for this scattering atom. In practice the free-gas approximation is typically used for secondary scattering atoms that have little impact on the scattering cross section, and processing codes may simply ignore the contribution of these scattering atoms. The legacy ENDF-6 format signals this with the B(7)=1.0 flag

Specifications for `freeGasApproximation`

Node name: `freeGasApproximation`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `freeGasApproximation`

```
<freeGasApproximation/>
```

20.7 Small α approximation

Codes reconstructing the incoherent inelastic cross section will likely encounter values of α below the lower limit given in $S(\alpha,\beta,T)$. While the short collision time approxi--mation may produce reasonable results for these small values of α, a better approach may be to use a power-law extrapolation, i.e. $S(\alpha) = C \cdot \alpha^N$ where C and N are de--termined from the lowest few $S(\alpha)$ points provided by the evaluator. This approach can help extend the range of $S(\alpha,\beta,T)$ a few orders of magnitude below what the evaluator provided, but for very small values of α (e.g. $\alpha < 10^{-10}$) the short collision time approximation falls off faster than a power law and is likely a better choice.

20.8 Placement of TSL data within a `reaction` element

The `thermalNeutronScatteringLaw1d` and `thermalNeutronScatteringLaw` nodes are used to link the cross section and outgoing neutron distribution data back to the double-differential thermal neutron scattering law data.

20.8.1 Thermal scattering law type: `thermalNeutronScatteringLaw1d`

Used to link the reaction `crossSection` to the corresponding thermal neutron scattering law entry in the `doubleDifferentialCrossSection`.

Specifications for `thermalNeutronScatteringLaw1d`

Node name: `thermalNeutronScatteringLaw1d`
Attributes: The list of additional allowed attributes is:
    `label` [`XMLName`, **required**] Style label for this element.
    `href` [`XMLName`, **required**] The URL this element links to
Child nodes: This element has no child nodes

XML Example(s) of `thermalNeutronScatteringLaw1d`

```
<reactions>
  <reaction label="n [thermalNeutronScatteringLaw incoherent-elastic]" ENDF_MT="2">
    <doubleDifferentialCrossSection>
      <thermalNeutronScatteringLaw_incoherentElastic
          label="eval" pid="n" productFrame="lab">
        ...
      </thermalNeutronScatteringLaw_incoherentElastic>
    </doubleDifferentialCrossSection>
    <crossSection>
      <thermalNeutronScatteringLaw1d label="eval"
          href="../../doubleDifferentialCrossSection
             /thermalNeutronScatteringLaw_incoherentElastic[@label='eval']"/>
    </crossSection>
    ...
```

20.8.2 Thermal scattering law type: `thermalNeutronScatteringLaw`

Used to link the outgoing neutron `distribution` to the corresponding thermal neutron scattering law entry in the `doubleDifferentialCrossSection`.

Specifications for `thermalNeutronScatteringLaw`

Node name: `thermalNeutronScatteringLaw`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**] Style label for this element.
  `href` [`XMLName`, **required**] The URL this element links to
Child nodes: This element has no child nodes

XML Example(s) of `thermalNeutronScatteringLaw`

```
<reactions>
  <reaction label="n [thermalNeutronScatteringLaw incoherent-inelastic]" ENDF_MT="4">
    <doubleDifferentialCrossSection>
      <thermalNeutronScatteringLaw_incoherentInelastic
          label="eval" pid="n" productFrame="lab">
        ...
      </thermalNeutronScatteringLaw_incoherentInelastic>
    </doubleDifferentialCrossSection>
    <crossSection>...</crossSection>
    <outputChannel genre="twoBody"
        process="thermalNeutronScatteringLaw incoherent-inelastic">
      <Q>..</Q>
      <products>
        <product pid="n" label="n">
          <multiplicity>..</multiplicity>
          <distribution>
            <thermalNeutronScatteringLaw label="eval"
                 href="../../../../../../doubleDifferentialCrossSection
                     /thermalNeutronScatteringLaw_incoherentInelastic[@label='eval']"/>
          </distribution>
        </product>
      </products>
    </outputChannel>
  </reaction>
</reactions>
```

21. Fission fragment data

Fission is a complicated reaction to model and store in evaluations, in part since it is really a sum of many different, distinguishable reactions that can produce any of more than a thousand products. In principle every possible combination of primary, secondary and sometimes ternary (A$\leq$4) fission products (and number of prompt neutrons and photons) could be treated as a separate reaction, but this would place a much higher burden on evaluators and interpretation in practical applications. For practical applications, fission is most often broken up into at most 4 reactions (i.e. 1st-, 2nd-, 3rd-, and 4th-chance fission), and more commonly evaluations only provide a full description of the outgoing products for 'total' fission.

Fission product yields are also a complex topic in nuclear data since the distribution of products evolves rapidly with elapsed time after fission. The original products are emitted immediately at the scission point and are sometimes called 'scission products'. These nuclei are typically in highly excited states, and often proceed to emit prompt neutrons on a very short time scale ($\leq 10^{-14}$ s). After prompt emission the 'prompt fission products' remain. These are typically neutron-rich nuclei, and proceed to β-decay back towards stability on a longer time frame. Evaluations therefore also often store 'cumulative yields' representing average populations after some time has elapsed.

In GNDS, fission is mostly treated like other `reaction` nodes. However, some special types of data are defined that only appear inside fission reactions. These include several fission-specific energy distribution forms for use inside the `uncorrelated` distribution (see section 18.9, and a `fissionFragmentData` section in the `outputChannel` that lists information about fission fragments, delayed neutron emission and the partition of fission energy release into various types of products.

Prior to the release of GNDS-1.9, some proposals had already been made for changing how fission-related data like fission product yields, average energy release and β-delayed neutrons would be stored. These proposals were intended to address some basic problems with fission data in GNDS-1.9:

- Induced fission product yields were stored in a separate file in GNDS-1.9, not together with other reaction data.
- β-delayed neutrons and photons were stored like all other reaction products in an `outputChannel`, but they are not direct products of the reaction and require an additional attribute (the characteristic decay constant).

- The Q-value for induced fission was complicated (see section fissionEnergyRe--leased) and unlike all other GNDS Q values.

The fissionFragmentData container was proposed to address these issues. It stores all information related to the heavy fission fragments and their subsequent decay, as well as information about the partition of energy into various types of particles following fission.

21.1 fissionFragmentData

21.1.1 Fission product type: fissionFragmentData

Nuclear fission generally produces some prompt neutrons and gammas and occasion--ally light ions, plus two large fission fragments. These fragments may then decay and emit additional radiation including delayed neutrons / gammas, betas, etc. The fissionFragmentData section describes the heavy (A > 4) fragments and their subse--quent decay. When used for particle-induced fission, this node appears inside the outputChannel node of a fission reaction. When used for spontaneous fission, this node appears in the nuclide node inside the PoPs database.

Specifications for fissionFragmentData

Node name: fissionFragmentData
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
delayedNeutrons: [optional, when present, must appear one time] List of delayed neutrons. Each has an associated decay rate, multiplicity and energy distri--bution.
fissionEnergyReleased: [optional, when present, must appear one time] Describes how energy released during fission is partitioned between various products.
productYields: [optional, when present, must appear one time] Container to hold lists of productYield, including evaluated or processed yields.

XML Example(s) of fissionFragmentData

```
<fissionFragmentData>
    <delayedNeutrons>...</delayedNeutrons>
    <fissionEnergyReleased>...</fissionEnergyReleased>
    <productYields>...</productYields></fissionFragmentData>
```

The delayedNeutrons and fissionEnergyReleased nodes are described in this chapter, and the productYields node is described in chapter 22.

21.2 Delayed Neutrons

Delayed neutrons (and photons) from fission must be treated differently from prompt reaction products. They are generally produced on a time scale from milliseconds to hours or days after the fission event by the β-decay of the fission products. In principal, if complete information about the structure and decay properties of those fission products were available, the multiplicity and outgoing spectra of the delayed products could be computed directly by sampling fission products and then following their decay. In practice, however, delayed products are grouped together into time-averaged products. For example, an evaluation may contain a `delayedNeutron` representing all neutrons emitted at an average rate of 0.1/s, and another for neutrons emitted at a rate of 1/s.

21.2.1 Fission product type: `delayedNeutrons`

Stores a list of delayed neutrons, each with an associated decay rate, multiplicity and energy distribution.

Specifications for `delayedNeutrons`

Node name: `delayedNeutrons`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`delayedNeutron`:** [**required**, must appear at least one time] Information about neu--trons emitted at the specified 'decay rate' after fission.

XML Example(s) of `delayedNeutrons`

```
<delayedNeutrons>
    <delayedNeutron>...</delayedNeutron></delayedNeutrons>
```

21.2.2 Fission product type: `delayedNeutron`

Stores a delayed neutron product, representing an average multiplicity and distribution for neutrons emitted with the specified decay rate.

Specifications for `delayedNeutron`

Node name: `delayedNeutron`
Attributes: The list of additional allowed attributes is:

`label` [`XMLName`, **required**] Unique label for this delayed neutron group.

Child nodes: The list of additional allowed child nodes is:

`rate`: [**required**, must appear one time] Decay rate (in decays/second or equivalent) for this delayed neutron group.

`product`: [**required**, must appear one time] Product describing this delayed neu--tron. Includes average multiplicity and distribution info.

XML Example(s) of `delayedNeutron`

```
<delayedNeutron
        label="...">
    <rate>...</rate>
    <product>...</product></delayedNeutron>
```

21.2.3 Fission product type: `rate`

Defines the decay rate for the parent `delayedNeutron`.

Specifications for `rate`

Node name: `rate`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`double`: [**required**, must appear one time] Defines the decay rate. Recommended units are ‘1/s’ or equivalent.

XML Example(s) of `rate`

```
<rate>
    <double>...</double></rate>
```

21.3 Energy release during fission

This section describes the energy released from fission. Fission is different from other reactions in that it produces a statistical set of products rather than a defined set. For example, an (n,2n) always produces two neutrons and a residual product. Because the mass of the products is fixed, the Q-value (based on the mass difference of its reactants and products) for the reaction is fixed. Fission produces a complex set of products that are typically defined within the evaluated data libraries in the sense of statistical averages.

Fission is generally understood to be the process where the nucleus splits into two or more *fission fragments*. While these are often thought of as roughly equal masses, in reality they can have a significant asymmetry. It is at the scission point, where the nucleus has unalterably become two (or three in ternary fission) smaller mass nuclei, that the issue of energy release begins. Because of the nature of this reaction there are typically more than a thousand different pairs of fission fragments that have a significant likelihood of occuring with individual fission event having a total Q-value defined by the specific fission fragments created. The distribution of Q-values has a significant spread about the average fission Q-value.

Because of their charge, the fragments quickly accelerate ($\sim 10^{-20}$ sec.) in opposite directions and deposit the bulk of the their energy locally (`fissionEnergyReleased`), typically within a few micrometers. At short times ($\sim 10^{-18}$ – 10^{-7} sec.), these highly excited *primary fission fragments* emit *prompt* neutrons (`promptNeutronKE`) and gamma rays (`promptGammaEnergy`) that transport a portion of the energy away, depositing it over a range of centimeters or longer. The first three fission energy release compo--nents (`promptProductKE`, `promptNeutronKE`, and `promptGammaEnergy`), must be correctly described to predict the energy deposition profile within a nuclear reactor or other fissioning system during a transient event, for example a criticality accident. The `KERMA` (`K`inetic `E`nergy `R`eleased to `MA`tter, typically expressed in units of eV-barns) defines the amount of energy deposited in the material and here is the average kinetic energy of the primary fission fragments times the likelihood of a fission reaction (that is, `fissionEnergyReleased` times the fission cross section). The energy from the prompt neutrons and gammas must be transported in order to accurately model its subsequent energy deposition (though crude models sometimes deposit this energy locally).

The initial *fission products* (fission fragments after prompt emissions) remain in an excited state and undergo subsequent radioactive decay. These *delayed* emissions are typical of neutron-rich nuclei and are dominated by beta emissions; beta emissions al--ways include an electron/neutrino or positron/anti-neutrino emission (`neutrinoEnergy`) and may include a beta delayed neutron (`delayedNeutronKE`) or proton emission. These beta decays occur on the timescales of milli-seconds to days or longer and transform the *primary fission products* into *secondary fission products* which often undergo an immediate sequence of internal transitions producing delayed gamma-rays (`nonNeutrinoEnergy`) or subsequent complex decays as well. For steady state systems like a nuclear reactor, this delayed energy can be a significant portion of the total energy during power production. However, for spent fuel or during a reactor shutdown, this delayed energy release becomes one of the primary safety hazards. Because of the complex time and energy dependence of the fission fragments which lead to the delayed emissions, accurate estimates of this energy deposition are best done by post processing codes.

Evaluated data for the fission energy release were not included in the initial ENDF/B nuclear data libraries. It was not until the ENDF-5 format (see the ENDF-102 Manual rev 1983) that the MF 1 MT 458 section was defined and data were added to the ENDF/B-V release. While it was well understood that these quantities would be energy dependent due to the changes in fission fragment yields as a function of the excitation energy, only rudimentary estimates of this energy dependence were available. The understanding of these quantities has since improved (if only marginally) and an explicit polynomial representation of the energy dependence was added in 2010 and an option to provide pointwise descriptions in 2017.

In GNDS-1.9, the `fissionEnergyReleased` node was stored inside the `Q` section for the fission `outputChannel`. `fissionEnergyReleased` behaves differently from other Q-values, however: rather than the actual energy released it stores details of how the energy release is divided among both prompt and delayed particles produced during fission.

The fission Q-value was therefore restricted to only storing a 1-dimensional function (i.e. `constant1d`, `XYs1d`, `regions1d` or `polynomial1d`). The `fissionEnergyReleased` section is moved inside the `fissionFragmentData`. The fission Q-value should only store the prompt energy release, equal to the sum of the `promptProductKE`, `promptNeutronKE` and `promptGammaEnergy` in the `fissionEnergyReleased` section.

21.3.1 Fission transport type: `fissionEnergyReleased`

Specifications for `fissionEnergyReleased`

Node name: `fissionEnergyReleased`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, optional]
Child nodes: The list of additional allowed child nodes is:
  `delayedBetaEnergy`: [optional, when present, must appear one time] Total energy released by delayed β's.
  `delayedGammaEnergy`: [optional, when present, must appear one time] Total energy released by the emission of delayed γ rays.
  `delayedNeutronKE`: [optional, when present, must appear one time] Kinetic energy of the delayed fission neutrons.
  `neutrinoEnergy`: [optional, when present, must appear one time] Energy carried away by neutrinos.
  `nonNeutrinoEnergy`: [optional, when present, must appear one time] Total energy less the energy of the neutrinos `(ET - ENU)`
  `promptGammaEnergy`: [optional, when present, must appear one time] Total energy released by the emission of prompt γ rays.
  `promptNeutronKE`: [optional, when present, must appear one time] Kinetic energy of the prompt fission neutrons.
  `promptProductKE`: [optional, when present, must appear one time] Kinetic energy of the fission products (following prompt neutron emission from the fission fragments).
  `totalEnergy`: [optional, when present, must appear one time] Sum of the partial energies. This sum is the total energy release per fission and by definition is the fission Q-value.

XML Example(s) of `fissionEnergyReleased`

```
<fissionEnergyReleased
        label="...">
    <delayedBetaEnergy>...</delayedBetaEnergy>
    <delayedGammaEnergy>...</delayedGammaEnergy>
    <delayedNeutronKE>...</delayedNeutronKE>
    <neutrinoEnergy>...</neutrinoEnergy>
    <nonNeutrinoEnergy>...</nonNeutrinoEnergy>
    <promptGammaEnergy>...</promptGammaEnergy>
    <promptNeutronKE>...</promptNeutronKE>
    <promptProductKE>...</promptProductKE>
    <totalEnergy>...</totalEnergy></fissionEnergyReleased>
```

21.3.2 Fission transport type: `delayedBetaEnergy`

Total energy released by delayed β's. This corresponds to ENDF's `EB`.

Specifications for `delayedBetaEnergy`

Node name: `delayedBetaEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `delayedBetaEnergy`

```
<delayedBetaEnergy>
    <polynomial1d>...</polynomial1d></delayedBetaEnergy>
```

21.3.3 Fission transport type: `delayedGammaEnergy`

Total energy released by the emission of delayed γ rays. This corresponds to ENDF's `EGD`.

Specifications for `delayedGammaEnergy`

Node name: `delayedGammaEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `delayedGammaEnergy`

```
<delayedGammaEnergy>
    <polynomial1d>...</polynomial1d></delayedGammaEnergy>
```

21.3.4 Fission transport type: `delayedNeutronKE`

Kinetic energy of the delayed fission neutrons. This corresponds to ENDF's `END`.

Specifications for `delayedNeutronKE`

Node name: `delayedNeutronKE`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `XYs1d`: [optional, when present, must appear one time]
  `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `delayedNeutronKE`

```
<delayedNeutronKE>
    <XYs1d>...</XYs1d>
    <polynomial1d>...</polynomial1d></delayedNeutronKE>
```

21.3.5 Fission transport type: `neutrinoEnergy`

Energy carried away by neutrinos. This corresponds to ENDF's `ENU`.

Specifications for `neutrinoEnergy`

Node name: `neutrinoEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `neutrinoEnergy`

```
<neutrinoEnergy>
    <polynomial1d>...</polynomial1d></neutrinoEnergy>
```

21.3.6 Fission transport type: `nonNeutrinoEnergy`

Total energy less the energy of the neutrinos (`ET - ENU`).

Specifications for `nonNeutrinoEnergy`

Node name: `nonNeutrinoEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `nonNeutrinoEnergy`

```
<nonNeutrinoEnergy>
    <polynomial1d>...</polynomial1d></nonNeutrinoEnergy>
```

21.3.7 Fission transport type: `promptGammaEnergy`

Total energy released by the emission of prompt γ rays. This corresponds to ENDF's `EGP`.

Specifications for `promptGammaEnergy`

Node name: `promptGammaEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `XYs1d`: [optional, when present, must appear one time]
 `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `promptGammaEnergy`

```
<promptGammaEnergy>
    <XYs1d>...</XYs1d>
    <polynomial1d>...</polynomial1d></promptGammaEnergy>
```

21.3.8 Fission transport type: `promptNeutronKE`

Kinetic energy of the prompt fission neutrons. This corresponds to ENDF's `ENP`.

Specifications for `promptNeutronKE`

Node name: `promptNeutronKE`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `XYs1d`: [optional, when present, must appear one time]
  `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `promptNeutronKE`

```
<promptNeutronKE>
    <XYs1d>...</XYs1d>
    <polynomial1d>...</polynomial1d></promptNeutronKE>
```

21.3.9 Fission transport type: `promptProductKE`

Kinetic energy of the fission products (following prompt neutron emission from the fission fragments). This corresponds to ENDF's EFR.

Specifications for `promptProductKE`

Node name: `promptProductKE`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `XYs1d`: [optional, when present, must appear one time]
  `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `promptProductKE`

```
<promptProductKE>
    <XYs1d>...</XYs1d>
    <polynomial1d>...</polynomial1d></promptProductKE>
```

21.3.10 Fission transport type: `totalEnergy`

Sum of the partial energies. This sum is the total energy release per fission and by definition is the fission Q-value. This corresponds to ENDF's ET.

Specifications for `totalEnergy`

Node name: `totalEnergy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `polynomial1d`: [optional, when present, must appear one time]

XML Example(s) of `totalEnergy`

```
<totalEnergy>
    <polynomial1d>...</polynomial1d></totalEnergy>
```

22. Fission product yields

Unlike other low-energy nuclear reactions, where the outgoing nuclides are defined by the reaction channel, fission can produce hundreds or thousands of different nuclides with non-negligible probability. As a result, this data is typically separated from the fission cross section as a probability distribution over products per fission event.

Immediately after fission, the initial products are typically well above the neutron separation energy. They emit prompt neutrons and gammas on a short timescale (approximately 10^{-14} s). After those prompt products are emitted, the resulting dis--tribution of product nuclides probabilities is referred to as the independent fission product yield (IFPY). These are normalised per fission event so that the yields, ignoring light-particle production, sum to 2.0.

The cumulative fission product yield (CFPY) is defined as:

$$CFPY(N,t) = IFPY(N) + \sum_{N'} B(N' \to N,t)CFPY(N,t), \tag{22.1}$$

where the nuclides N represent a charge, mass and isomeric state, the branching matrix B is populated with entries between 0 and 1 that represent the probability of producing the decay product through one decay, and the time t may be used to limit fraction of decays that occur based on the timescale of an application. It should be stressed that the normalisation of the CFPY is strictly greater than 2.0 and that these are not yields for any given time t, but the integral yield, ignoring decay, of each product between the time of the independent yield and the decay time t.

It is common to write the CFPY definition as a matrix expression,

$$\begin{aligned} \mathbf{CFPY}(t) = &\quad \mathbf{IFPY} + \mathbf{B}(t)\mathbf{CFPY}(t) \\ \mathbf{CFPY}(t) = &\quad (\mathbf{1} - \mathbf{B}(t))^{-1}\mathbf{IFPY} \\ \equiv &\quad \mathbf{Q}(t)\mathbf{IFPY}, \end{aligned} \tag{22.2}$$

where $\mathbf{Q}(t)$ represents the identity matrix plus the branching probabilities through any number of decay events. The $\mathbf{Q}$-matrix is typically defined this way, in terms of the single-decay matrix $\mathbf{B}$, since the latter data exist in well-known evaluated libraries[29]. This is also how the decay data is described in GNDS (c.f. Section 12.2.2).

[29]Note that the matrix of decay probabilities $\mathbf{B}(t)$ is taken from an evaluated nuclear data sub-library that must be used in the translation between independent and cumulative yields.

22.1 General fission product yield markups

The nodes used to collect fission product yield together are similar whether fission is induced or happens spontaneously. In this section the nodes both cases have in common are discussed. In the subsequent sections their use is elaborated upon and on the nodes that are specific to induced or spontaneous fission are discussed.

22.1.1 Fission product type: `productYields`

Container to hold lists of `productYield`, including evaluated or processed yields.

Specifications for `productYields`

Node name: `productYields`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`productYield`**: [**required**, must appear at least one time] One 'style' of product yield data.

XML Example(s) of `productYields`

```
<productYields>
    <productYield>...</productYield></productYields>
```

22.1.2 Fission product type: `productYield`

This element actually contains the yields, broken out by the time at which the yield is considered.

Specifications for `productYield`

Node name: `productYield`
Attributes: The list of additional allowed attributes is:
- **`label`** [`XMLName`, **required**] Denotes the `style` of the evaluated data.

Child nodes: The list of additional allowed child nodes is:
- **`nuclides`**: [optional, when present, must appear one time] List of nuclide id's. For spontaneous yields, all `elapsedTimes` use the same list of nuclides.
- **`elapsedTimes`**: [**required**, must appear one time] Container to hold lists of `elapsed-` `-Time` elements.

XML Example(s) of `productYield`

```
<productYield
        label="...">
    <nuclides>...</nuclides>
    <elapsedTimes>...</elapsedTimes></productYield>
```

22.1.3 Fission product type: `nuclides`

List of nuclide id's, or href pointing to another `nuclides` node. GNDS permits using a different list of nuclides for different incident energies and/or elapsed times, but also supports linking if the evaluator wishes to use the same list of nuclides throughout.

Specifications for `nuclides`

Node name: `nuclides`
Attributes: The list of additional allowed attributes is:
- **`href`** [`bodyText`, optional] xPath link to another `nuclides` node, indicating to use the same list of nuclides.

Child nodes: This element has no child nodes
Body text: A space delimited list of nuclide id's (e.g. `"Cu79 Cu80 ..."`), arranged in the same order as the appropriate values tabulated in the `values` container(s). For particle induced yields, the `values` container is co-located with this `nuclide` container. For spontaneous yields, the `values` container is inside the appropriate `yields` container. If the 'href' attribute is present, the body text must be empty.

XML Example(s) of `nuclides`

```
<nuclides> V66 V67 Cr66 ... </nuclides>
```

22.1.4 Fission product type: `elapsedTimes`

Container to hold lists of `elapsedTime` elements.

Specifications for `elapsedTimes`

Node name: `elapsedTimes`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`elapsedTime`:** [**required**, must appear at least one time] The collection of yield data corresponding to one time point.

XML Example(s) of `elapsedTimes`

```
<elapsedTimes>
    <elapsedTime>...</elapsedTime></elapsedTimes>
```

22.1.5 Fission product type: `elapsedTime`

The collection of yield data at a given time after fission. For spontaneous fission, the `elapsedTime` node contains a `time` and a `yields` section. For induced fission, each `elapsedTime` contains a `time` and an `incidentEnergies`.

Specifications for `elapsedTime`

Node name: `elapsedTime`
Attributes: The list of additional allowed attributes is:
    `label` [`XMLName`, optional] This labels what the time element corresponds to, namely `"initial"` (e.g. independent) or `"unspecified"` (e.g. cumulative)
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):
    `time`: [**required**, must appear one time] Time corresponding to yields in question.
    `yields`: [optional, when present, * one of marked children must appear one time] For spontaneous fission, this collects the yields.
    `incidentEnergies`: [optional, when present, * one of marked children must appear one time] For induced fission, this collects the data for each of the projectile's incident energies. Contains the list of `incidentEnergy` elements.

XML Example(s) of `elapsedTime`

First example, for neutron-induced fission:

```
<elapsedTime
        label="...">
    <time>...</time>
    <incidentEnergies>...</incidentEnergies></elapsedTime>
```

A second example, for spontaneous fission:

```
<elapsedTime
        label="...">
    <time>...</time>
    <yields>...</yields></elapsedTime>
```

22.1.6 Fission product type: `time`

Time corresponding to the yields in question. Generally yields with time of 0 s are scission yields, times between 0 s and 10^{-14} s are independent fission yields and those with longer times are cumulative yields. Cumulative yields with an unspecified time are denoted with a `string` child node containing “unspecified”.

Specifications for `time`

Node name: `time`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is (only one of the children marked with * for each unique style label is allowed. However, the style label can be omitted and empty, in which case only one of the marked children is allowed):
`double`: [optional, when present, * one of marked children must appear one time]
`string`: [optional, when present, * one of marked children must appear one time]

XML Example(s) of `time`

```
<time>
    <double label="initial" value="0.0" unit="s"/></time>
```

For cumulative yields, the time at which they are relevant may be ill-defined so use the following:

```
<time>
     <string label="unspecified" value="unspecified" unit="s"/></time>
```

22.2 Particle induced fission product yield data

22.2.1 Fission product type: `incidentEnergies`

For induced fission, this collects the data for each of the projectile’s incident energies. Contains the list of `incidentEnergy` elements.

Specifications for `incidentEnergies`

Node name: `incidentEnergies`
Attributes: This element has no attributes

Child nodes: The list of additional allowed child nodes is:
  `incidentEnergy`: [**required**, must appear at least one time] Yield data from one incident energy.

XML Example(s) of `incidentEnergies`

```
<incidentEnergies>
    <incidentEnergy>...</incidentEnergy></incidentEnergies>
```

22.2.2 Fission product type: `incidentEnergy`

Stores yield data for a single incident energy.

Specifications for `incidentEnergy`

Node name: `incidentEnergy`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, **required**] The index of this element.
Child nodes: The list of additional allowed child nodes is:
  `energy`: [**required**, must appear one time] The projectile's incident energy.
  `yields`: [**required**, must appear one time] Nuclides, yields and uncertainties for this elapsed time / incident energy.

XML Example(s) of `incidentEnergy`

```
<incidentEnergy
        label="...">
    <energy>...</energy>
    <yields>...</yields></incidentEnergy>
```

22.2.3 Fission product type: `energy`

The projectile's incident energy.

Specifications for `energy`

Node name: `energy`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `double`: [**required**, must appear one time] The actual energy.

XML Example(s) of `energy`

```
<energy>
    <double>...</double></energy>
```

22.2.4 Fission product type: `yields`

Container for the nuclides and corresponding yield values. This node appears in-side either a `elapsedTime` node (spontaneous fission) or `incidentEnergy` node (induced fission).

Specifications for `yields`

Node name: `yields`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`nuclides`:** [**required**, must appear one time] List of nuclide id's or href pointing to another `nuclides` node.
- **`values`:** [**required**, must appear one time] The actual values, in the order given by the corresponding `nuclides` element.
- **`uncertainty`:** [optional, when present, must appear one time] Uncertainty/covari-ance on the actual values, in the order given by the corresponding `nuclides` element.

XML Example(s) of `yields`

```
<yields>
    <nuclides>...</nuclides>
    <values>...</values>
    <uncertainty>...</uncertainty></yields>
```

22.3 Example fission fragment data for induced fission

Below a more fleshed out example of induced fission is provided, demonstrating the nesting of the `elapsedTime`, `incidentEnergies` and `incidentEnergy` nodes. In this case each incident energy links back to a common list of `nuclides`, but each could define its own list if needed.

```
<outputChannel>
  ...
  <fissionFragmentData>
    <delayedNeutrons>...</delayedNeutrons>
    <fissionEnergyReleased>...</fissionEnergyReleased>
    <productYields>
      <productYield label="eval">
        <nuclides> V66 V67 Cr66 ... </nuclides>
        <elapsedTimes>
          <elapsedTime label="initial">
            <time>
              <double label="initial" value="0.0" unit="s"/></time>
            <incidentEnergies>
              <incidentEnergy label="0">
                <energy>
                  <double label="0" value="500000.0" unit="eV"/></energy>
                <yields>
                  <nuclides href="../../../../../../nuclides"/>
                  <values>1.87064e-17 2.02328e-18 6.25013e-13 ... </values>
                  <uncertainty>
                    <covariance>
                      <array shape="1267,1267" compression="diagonal">
                        <values>...</values></array></covariance>
                  </uncertainty>
                </yields>
              </incidentEnergy>
              <incidentEnergy label="1">...</incidentEnergy>
            .
            .
            .
```

22.4 Example fission fragment data for spontaneous fission

Spontaneous fission data is found within the `nuclide` or `nucleus` node of a PoPs database.

Below a more fleshed out example of spontanteous fission is provided, demonstrating where the `yields` node resides:

```
<nuclide id="Cm244">
  <fissionFragmentData>
    <productYields>
      <productYield label="eval">
        <nuclides> V66 V67 Cr66...</nuclides>
        <elapsedTimes>
          <elapsedTime label="0">
            <time>
              <double label="" value="0.0" unit="s"/></time>
            <yields>
              <nuclides href="../../../../nuclides"/>
              <values>1.37782e-17 6.56914e-19 7.33989e-13...</values>
              <uncertainty>
                <covariance>...</covariance></uncertainty>
            </yields>
          <elapsedTime label="1">
          .
```

```
.
.
```

23. Charged particle transport data

Elastic scattering of charged particles includes components from Coulomb scattering, nuclear scattering, and the interference between them. The Coulomb scattering quan--tum mechanical amplitude is divergent at short distances, leading to divergencies in both the Coulomb and interference components at low energies and small scattering angles. The Coulomb scattering cross section is represented by the Rutherford formula when electronic screening is ignored (as is the case here), but in other cases charged particle elastic scattering can be quite complicated.

In this chapter, the following parameters are defined:

$\sigma_{cd}(\mu, E)$	differential Coulomb scattering cross section in the *center-of-mass* refer--ence frame for distinguishable particles, a.k.a the Rutherford scattering cross section
$\sigma_{ci}(\mu, E)$	differential Coulomb scattering cross section in the *center-of-mass* ref--erence frame for identical particles
E	energy of the incident particle in the *laboratory* reference frame
μ	cosine of the scattering angle in the *center-of-mass* reference frame
m_1	incident particle mass
Z_1, Z_2	charge numbers of the incident particle and target, respectively
s	spin (only necessary for identical particles, $s = 0, \frac{1}{2}, 1, \frac{3}{2}$, *etc.*)
A	target/projectile mass ratio
k	particle wave number
η	dimensionless Coulomb parameter
$u, c,\hbar, \alpha$	fundamental constants.

The differential cross sections can then be written:

$$\sigma_{cd}(\mu, E) = \frac{\eta^2}{k^2(1-\mu)^2} \tag{23.1}$$

$$\sigma_{ci}(\mu, E) = \frac{2\eta^2}{k^2(1-\mu^2)}\left[\frac{1+\mu^2}{1-\mu^2} + \frac{(-1)^{2s}}{2s+1}\cos\left(\eta \ln\frac{1+\mu}{1-\mu}\right)\right] \tag{23.2}$$

$$\text{where} \quad k = \frac{A}{1+A}\sqrt{\frac{2}{\hbar^2}\frac{u}{c^2}\, m_1 E} \times 10^{-14} \tag{23.3}$$

$$\eta = Z_1 Z_2 \sqrt{\frac{\alpha^2 u}{2}\,\frac{m_1}{E}} \tag{23.4}$$

Note that $A = 1$ and $Z_1 = Z_2$ for identical particles.

Note: like ENDF-6, GNDS provides an explicit flag to denote that the target and projectile are identical. While users of the data can check for themselves whether the target and projectile are identical, this flag is provided in the interest of explicitly marking up elastic scattering of identical particles. It is up to evaluators and library maintainers to ensure that this flag is set correctly.

23.1 General markup for charged particle elastic scattering

23.1.1 Charged particle transport type: `CoulombPlusNuclearElastic`

This is either a container holding the differential charged particle scattering cross section or a reference to said container. It may appear inside a `doubleDifferential--CrossSection`, `crossSection` or `distribution` node.

Specifications for `CoulombPlusNuclearElastic`

Node name: `CoulombPlusNuclearElastic`
Attributes: The list of additional allowed attributes is:

- **`href`** [`XMLName`, optional] When `CoulombPlusNuclearElastic` is used in place of an actual distribution this link is used to refer back to the parameterisation that defines the distribution.
- **`identicalParticles`** [`Boolean`, optional, default is “false”] If both outgoing products are the same type of particle, for example in the reaction ‘`p + p -> p + p`’. If true, the angular dependence is symmetric and only applies to $0 \leq \mu \leq \mu_{cutoff}$. Otherwise, the angular dependence is not symmetric and applies $-1 \leq \mu \leq \mu_{cutoff}$.
- **`label`** [`XMLName`, **required**] Style label for this cross section.
- **`pid`** [`XMLName`, **required**] The particle identifier of the outgoing particle.
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

`RutherfordScattering`: [optional, when present, must appear one time] This differential cross section is given by the Rutherford scattering cross section.

`nuclearAmplitudeExpansion`: [optional, when present, must appear one time] This differential cross section is given by the nuclear amplitude expansion.

`nuclearPlusInterference`: [optional, when present, must appear one time] This differential cross section is given by the nuclear plus interference approach.

XML Example(s) of `CoulombPlusNuclearElastic`

```
<CoulombPlusNuclearElastic label="eval" pid="H2" productFrame="centerOfMass">
    <RutherfordScattering/>
    <nuclearPlusInterference muCutoff="0.94">
        <crossSection>
            <XYs1d>...</XYs1d>
        </crossSection>
        <distribution>
            <XYs2d>...</XYs2d>
        </distribution>
    </nuclearPlusInterference>
```

Another example, this time of `CoulombPlusNuclearElastic` being used as a link

```
<CoulombPlusNuclearElastic
     label="eval"
     href="../../doubleDifferentialCrossSection/CoulombPlusNuclearElastic[@label='eval']"/>
```

23.1.2 Charged particle transport type: `RutherfordScattering`

Although strictly speaking the Rutherford scattering cross section is the Coulomb scattering cross section for two non-identical charged particles (see equation (23.1)), this markup is also used to describe the case of identical charged particle Coulomb scattering (see equation (23.2)). In either case, the Coulomb scattering cross section is analytic.

Specifications for `RutherfordScattering`

Node name: `RutherfordScattering`
Attributes: This element has no attributes
Child nodes: This element has no child nodes

XML Example(s) of `RutherfordScattering`

```
<RutherfordScattering/>
```

23.2 Nuclear amplitude expansion

23.2.1 Charged particle transport type: `nuclearAmplitudeExpansion`

The net elastic scattering cross section for distinguishable particles may be written as:

$$\begin{aligned}\sigma_{ed}(\mu,E)=\sigma_{cd}(\mu,E) \quad &- \quad \frac{2\eta}{1-\mu}\mathrm{Re}\left\{\exp\left(i\eta\ln\frac{1-\mu}{2}\right)\sum_{l=0}^{\mathrm{NL}}\frac{2l+1}{2}a_l(E)P_l(\mu)\right\}\\ &+ \quad \sum_{l=0}^{2\mathrm{NL}}\frac{2l+1}{2}b_l(E)P_l(\mu)\end{aligned} \tag{23.5}$$

and the cross section for identical particles is:

$$\begin{aligned}\sigma_{ei}(\mu,E) \quad &= \quad \sigma_{ci}(\mu,E)\\ &- \quad \frac{2\eta}{1-\mu^2}\;\mathrm{Re}\left\{\sum_{l=0}^{\mathrm{NL}}\left[\begin{array}{r}(1+\mu)\exp\left(i\eta\ln\frac{1-\mu}{2}\right)\\ +(-1)^l(1-\mu)\exp\left(i\eta\ln\frac{1+\mu}{2}\right)\end{array}\right]\frac{2l+1}{2}a_l(E)P_l(\mu)\right\}\\ &+ \quad \sum_{l=0}^{\mathrm{NL}}\frac{4l+1}{2}\;b_l(E)P_{2l}(\mu)\end{aligned} \tag{23.6}$$

where the a_l are complex coefficients for expanding the trace of the nuclear scattering amplitude matrix and the b_l are real coefficients for expanding the nuclear scattering cross section. The value of NL represents the highest partial wave contributing to nuclear scattering. Note that $\sigma_{ei}(-\mu,E)=\sigma_{ei}(\mu,E)$. The three terms in Equations (23.5) and (23.6) are Coulomb, interference, and nuclear scattering, respectively.

Specifications for `nuclearAmplitudeExpansion`

Node name: `nuclearAmplitudeExpansion`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`nuclearTerm`:** [**required**, must appear one time] The coefficients $b_l(E)$ in the nuclear term of equations (23.5) or (23.6)
- **`realInterferenceTerm`:** [**required**, must appear one time] The coefficients $\mathtt{Re}(a_l(E))$ in the interference term of equations (23.5) or (23.6)
- **`imaginaryInterferenceTerm`:** [**required**, must appear one time] The coefficients $\mathtt{Im}(a_l(E))$ in the interference term of equations (23.5) or (23.6)

XML Example(s) of `nuclearAmplitudeExpansion`

```
<nuclearAmplitudeExpansion>
    <nuclearTerm>...</nuclearTerm>
    <realInterferenceTerm>...</realInterferenceTerm>
    <imaginaryInterferenceTerm>...</imaginaryInterferenceTerm>
</nuclearAmplitudeExpansion>
```

23.2.2 Charged particle transport type: `nuclearTerm`

The coefficients $b_l(E)$ in the nuclear term of equations (23.5) or (23.6)

Specifications for `nuclearTerm`

Node name: `nuclearTerm`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`XYs2d`:** [optional, when present, must appear one time] The coefficients $b_l(E)$ given as a Legendre moment expansion inside an `XYs2d`
- **`regions2d`:** [optional, when present, must appear one time] The coefficients $b_l(E)$ given as a Legendre moment expansion inside an `regions2d`

XML Example(s) of `nuclearTerm`

```
<nuclearTerm>
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d></nuclearTerm>
```

23.2.3 Charged particle transport type: `realInterferenceTerm`

The coefficients $\texttt{Re}(a_l(E))$ in the interference term of equations (23.5) or (23.6)

Specifications for `realInterferenceTerm`

Node name: `realInterferenceTerm`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`XYs2d`:** [optional, when present, must appear one time] The coefficients $\texttt{Re}(a_l(E))$ given as a Legendre moment expansion inside an `XYs2d`
- **`regions2d`:** [optional, when present, must appear one time] The coefficients $\texttt{Re}(a_l(E))$ given as a Legendre moment expansion inside an `regions2d`

XML Example(s) of `realInterferenceTerm`

```
<realInterferenceTerm>
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d></realInterferenceTerm>
```

23.2.4 Charged particle transport type: `imaginaryInterferenceTerm`

The coefficients $\mathtt{Im}(a_l(E))$ in the interference term of equations (23.5) or (23.6)

Specifications for `imaginaryInterferenceTerm`

Node name: `imaginaryInterferenceTerm`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`XYs2d`:** [optional, when present, must appear one time] The coefficients $\mathtt{Im}(a_l(E))$ given as a Legendre moment expansion inside an `XYs2d`
- **`regions2d`:** [optional, when present, must appear one time] The coefficients $\mathtt{Im}(a_l(E))$ given as a Legendre moment expansion inside an `regions2d`

XML Example(s) of `imaginaryInterferenceTerm`

```
<imaginaryInterferenceTerm>
    <XYs2d>...</XYs2d>
    <regions2d>...</regions2d></imaginaryInterferenceTerm>
```

23.3 Residual amplitude expansion

When only experimental data are available, it is convenient to remove the infinity due to σ_C by subtraction and to remove the remaining infinity in the interference term by multiplication, thereby obtaining the residual cross sections:

$$\sigma_{Rd}(\mu, E) = (1 - \mu) \left[\sigma_{ed}(\mu, E) - \sigma_{cd}(\mu, E)\right] \tag{23.7}$$

and

$$\sigma_{Ri}(\mu, E) = (1 - \mu^2) \left[\sigma_{ei}(\mu, E) - \sigma_{ci}(\mu, E)\right] \tag{23.8}$$

Then σ_R can be given as a Legendre polynomial expansion in the forms:

$$\sigma_{Rd}(\mu, E) = \sum_{l=0}^{\mathrm{NL}} \frac{2l+1}{2} c_{ld}(E) P_l(\mu) \tag{23.9}$$

and

$$\sigma_{Ri}(\mu, E) = \sum_{l=0}^{\mathrm{NL}} \frac{4l+1}{2} c_{li}(E) P_{2l}(\mu) \tag{23.10}$$

Because the interference term oscillates as μ goes to 1, the limit of the Legendre representation of the residual cross section at small angles may not be well defined. However, if the coefficients are chosen properly, the effect of this region will be small because the Coulomb term is large.

Note: This format is described in the ENDF-6 manual but does not appear to be used in practice. For this reason, FUDGE does not translate this data and no GNDS format has yet been defined.

23.4 Nuclear plus interference approach

This approach breaks the distribution up into an effective cross section and effective dis- -tribution. The distribution is defined up to a cut-off angle stored as $\mu_{cutoff} = cos(\theta_{cutoff})$, and the cross section is equal to the integral of $d\sigma/d\Omega$ from $\mu = -1$ to μ_{cutoff}. Nuclear plus interference distribution with P_{NI} in μ is given by equations (23.11) and (23.12)

23.4.1 Charged particle transport type: `nuclearPlusInterference`

It is possible to represent experimental data using the “nuclear plus interference” cross section and angular distribution in the CM system defined by:

$$\sigma_{NI}(E) = \int_{\mu_{\mathrm{min}}}^{\mu_{\mathrm{max}}} \Big[\sigma_e(\mu, E) - \sigma_c(\mu, E)\Big] \, , d\mu \tag{23.11}$$

and

$$P_{NI}(\mu, E) = \begin{cases} \dfrac{\sigma_e(\mu, E) - \sigma_c(\mu, E)}{\sigma_{NI}(E)} & \mu_{\mathrm{min}} \le \mu \le \mu_{\mathrm{max}} \\ 0 & \text{otherwise} \end{cases} \tag{23.12}$$

where $\mu_{min} = -1$ for different particles and 0 for identical particles. The maximum cosine should be as close to 1.0 as possible, especially at high energies where Coulomb scattering is less important. The Coulomb cross section $\sigma_c(\mu, E)$ is to be computed using equations (23.1) or (23.2) for different or identical particles, respectively.

Specifications for `nuclearPlusInterference`

Node name: `nuclearPlusInterference`
Attributes: The list of additional allowed attributes is:

`muCutoff` [`Float64`, **required**] Cosine of the cutoff angle. Above the cutoff angle, the distribution is assumed to be pure Coulomb scattering.

Child nodes: The list of additional allowed child nodes is:

`crossSection`: [**required**, must appear one time] The effective cross section $\sigma_{NI}(E)$

`distribution`: [**required**, must appear one time] The effective angular distribution $P_{NI}(\mu, E)$.

XML Example(s) of `nuclearPlusInterference`

```
<nuclearPlusInterference
        muCutoff="...">
    <crossSection>...</crossSection>
    <distribution>...</distribution></nuclearPlusInterference>
```

24. Atomic reaction data

This chapter describes photo-atomic and electro-atomic reactions. Photon interaction data encompass "smooth" cross sections as well as coherent scattering form factors and incoherent scattering functions.

Electron interaction data consists of smooth cross sections for elastic scattering, bremsstrahlung, excitation, and the ionisation of different atomic subshells and a variety of outgoing distributions. These distributions include the angular distribution for elastically scattered electrons, the outgoing photon spectra and energy loss for bremsstrahlung, the energy transfer for excitation, and the spectra of the scattered and recoil electrons associated with subshell ionisation.

Both photo-atomic and electro-atomic reactions can leave the atom in an ionised state. See the ENDF-6 Manual Chapter 28 (Cross Section Evaluation Working Group, 2018) for a description of the atomic relaxation data needed to compute the outgoing X-ray and electron spectra as an ionised atom relaxes back to neutrality. Note that ENDF's subshell binding energies (equal to the photoelectric edge energy) are converted to reaction Q-values for the respective ionisation reactions.

The legacy ENDF format divides this data into a confusing collection of ENDF MF/MT's reflecting both the fact that atomic scattering data was added into ENDF much later in the library development process and that this data never recieved the detailed attention of other scattering data. Table 24.1 summarised the contents of the ENDF library and ENDF-6 format.

24.1 Electron scattering

On the face of it, electron scattering off atoms is very different from neutron induced reactions on nuclei. However, from the particle transport perspective, an electron is "just another changed particle" and can be handled with existing GNDS structures for both cross sections and outgoing particle distributions. In particular:

- Secondary photons or electrons emitted after electro-atomic reactions, energy given to the residual atom, and the energy transfer associated with excitation are already supported by existing distributions.

Sublibrary	ENDF MT	Reaction Description
atomic_relax	533	Atomic relaxation data
photo-at	501	Total photon scattering
	502	Coherent photon scattering, including form fac--tor (MF=23, 27)
	505	Coherent photon scattering real anomolous scattering factor (MF=27)
	506	Coherent photon scattering imaginary anomolous scattering factor (MF=27)
	504	Incoherent photon scattering, including inco--herent scattering factor (MF=23, 27)
	515	Pair production in an electron field
	516	Total pair production (sum of MT=515 and 517)
	517	Pair production in a nuclear field
	522	Photo-electron absorption
	523	Photo-excitation cross sections
	534-572	Subshell ionisation
electrons	500	Total stopping
	526	Electro-atomic elastic scattering
	527	Electro-atomic bremsstrahlung
	528	Electro-atomic excitation cross section
	534-572	Photo-electric or electro-atomic subshell ionisa--tion

Table 24.1: Summary of ENDF files (MF) and tapes (MT) containing atomic relaxation and scattering data.

- Elastic scattering is represented by the normalised angular distribution for the scattered electron given in tabulated form typically for scattering cosines ranging from -1 to .999999. Because of the very large mass of the residual atom with respect to the mass of the electron, it can be assumed that the electron scatters without a change of energy, and there is no energy transfer to the residual atom.
- Bremsstrahlung is represented using two subsections. The electron is assumed to scatter straight ahead with an energy loss. This is described in ENDF-6 using the LAW=8 format (Cross Section Evaluation Working Group, 2018). The photon is usually assumed to be emitted isotropically with spectra given as tabulated distributions. Energy transfer to the residual atom is often ignored.
- Excitation occurs when the incident electron loses some of its energy by exciting the outer electrons of the atom to higher energy states. The energy transfer to the residual atom is stored in 17.4.1 The electron is assumed to continue in the straight-ahead direction.
- Ionisation is represented by using a separate `reaction` for each subshell. There are two electrons coming out of each ionisation reaction: the scattered electron and the recoil electron. Because these two particles are identical, it is arbitrarily assumed that the particle with the lower energy is the “recoil” electron, and the one with the higher energy is the “scattered” electron. If E_k is the binding energy for the sub-shell, the energy of the recoil electron varies from 0 to $(E - E_k)/2$, and the energy of the scattered electron varies from $(E - E_k)/2$ to $E - E_k$. Only the distribution for the “recoil” electron needs to be given. The user can select a recoil energy E_r from the distribution and then generate the corresponding scattered electron with energy $E - E_k - E_r$. The value of E_k is given by the Q-value of the reaction. It is assumed that both the scattered and the recoil electrons continue in the direction of the incident electron, and that no kinetic energy is transferred to the residual atom.
- The relaxation of the residual atom left after ionisation results in the emission of additional X-rays and electrons. Those spectra can be computed using the atomic relaxation data described in Chapter 28 of the ENDF-6 Formats Manual (Cross Section Evaluation Working Group, 2018).

24.2 Incoherent photon (X-ray) scattering

Photons are assumed to interact with the electron cloud of a target atom and the effect of the scattering off the electron cloud can be encapsulated in the incoherent scattering function and coherent scattering factor(s).

GNDS allows two alternatives for storing angular distribution data. One is by proba--bility per unit cos(θ) vs. cos(θ), and the other is by Legendre coefficients. Neither of these is a “natural” method for photons. An alternative way of presenting the photon scattering data is to tabulate incoherent scattering functions and form factors. Users could then provide processing codes to generate the cross sections from this informa--tion. The calculation is quite straightforward and allows the user to generate all the scattering data from a relatively small table of numbers. These are discussed below.

24.2.1 Atomic type: `incoherentPhotonScattering`

The double differential cross section for incoherent (a.k.a. inelastic) scattering is given by:

$$\frac{d^2\sigma_{\text{incoh}}(E,E',\mu)}{d\mu\; dE'} = \frac{d\sigma_{\text{incoh}}(E,\mu)}{d\mu}\;\delta(E'-E'(E,\mu)) \tag{24.1}$$

where $\delta(x)$ is the Dirac delta function, $E'(E,\mu)$ is the energy of the scattered photon (see definition of $k' \equiv k'(k,\mu)$ below), and the differential cross section $d\sigma_{\text{incoh}}(E,\mu)/d\mu$ is given by

$$\frac{d\sigma_{\text{incoh}}(E,\mu)}{d\mu} = S(x,Z)\frac{d\sigma_{KN}(E,\mu)}{d\mu} \tag{24.2}$$

here

$d\sigma_{KN}/d\mu$ the Klein-Nishina cross section (Hubbell et al., 1975; Klein et al., 1929) which can be written in a closed form as

$$\frac{d\sigma_{KN}(E,E',\mu)}{d\mu} = \pi r_e^2\left(\frac{k'}{k}\right)^2\left[1+\mu^2+kk'(1-\mu)^2\right] \tag{24.3}$$

$S(x,Z)$ the incoherent scattering function. For large x, S approaches Z. For small x, $S(0,Z)=0$.

x a quantity related to the momentum of the recoil electron. x is an inverse length, given in inverse Ångströms as customarily reported in the literature.

$$x = \frac{E}{hc}\sin(\theta/2) = \frac{E}{hc}\left[\frac{(1-\mu)}{2}\right]^{1/2} \tag{24.4}$$

r_e the classical radius of the electron, e^2/m_ec^2.

k the incident photon energy, in units of the electron rest mass, E/m_ec^2

k' the scattered photon energy, in units of the electron rest mass, $E'/m_ec^2 = k/(1+k(1-\mu))$

E' scattered photon energy,

μ $\cos\theta$ of the scattered photon.

Using 24.2, the total cross section and angular distribution can then easily be calculated. Values of $S(x,Z)$ are tabulated as a function of x. The user presumably will have subroutines available for calculating x for energies and angles of interest and for calculating Klein-Nishina cross sections. The user will then generate the cross sections for the appropriate cases by calculating x's, looking up the appropriate values of S, and substituting them in the above formula. The quantity x is related to the the momentum of the recoil electron q. In units of $m_e c$, the recoil momentum is

$$q = k' \left[1 + \left(\frac{k}{k'}\right)^2 - 2\mu\left(\frac{k}{k'}\right)\right]^{1/2}. \tag{24.5}$$

In the limit of coherent (elastic) scattering, $E = E'$ and θ is small so one has

$$q = k\left[2(1-\mu)\right]^{1/2} = 2hx/m_e c. \tag{24.6}$$

Specifications for `incoherentPhotonScattering`

Node name: `incoherentPhotonScattering`
Attributes: The list of additional allowed attributes is:

- **`href`** [`bodyText`, optional] When `incoherentPhotonScattering` is used in place of an actual distribution this link is used to refer back to the parameterisation that defines the distribution.
- **`label`** [`XMLName`, optional] Style label for this cross section.
- **`pid`** [`XMLName`, optional] The particle identifier of the outgoing particle.
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- **`scatteringFactor`:** [optional, when present, must appear one time] $S(x, Z)$, the incoherent scattering function.

XML Example(s) of `incoherentPhotonScattering`

```
<incoherentPhotonScattering
        href="..."
        label="..."
        pid="..."
        productFrame="...">
    <scatteringFactor>...</scatteringFactor></incoherentPhotonScattering>
```

24.2.2 Atomic type: `scatteringFactor`

$S(x, Z)$, the incoherent scattering function. For large x, S approaches Z. For small x, $S(0, Z) = 0$.

Specifications for `scatteringFactor`

Node name: `scatteringFactor`
Attributes: This element has no attributes

Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`:** [optional, when present, must appear one time] The incoherent scattering function given as a function of incident energy as an interpolation table.
- **`regions1d`:** [optional, when present, must appear one time] The incoherent scat--tering function given as multiple regions.

XML Example(s) of `scatteringFactor`

```
<scatteringFactor>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></scatteringFactor>
```

24.3 Coherent photon (X-ray) scattering

24.3.1 Atomic type: `coherentPhotonScattering`

The double differential cross section for coherent (a.k.a. elastic) scattering is given by (Hubbell et al., 1975; Kissel et al., 1995):

$$\frac{d^2\sigma_{\text{coh}}(E,E',\mu)}{d\mu\, dE'} = \frac{d\sigma_{\text{coh}}(E,\mu)}{d\mu}\,\delta(E'-E). \tag{24.7}$$

Here $\delta(x)$ is the Dirac delta function and the differential cross section $d\sigma_{\text{coh}}(E,\mu)/d\mu$ is given by

$$\frac{d\sigma_{\text{coh}}(E,\mu)}{d\mu} = \frac{d\sigma_T(E)}{d\mu}\left\{\left[F(x,Z)+F'(E)\right]^2 + F''(E)^2\right\} \tag{24.8}$$

where

$d\sigma_T/d\mu$ the Thompson cross section [2] which can be written in a closed form as

$$\frac{d\sigma_T(E)}{d\mu} = \pi r_e^2\left(1+\mu^2\right) \tag{24.9}$$

$F(x,Z)$ a form factor, which can be easily tabulated. At high x, F approaches zero. In the other limit $F(0,Z)$ tends to Z.,

$F'(E)$ the real part of the anomalous scattering factor.,

$F''(E)$ the imaginary part of the anomalous scattering factor.,

Using 24.8, the total cross section and angular distribution can then easily be calculated. The incoherent and coherent scattering data should always be presented as scatter--ing functions and form factors, respectively, whether or not data are included. The anomalous scattering factors are assumed to be isotropic. In addition, they smoothly approach zero at 1.0 MeV and can be assumed to be zero at higher energies, (Cullen, 1989; Cullen et al., 1997).

Specifications for `coherentPhotonScattering`

Node name: `coherentPhotonScattering`
Attributes: The list of additional allowed attributes is:

- **`href`** [`bodyText`, optional] When `coherentPhotonScattering` is used in place of an actual distribution this link is used to refer back to the parameterisation that defines the distribution.
- **`label`** [`XMLName`, optional] Style label for this cross section.
- **`pid`** [`XMLName`, optional] The particle identifier of the outgoing particle.
- **`productFrame`** [`frame`, **required**] The frame that the product data are defined in.

Child nodes: The list of additional allowed child nodes is:

- **`formFactor`**: [optional, when present, must appear one time] The form factor $F(x, Z)$.
- **`realAnomalousFactor`**: [optional, when present, must appear one time] $F'(E)$, the real part of the anomalous scattering factor.
- **`imaginaryAnomalousFactor`**: [optional, when present, must appear one time] $F''(E)$, the imaginary part of the anomalous scattering factor.

XML Example(s) of `coherentPhotonScattering`

```
<coherentPhotonScattering
        href="..."
        label="..."
        pid="..."
        productFrame="...">
    <formFactor>...</formFactor>
    <realAnomalousFactor>...</realAnomalousFactor>
    <imaginaryAnomalousFactor>...</imaginaryAnomalousFactor>
</coherentPhotonScattering>
```

24.3.2 Atomic type: `formFactor`

A form factor $F(x, Z)$, which can be tabulated. At high x, F approaches zero. In the other limit $F(0, Z)$ tends to Z.

Specifications for `formFactor`

Node name: `formFactor`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`**: [optional, when present, must appear one time] The form factor given as a function of incident energy as an interpolation table.
- **`regions1d`**: [optional, when present, must appear one time] The form factor given as multiple regions.

XML Example(s) of `formFactor`

```
<formFactor>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></formFactor>
```

24.3.3 Atomic type: `realAnomalousFactor`

$F'(E)$, the real part of the anomalous scattering factor.

Specifications for `realAnomalousFactor`

Node name: `realAnomalousFactor`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`:** [optional, when present, must appear one time] The real part of the anoma--lous scattering factor given as an interpolation table of F'' vs. E
- **`regions1d`:** [optional, when present, must appear one time] The real part of the anomalous scattering factor given as several regions.

XML Example(s) of `realAnomalousFactor`

```
<realAnomalousFactor>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></realAnomalousFactor>
```

24.3.4 Atomic type: `imaginaryAnomalousFactor`

$F''(E)$, the imaginary part of the anomalous scattering factor.

Specifications for `imaginaryAnomalousFactor`

Node name: `imaginaryAnomalousFactor`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`XYs1d`:** [optional, when present, must appear one time] The imaginary part of the anomalous scattering factor given as an interpolation table of F'' vs. E
- **`regions1d`:** [optional, when present, must appear one time] The imaginary part of the anomalous scattering factor given as several regions

XML Example(s) of `imaginaryAnomalousFactor`

```
<imaginaryAnomalousFactor>
    <XYs1d>...</XYs1d>
    <regions1d>...</regions1d></imaginaryAnomalousFactor>
```

25. Covariance data

25.1 The `covarianceSuite` root node

For many applications, a nuclear data evaluation is viewed as incomplete unless it includes estimates of the uncertainties on values and the covariances between those values. Covariance matrices in nuclear data give correlated uncertainties for quantities like the cross section and average multiplicity (where the correlation is between different incident energies), outgoing product energies (where the correlation is between outgoing energies), and for model parameters. Evaluations may also include 'cross-terms' that link two different quantities (e.g. that link the cross sections for two different reactions or for reactions on different targets).

GNDS supports storing covariance information together with the corresponding data container. For example, a point-wise cross section may contain a `covariance` in addition to its list of X-Y pairs. That covariance may contain a full matrix, or it may be a link to another `covariance` in a separate file, the `covarianceSuite`. Using the `covarianceSuite` to store covariances externally helps reduce size and file access time for `reactionSuite` files, and also helps support storing cross-covariances between different quantities in a transparent way. Inside the `covarianceSuite`, covariances are organised into `section` nodes. Each `section` corresponds to either a single quantity (e.g. reaction cross section) or to the combination of two quantities (i.e. a cross-term between two different reaction cross sections). The `covarianceSuite` also contains `styles` and an `externalFiles` list telling what other `reactionSuite` and possible `covarianceSuite` files it refers to.

25.1.1 Covariance type: `covarianceSuite`

Stores covariance data for various reaction and materials. Covariance matrix data my be given for the combination of a projectile and target given, as well as for any other combination. Therefore, one or more `reactionSuite` elements and other related files may be listed in the `externalFiles` link.

Specifications for `covarianceSuite`

Node name: `covarianceSuite`
Root node: This node may be the root node of a GNDS file
Attributes: The list of additional allowed attributes is:

- **`evaluation`** [`XMLName`, optional] Name of the evaluation, e.g. 'ENDF-VIII.0'
- **`projectile`** [`XMLName`, optional] Projectile particle id for the most of the covariances matrices in this suite.
- **`target`** [`XMLName`, optional] Target particle id for most of the covariance matrices in this suite.
- **`interaction`** [`interaction`, **required**] Defines the type of interaction for the reac--tion data linked from within the covarianceSuite. Options are 'nuclear', 'atomic' and 'thermalNeutronScatteringLaw'. For example, a covarianceSuite containing covariances for photo-nuclear reactions has a value of 'nuclear' while a covarianceSuite for photo-atomic reactions has a value of 'atomic'. In principal a covarianceSuite could contain cross-terms between two different types of interaction, but such cross-terms are not present in any current libraries. If the need arises, they could be handled by allowing the interaction to be 'mixed'.
- **`version`** [`Float64`, optional] GNDS format version, e.g. '2.1'

Child nodes: The list of additional allowed child nodes is:

- **`externalFiles`:** [optional, when present, must appear one time] Stores a list of ex--ternal files related to this `covarianceSuite`. Often used to link a `reactionSuite` to one or more `covarianceSuite` files.
- **`styles`:** [optional, when present, must appear one time] Element containing a list of styles inside this covarinceSuite. Each style describes information about the evalulated data (e.g. library, version) representation and each processed data representation.
- **`covarianceSections`:** [optional, when present, must appear one time] Stores co--variance matrices for continuous data.
- **`parameterCovariances`:** [optional, when present, must appear one time] contains covariance for paramters. This may be resoanance parameters or covariance matrices for other nuclear data described by parameters

XML Example(s) of `covarianceSuite`

```
<covarianceSuite
        evaluation="..."
        projectile="..."
        target="..."
        interaction="..."
        version="...">
    <externalFiles>...</externalFiles>
    <styles>...</styles>
    <covarianceSections>...</covarianceSections>
    <parameterCovariances>...</parameterCovariances></covarianceSuite>
```

25.2 General covariance containers

25.2.1 Covariance type: `covarianceSections`

Stores covariance matrices for continuous data. The grid on which the covariance data are given will usually be different from the one the orignal data are given on. Therefore the axes data will most often be given as `grid` elements. This element can contain covariance matrices and cross covariance matrices for many different types of data.

Specifications for `covarianceSections`

Node name: `covarianceSections`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`covarianceSection`**: [optional, when present, must appear at least one time] Each covarianceSection represents either the covariance for a single quantity, or a cross-term between two different quantities.

XML Example(s) of `covarianceSections`

```
<covarianceSections>
    <covarianceSection>...</covarianceSection></covarianceSections>
```

25.2.2 Covariance type: `covarianceSection`

Within the `covarianceSections`, most covariance data are organised into covariance--Section nodes. Each covarianceSection represents either the covariance for a single quantity, or a cross-term between two different quantities.

Specifications for `covarianceSection`

Node name: `covarianceSection`
Attributes: The list of additional allowed attributes is:

- **`crossTerm`** [`Boolean`, optional, default is "false"] Indicates whether the covariance is a cross term connecting two different quantities.
- **`label`** [`XMLName`, optional] The label must be unique among all `covarianceSection` elements in a `crossSections` element. No other restrictions are placed on the label.

Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):

`rowData`: [**required**, must appear one time] defines what data lie along the rows of the covariance

`columnData`: [optional, when present, must appear one time] Defines what data lie along the columns of the covariance. Required if crossTerm is true. Otherwise it is assumed to be the same as rowData.

`covarianceMatrix`: [optional, when present, * one of marked children must appear one time] Contains the covariance data if only one `covarianceMatrix` element is needed to describe the covariance data for this covarianceSection and the desired style.

`sum`: [optional, when present, * one of marked children must appear one time] The data references by this covariance matrix are given as a sum of other data.

`mixed`: [optional, when present, * one of marked children must appear one time] Contains more than one covariance matrix. All matrices need to be added in order to get the full covariance matrix.

XML Example(s) of `covarianceSection`

```
<covarianceSection
        crossTerm="..."
        label="...">
    <rowData>...</rowData>
    <columnData>...</columnData>
    <covarianceMatrix>...</covarianceMatrix>
    <sum>...</sum>
    <mixed>...</mixed></covarianceSection>
```

25.2.3 Covariance type: `rowData`

The rowData contains a link to the quantity (cross section, multiplicity, etc.) corre--sponding to the rows of the covariance matrix. One or more `slice` child nodes are necessary when a higher-dimensional covariance such as for the outgoing energy spectrum $P(E'|E)$ is divided into multiple `covarianceSection` nodes, each of which applies to part of the incident energy domain. Each `slice` must contain a `dimension` attribute, along with an optional `domainUnit` and either a `domainValue` or both `domainMin` and `domainMax`.

Specifications for `rowData`

Node name: `rowData`
Attributes: The list of additional allowed attributes is:

`ENDF_MFMT` [`XMLName`, optional] the ENDF MF and MT numbers, stored as a comma-joined list (i.e. “MF,MT”)

`href` [`XMLName`, optional] An xPath expression linking to the covariant quantity. The link may contain a reference to an external file referenced in `externalFiles`.

For some covariance matrices, such as a `parameterCovariance` this attribute may be omitted, as more explicit references are given for the parameters.

dimension [`Integer32`, optional] If the href values is refering to a multi-dimensional object, (`array` or `regions2d` for example), the dimension to which this row refers. The number should be the same index as the axis, for which this dimension is indexed.

Child nodes: The list of additional allowed child nodes is:

slices: [optional, when present, must appear one time] If the href values is refer--ing to a multi-dimensional object, list of slices along the fixed dimensions.

XML Example(s) of `rowData`

```
<rowData
        ENDF_MFMT="..."
        href="..."
        dimension="...">
    <slices>...</slices></rowData>
```

25.2.4 Covariance type: `columnData`

The columnData contains a link to the quantity (cross section, multiplicity, etc.) cor--responding to the columns of the covariance matrix. The element `columnData` is only needed if this is a cross term connecting two different quantities.

Specifications for `columnData`

Node name: `columnData`

Attributes: The list of additional allowed attributes is:

ENDF_MFMT [`XMLName`, optional] Same as for `rowData`.

href [`XMLName`, optional] Same as for `rowData`.

dimension [`Integer32`, optional] Same as for `rowData`.

Child nodes: The list of additional allowed child nodes is:

slices: [optional, when present, must appear one time] If the href values is refer--ing to a multi-dimensional object, list of slices along the fixed dimensions.

XML Example(s) of `columnData`

```
<columnData
        ENDF_MFMT="..."
        href="..."
        dimension="...">
    <slices>...</slices></columnData>
```

25.2.5 Covariance type: `slices`

List of `slice` nodes, one for each fixed dimension of a covariance for a multi-dimensional function.

Specifications for `slices`

Node name: `slices`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `slice`: [**required**, must appear at least one time]

XML Example(s) of `slices`

```
<slices>
    <slice>...</slice></slices>
```

25.2.6 Covariance type: `slice`

If giving a covariance matrix for a multidimensional object, each `slice` describes one dimension that is kept constant, along with the value or range of values along that dimension to which the covariance applies.

Specifications for `slice`

Node name: `slice`
Attributes: The list of additional allowed attributes is:
 `domainMin` [`Float64`, optional] The upper end of the domain
 `domainMax` [`Float64`, optional] The lower end of the domain.
 `domainValue` [`Float64`, optional] Value of the domain (e.g. Legendre coefficient for an angular distribution covariance). This attribute should be used instead of `domainMin` and `domainMax` if the covariance applies to a single point in the given domain.
 `domainUnit` [`XMLName`, optional] The unit for `domainMin`, `domainMax` or `domainValue`.
 `dimension` [`Integer32`, **required**] The fixed dimension to which this dimension object refers. The number should be the same index as the axis, for which this dimension is indexed
Child nodes: This element has no child nodes

XML Example(s) of `slice`

```
<slice
        domainMin="..."
        domainMax="..."
        domainValue="..."
        domainUnit="..."
        dimension="...">
</slice>
```

25.2.7 Multi-dimensional covariance section example

The following example illustrates how a multi-dimensional covariance matrix (e.g. for a prompt fission neutron spectrum or for S$_{(\alpha,\beta,T)}$) can be decomposed into multiple sections:

```
<section label="S_ab matrix 0">
  <rowData ENDF_MFMT="37,4" dimension="1" href="...">
    <slices>
      <!-- define temperature range -->
      <slice dimension="3" domainMin="283.6" domainMax="350" domainUnit="K"/>
      <!-- define beta range -->
      <slice dimension="2" domainMin="0" domainMax="1e-5" domainUnit=""/>
    </slices>
  </rowData>
  <covarianceMatrix>
    ... <!-- (covariance for S(alpha,alpha') at given T and beta) -->
  </covarianceMatrix>
</section>
...
<section label="S_ab cross term 0" crossTerm="true">
  <rowData ENDF_MFMT="37,4" dimension="1" href="...">
    <slices>
      <slice dimension="3" domainMin="283.6" domainMax="350" domainUnit="K"/>
      <slice dimension="2" domainMin="0" domainMax="1e-5" domainUnit=""/>
    </slices>
  </rowData>
  <!-- columnData defines a different range of beta: -->
  <columnData ENDF_MFMT="37,4" dimension="1" href="...">
    <slices>
      <slice dimension="3" domainMin="283.6" domainMax="350" domainUnit="K"/>
      <slice dimension="2" domainMin="1e-5" domainMax="5e-5" domainUnit=""/>
    </slices>
  </rowData>
  <covarianceMatrix> ... </covarianceMatrix>
</section>
```

As this example demonstrates, a multi-dimensional covariance function can be decomposed into many 2-dimensional sections where each section corresponds to a fixed range of all higher dimensions. This may require defining a large number of sections and could become very verbose compared to storing the covariance in a multi-dimensional array. However, it offers a potential advantage in that different independent variable grids can be chosen for each section.

25.2.8 Covariance type: `covarianceMatrix`

A covariance matrix can be either absolute or relative with the respect to the data it referes to. Choices are:

`absolute`: the covariance matrix data are given absolute.
`relative`: the covariance matrix data are given relative to the data it refers to.

Specifications for `covarianceMatrix`

Node name: `covarianceMatrix`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, optional] If `covarianceMatrix` is a direct child of `covarianceSection`, the label refers to the style of the evalutions this covariance matrix refers to. In this case, the label is required, otherwise if `covarianceMatrix` is a child of `mixed`, it is optional. In the latter case it may be used to ensure that each `covarianceMatrix` in `mixed` has a unique label by which it can be refered to in a link
- **`productFrame`** [`frame`, optional] The frame that the product data are defined in. Required for covariances on energy/angular distributions.
- **`type`** [`XMLName`, **required**] Options are 'relative' or 'absolute'

Child nodes: The list of additional allowed child nodes is:

- **`gridded2d`:** [optional, when present, * one of marked children must appear one time] A two-dimensional array that gives the actual covariance data.
- **`sandwichProduct`:** [optional, when present, * one of marked children must appear one time] Covariance matrix decomposed into a matrix sandwich product.

XML Example(s) of `covarianceMatrix`

```
<covarianceMatrix
        label="..."
        productFrame="..."
        type="...">
    <gridded2d>...</gridded2d>
    <sandwichProduct>...</sandwichProduct></covarianceMatrix>
```

25.2.9 Covariance type: `sandwichProduct`

Contains a matrix decomposed into the sandwich product $R \times c \times C$. In many cases C is equal to the transpose of R and need not be stored explicitly. This representation can be used to store covariances more compactly. For example, if eigenvalue decomposition reveals that only a few eigenvalues are significant, this representation can be used to store only those eigenvalues and their corresponding eigenvectors. If a covariance was computed by varying some parameters and determining the sensitivity of a final quan--tity to each parameter, this representation permits storing the parameter covariance and sensitivity explicitly.

Specifications for `sandwichProduct`

Node name: `sandwichProduct`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`axes`**: [**required**, must appear one time] An `axes` element containing the `grid` elements. The grid applies to the final matrix after computing the matrix product
- **`covariance`**: [**required**, must appear one time] Array containing the inner portion of the sandwich product.
- **`rowSensitivity`**: [**required**, must appear one time] Array containing the left-hand side of the sandwich product.
- **`columnSensitivity`**: [optional, when present, must appear one time] Array con--taining the right-hand side of the sandwich product. May be omitted if the right-hand side is equal to the transpose of the `rowSensitivity`. If a columnSensitivity is supplied, its dimensions must be consistent with the transpose of the rowSensitivity.

XML Example(s) of `sandwichProduct`

```
<sandwichProduct>
    <axes>...</axes>
    <covariance>...</covariance>
    <rowSensitivity>...</rowSensitivity>
    <columnSensitivity>...</columnSensitivity></sandwichProduct>
```

25.2.10 Covariance type: `covariance`

Specifications for `covariance`

Node name: `covariance`
Attributes: This element has no attributes

Child nodes: The list of additional allowed child nodes is:

`array`: [**required**, must appear one time] Array storing the central term in a sand--wich product. The array must be square and may be symmetric or diagonal.

XML Example(s) of `covariance`

```
<covariance>
    <array>...</array></covariance>
```

25.2.11 Covariance type: `rowSensitivity`

Specifications for `rowSensitivity`

Node name: `rowSensitivity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`array`: [**required**, must appear one time] Array storing the left-hand term in a sandwich product. The array is generally rectangular, with the number of columns equal to the dimension of the corresponding `covariance` array.

XML Example(s) of `rowSensitivity`

```
<rowSensitivity>
    <array>...</array></rowSensitivity>
```

25.2.12 Covariance type: `columnSensitivity`

Required only if the right-hand term in the sandwich product is not equal to the transpose of the left-hand term.

Specifications for `columnSensitivity`

Node name: `columnSensitivity`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

`array`: [**required**, must appear one time] Array storing the right-hand term in a sandwich product. The array is generally rectangular, with the number of rows equal to the dimension of the corresponding `covariance` array.

XML Example(s) of `columnSensitivity`

```
<columnSensitivity>
    <array>...</array></columnSensitivity>
```

25.2.13 Covariance type: `sum`

If, in a given range, the underlying data can be described by a sum over a different set of data, then the covariance matrix can be calculated from the covariance information of this set of data. This implies cross correlation between these different sets of data.

Specifications for `sum`

Node name: `sum`
Attributes: The list of additional allowed attributes is:

- **`domainMin`** [`Float64`, **required**] The lower end of the domain the sum applies to.
- **`domainMax`** [`Float64`, **required**] The upper end of the domain the sum applies to.
- **`domainUnit`** [`XMLName`, **required**] The unit for `domainMin` and exttdomainMax
- **`label`** [`XMLName`, optional] If `sum` is a direct child of `covarianceSection`, the label refers to the style of the evalutions this sum relationship refers to. In this case, the label is required, otherwise if `sum` is a child of `mixed`, it is optional. In the latter case it may be used to ensure that each `sum` in `mixed` has a unique label by which it can refered to in a link

Child nodes: The list of additional allowed child nodes is:

- **`summand`:** [optional, when present, must appear at least one time] Identifies one of the summands to calculate the covariance data.

XML Example(s) of `sum`

```
<sum
        domainMin="..."
        domainMax="..."
        domainUnit="..."
        label="...">
    <summand>...</summand></sum>
```

25.2.14 Covariance type: `summand`

Indicates how the constituents for this covariance matrix data are to be summed.

Specifications for `summand`

Node name: `summand`
Attributes: The list of additional allowed attributes is:

- **`ENDF_MFMT`** [`XMLName`, optional]
- **`coefficient`** [`Float64`, optional] The coefficient by which to weigh this reaction data in this range to calculate the final covariance data.
- **`href`** [`bodyText`, optional]

Child nodes: This element has no child nodes

XML Example(s) of `summand`

```
<summand
        ENDF_MFMT="..."
        coefficient="..."
        href="...">
</summand>
```

25.2.15 Covariance type: `mixed`

If more than one explicit covariance matrix, short term scaling data or different sum relationships are needed to describe the covariance information, the data needs to be put into a `mixed` container. All child elements of `mixed` can be processed separately but need to be added together to yield the covariance matrix for the covarianceSection.

Specifications for `mixed`

Node name: `mixed`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] The label refers to the style of the evalutions this covari--ance matrix refers to and it is required.

Child nodes: The list of additional allowed child nodes is:

- **`covarianceMatrix`:** [optional, when present, must appear at least one time] Partial explicit covariance matrix
- **`shortRangeSelfScalingVariance`:** [optional, when present, must appear at least one time] Short range scaling data for this covariance data
- **`sum`:** [optional, when present, must appear at least one time] Sum relationship for the underlying data to be used in some range of the covariance domain.

XML Example(s) of `mixed`

```
<mixed
        label="...">
    <covarianceMatrix>...</covarianceMatrix>
    <shortRangeSelfScalingVariance>...</shortRangeSelfScalingVariance>
    <sum>...</sum></mixed>
```

25.2.16 Covariance type: `shortRangeSelfScalingVariance`

Gives a covariance matrix for a short-range self-scaling variance. The matrix will only have elements on the diagonal given on an evalutor grid ΔE_k with diagonal elements F_k. In order to calculate the covariance matrix on the user energy grid of ΔE_{kj} there are two options:

- The variance contribution $\text{Var}(X_{jj})$ to the processed group variance for the energy group (E_j, E_{j+1}) is inversely proportional to its width ΔE_j when (E_j, E_{j+1}) lies within (E_k, E_{k+1}) and is obtained from the relation:

  $$\text{Var}(X_{jj}) = F_k \frac{\Delta E_k}{\Delta E_j}$$

  where $E_k \leq E_j \leq E_{j+1} \leq E_{k+1}$. This form is applicable in the resonance range where the covariances in the other sub-subsections define "average" coarse energy-grid uncertainties, while the actual pointwise cross-section values may fluctuate by orders of magnitude. The evaluator must be aware that the actual uncertainty in the cross sections depends on the user's energy-grid. The user should be aware of possible processing problems, for example when the union grid of the user's energy group structure and the covariance grid nearly coincide at some energy. This form should not be used to specify the uncertainty on the actual value of pointwise cross sections.
- The variance contribution $\text{Var}(X_{jj})$ to the processed group variance for the energy group (E_j, E_{j+1}) is directly proportional to the width ΔE_j when (E_j, E_{j+1}) lies within (E_k, E_{k+1}). It is obtained from the relation:

  $$\text{Var}(X_{jj}) = F_k \left[1 - \frac{\Delta E_j}{\Delta E_k}\right]$$

  where $E_k \leq E_j < E_{j+1} \leq E_{k+1}$. This form is applicable in cases when experimental evidence suggests the possibility of structure in the cross sections, but the ex--perimental resolution is not sufficient to determine the detailed shape, which is then approximated by a smooth curve. It describes the maximum uncertainty in the cross section due to possible fine-structure, which vanishes when the user's energy grid is equal or coarser than the covariance grid. The physical consequence of this property is that it increases the absolute variance when the user defines an energy grid more refined than the covariance grid (thus avoiding zero-eigenvalue problems), but will not affect any coarse energy-group uncertainties. The increase in the variance remains finite (limited to F_k) and is applicable to defining the uncertainties of pointwise cross sections.

Note that the Var(X_{jj}) are variances in average cross sections. This rule suffices for arbitrary group boundaries if subgroup boundaries are chosen to include all the E_k. No contributions to off-diagonal multigroup covariance matrix elements are generated by this covariance section.

Specifications for `shortRangeSelfScalingVariance`

Node name: `shortRangeSelfScalingVariance`
Attributes: The list of additional allowed attributes is:
- **`dependenceOnProcessedGroupWidth`** [`XMLName`, optional]
- **`label`** [`XMLName`, optional] Only used to ensure that the `shortRangeSelfScalingVari-` `-anc` is unique within `mixed`. No other restrictions are put on the label.
- **`type`** [`XMLName`, optional] Options are ‘relative’ or ‘absolute’ to indicate whether the data are given absolute or relative with respect to the underlying data.

Child nodes: The list of additional allowed child nodes is:
- **`gridded2d`:** [optional, when present, must appear one time]

XML Example(s) of `shortRangeSelfScalingVariance`

```
<shortRangeSelfScalingVariance
        dependenceOnProcessedGroupWidth="..."
        label="..."
        type="...">
    <gridded2d>...</gridded2d></shortRangeSelfScalingVariance>
```

25.3 Covariances for model parameters

25.3.1 Covariance type: `parameterCovariances`

Specifications for `parameterCovariances`

Node name: `parameterCovariances`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
- **`averageParameterCovariance`:** [optional, when present, must appear at least one time] Gives the covariance matrix data on average parameters, for example average resonance parameters in the resolved range
- **`parameterCovariance`:** [optional, when present, must appear at least one time] Gives the covariance matrix for parameters, for example resonance parame- -ters in the resolved range.

XML Example(s) of `parameterCovariances`

```
<parameterCovariances>
    <averageParameterCovariance>...</averageParameterCovariance>
    <parameterCovariance>...</parameterCovariance></parameterCovariances>
```

25.3.2 Covariance type: `parameterCovariance`

Container for `parameterCovarianceMatrix` containing the actual parameter covariance matrix. Storing the data in the container, allows to give different parameter covariance matrices, depending on the evaluation.

Specifications for `parameterCovariance`

Node name: `parameterCovariance`
Attributes: The list of additional allowed attributes is:
  `label` [`XMLName`, optional] Used to make the `covariance`parameterCovariance unique within `parameterCovariances`. No other restrictions on the label exist.
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):
  `rowData`: [**required**, must appear one time] Gives a reference to the resonance parameters to which this matrix applies.
  `parameterCovarianceMatrix`: [**required**, must appear at least one time] Contains the parameter covariance matrix itself.

XML Example(s) of `parameterCovariance`

```
<parameterCovariance
        label="...">
    <rowData>...</rowData>
    <parameterCovarianceMatrix>...</parameterCovarianceMatrix>
</parameterCovariance>
```

25.3.3 Covariance type: `parameterCovariancMatrix`

The parameter covariance matrix can be either absolute or relative with the respect to the . Choices are:

`absolute`: the parameters covariance matrix data is given absolute.
`absoluteCovariance`: the parameters covariance matrix data is given absolute.
`relative`: the parameter covariance matrix data are given relative to the param- -eters.

Specifications for `parameterCovariancMatrix`

Node name: `parameterCovariancMatrix`
Attributes: The list of additional allowed attributes is:

- **`label`** [`XMLName`, **required**] The label refers to the style of the evalutions this covari--ance matrix refers to.
- **`type`** [`XMLName`, optional] Options ’absolute’, ’absoluteCovariance’, or ’relative’.

Child nodes: The list of additional allowed child nodes is:

- **`parameters`**: [**required**, must appear one time] Cross references the matrix indices to the parameter they refer to.
- **`array`**: [**required**, must appear one time] The two-dimensional array containing the actual parameter covariance data.

XML Example(s) of `parameterCovariancMatrix`

```
<parameterCovariancMatrix
        label="..."
        type="...">
    <parameters>...</parameters>
    <array>...</array></parameterCovariancMatrix>
```

25.3.4 Covariance type: `parameters`

Collects the elemlinkparameterLink elements that allow to link the indices in the parameter covariance matrix to the correct parameters.

Specifications for `parameters`

Node name: `parameters`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:

- **`parameterLink`**: [optional, when present, must appear at least one time] Links the indices in the parameter covariance matrix to the correct parameter.

XML Example(s) of `parameters`

```
<parameters>
    <parameterLink>...</parameterLink></parameters>
```

25.3.5 Covariance type: `parameterLink`

The reference given can be of several different types. Depending on the type, the numbering of the parameters is as follows:

> **`table:`** The table is counted row-major and all entries are counted. In this case `nParameters` should be equal to row count times column count. For example, the table object for `BreitWigner` resonance parameters lists L and J values, which are not usually referenced in the covariance matrix. However, the covariance matrix must list a zero entry for all terms involving L and J. This allows for easy referencing of an entire table object, rather than having to give an explicit reference to each individual resonance parameter.
> **`scatteringRadius:`** It is aassumed that the scattering radius contains only a `con-` `-stant1d` element for the selected evaluation. In this case `nParameters` should be 1
> Any references that references exactly one value, for example a `constant1d`

Specifications for `parameterLink`

Node name: `parameterLink`
Attributes: The list of additional allowed attributes is:
- **`href`** [`XMLName`, optional] Link to the data container (usually a table) where the parameters are defined.
- **`label`** [`XMLName`, optional] The link is used to make each `covariance`parameterLink unique within `parameters`. No other restrictions are put on the value.
- **`matrixStartIndex`** [`Integer32`, optional] The starting index for this set of parame- -ters in the covariance matrix given in the `array` element of the `parameterCo-` `-varianceMatrix`
- **`nParameters`** [`Integer32`, optional, default is "1"] The number of parameters that are refered to in this link

Child nodes: This element has no child nodes

XML Example(s) of `parameterLink`

```
<parameterLink
        href="..."
        label="..."
        matrixStartIndex="..."
        nParameters="...">
</parameterLink>
```

25.3.6 Covariance type: `averageParameterCovariance`

Covariance data are given for average parameters, as used in the unresolved resonance range. Even so the unresolved resonance parameters may be given as energy dependent values, the covariance data are given for energy indedependent, i.e. energy averaged, parameters.

Specifications for `averageParameterCovariance`

Node name: `averageParameterCovariance`
Attributes: The list of additional allowed attributes is:
  `crossTerm` [`Boolean`, optional, default is "false"]
  `label` [`XMLName`, optional]
Child nodes: The list of additional allowed child nodes is (only one of children marked with * for each unique style label is allowed):
  `columnData`: [optional, when present, must appear one time]
  `covarianceMatrix`: [optional, when present, * one of marked children must appear one time]
  `rowData`: [optional, when present, must appear one time]

XML Example(s) of `averageParameterCovariance`

```
<averageParameterCovariance
        crossTerm="..."
        label="...">
    <columnData>...</columnData>
    <covarianceMatrix>...</covarianceMatrix>
    <rowData>...</rowData></averageParameterCovariance>
```

26. Application-specific data

26.1 Denoting site specific data

The `applicationData` and related elements were added to allow an institution to try out new things, providing a workspace for new format development, or to store content which is not supported by the current format specifications.

26.1.1 Application data type: `applicationData`

Specifications for `applicationData`

Node name: `applicationData`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
 `institution`: [optional, when present, must appear one time]

XML Example(s) of `applicationData`

```
<applicationData>
    <institution>...</institution></applicationData>
```

26.1.2 Application data type: `institution`

Specifications for `institution`

Node name: `institution`
Attributes: The list of additional allowed attributes is:
 `label` [`XMLName`, optional]
Child nodes: The list of additional allowed child nodes is:
 `ENDFconversionFlags`: [optional, when present, must appear one time]

XML Example(s) of `institution`

```
<institution
        label="...">
    <ENDFconversionFlags>...</ENDFconversionFlags></institution>
```

26.2 Backwards compatibility with ENDF

Currently the main use of the `applicationData` section is to store 'compatibility flags' to support better translations from GNDS back to ENDF-6. The ENDF-6 format is not as flexible and expressive as GNDS, and in some cases it has more than one way of storing data. Therefore translation between the two languages often requires "help" deciding where to store the data in ENDF-6 (useful for minimising differences between original and re-translated ENDF-6 files). The `ENDFconversionFlags` markup is a mechanism to provide this help.

26.2.1 Application data type: `conversion`

Specifications for `conversion`

Node name: `conversion`
Attributes: The list of additional allowed attributes is:
  `flags` [`XMLName`, optional]
  `href` [`bodyText`, optional]
Child nodes: This element has no child nodes

XML Example(s) of `conversion`

```
<conversion
        flags="..."
        href="..."></conversion>
```

26.2.2 Application data type: `ENDFconversionFlags`

Specifications for `ENDFconversionFlags`

Node name: `ENDFconversionFlags`
Attributes: This element has no attributes
Child nodes: The list of additional allowed child nodes is:
  `conversion`: [optional, when present, must appear one time]

XML Example(s) of `ENDFconversionFlags`

```
<ENDFconversionFlags>
    <conversion>...</conversion></ENDFconversionFlags>
```

27. Map files

27.1 Collecting evaluations into a library using a map file

The GNDS specification includes a method for summarising a library of evaluations. This is accomplished using a ‘map file’, somewhat similar to the well-known ‘xsdir’ files used by the MCNP code. Like the GNDS `reactionSuite` nodes, map files are primarily stored in XML although they could be written out to other hierarchical forms as well.

Map files store a checksum for each protare included in the map, as well as for other map files. These checksums are intended to help detect changes in any of the protares making up a library. Checksums on protares are computed for each protare file. Checksums on map files are computed differently. Since adding a checksum to the file would change the checksum, the sum is instead computed by concatenating all checksums in the map together in the order that they appear in the map file, and computing the checksum on the resulting string. In addition to a `checksum`, each map file must also supply a `algorithm` telling how the sum was computed (e.g. message-digest algorithm 5 [MD5] or secure hash algorithm 1 [SHA1]). The `algorithm` is optional for nodes inside the map file. If no `algorithm` is supplied, then the map file `algorithm` is used.

The structure of a map file is very simple. It only has 3 types of sub-nodes. They are:

- protare: Specifies a GNDS file by listing the projectile ID, target ID, evaluation string, interaction, a path to a GNDS file and the checksum for that file.
- TNSL: Like the ‘protare’ sub-node but used for TNSL files. In addition to projectile ID, target ID, evaluation string, path and checksum, it adds standardTarget and standardEvaluation. The GNDS file pointed to by the path only has the TNSL data. standardTarget and standardEvaluation, along with an implicit ‘projectile=“n”’, specify the information needed to access the GNDS for the data needed when the projectile energy or target temperature are outside the domain of the TNSL file.
- import: Specifies a path to another map file along with the checksum for that map file (they can be nested to any level).

All three sub-nodes can be mixed and matched in any order inside a map file.

27.1.1 Map files type: `map`

A map file organises multiple files into a library. Individual files are called ‘protares’: PROjectile + TARget + Evaluation.

Specifications for `map`

Node name: `map`
Root node: This node may be the root node of a GNDS file
Attributes: The list of additional allowed attributes is:

- **`library`** [`XMLName`, **required**] String identifier for the library defined in this map file, e.g. ’ENDF/B-VIII.0’.
- **`format`** [`XMLName`, **required**] Map file format version id.
- **`checksum`** [`checksum`, **required**] Checksum for the map file, computed by string-concatenating all checksums inside the file (in the order they appear) and computing the checksum of the resulting string.
- **`algorithm`** [`algorithm`, **required**] Algorithm used to compute the checksum.

Child nodes: The list of additional allowed child nodes is:

- **`import`:** [optional, when present, may appear any number of times] Imports an--other map file using a path (relative or absolute).
- **`protare`:** [optional, when present, may appear any number of times] Adds one evaluation to the map file, using relative or absolute path.
- **`TNSL`:** [optional, when present, may appear any number of times] Adds a thermal neutron scattering law evaluation to the map file, using relative or absolute paths.

XML Example(s) of `map`

```xml
<map
        library="..."
        format="..."
        checksum="..."
        algorithm="...">
    <import>...</import>
    <protare>...</protare>
    <TNSL>...</TNSL></map>
```

27.1.2 Map files type: `import`

Imports contents of another map file into this map file. Care must be taken to avoid circular imports.

Specifications for `import`

Node name: `import`
Attributes: The list of additional allowed attributes is:

- **`path`** [`XMLName`, **required**] Path to another map file imported by this file.
- **`checksum`** [`checksum`, **required**] Checksum of the imported map file, identical to the `checksum` attribute in the top node of that file.
- **`algorithm`** [`algorithm`, optional] Algorithm used to compute the checksum. De--faults to the `algorithm` of the parent `map` file.

Child nodes: This element has no child nodes

XML Example(s) of `import`

```
<import
        path="..."
        checksum="..."
        algorithm="...">
</import>
```

27.1.3 Map files type: `protare`

Adds a single reactionSuite (or 'ProTarE') to the map file.

Specifications for `protare`

Node name: `protare`
Attributes: The list of additional allowed attributes is:

- **`projectile`** [`XMLName`, **required**] Particle id of the projectile.
- **`target`** [`XMLName`, **required**] Particle id of the target.
- **`evaluation`** [`XMLName`, **required**] Evaluation label, usually equal to the library name e.g. 'ENDF/B-8.0'.
- **`path`** [`XMLName`, **required**] Path (absolute or relative) to the reactionSuite associated with this protare.
- **`interaction`** [`XMLName`, **required**] Type of interaction. Options are: 'nuclear' and 'atomic'.
- **`checksum`** [`checksum`, **required**] Checksum for the protare file located at the given `path`.
- **`algorithm`** [`algorithm`, optional] Algorithm used to compute the checksum. De--faults to the `algorithm` of the parent `map` file.

Child nodes: This element has no child nodes

XML Example(s) of `protare`

```
<protare
        projectile="..."
        target="..."
        evaluation="..."
        path="..."
        interaction="..."
        checksum="..."
        algorithm="...">
</protare>
```

27.1.4 Map files type: `TNSL`

A thermal neutron scattering law (TNSL) target to add to the map file. Requires also listing the ‘standard’ protare to be used outside the limits of the TNSL data.

Specifications for `TNSL`

Node name: `TNSL`
Attributes: The list of additional allowed attributes is:

- **`projectile`** [`XMLName`, **required**] Particle id of the projectile.
- **`target`** [`XMLName`, **required**] Particle id of the target.
- **`evaluation`** [`XMLName`, **required**] Evaluation label, usually equal to the library name e.g. ‘ENDF/B-8.0‘.
- **`path`** [`XMLName`, **required**] Path (absolute or relative) to the reactionSuite associated with this protare.
- **`checksum`** [`checksum`, **required**] Checksum for the TNSL file located at the given `path`.
- **`algorithm`** [`algorithm`, optional] Algorithm used to compute the checksum. De--faults to the `algorithm` of the parent `map` file.
- **`standardTarget`** [`XMLName`, **required**] Particle id of the protare to use for reactions / incident energies outside the range of this TNSL target.
- **`standardEvaluation`** [`XMLName`, **required**] Evaluation label for the protare to use for reactions / incident energies outside the range of this TNSL target.

Child nodes: This element has no child nodes

XML Example(s) of `TNSL`

```
<TNSL
        projectile="..."
        target="..."
        evaluation="..."
        path="..."
        checksum="..."
        algorithm="..."
        standardTarget="..."
        standardEvaluation="...">
</TNSL>
```

27.2 Example map files

This example only has import sub-nodes:

```
<map library="MG_MC_imports" format="0.2"
        checksum="c9970a819ccb6663f70d428983a809a3d5f7ef4a" algorithm="sha1">
  <import path="neutrons/all.map"
        checksum="da39a3ee5e6b4b0d3255bfef95601890afd8070"/>
  <import path="protons/all.map"
        checksum="84a516841ba77a5b4648de2cd0dfcb30ea46dbb4"/>
  <import path="photo-nuclear/all.map"
        checksum="1b30468cc7a3aaf9c0b86af09191d0088e4cd4a5"/>
  <import path="photo-atomic/all.map"
        checksum="c17faf38ae944a80ed979c8b3c601f98d4d12f17"/>
</map>
```

The next example (from the neutrons/all.map in the General Interaction Data Interface [GIDI] test directory) only has protare and TNSL sub-nodes:

```
<map library="neutrons" format="0.2"
        checksum="c180fbca8661de7e053ddbe1a7101ac371e79428" algorithm="sha1">
  <protare projectile="n" target="H1" evaluation="ENDF/B-8.0"
        path="n-001_H_001.xml" interaction="nuclear"
        checksum="32518f6f6ec82c801531ae120618bd99d3589304"/>
  <TNSL projectile="n" target="HinCH2" evaluation="ENDF/B-8.0"
        path="tsl-HinCH2.xml"
        checksum="dea956df66b3cef44d690efba983d537dca0a997"
        standardTarget="H1" standardEvaluation="ENDF/B-8.0"/>
  <protare projectile="n" target="O16" evaluation="ENDF/B-7.1"
        path="n-008_O_016.xml" interaction="nuclear"
        checksum="ff118f6851181c244e494921c6e56101dabf6200"/>
  <protare projectile="n" target="Al27" evaluation="ENDF/B-8.0"
        path="n-013_Al_027.xml" interaction="nuclear"
        checksum="c3bd9de97a51a5a7c05489a3b5134bd02719942d"/>
  <TNSL projectile="n" target="tnsl-Al27" evaluation="ENDF/B-8.0"
        path="tsl-013_Al_027.xml"
        checksum="738572faa8fb763418a19b1a20d84165b02bdbe4"
        standardTarget="Al27" standardEvaluation="ENDF/B-8.0"/>
  <protare projectile="n" target="Fe56" evaluation="ENDF/B-7.1"
        path="n-026_Fe_056.xml" interaction="nuclear"
        checksum="c3ab2d67e59d5a8bb5820cafe67a0b8d498f0870"/>
  <protare projectile="n" target="Th227" evaluation="JENDL-7.1"
        path="n-090_Th_227.xml" interaction="nuclear"
        checksum="1dca182331b2d482d2dfa6d7142f1a36fc833dd9"/>
</map>
```

The final example contains one protare and an import. This is useful for changing a single evaluation within a library, for example when using random 'realizations' (i.e. GNDS files with modifications for UQ studies). In this case, the file pointed to by the protare sub-node (representing 'n + Pu239') is a modification of the one found in the imported map file. When using GIDI or the For Updating Data and Generating Evaluations (FUDGE) codes, if one specifies a 'projectile' of 'n' and 'target' of 'Pu239' without an 'evaluation' specified, then the first match is returned. In this example, if one specifies 'n + O16', it will be found in the imported map file and returned. The example also shows how to override the checksum `algorithm` for a node in the map file.

```
<map library="ENDF/V-III.0 realization_2.0 nubar_0.0" format="0.1"
        checksum="cdf4ceeccf2e87501b66ee501413f2ee256ef021" algorithm="sha1">
  <protare projectile="n" target="Pu239"
        evaluation="ENDF/V-III.0 realization_2.0 nubar_0.0"
        path="n-094_Pu_239.xml.proc"
        checksum="6656018ac96d0295d887ff2893ba9af66ae16af2"/>
  <import path="/path/to/ENDFB-VIII.0/all.map"
        checksum="c7561db7a418dd39b2201dfe110ab4a4" algorithm="md5"/>
</map>
```

A. Connection of GNDS types to ENDF-6 data types

Those familiar with the legacy ENDF-6 format will find a lot of commonality with data in this GNDS data structures. Here the list of Files (denoted by MF) in the ENDF-6 manual (Cross Section Evaluation Working Group, 2018) and their correspondence with GNDS data structures is given.

A.1 GNDS analogues of low level ENDF-6 data containers

TEXT free text fields in ENDF-6 only occur in the documentation MF/MT 1/451. GNDS `<documentation>` elements are much more expressive and discussed in Chapter 8.

CONT a CONT record is a low-level basic format in ENDF-6, used to build other elements such as TAB1 fields. GNDS has no direct analogue.

HEAD a HEAD record in ENDF-6 signals the beginning of an ENDF-6 section. The beginning of element GNDS markups depend on the format used (XML, HDF5, etc.)

END In ENDF-6, these markups include the SEND, FEND, MEND and TEND records. Each denote the end of an organisational element.

- **SEND** signals the end of an ENDF-6 section. This is analogous to the end markup of a GNDS `reaction` element. The actual markup depends on the format (XML, HDF5, etc.) for the GNDS file.
- **FEND** signals the end of an ENDF-6 file. This is analogous to the end markup of many GNDS end-of-data structure elements.
- **MEND** signals the end of an ENDF-6 material. This is analogous to the end markup of a `reactionSuite` data container. The actual markup depends on the format (XML, HDF5, etc.) for the GNDS file. Note that the MEND field is important when concatenating materials together in one ENDF file. GNDS files are never grouped together in this manner so the MEND field is obsolete.
- **TEND** signals the end of an ENDF-6 tape. ENDF libraries are no longer distributed on magnetic tapes. The concept is obsolete and has no GNDS analog.

DIR this is a special type of CONT field that provides a directory of ENDF-6 files and sections. The hierarchical nature of GNDS renders this markup obsolete.

LIST this is used to store a list of numbers in ENDF-6. The analogous format in GNDS is described in Section 5.2.

TAB1 this is used to store a 2-D interpolation tables in ENDF-6. The analogous formats in GNDS are described in Sections 6.1-6.2.

TAB2 this is used to store multi-dimensional interpolation tables in ENDF-6. The analogous formats in GNDS are described in Sections 6.3-6.5.

INTG this is used for compressed storage of ENDF-6 correlation matrices. GNDS has several different matrix compression schemes including sparse matrix formats. See Chapter 25 and Section 5.3 for more detail.

A.2 GNDS analogues of high level ENDF-6 organisational ele--ments

MF=1 *General information*

MT=451 *Descriptive data* – Unlike the legacy ENDF format, documentation is al--lowed at nearly all levels in the new data structure. See Chapter 8 for a discussion of the `documentation` element.

MT=452,455,456 *Number of neutrons per fission* – These are product data of a fission reaction. Chapter 21 discusses fission reactions, Chapter 22 discusses product elements and Section 17.3 discusses particle multiplicities.

MT=458 *Energy release due to fission* – This is product data of a fission reaction. Chapter 21 discusses fission reactions, Chapter 18 discusses product elements. Sections 18.7–18.9 discuss specific energy distributions relevant for fission reactions.

MT=460 *Delayed photon data* – This is product data of a fission reaction. Chapter 21 discusses fission reactions, Chapter 17 discusses product elements and other reaction outputs.

MF=2 *Resonance parameters* – See Chapter 19 for the discussion of the `resonances` ele--ment.

MF=3 *Reaction cross sections*

- *Neutral particle scattering* – All cross section data structures are described in Chapter 16.
- *Background cross sections in the resonance region* – All are stored with the reso--nances themselves as described in Chapter 19.
- *Charged particle scattering* – These cross sections are treated in the same manner as neutral particle scattering for the most part. The elastic `reaction` is problematic because of the divergences in the differential cross section. This special case is explained in Chapter 23.

MF=4 *Angular distributions* – These are product data of a reaction. The enclosing ele--ments are described in Chapter 17 and the distributions in Chapter 18.

MF=5 *Energy distributions* – These are product data of a reaction. The enclosing elements are described in Chapter 17 and the distributions in Chapter 18.

MF=6 *Product energy-angle distributions* – These are product data of a reaction. The enclosing elements are described in Chapter 17 and the distributions in Chapter 18.

MF=7 *Thermal neutron scattering law data* – This data describes scattering off of macro--scopic materials by very low energy neutrons. This data has many unique specifi-

-cations that are covered in Chapter 20.

MF=8 *Decay and fission product yields*

- *Decay data* – These are particle decay properties, see Section 12.2.
- *Fission product yields* – Although this is product data of the fission reaction, it has unique and detailed specifications covered in Chapter 22.

MF=9 *Multiplicities of radionuclide products* – This is product data of a reaction. Section 17.2 discusses `product` elements and Section 17.3 discusses particle multiplicities.

MF=10 *Production cross sections for radionuclides* – Production data are given within a `productions` reaction list as described in Section 14.2.

MF=12 *Photon production yield data* – This is product data of a reaction and Section 17.3 discusses particle multiplicities. However, the branching fractions from discrete level de-excitation are particle properties and covered in Section 12.2.

MF=13 *Photon production cross sections* – GNDS does not have an exact analogue to MF13. GNDS does store the photon multiplicity so when translating ENDF, one must divide the photon production cross section by the reaction cross section to get photon multiplicity. The exception to this rule is the total photon production cross section, which would be stored in the `productions` reaction list as described in Section 14.2.

MF=14 *Photon angular distributions* – This is product data of a reaction. The enclosing elements are described in Chapter 17 and the distributions in Chapter 18.

MF=15 *Continuous photon energy spectra* – This is product data of a reaction. The en--closing elements are described in Chapter 17 and the distributions in Chapter 18.

MF=26 *Secondary distributions for photo- and electro- atomic data* – This is product data of a reaction. The enclosing elements are described in Chapter 17 and the distributions in Chapter 18. In Chapter 24, the additional specifications for use with atomic reaction data are given.

MF=27 *Atomic form factors of scattering functions* – Scattering functions and form factors modify cross sections for photon scattering and are described in Chapter 24 while the cross section elements are described in Chapter 16.

MF=28 *Atomic relaxation data* – These are particle properties, so see Section 12.1 and subsection 12.1.13 in particular.

MF=30 *Covariances of model parameters* – See Section 25.3 for a description of the unified covariance structure for model parameters.

MF=31 *Covariances of fission $\overline{\nu}$* – See Section 25.2 for a description of the unified covari--ance structure.

MF=32 *Covariances of resonance parameters* – See Section 25.3 for a description of the unified covariance structure for model parameters, including resonance parameter covariance.

MF=33 *Covariances of neutron cross sections* – See Section 25.2 for a description of the unified covariance structure.

MF=34 *Covariances for angular distributions* – See Section 25.2 for a description of the unified covariance structure.

MF=35 *Covariances for energy distributions* – See Section 25.2 for a description of the unified covariance structure.

MF=40 *Covariances for radionuclide production* – See Section 25.2 for a description of the unified covariance structure.

B. GNDS versions

B.1 GNDS versioning

B.1.1 Adoption of semantic versioning

The Expert Group on GNDS has adopted semantic versioning (SemVer, 2019) as a tool to organise GNDS versions. Each format release is given a version number of the form MAJOR.MINOR[.PATCH]. The new GNDS-2.0 refers to the GNDS release with MAJOR version number '2' and MINOR version number '0'. The patch number is optional. Version numbers have the following meaning:

MAJOR version changes when backwards incompatible changes to the format are made.
MINOR version changes when functionality is added in a backwards compatible manner.
PATCH version changes when backwards compatible bug fixes are made. As these releases are rare, the patch number is often omitted.

All major, minor and patch versions are integers. Additional labels for pre-release and in-house formats are available as extensions to the MAJOR.MINOR[.PATCH] format. While the format is known by the designator 'GNDS-MAJOR.MINOR', 'GNDS-MAJOR.MINOR' files in the format carry the format designator 'MAJOR.MINOR'. The version number used for a GNDS file is given as an attribute in root nodes such as the `reactionSuite` and `covarianceSuite` elements.

B.1.2 In-house development versions

Institutions may wish to experiment with modifications to the GNDS format before they are officially approved by the GNDS Expert Group. For example, in the preparation of the GNDS-2.0 specifications, LLNL has developed an in-house version, called 'GNDS-1.10', which has several (but not all) format proposals implemented that were later approved for inclusion in GNDS-2.0. Test files labelled '1.10' have been distributed outside LLNL. It is straightforward to detect non-standard GNDS files using a schema validator along with a schema derived from the official format specification. However, the Expert Group has suggested a labelling convention to make it easier to understand why files do not comply with official GNDS specifications and hopefully saves the user time. To avoid confusion between these test versions and officially approved versions of GNDS, a convention for labelling these internal versions is given here.

The suggested versioning system is as follows: append institution and optional internal version number to latest official GNDS version number. For example, instead of 'GNDS-1.10' an organisation such as LLNL could use 'GNDS-1.9.LLNL' or 'GNDS-1.9.LLNL.x', where 'x' is an integer that is incremented as internal changes are made at LLNL. To be consistent with semantic versioning standards (SemVer, 2019), both the in-house designator and other refinements should restrict arguments to match the following regular expression: '[a-zA-Z0-9-]'.

B.2 Versions prior to GNDS-1.9

Prior to the establishment of the GNDS Expert Group, the GND/GNDS format was managed exclusively by Lawrence Livermore National Laboratory (LLNL). While LLNL did maintain strict versioning using its own in-house semantic versioning, the format of the version numbers varied over the years. While under LLNL maintenance, the GNDS format was known by the abbreviation 'GND'.

B.2.1 LLNL in-house formats

In this section, notable LLNL in-house versions of GNDS are listed. These formats are notable mainly because a sizeable number of files in each of these formats were widely distributed.

B.2.2 GND-1.0

'GND-1.0' was the first widely distributed GNDS version. ENDF/B-VII.1 (Chadwick et al., 2011) files are posted on Brookhaven National Laboratory's National Nuclear Data Center (NNDC) website at https://www.nndc.bnl.gov/endf/b7.1/gndFiles.html. Files using this version carry the format identifier 'gnd version 1.0'. This format has limited supported transport data, namely support for the ENDF/B neutron and photonuclear data libraries and does not including thermal scattering law data.

B.2.3 GND-1.2

'GND-1.2' was the second widely distributed GNDS version and is primarily available through the EXFOR/ENDF web application at https://www.nndc.bnl.gov/exfor/endf00.jsp. This format is notable since it has an XML stylesheet transform that can be used to generate an HTML overview of the GNDS file. Files using this version carry the format identifier ‘gnd version 1.2’. This format does not support thermal scattering law, atomic scattering or decay data.

B.2.4 Transition from “GND” to “GNDS”

Beginning with the establishment of the Expert Group in 2016, the name of the format ‘Generalised Nuclear Data’ (GND) has formally changed to ‘Generalised Nu--clear Database Structure’ (GNDS). This is explained in the EG-GNDS mandate https://www.oecd-nea.org/science/wpec/gnds/files/EGGNDS_GB.pdf. All GNDS formats de--veloped since the establishment of EG-GNDS use the “GNDS” abbreviation.

B.3 GNDS-1.9

GNDS-1.9 is the first official format, fully managed by the GNDS Expert Group. This format revised the LLNL in-house GND format to meet, as much as was possible given time constraints, the requirements laid out by the WPEC Subgroup 38.

GNDS-1.9 is the first official version of GNDS that supported all formats in modern ENDF libraries. As a result, ENDF/B-VIII.0 and JEFF-3.3 was released in this format. Files using this version carry the format identifier ‘1.9’.

This version was approved at the June 2019 GNDS Expert Group meeting at the NEA headquarters in Paris, France. The specifications are published in (NEA, 2020).

B.3.1 GNDS-1.10

GNDS-1.10 is an in-house LLNL development format that was used to prototype GNDS-2.0 formats. All features from GNDS-1.10 and most from GNDS-2.0 are present in this version. Files using this version carry the format identifier ‘1.10’.

B.4 GNDS-2.0

Following the June 2019 WPEC meeting, EG-GNDS set out to bring the GNDS format into alignment both with the Subgroup 38 requirements and the LLNL in-house development version ‘GNDS-1.10’. In all, Expert Group made 149 formal merge requests and 29 of the merge requests were substantive format proposal requiring peer review from the Expert Group and domain experts from the larger nuclear data community. The changes made to GNDS centered on nine themes, summarised in Table B.1. Interestingly, the entire development of GNDS-2.0 took place during the COVID-19 pandemic so no in-person

Expert Group meetings were held until May 2022 when the Expert Group issued its final approval of GNDS-2.0 and submitted the manuscript to NEA for publication.

Twelve substantive changes were completed and approved at the May 2020 EG-GNDS meeting, which was held as a videoconference due to the COVID-19 pandemic. These include:

1. `proposal/multiGroupStyleFix`: improve multi-group data styles;
2. `proposal/atomicConfigurations`: support for electron sub-shells, per the ENDF-6 format;
3. `proposal/externalFiles`: denote external resources that may need to be read before processing interaction;
4. `proposal/pids`: resolve possible discrepancy between branchings and PoPs;
5. `proposal/TNSL`: improve thermal neutron scattering markup, per requirements document;
6. `proposal/interaction`: denote what kind of reactionSuite that is being used;
7. `proposal/orphanProduct`: fix inconsistency in orphan product organisation;
8. `proposal/documentation`: improve documentation markup, per the requirements document;
9. `proposal/childFunctions`: add functional container organisation to low-level con--tainers;
10. `proposal/sandwichProduct`: support for ‘sandwich product’ covariance, per the requirements document;
11. `proposal/fissionFragmentData`: consolidate and clean up fission data support; and
12. `proposal/outerDomainValue`: rename ‘value’ attribute for clarity in multi-dimensional containers.

Two proposals did not have their reviews completed in time for the May 2020 EG-GNDS meeting. Coupled with the fact that the change from GNDS-1.9 to GNDS-2.0 was already large, the Expert Group allowed additional time for format proposals. Additional formats were approved in the November 2020 virtual meeting, including:

1. `proposal/primaryGamma`: add a `finalState` attribute;
2. `proposal/add-appendix`: add appendices including a change log and mapping between ENDF and GNDS; and
3. `proposal/URR_style`: add `URR_probabilityTables` as allowed `<style>`.

In preparation for the 2.0 release of the GNDS specifications, the Expert Group issued a feature freeze at the virtual May 2021 WPEC meeting. This feature freeze excluded resonance related features. Eight format proposals completed peer review at this meeting, including refinements of GNDS-2.0 and proper addition of support for thermal neutron scattering covariance.

1. `proposal/MultiDimCovariance`: ensure proper support for multi-dimensional co--variances such as encountered in TNSL covariance data;
2. `proposal/renameSums`: stylist improvements to element names, making them com--ply with conventions elsewhere in the specifications;

Table B.1: Themes of format proposals and bug fixes in the development of the GNDS-2.0 specifications. The merge requests summarised here include merge requests related to fixing typos, spelling, and grammar, provided they are directly related to these themes.

Theme	Format proposals	# Related merge requests
Atomic data	`proposal/atomicConfigurations`	4
Covariances	`proposal/sandwichProduct`, `proposal/MultiDimCovariance`, `proposal/renameCovarianceSection`	3
Documentation	`proposal/documentation`, `proposal/documentationContainer`, `proposal/docTextHasAToIsA`, `proposal/dateFormatting`, `proposal/documentation_and_metadata`	10
File management	`33-map-file-format`, `proposal/externalFile_checksums`, `proposal/externalFiles`	6
Gamma data	`proposal/primaryGamma`	2
Resonances	`proposal/URR_style`, `feature/radii`, `feature/boundary`, `feature/externalRMatrix`	6
SG-38 Compliance	`proposal/fissionFragmentData`, `proposal/multiGroupStyleFix`, `proposal/add-appendix`	3
Thermal neutron scattering	`proposal/TNSL`, `112-minor-tnsl-fixes-in-map-files`	5
Usability	`proposal/pids`, `proposal/interaction`, `proposal/orphanProduct`, `proposal/childFunctions`, `proposal/outerDomainValue`, `proposal/renameSums`, `81-pops-shell-node-needs-to-support-uncertainty`	7

3. `proposal/dateFormatting`: change the GNDS date formats to comply with existing standards (International Standards Organisation, 2019b);
4. `proposal/renameCovarianceSection`: simplify the covariance section naming;
5. `proposal/externalFile_checksums`: add support for file checksums;
6. `proposal/documentationContainer`: many small corrections, clarifications and re-namings to improve the usability of the documentation markups;
7. `proposal/docTextHasAToIsA`: simplify the various documentation markups; and
8. `33-map-file-format`: add support for `map` files, the GNDS analog of MCNP's `xsdir` file (Monte Carlo Code Group, n.d.).

Final approval of the format specifications, closing of all open issues and add a cross walk between GNDS documentation metadata and the DataCite.org metadata schema (DataCite Metadata Working Group, 2021b) (see appendix C) were performed at the October 2021 virtual Expert Group meeting.

1. `feature/radii`: consolidate the ENDF-6 "scattering radius" and "hard sphere ra--dius" concepts into one R-matrix radius;
2. `feature/boundary`: expand options for the boundary condition of the R-matrix;
3. `feature/externalRMatrix`: add support for the various background R-matrix treat--ments recently added to the ENDF-6 format; and
4. `proposal/documentation_and_metadata`: align the GNDS `documentation` node and decedents with DataCite.org's schema (DataCite Metadata Working Group, 2021b).

At the December 2021 virtual Expert Group meeting, the final editing process began with the aim of submitting the specifications manuscript at the May 2022 meeting to NEA for publication. In the final preparation process, four bug fixes and two ex--tremely minor format proposals were made. The two format proposals were approved unanimously by acclimation at the May 2022 WPEC meeting. They were:

1. `81-pops-shell-node-needs-to-support-uncertainty`: Add `uncertainty` as allowed child node of `shell` in the PoPs database; and
2. `112-minor-tnsl-fixes-in-map-files`: TNSL erroneously required a special `inter--action` attribute in the main `map` container.

C. Comparison of metadata schema

A Digital Object Identifier (DOI) is a persistent identifier that refers to a unique document or dataset. With the DOI for a document or dataset, one can easily locate the referenced file using web resources such as https://dx.doi.org and https://www.osti.gov/dataexplorer/. DOIs are an essential part of national and international Open Data efforts as they enable the 'Findable' part of the 'Findability, Accessibility, Interoperability, and Reusability (FAIR)' doctrine. One of the implicit goals for GNDS is to enable Open Data stewardship efforts in the area of nuclear reaction data.

Datasets with DOIs can be cited much like a journal article. The DataCite organisation recommends (Office of Scientific and Technical Information, 2020b) that datasets be cited as

Creator (PublicationYear). Title. Publisher. DOI,

or:

Creator (PublicationYear). Title. Version. Publisher. ResourceType. DOI

The DOI system functions with the help of national and international DOI registration organisations such as DOI.org, DataCite.org and OSTI.gov. These organisations work with data providers to set up dedicated webpages (i.e. landing pages) for each dataset or document, to assemble the needed metadata to describe the data and to register and assign the DOIs. In this appendix the metadata available within GNDS will be compared with the requirements and recommendations of these registration organisations.

For datasets, DOIs are assigned by the DataCite organisation. Data providers such as CSEWG, JEFF or JAEA must work through their local registrants rather than through DataCite directly, and these registrants interface with DataCite. In the United States, data creators must work through the US Department of Energy Data ID Services (Office of Scientific and Technical Information, 2021) for Department of Energy funded research or the Interagency Data (IAD) ID Service (Office of Scientific and Technical Information, 2020b) for research funded by other U.S. Federal Agencies.

The DOE Data ID Services, the Interagency Data ID Service and DataCite all have slightly different metadata schema. In this appendix, these different schema will not be reviewed in detail. The commonalities will be identified to simplify the job of translating metadata in GNDS into the required (and additional) metadata schema for DOI registration. Some metadata would be useful additions to the GNDS standard. In particular,

- date schema to denote important events in the history of the dataset in question;
- authorship and contributor schema to properly give credit to the originators of the dataset and others who have contributed to the dataset in various forms including performing bug fixes, sponsoring work or several other classes of contribution; and
- schema that denote interrelationships between datasets.

C.1 Required metadata for DOI assignment

The three DOI registering organiations (DataCite, DOE Data ID Services and the Intera--gency Data ID Service) each have different minimal metadata requirements to register a DOI. In Table C.1 a superset of the three main sources metadata requirements is provided. Some of this data is for internal use by OSTI and is a requirement specifically for DOE funded data generators. This metadata and other metadata described later in this appendix are all presented to users on the landing pages and on the search results of the various search pages.

As this metadata is required for DOI generation, a strategy for populating these fields from a typical GNDS file is required. Below is a list of metadata and an initial scheme for populating the field:

doi: This is assigned by DataCite.
title: Take from reactionSuite/styles/evaluated/documentation/title
creatorsblock: Populate from reactionSuite/styles/evaluated/documentation/authors
dataset_type: Required only by OSTI. OSTI Data Data ID schema (Office of Scientific and Technical Information, 2021) defines this as ‘ND’ for numerical data
publication_date: This corresponds to ENDF’s ENDATE field, populate from reactionSuite/styles/evaluated/documentation/publicationYear.
language: For the ENDF library, this defaults to English. For other ENDF formatted libraries, this likely defaults to English as well.
country: For the ENDF library, this defaults to USA. As this is optional for DataCite, it may be ‘none’.
sponsor: Required only by OSTI. Populate from reactionSuite/styles/evaluated/documentation/authors/acknowledgements.
originating_research_org: Required only by OSTI. Populate from reactionSuite/styles/evaluated/documentation/authors/affiliations.
sponsor_org: Required only by OSTI. Populate from reactionSuite/styles/evaluated/documentation/acknowledgements.
contract_nos: Required only by OSTI. Populate from reactionSuite/styles/evaluated/documentation/acknowledgements.
publisher: This is the dataset publisher. For example, it may be the NNDC, NEA, IAEA,

Table C.1: List of required metadata from (Office of Scientific and Technical Information, 2021) and (Office of Scientific and Technical Information, 2020a). Each concept is cross referenced by the metadata tag in the respective schema. Each block may have sub-properties that are child nodes of some form, depending on the schema language in question. Concepts marked with an * are the bare minimum for all three sources.

Concept	DataCite name	IAD Service name	DOE Data ID name	Description
Identifier DOI*	identifier	doi	doi	The Digital Object Identifier (DOI)
Title*	Title	title	title	The name or title by which a resource is known.
Publication Date*	PublicationYear	publication_-date	publication_-date	The year when the data was or will be made publicly available.
Site URL*	n/a	site_url	site_url	The URL containing the record content, which the DOI will redirect to. Given to DataCite as DOI destination.
Dataset Type*	resourceType-General	product_type	dataset_type	The general type of the resource.
Authors	creator	authors	creatorsblock	The main researchers involved in producing the data, or the authors of the publication, in priority order.
Originating Re--search Organi--sation	n/a	research_organi--zation	originating_re--search_org	If credited, the organisation name that performed the research
Language	language	language	language	The primary language in which the resource is given.
Publisher	Publisher	publisher	n/a	The name of the entity that holds, archives, publishes prints, dis--tributes, releases, issues, or produces the resource.
Country of Publi--cation	n/a	country	country	The country in which this version of the record was published; Not given to DataCite
Dataset Product Number(s)	alternate-Identifier	accession_num--ber	product_nos	Identifying number; “none” is acceptable.
Sponsoring Organisation(s)	n/a	sponsor_org	sponsor_org	Not given to DataCite
Contract Num--ber(s)	alternate-Identifier	contract_num--bers	contract_nos	Any contract or funding numbers associated with this record
Contact Name and Position	n/a	n/a	contact_name	Internal tag for OSTI records only; will not be publicly displayed.
Contact email	n/a	n/a	contact_email	Internal tag for OSTI records only; will not be publicly displayed.
Contact phone	n/a	n/a	contact_phone	Internal tag for OSTI records only; will not be publicly displayed.

JAEA, etc.

product_nos: This is assigned by the dataset publisher. For example, it may be the NNDC, NEA, IAEA, JAEA, etc. It may be 'none'.

site_url: This is assigned by the dataset publisher. For example, it may be the NNDC, NEA, IAEA, JAEA, etc.

contact_name: Required only by OSTI. This is determined by the dataset publisher. For example, it may be the NNDC, NEA, IAEA, JAEA, etc.

contact_email: Required only by OSTI. This is determined by the dataset publisher. For example, it may be the NNDC, NEA, IAEA, JAEA, etc.

contact_phone: Required only by OSTI. This is determined by the dataset publisher. For example, it may be the NNDC, NEA, IAEA, JAEA, etc.

C.2 Authorship and other contributions

It is important to give proper credit to the contribution of each person who creates or modifies an evaluation. The ENDF-6 format does not provide much flexibility regarding such credit. There is a 32 character wide field for listing all authors and contributors to an evaluation. ENDF-6 provides a free text field to list authors, but this field is not standardised and therefore not easily machine readable. As a side effect of these format restrictions, it is not uncommon for the last person to change a file to 'assert ownership' over an evaluation simply because there is no way to denote the scope of the changes made nor to preserve the contributions of previous evaluators.

The schema provided by DataCite overcomes this limitation by providing two classes of contribution: Authors (i.e. creators) and Contributors. The Author fields can then be reserved for significant contributors to the evaluation, while the Contributor fields can be used for fine-grained description of the nature of the contribution using the contributerType attribute. Allowed values of contributorType, as given in the DataCite schema are given in Table C.2. Note that this list can be expanded and adapted for the purpose of GNDS as the Contributor fields are not a requirement of any DOI registrar schema mentioned in this appendix.

Table C.2: Allowed contributorType values are given be-
-low. This table is reproduced from (DataCite Metadata Working Group, 2021a).

Option	Description	Usage Notes
ContactPerson	Person with knowl- -edge of how to access, troubleshoot, or otherwise field issues related to the resource	May also be the 'Point of Contact' in an organisation that controls access to the resource, if that organisation is different from the Publisher, Distribu- -tor, and Data Manager.

Continued on next page

Option	Description	Usage Notes
DataCollector	Person/institution responsible for finding or gathering/collecting data under the guidelines of the author(s) or Principal Investigator (PI)	May also be used when crediting survey conductors, interviewers, event or condition observers, or persons responsible for monitoring key instrument data.
DataCurator	Person tasked with reviewing, enhancing, cleaning, or standardizing metadata and the associated data submitted for storage, use, and maintenance within a data centre or repository	While the DataManager is concerned with digital maintenance, the DataCurator role encompasses quality assurance focused on content and metadata. DataCurator responsibilities include: checking completeness of the submitted dataset against the content as described by the submitter; verifying standard metadata according to the applicable system or schema; adding or verifying specialised metadata to add value and ensure access across disciplines; and determining how the metadata might map to search engines, database products, and automated feeds. Repository managers as well as data librarians working in the repository fall within this category.
DataManager	Person (or organisation with a staff of data managers, such as a data centre) responsible for maintaining the finished resource	The work done by this person or organisation ensures that the resource is periodically 'refreshed' in terms of software/hardware support, is kept available or is protected from unauthorised access, is stored in accordance with industry standards, and is handled in accordance with the records management requirements applicable to it.
Distributor	Institution tasked with responsibility to generate/disseminate copies of the resource in either electronic or print form	Works stored in more than one archive/repository may credit each as a distributor.

Continued on next page

Option	Description	Usage Notes
Editor	A person who oversees the details related to the publication format of the resource	Note: if the Editor is to be credited in place of multiple creators, the Editor's name may be supplied as Creator, with '(Ed.)' appended to the name.
HostingInstitution	Typically, the organisation allowing the resource to be available on the internet through the provision of its hardware/software/operating support	This role normally falls on the University, research centre or organisation where the data centre and/or data repository belongs. EXAMPLE: Université Grenoble Alpes (UGA) May also be used for an organisation that stores the data offline - often a data centre if that data centre is not the 'publisher' of the resource.
Producer	Typically, a person or organisation responsible for the artistry and form of a media product	In the data industry, this may be a company 'producing' DVDs that package data for future dissemination by a distributor.
ProjectLeader	Person officially designated as head of project team or subproject team instrumental in the work necessary to development of the resource	The Project Leader is not 'removed' from the work that resulted in the resource; he or she remains intimately involved throughout the life of the particular project team.
ProjectManager	Person officially designated as manager of a project. Project may consist of one or many project teams and sub-teams.	The manager of a project normally has more administrative responsibility than actual work involvement.
ProjectMember	Person on the membership list of a designated project/project team	This vocabulary may or may not indicate the quality, quantity, or substance of the person's involvement.
RegistrationAgency	Institution/organisation officially appointed by a Registration Authority to handle specific tasks within a defined area of responsibility	DataCite is a Registration Agency for the International DOI Foundation (IDF). One task assigned to DataCite is to assign DOI prefixes to the allocating agents who then assign the full, specific character string to data clients, provide metadata back to the DataCite registry, etc.

Continued on next page

Option	Description	Usage Notes
RegistrationAuthority	A standards-setting body from which Registration Agencies obtain official recognition and guidance	The IDF serves as the Registration Authority for the International Standards Organisation (ISO) in the area/domain of Digital Object Identifiers.
RelatedPerson	A person without a specifically defined role in the development of the resource, but who is someone the author wishes to recognise	This person could be an author's intellectual mentor, a person providing intellectual leadership in the discipline or subject domain, etc.
Researcher	A person involved in analysing data or the results of an experiment or formal study. May indicate an intern or assistant to one of the authors who helped with research but who was not so 'key' as to be listed as an author.	Should be a person, not an institution. Note that a person involved in the gathering of data would fall under the contributorType 'DataCollector.' The researcher may find additional data online and correlate it to the data collected for the experiment or study, for example.
ResearchGroup	Typically refers to a group of individuals with a lab, department, or division that has a specifically defined focus of activity.	May operate at a narrower level of scope; may or may not hold less administrative responsibility than a project team.
RightsHolder	Person or institution owning or managing property rights, including intellectual property rights over the resource	
Sponsor	Person or organisation that issued a contract or under the auspices of which a work has been written, printed, published, developed, etc.	Includes organisations that provide in-kind support, through donation, provision of people or a facility or instrumentation necessary for the development of the resource, etc.

Continued on next page

Option	Description	Usage Notes
Supervisor	Designated administrator over one or more groups/teams working to produce a resource, or over one or more steps of a development process	
WorkPackageLeader	A Work Package is a recognised data product, not all of which is included in publication. The package, instead, may include notes, discarded documents, etc. The Work Package Leader is responsible for ensuring the comprehensive contents, versioning, and availability of the Work Package during the development of the resource.	
Other	Any person or institution making a significant contribution to the development and/or maintenance of the resource, but whose contribution is not adequately described by any of the other values for contributorType	Could be a photographer, artist, or writer whose contribution helped to publicise the resource (as opposed to creating it), a reviewer of the resource, someone providing administrative services to the author (such as depositing updates into an online repository, analysing usage, etc.), or one of many other roles.

C.3 Dates

Each ENDF evaluation contains a series of dates that denote significant events in the preparation or publication of a dataset. The ENDF-6 format requires that files provide four separate DATE fields with for different purposes:

EDATE Date of evaluation given in the form EVAL-DEC74 in col. 23-32.
DDATE Original distribution date given in the form DIST-DEC74 in col. 23-32.

RDATE Date and number of the last revision to this evaluation in col. 34-43 in the form REV2-DEC74, where 2 in column 37 is the release number and must be equal to LREL. The term 'revision' (IREV), which appeared in earlier editions of this manual, was synonymous with 'release' (LREL) and has been dropped in favour of the more-frequently used term 'release'.

ENDATE Master File entry date in the form yyyymmdd right adjusted in col. 56-63. The Master File entry date is assigned by NNDC for ENDF/B libraries.

DataCite provides their metadata schema (DataCite Metadata Working Group, 2021b) and lists several DATE-like formats/fields. The DataCite PublicationYear field is equiva--lent to the ENDF ENDATE. However, DataCite also provides a dateType attribute and date elements as described on page 46 of the schema documentation (DataCite Meta--data Working Group, 2021a). This attribute denotes the type of DATE and allowed values are tabulated in Table C.3. For example: `<date dateType="created">2012-03-01/2012-03-05</date>` The mappings between the DataCite and ENDF-6 dates are given in Table C.4.

Given this, the DataCite system is clearly more expressive and should be considered for adoption in GNDS. Note also that DataCite uses a modified schema for specifying date ranges based on the RKMS-ISO8601 standard (see http://www.lyberty.com/meta/iso_8601.htm)

C.4 Capturing the relationship between datasets

GNDS currently has a `derivedFrom` element to denote which `style` is derived from another within a file. For the purpose of tracking derived data styles, this is sufficient. Indeed the legacy ENDF-6 format has only a simple LDRV flag to denote that data is derived. Neither GNDS-1.9 nor the legacy ENDF6 format provide more detailed interrelationships between data.

There are many other relationship types that are captured in the DataCite or US Department of Energy Data ID schemas that should be considered for inclusion in GNDS. The relationType attribute described in Table C.5 is a modifier to the Relate--dIdentifier tag. This tag also includes the URL or DOI of the dataset referred to by the RelatedIdentifier. Adoption of this schema will allow GNDS to track, within the metadata, which evaluations were adopted from other older libraries, ensuring that these older libraries and evaluators receive proper credit for their efforts.

Table C.3: Allowed dateType values are given below. This table is reproduced from (DataCite Metadata Working Group, 2021a).

Option	Description	Usage Notes
accepted	The date that the publisher accepted the resource into their system	To indicate the start of an embargo period, use Sub--mitted or Accepted, as ap--propriate.
available	The date the resource is made publicly available. May be a range.	To indicate the end of an embargo period, use Avail--able.
copyrighted	The specific, documented date at which the resource receives a copy--righted status, if applicable	
collected	The date or date range in which the resource content was collected	To indicate precise or particular timeframes in which research was conducted.
created	The date the resource itself was put to--gether; this could refer to a timeframe in ancient history, a date range, or a single date for a final component, e.g. the finalised file with all the data.	Recommended for discov--ery.
issued	The date that the resource is pub--lished or distributed, e.g. to a data centre	
submitted	The date the creator submits the re--source to the publisher. This could be different from Accepted if the pub--lisher then applies a selection pro--cess.	Recommended for discov--ery. To indicate the start of an embargo period, use Submitted or Accepted, as appropriate.
updated	The date of the last update to the re--source, when the resource is being added to. May be a range.	
valid	The date or date range during which the dataset or resource is accurate.	
withdrawn	The date the resource is removed	It is good practice to in--clude a Description that in--dicates the reason for the retraction or withdrawal.

Table C.4: Mapping between the ENDF-6 and DataCite `dateType` schema.

ENDF DATE	dataType	Comment
EDATE	created or updated	The ENDF format doesn't clearly distinguish between original evaluations and small up--dates
DDATE	issued	
RDATE	updated	In ENDF-6, in many cases, RDATE should have been changed rather than EDATE
ENDATE	n/a	Closest analogue is PublicationYear in the Dat--aCite schema

Table C.5: Allowed relationType values are given below. Description of the relationship of the resource being registered (A) and the related resource (B). This table is reproduced from (DataCite Metadata Working Group, 2021a).

Option	Description	Usage Notes
IsCitedBy	indicates that B includes A in a citation	Recommended for discovery.
Cites	indicates that A includes B in a citation	Recommended for discovery.
IsSupplementTo	indicates that A is a supple--ment to B	Recommended for discovery.
IsSupplementedBy	indicates that B is a supple--ment to A	Recommended for discovery.
IsContinuedBy	indicates A is continued by the work B	
Continues	indicates A is a continua--tion of the work B	
Describes	indicates A describes B	
IsDescribedBy	indicates A is described by B	
HasMetadata	indicates resource A has additional metadata B	
IsMetadataFor	indicates additional meta--data A for a resource B	

Continued on next page

Option	Description	Usage Notes
HasVersion	indicates A has a version (B)	The registered resource such as a software package or code repos--itory has a versioned instance (indicates A has the instance B). It may be used, e.g. to relate an un-versioned code repository to one of its specific software ver--sions.
IsVersionOf	indicates A is a version of B	The registered resource is an in--stance of a target resource (in--dicates that A is an instance of B). It may be used, e.g. to re--late a specific version of a soft--ware package to its software code repository.
IsNewVersionOf	indicates A is a new edition of B, where the new edition has been modified or up--dated	
IsPreviousVersion Of	indicates A is a previous edition of B	
IsPartOf	indicates A is a portion of B; may be used for elements of a series	Primarily this relation is applied to container-contained type re--lationships. Note: May be used for individual software modules; note that code repository-to-version relationships should be modelled using IsVersionOf and HasVersion Recommended for discovery.
HasPart	indicates A includes the part B	Primarily this relation is applied to container-contained type re--lationships. Note: May be used for individual software modules; note that code repository-to-version relationships should be modelled using IsVersionOf and HasVersion Recommended for discovery.
IsPublishedIn	indicates A is published in--side B, but is independent of other things published inside of B	

Continued on next page

Option	Description	Usage Notes
IsReferencedBy	indicates A is used as a source of information by B	
References	indicates B is used as a source of information for A	
IsDocumentedBy	indicates B is documentation about/ explaining A; e.g. points to software documentation	
Documents	indicates A is documentation about B; e.g. points to software documentation	
IsCompiledBy	indicates B is used to compile or create A	Note: This may be used to indicate either a traditional text compilation, or the compiler program used to generate executable software.
Complies	indicates B is the result of a compile or creation event using A	Note: may be used for software and text, as a compiler can be a computer program or a person.
IsVariantFormOf	indicates A is a variant or different form of B	Use for a different form of one thing. May be used for different software operating systems or compiler formats, for example.
IsOriginalFormOf	indicates A is the original form of B	May be used for different software operating systems or compiler formats, for example.
IsIdenticalTo	indicates that A is identical to B, for use when there is a need to register two separate instances of the same resource	IsIdenticalTo should be used for a resource that is the same as the registered resource but is saved on another location, maybe another institution.
IsReviewedBy	indicates that A is reviewed by B	
Reviews	indicates that A is a review of B	
IsDerivedFrom	indicates B is a source upon which A is based	IsDerivedFrom should be used for a resource that is a derivative of an original resource.
IsSourceOf	indicates A is a source upon which B is based	IsSourceOf is the original resource from which a derivative resource was created.
IsRequiredBy	Indicates A is required by B	Note: May be used to indicate software dependencies.

Continued on next page

Option	Description	Usage Notes
Requires	Indicates A requires B	Note: May be used to indicate software dependencies.
Obsoletes	Indicates A replaces B	
IsObsoletedBy	Indicates A is replaced by B	

Index of formats

References

American Physical Society (2021), “PhySH - Physics Subject Headings – Version 2.2.0”, physh.org/ (visited on 1 February 2022).

Blokhin, A. et al. (2016), “New Version of Neutron Evaluated Data Library BROND-3.1”, *Problems of Atomic Science and Technology, Series: Nuclear Constants*, 2, 62–93, vant.ippe.ru/images/pdf/2016/2-5.pdf (visited on 1 February 2022).

Brown, D. et al. (2018), “ENDF/B-VIII.0: The 8th major release of the nuclear reaction data library with CIELO-project cross sections, new standards and thermal scattering data” [Special Issue on Nuclear Reaction Data], *Nuclear Data Sheets*, *148*, 1–142, DOI:10.1016/j.nds.2018.02.001.

Cacuci, D. (Ed.)(2010), “Handbook of nuclear engineering” [ISBN 978-0-387-98149-9], Spinger, DOI:10.1007/978-0-387-98149-9.

Cerf, V. (1969), “ASCII format for Network Interchange”, www.faqs.org/rfcs/rfc20.html (visited on 1 February 2022).

Chadwick, M. B. et al. (2006), “ENDF/B-VII.0: Next generation evaluated nuclear data library for nuclear science and technology”, *Nuclear Data Sheets*, *107*, 2931, DOI: 10.1016/j.nds.2006.11.001.

Chadwick, M. B. et al. (2011), “ENDF/B-VII.1 nuclear data for science and technology: Cross sections, covariances, fission product yields and decay data”, *Nuclear Data Sheets*, *112*, 2887, DOI:10.1016/j.nds.2011.11.002.

Chadwick, M. B., Young, P. G., & Chiba, S. (1995), “Photonuclear angular distribution systematics in the quasideuteron regime”, *Journal of Nuclear Science and Technology*, *32*(11), 1154–1158, DOI:10.1080/18811248.1995.9731830.

Coste-Delclaux, M. (2009), “GALILEE: A nuclear data processing system for transport, depletion and shielding codes”, In *Proceedings of the Second international Workshop on Nuclear Data Evaluation for Reactor applications (Wonder 2009)*, inis.iaea.org/search/search.aspx?orig_q=RN:42026936 (visited on 1 February 2022).

Cross Section Evaluation Working Group (2018), “ENDF-6 Formats Manual - Data Formats and Procedures for the Evaluated Nuclear Data Files ENDF/B-VI, ENDF/B-VII and ENDF/B-VIII” (Edited by A. Trkov, M. Herman and D. A. Brown) [CSEWG Document ENDF-102, Report BNL-203218-2018-INRE], CSEWG Document ENDF-102, Report BNL-203218-2018-INRE.

Cullen, D. (1989), “Program SCATMAN: A Code Designed to Calculate Photon Coher--ent Scattering Anomalous Scattering Factors and Cross Sections” (Computer software UCRL-ID-103422), Lawrence Livermore National Laboratory, CA, USA.

Cullen, D., Hubbell, J., & Kissel, L. (1997), "EPDL97: The Evaluated Photon Data Library '97 Version" (Report UCRL-LR-50400 Vol. 6 Rev. 5), Lawrence Livermore National Laboratory, CA, USA.

DataCite Metadata Working Group (2021a), "Datacite metadata schema documentation for the publication and citation of research data and other research outputs v4.4", DOI:10.14454/3W3Z-SA82.

DataCite Metadata Working Group (2021b), "Datacite metadata schema for the pub--lication and citation of research data and other research outputs v4.4", DOI: 10.14454/FXWS-0523.

ECMA International (2021), "ECMA-376 Office Open XML file formats", www.ecma-international.org/publications-and-standards/standards/ecma-376/ (visited on 1 February 2022).

Fröhner, F. H., & Bouland, O. (2001), "Treatment of external levels in neutron resonance fitting: Application to the nonfissile nuclide Cr-52", *Nuclear Science and Engineering*, *137*(1), 70–88, DOI:10.13182/NSE01-A2176.

Ge, Z. et al. (2017), "CENDL project, the chinese evaluated nuclear data library", *EPJ Web Conf.*, *146*, 02002, DOI:10.1051/epjconf/201714602002.

Gundersen, B. B., & Hendrikse, Z. (2007), "BibTeX as XML markup", bibtexml.sourceforge.net (visited on 1 February 2022).

Hedstrom, G., Beck, B., & Mattoon, C. (2016), "Merced, version 00." (Computer software LLNL-CODE-725546), Lawrence Livermore National Laboratory, CA, USA.

Hehr, B. D. (2010), "Development of the Thermal Neutron Scattering Cross Sections of Graphitic Systems using Classical Molecular Dynamics Simulations" (Doctoral dissertation), North Carolina State University, repository.lib.ncsu.edu/handle/1840.16/7422 (visited on 1 February 2022).

Hove, L. V. (1954), "Correlations in space and time and born approximation scattering in systems of interacting particles", *Physical Review*, 95, 249, DOI:10.1103/PhysRev.95.249.

Hubbell, J. H. et al. (1975), "Atomic form factors, incoherent scattering functions, and photon scattering cross sections", *Journal of Physical and Chemical Reference Data*, *4*(3), 471–538, DOI:10.1063/1.555523.

IAEA (2000), "Handbook on Photonuclear Data for Applications Cross-sections and Spectra", www.iaea.org/publications/6043/handbook-on-photonuclear-data-for-applications-cross-sections-and-spectra (visited on 1 February 2022).

Institute of Electrical and Electronics Engineers (2008), "IEEE Standard for Floating-Point Arithmetic", DOI:10.1109/IEEESTD.2008.4610935.

International Standards Organisation (2019a), "Information and documentation — Digital object identifier system", www.iso.org/standard/43506.html (visited on 1 February 2022).

International Standards Organisation (2019b), "ISO 8601 Date and time format", www.iso.org/iso-8601-date-and-time-format.html (visited on 1 February 2022).

International Standards Organisation (2017), "Information technology — Universal Coded Character Set (UCS)", www.iso.org/standard/69119.html (visited on 1 February 2022).

Kalbach, C. (1988), "Systematics of continuum angular distributions: Extensions to higher energies", *Phys. Rev. C*, *37*, 2350–2370, DOI:10.1103/PhysRevC.37.2350.

Kalbach, C., & Mann, F. M. (1981), "Phenomenology of continuum angular distributions. I. Systematics and parametrization", *Phys. Rev. C*, *23*, 112–123, DOI:10.1103/PhysRevC.23.112.

Kissel, L. et al. (1995), "The validity of form-factor, modified-form-factor and anomalous-scattering-factor approximations in elastic scattering calculations", *Acta Crystal--lographica Section A*, *51*(3), 271–288, DOI:10.1107/S010876739400886X.

Klein, O. et al. (1929), "Über die streuung von strahlung durch freie elektronen nach der neuen relativistischen quantendynamik von dirac", *Zeitschrift für Physik*, *52*(11), 853–868, DOI:10.1007/BF01366453.

Koning, A. et al. (2019), "TENDL: Complete Nuclear Data Library for Innovative Nuclear Science and Technology" [Special Issue on Nuclear Reaction Data], *Nuclear Data Sheets*, *155*, 1–55, DOI:10.1016/j.nds.2019.01.002.

Lamport, L. (1994), "*latex: A document preparation system: User's guide and reference.*", Addison-Wesley Professional, Reading, MA, USA.

Larson, N. (2008), "Updated users' guide for SAMMY: Multilevel r-matrix fits to neutron data using Bayes' equations" (Report ORNL/TM-9179/R8), Oak Ridge National Laboratory, TN, USA.

MacFarlane, R. et al. (2017), "The NJOY nuclear data processing system, version 2012" (Report LA-UR-12-27079), Los Alamos National Laboratory, NM, USA.

Monte Carlo Code Group (n.d.), "A general Monte Carlo N-particle (MCNP) transport code, version 6.1", laws.lanl.gov/vhosts/mcnp.lanl.gov/index.shtml (visited on 1 February 2022).

NEA (2020), "Specifications for the Generalised Nuclear Database Structure (GNDS) – Version 1.9" [NEA No. 7519], OECD Publishing, Paris, DOI:10.1787/94d5e451-en.

NEA (2017), "Specifications for an XML markup for documentation elements in the GNDS and related formats", OECD Publishing, Paris, www.oecd-nea.org/science/wpec/sg38 (visited on 1 February 2022).

NEA (2016a), "Detailed requirements for a next generation nuclear data structure" (Re--port BNL-112394-2016-IR), OECD Publishing, Paris, www.oecd-nea.org/science/wpec/sg38 (visited on 1 February 2022).

NEA (2016b), "Requirements and specifications for a particle database", OECD Publishing, Paris, www.oecd-nea.org/science/wpec/sg38 (visited on 1 February 2022).

Nic, M., Jirat, J., & Kosata, B. (2018), "IUPAC compendium of chemical terminology - the Gold Book", International Union of Pure; Applied Chemistry, DOI:10.1351/goldbook.

Office of Scientific and Technical Information (2020a), "Interagency Data ID Service Mapping", www.osti.gov/data-services (visited on 1 February 2022).

Office of Scientific and Technical Information (2020b), "Interagency Data ID Services", www.osti.gov/data-services (visited on 1 February 2022).

Office of Scientific and Technical Information (2021), "DOE Data ID Service Manual", www.osti.gov/data-services (visited on 1 February 2022).

Otuka, N. et al. (2014), "Towards a More Complete and Accurate Experimental Nuclear Reaction Data Library (EXFOR): International Collaboration Between Nuclear Reaction Data Centres (NRDC)", *Nuclear Data Sheets*, *120*, 272–276, DOI:10.1016/j.nds.2014.07.065.

Patashnik, O. (1988), "Bibtexing", texdoc.org/serve/btxdoc/0 (visited on 1 February 2022).

Plompen, A. et al. (2020), “The Joint Evaluated Fission and Fusion Nuclear Data Library, JEFF-3.3”, *European Physical Journal A*, 56, 181, DOI:10.1140/epja/s10050-020-00141-9.

Seco, J., & Verhaegen, F. (2016), “Monte Carlo Techniques in Radiation Therapy”, CRC Press, Boca Raton, DOI:10.1201/b13961.

SemVer (2019), “Semantic Versioning 2.0.0”. semver.org (visited on 1 February 2022).

Shibata, K. et al. (2011), “JENDL-4.0: A new library for nuclear science and engineering”, *Journal of Nuclear Science and Technology*, 48(1), 1–30, DOI:10.1080/18811248.2011.9711675.

Tuli, J. (2001), “Evaluated Nuclear Structure Data File; A Manual for Preparation of Data Sets” (Report BNL-NCS-51655-01/02-Rev), Brookhaven National Laboratory, NY, USA.

World Wide Web Consortium (2010), “XML Linking Language (XLink) Version 1.1”, www.w3.org/TR/xlink11/ (visited on 1 February 2022).

WPEC Subgroup #38 (2016), “General-Purpose Data Containers for Science and Engi--neering (draft – 4 May 2015)”. www.oecd-nea.org/science/wpec/sg38/Meetings/2015_May/generalDataContainers.pdf (visited on 1 February 2022).

Zabrodskaya, S. et al. (2007), “ROSFOND - Russian National Library of Evaluated Neutron Data”, *Problems of Atomic Science and Technology. Series: Nuclear Constants*, 1, 1–18, vant.ippe.ru/images/pdf/2007/1.pdf (visited on 1 February 2022).

Zerkin, V., & Pritychenko, B. (2018), “The experimental nuclear reaction data (exfor): Extended computer database and web retrieval system”, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 888, 31–43, DOI:10.1016/j.nima.2018.01.045.

Specifications for the Generalised Nuclear Database Structure (GNDS)

Knowledge of basic nuclear physics data is essential for the modelling and safe operation of all types of nuclear facilities. The de facto international standard format, Evaluated Nuclear Data File 6 (ENDF-6) format, was designed originally for 1960s era punch-card readers. The replacement of the system of codes built off this format has been recognised as an important initiative.

The ability to use increasingly high-fidelity nuclear physics, coupled to accurate uncertainties, is crucial for advanced simulations. This in turn requires more detailed and accurate data, then requiring improvements to the data storage standards, simultaneously enabling robust Quality Assurance and transfer of knowledge to the next generation.

In 2013, the NEA Working Party on International Nuclear Data Evaluation Co-operation (WPEC) launched a project to review the requirements for an international replacement for ENDF-6. The recommendations prompted the creation of a new Expert Group on a Generalised Nuclear Data Structure (GNDS) in 2016 that has used these requirements as the framework for a new format specification. Following rigorous international review, version 1.9 was unanimously approved as the first official published format in 2020. Since then, some 149 formal change requests were made to add new features and clarifications to the specifications, which were unanimously approved for publication in this release as version 2.0.

Nuclear Energy Agency (NEA)
46, quai Alphonse Le Gallo
92100 Boulogne-Billancourt, France
Tel.: +33 (0)1 73 21 28 19
nea@oecd-nea.org **www.oecd-nea.org**

NEA No. 7647
ISBN 9789264901971