### RESQML Overview

An industry-developed, vendor-neutral format that facilitates data exchange among the many software applications used along the E&P subsurface workflow, which helps promote interoperability and data integrity among these applications and improve workflow efficiency.

<table>
<thead>
<tr>
<th>Version:</th>
<th>Usage Guide for RESQML v.1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract:</td>
<td>For technical people, such as programmers and architects, implementing RESQML. An overview of key concepts and technologies, and a high level process for implementing the RESQML data exchange standard into a software package.</td>
</tr>
<tr>
<td>Prepared by:</td>
<td>Energistics and RESQML SIG</td>
</tr>
<tr>
<td>Date published:</td>
<td>17 October 2011</td>
</tr>
<tr>
<td>Document type:</td>
<td>Usage Guide</td>
</tr>
<tr>
<td>Keywords:</td>
<td>standards, energy, data, information, process, reservoir model, shared earth model</td>
</tr>
</tbody>
</table>
Acknowledgements
This version of the RESQML standard and documentation were developed by the Energistics RESQML Special Interest Group (SIG), which includes these members: BP, Chevron, Computer Modeling Group (CMG), Dynamic Graphics (DGI), Halliburton/Landmark, IFP Energies Nouvelles (French Institute of Petroleum), Paradigm, Roxar, Schlumberger, Shell, Total, Transform Software.

Usage, Intellectual Property Rights, and Copyright
The material described in this document was developed by and is the intellectual property of Energistics. Energistics develops material for open, public use so that the material is accessible and can be of maximum value to everyone.

Use of the material in this document is governed by the Energistics Intellectual Property Policy document and the Product Licensing Agreement, both of which can be found on the Energistics website, http://www.energistics.org/legal-policies.

All Energistics published materials are freely available for public comment and use. Anyone may copy and share the materials but must always acknowledge Energistics as the source. No one may restrict use or dissemination of Energistics materials in any way.

Trademarks
Energistics™, POSC®, Epicentre®, WITSML™, PRODML™, Upstream Standards. Bottom Line Results™, The Energy Standards Resource Centre™ and their logos are trademarks or registered trademarks of Energistics. Access, receipt, and/or use of these documents and all Energistics materials are generally available to the public and are specifically governed by the Energistics Product Licensing Agreement (http://www.energistics.org/product-license-agreement)
## Amendment History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comment</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31 Dec 2010</td>
<td>First publication for version 1.0 of the RESQML standard.</td>
<td>Energistics and RESQML SIG</td>
</tr>
<tr>
<td>1.1</td>
<td>17 Oct 2011</td>
<td>Updated as necessary for version 1.1 of RESQML.</td>
<td>Energistics and RESQML SIG</td>
</tr>
</tbody>
</table>
# Table of Contents

## 1 Introduction

1.1 Audience, Purpose, and Scope ................................................................. 1  
1.2 Documentation Set and Resources ........................................................... 1  
1.2.1 Links in this Document ................................................................. 1

## 2 Overview of RESQML

2.1 Standards and Technology ......................................................................... 2  
2.1.1 XML ....................................................................................... 2  
2.1.2 HDF5 .................................................................................... 2  
2.1.3 RESQML Document, Files, and Conventions .................................. 2  
2.1.4 Standards Used in RESQML ..................................................... 3

2.2 Schema Overview ..................................................................................... 4

2.3 RESQML Data Objects and Key Concepts ................................................ 5  
2.3.1 Faults and Horizons ...................................................................... 5  
2.3.2 Grids ..................................................................................... 6  
2.3.3 Wells ..................................................................................... 6  
2.3.4 Property Data ........................................................................... 7  
2.3.5 Coordinate Reference System ..................................................... 7  
2.3.6 Bin Grids .............................................................................. 7  
2.3.7 Feature, Version, and Representations ........................................... 7

## 3 Implementing RESQML in a Software Package

3.1 File Types and Technology ........................................................................ 8

3.2 Implementation Workflow .......................................................................... 8

3.3 Reading and Writing RESQML Documents............................................... 9  
3.3.1 Basic Read Process with Proxy Classes and HDF5 .......................... 9  
3.3.2 Basic Write Process with Proxy Classes and HDF5 ....................... 10

3.3.3 Basic Read Process with XML Parser ........................................... 10

3.3.4 Basic Write Process with XML Parser .......................................... 10

## 4 Implementing HDF5

4.1 Overview of RESQML HDF5 .................................................................... 11

4.2 Adding HDF5 to your Code ...................................................................... 11

4.3 HDF5 Handles ....................................................................................... 12

4.4 Reading from a RESQML HDF5 File ...................................................... 12  
4.4.1 Create the HDF5 Types Required for RESQML ......................... 12  
4.4.2 Reading from a RESQML HDF5 File ........................................... 12

4.5 Writing Data to a RESQML HDF5 File .................................................. 13  
4.5.1 Enabling Compression ............................................................... 13  
4.5.2 Creating the Groups that Contain the Dataset ............................... 13

4.5.3 Creating the Dataspace ................................................................ 14  
4.5.4 Creating the Dataset .................................................................. 14

4.5.5 Writing the Data ........................................................................ 14

4.6 HDF5 File Conventions and Formats ...................................................... 14

4.6.1 Group/Dataset Hierarchy and Name Guidelines ................................ 14

4.6.2 Common Types ......................................................................... 15

4.6.3 Structural Types ........................................................................ 15

4.6.4 Structural Representations .......................................................... 15

4.6.5 Grid Types .............................................................................. 16

4.6.6 Grid Representations .................................................................. 18

4.6.7 Property Representations ............................................................ 19

## 5 Using RESQML Property Kinds

5.1 Available Files ..................................................................................... 21
6 Implementing Coordinate Reference Systems and Coordinate Systems ........................................26
   6.1 Introduction .................................................................................................................26
   6.1.1 Definitions ...........................................................................................................26
   6.2 Local Coordinate Reference System ..........................................................................26
   6.3 Global Coordinate Reference System ......................................................................27
      6.3.1 Defining the Global 2D CRS ...........................................................................27
      6.3.2 Defining the Global Vertical CRS ....................................................................27
      6.3.3 Defining Global Time CRS ...............................................................................27
   6.4 Defining the Local Coordinate Reference System .....................................................28
      6.4.1 Areal Local Coordinate Reference System ...................................................28
      6.4.2 Local Depth Coordinate System .....................................................................35
      6.4.3 Local Time Coordinate System ......................................................................36
7 Bin Grids ..........................................................................................................................37
   7.1 Definition ................................................................................................................37
   7.2 Use of Bin Grids .......................................................................................................38
Appendix A. Reference Standards .....................................................................................39
Appendix B. Dublin Core Elements ...............................................................................40
1 Introduction

RESQML is an XML-based data exchange standard that helps address the data-incompatibility and data-integrity challenges faced by petro-technical professionals when using the multiple software technologies required along the entire subsurface workflow, for analysis, interpretation, modeling, and simulation.

This document provides an overview of the RESQML standard and guidelines for implementing it into a software package.

- For a complete introduction to RESQML, see the RESQML Overview Guide.
- To review published use cases supported by RESQML, see the RESQML Use Case Guide.

1.1 Audience, Purpose, and Scope

This document:

- Is for information technology (IT) professionals—programmers, developers, architects and others—who are implementing RESQML into a software package.
- Provides an overview of the RESQML standard, the schema, and key concepts.
- Provides a high-level process for implementing RESQML into a software package.
- Contains example code to help with implementation.

To ensure you are reading the latest version of this document, visit the Energistics website, http://www.energistics.org/current-resqml-standards.

1.1.1 Audience Assumptions

This guide assumes that the reader has a good general understanding of programming and XML, and a basic understanding of the exploration and production (E&P) subsurface domain and workflow.

1.2 Documentation Set and Resources

The RESQML XML schema contains definitions and documentation of schema elements and their use.

The RESQML documentation set includes the following, which can be found at http://www.energistics.org/current-resqml-standards.

<table>
<thead>
<tr>
<th>Document</th>
<th>Purpose/Audience</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESQML Overview Guide</td>
<td>Overview of workflow and business processes that RESQML was designed to facilitate. A useful introduction to RESQML for both petro-technical and IT technical professionals.</td>
</tr>
<tr>
<td>RESQML Usage Guide</td>
<td>Overview of key technical concepts for implementing the RESQML standard in a software package. For IT professionals implementing RESQML.</td>
</tr>
<tr>
<td>RESQML Use Case Guide</td>
<td>Lists RESQML use cases. For use by both petro-technical and IT professionals to understand the business/domain application of RESQML v1.0.</td>
</tr>
</tbody>
</table>

1.2.1 Links in this Document

Though no special text formatting convention is used, all section numbers and page numbers in this documents are links.
2 Overview of RESQML

This chapter describes key standards, technology, data objects and concepts for the current version of RESQML.

2.1 Standards and Technology

RESQML leverages existing standards and technology, which contributes to its rigor and flexibility. This section provides an overview of these standards and their roles in RESQML.

For a list of these standards, their sponsoring organizations, and websites, see Appendix A, page 39.

2.1.1 XML

The primary transfer document is XML. The RESQML XML schemas are based on the design patterns, common types, and reference data from the previously published Energistics-stewarded standards, WITSML (Well Information Transfer Markup Language, for drilling and completions data) and PRODML (Production Markup Language).

Leveraging these existing standards and schemas allows integration of a rich set of data objects for cross-domain workflows and makes it possible for RESQML to adopt the WITSML version of well data (such as logs, directional surveys, formation markers, etc.), instead of developing new ones.

To download the latest version of the RESQML schema from the Energistics website, see http://www.energistics.org/current-resqml-standards.

2.1.2 HDF5

To handle multi-million cell models, the XML file can optionally reference an auxiliary Hierarchical Data Format 5 (HDF5) file, where geometry and properties can be stored.

HDF5 is a set of open file formats and libraries designed to store and organize large amounts of numerical data, and improve speed and efficiency of data processing. Specifically, it provides:

- Machine/architecture-independent "binary" format (supported on Windows, Linux, etc. APIs are available in C++, Java, and .NET, though the .NET implementation is a prototype as of January 2011)
- Hyper-slabbing of array data so that IJK sub-arrays may be extracted from the RESQML document without reading the entire data file; and built-in data compression.

For more information on how to implement HDF, available tools and tutorials, or to download the libraries, see the HDF Group website at: http://www.hdfgroup.org/HDF5/.

For more information specific to RESQML, see Chapter 4, page 11.

2.1.3 RESQML Document, Files, and Conventions

A RESQML document may have up to 2 files, as described in the table below.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Required/Optional</th>
<th>Naming Convention</th>
<th>Example File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>Required</td>
<td>File extension = .resqml</td>
<td>Myproject.resqml</td>
</tr>
<tr>
<td>HDF5</td>
<td>Optional (though will most likely be used)</td>
<td>File name = the RESQML file name, including the extension. File extension = .h5</td>
<td>Myproject.resqml.h5</td>
</tr>
</tbody>
</table>

2.1.3.1 Other Conventions

RESQML documents also use these conventions:

- All units of measure are in SI.
• RESQML document must have a global CRS defined but it can be defined as “unknown” (see Chapter 6, page 26).

2.1.4 Standards Used in RESQML
In addition to XML and HDF5, RESQML also leverages the standards listed in this section. For a complete list of all standards used in RESQML and their sponsoring organizations, see Appendix A, page 39.

2.1.4.1 EPSG Codes
The RESQML schema can use EPSG codes for geolocalization as part of defining a coordinate reference system (CRS).

The European Petroleum Survey Group (EPSG) publishes a database that includes codes for easy reference to well known global locations, such as the elevation of the mean sea level of a specific body of water or of the summit of a specific mountain.

The EPSG (http://www.epsg.org/), the globally recognized experts on geodetic issues, has been absorbed into the Surveying and Position Committee of the International Association of Oil & Gas Producers (OGP), which is now the owner of the EPSG database of Geodetic Parameters and assigned codes. For more information, see Chapter 6, page 26.

2.1.4.2 Geographic Markup Language
The OpenGIS® Geography Markup Language Encoding Standard (GML) is an XML grammar for expressing geographical features from the Open Geospatial Consortium (OGC) http://www.opengeospatial.org/.

2.1.4.3 Dublin Core Elements
Dublin Core is a metadata standard that has been adapted for use with RESQML to provide traceability. A set of standard metadata elements—for example, creator, date created, date modified, etc.—can be assigned to a RESQML document and to key data objects within documents to provide traceability for documents and objects for information such as users, software used, time/date stamp, and others.

For a complete list of Dublin Core Elements and their use in RESQML, see Appendix B, page 40.

2.1.4.4 Global Unique Identifiers (GUIDs)
RESQML uses global unique identifiers (GUID) for each RESQML document and each main object within a document such as faults, horizons and grids. Use of GUIDs makes it possible to manipulate objects independently of a grid (an issue with RESQML predecessor standard, RESCUE) and works with Dublin Core elements to support traceability.
2.2 Schema Overview

Figure 1 shows a high level overview of a RESQML document.

![Figure 1 High-level RESQML schema.](image)

A RESQML document contains one required and several optional data objects grouped into sets, as shown in Figure 1 and described the table below. Each data object can only be specified once. RESQML currently uses one XML schema for all objects.

The RESQML document and each object in it has a global unique identifier (GUID) and Dublin Core elements attached to them for traceability. (For a complete list of Dublin Core elements, see Appendix B-Dublin Core Elements, page 40.)

<table>
<thead>
<tr>
<th>Schema Element</th>
<th>Optionality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Root document content&gt;</code></td>
<td></td>
<td>The root resqml document contains non-contextual content of the document including the global UID of the document and Dublin Core metadata.</td>
</tr>
<tr>
<td>Area of Interest</td>
<td>Optional</td>
<td>A GML-based feature representing the area of interest. The area of interest contains a mandatory bounding box and an optional polygon outline.</td>
</tr>
<tr>
<td>Spatial Reference Set</td>
<td>Required</td>
<td>The set of coordinate reference system (CRS) definitions that are used in a document. Any referenced well-known systems will not be included in this set. For more information about CRSs, see Chapter 5.</td>
</tr>
<tr>
<td>Interface Feature Set</td>
<td>Optional</td>
<td>A set of interface features, such as faults and horizons, and their geometry. These data objects by themselves do not represent a coherent model. They are building blocks which can be used to create a model.</td>
</tr>
<tr>
<td>Gridded Volume Set</td>
<td>Optional</td>
<td>A container for 3D grids and related properties.</td>
</tr>
</tbody>
</table>
### RESQML Data Objects and Key Concepts

This section provides an overview of the data objects and key concepts.

A RESQML document may contain faults, horizons, and grids and their respective properties (Figure 2).

![RESQML Data Objects](image)

**Figure 2 Data objects supported in the current version of RESQML.**

#### 2.3.1 Faults and Horizons

Faults and horizons are considered to be interface features, and they may have multiple versions and multiple geometry representations of each version. (For definitions of features, versions and representations, see section 2.3.7, page 7.)
Geometry representations include:

- For horizons: 3D point sets, orthogonal 2D grids, triangulated meshes, hybrid (orthogonal grid plus triangles) and well marker sets (x, y, z and/or md location of horizon markers along wells).
- For faults: 3D point sets, orthogonal 2D grids, triangulated meshes, pillar sets (collection of 3D polylines) and well marker sets (x, y, z and/or md location of horizon markers along wells).

Each fault or horizon representation may have its own set of associated properties. This approach makes it possible to transfer information about the main structural features of an earth model. In the current version of RESQML, faults and horizons may now be independently transferred—a grid is not required (as it was in RESCUE).

However, it is not yet possible to declare that a group of individual features together represent an earth model. (A future version of RESQML will define the concepts of structural organization and structural framework to handle this concept of consistent earth model.)

To summarize, RESQML captures the geometric and topological description of some structural elements of the reservoir, but this description does not include a consistency between these elements or a consistency between these elements and the geocellular (static model) and the reservoir simulation (dynamic model) grid.

### 2.3.2 Grids

RESQML supports the following for grids:

- The grid is a general corner point grid with each cell described by its eight cell nodes and each node, by default, lying on a coordinate line.
- A new indicator clearly shows whether the cell geometry is defined for a cell (addressing a known ambiguity issue in RESCUE).
- Topological indicators are supplied with the grid, not just a geometric description, thereby removing ambiguities in the interpretation of the internal adjacency of a model.
- The grid index origin is preserved and cannot be swapped, thereby preserving the connection to reservoir simulation data, which is often index-based but otherwise not included within a RESQML document.
- For unfaulted grids, each node is described only once but shared among 8 cells.
- Coordinate lines need not be straight and need not be monotonic functions of depth. Additional coordinate line node lists are used on faults to describe fault throw. Additional nodes may also be described individually, e.g., to allow for the description of stair-step faults without forcing the introduction of additional coordinate line node lists.
- XML documents contain explicit information about the complexity of a grid, so that the reading software knows how complex the grid is without a cell-by-cell analysis of the grid geometry.
- Support for multiple grids, e.g., to support multiple reservoir problems, and flexible definition of local grids.
- Local grid refinement (LGR) allows a cell or group of cells within a grid to be further sub-divided for more detailed study.
- Non-standard adjacency list consists of a cell-face-pair list that explicitly represents the adjacency between "non-standard" cells, which avoids the differences in interpretation between different applications for faulted corner point grids.

### 2.3.3 Wells

RESQML handles blocked wells, which consist of the cell list that is intersected by a wellbore, ordered by increasing measured depth, and includes support for multi-lateral wells.

Integration of detailed well objects (wells, wellbores, trajectories, etc.) will be addressed in future versions of RESQML by leveraging the WITSML standard. The RESQML standard has been developed to be compatible with the WITSML standard.
### 2.3.4 Property Data

Property data is implemented in RESQML with property values and property kinds.

Property values, which may be static or dynamic, may be associated with a horizon, a fault, a grid, a blocked wellbore list or a non-standard adjacency list.

Property groups are used to group, for instance, all of the properties at one time or simulation time step within a RESQML document.

Property kinds refer to a RESQML list of standard property names that represent the basis for the commonly used properties in the E&P subsurface workflow. Use of this list allows programmers implementing RESQML to map their software property names to the RESQML standard property software names, which makes it possible for RESQML-enabled software packages to translate property names between each other.

### 2.3.5 Coordinate Reference System

Use of a global CRS allows models to be accurately located on the Earth. The CRS is specified using Geographic Markup Language (GML), which includes EPSG codes to identify coordinates for specific, well known global locations. Each geometric representation is specified with respect to a local coordinate system, which is embedded in the global CRS (with optional rotation, translation and change of units of measure). A Local CRS vertical axis may represent depth (always increases downward, thereby removing the ambiguity that occurred in the previous standard, RESCUE, which permitted depth or elevation) or it can represent time (a frequent user request for RESCUE).

For more information, see Chapter 6, page 26.

### 2.3.6 Bin Grids

Horizons and faults can be represented with 2D grid representations. Most of the time, these grids have a strong meaning: they correspond to the seismic bin grid where the interfaces have been picked.

Because these bin grids can be used to support several 2D grid representations of several interfaces, they are defined independently of any interface. That’s why they are not contained within one particular interface but within the interface feature set of the RESQML document. Thus, any 2D grid representation of any interface can reference any bin grid.

For more information, see Chapter 7, page 37.

### 2.3.7 Feature, Version, and Representations

RESQML uses the following constructs and terminology to most accurately depict data objects in the modeling process.

<table>
<thead>
<tr>
<th>Construct/Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature</td>
<td>A feature represents a concept that is associated with subsurface phenomena. Features may be geological or artificial. Geological features represent domain concepts such as faults and horizons. Artificial features describe non-geologic features such as boundaries and areas of interest.</td>
</tr>
<tr>
<td>Version</td>
<td>A version—which may also (informally) be called an interpretation or opinion—is a single consistent description of a feature. This single consistent description may be the interpretation or opinion of a person or a software package (e.g., an instance in a time step). A feature may have one or more versions.</td>
</tr>
<tr>
<td>Representation</td>
<td>A representation is a digital description of a feature or a version. For example, in the current version of RESQML, a horizon version may be represented by a point set, triangulated surfaces or orthogonal grids. Representations in RESQML must be either in time or in depth; a mixture of time and depth representations is not allowed.</td>
</tr>
</tbody>
</table>
3 Implementing RESQML in a Software Package

This section provides a high-level workflow for implementing RESQML into your software package.

Implementing RESQML requires that you use special software tools to read the schemas provided by Energistics and, optionally, integrating special libraries for HDF5.

Additionally, you can map property names in your software to property kinds in RESQML. For more information, see Chapter 5, page 21.

3.1 File Types and Technology

As a reminder from Chapter 2, a RESQML document may have up to 2 files, as described in the table below.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Required/Optional</th>
<th>Naming Convention</th>
<th>Example File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>Required</td>
<td>File extension = .resqml</td>
<td>Myproject.resqml</td>
</tr>
<tr>
<td>HDF5</td>
<td>Optional (Though most vendors will implement)</td>
<td>File name = the RESQML file name, including the extension. File extension = .h5</td>
<td>Myproject.resqml.h5</td>
</tr>
</tbody>
</table>

Ideally, developers implementing RESQML into a software package, will want to use both XML and HDF5 (for “heavy” data). Using only XML is really only suitable for very small files.

3.2 Implementation Workflow

This high-level workflow describes the basic steps to implement RESQML in your software package.

Step 1: Download Necessary Files

Follow these steps:

1. Download RESQML files from Energistics website (http://www.energistics.org/current-resqml-standards). These files include:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESQML data schema.zip</td>
<td>This zip file includes:</td>
</tr>
<tr>
<td></td>
<td>- RESQML schema files, which include imbedded documentation about RESQML data objects and their representation.</td>
</tr>
<tr>
<td></td>
<td>- Other supporting files for implementing RESQML</td>
</tr>
<tr>
<td>Documentation Files</td>
<td>PDF files containing additional information for domain and IT users about the use of RESQML. For a complete list of documentation, see Section 1.2, page 1.</td>
</tr>
</tbody>
</table>

2. Download HDF5 files from the HDF Group website at http://www.hdfgroup.org/HDF5/.

   HDF5 provides libraries for C, C++, C# and Java. The files required will depend on the programming language in which your software package is developed.

   You may download the pre-built HDF5 libraries, or you may download the source and build the libraries yourself. **Note:** If you build the libraries yourself, you must include both the SZIP and ZLIB external binaries to be able to read compressed RESQML HDF files. Use of SZIP in a commercial
writer requires an additional license through the SZIP group; for more information see http://www.hdfgroup.org/doc_resource/SZIP/Commercial_szip.html.

For more information on how to implement HDF, available tools and tutorials, or to download the libraries, see the HDF Group website at: http://www.hdfgroup.org/HDF5/.

For more information specific to RESQML, see chapter 4, Implementing HDF5, page 11.

**Step 2: Implement XML in Your Software Package**
Implementing XML in your software can be done using one of these technologies:

- **XML parsers** are special software components that can be used with your software package to read and write XML documents such as RESQML. For instructions on how to use an XML parser in your software package, see the instructions from the parser vendor.
- **A proxy generator** may be used to create proxy classes that can serialize and deserialize RESQML objects directly.
  
  In general, use of parsers may be more flexible than proxy generators, but may require more effort to develop and maintain in a software package.

With a **proxy generator**, follow these steps to implement RESQML into your software package.

1. Use one of the many available tools, such as Microsoft XSD or gSOAP, to:
   - Generate RESQML proxy classes that can serialize and deserialize RESQML documents directly. Depending on the tool, proxy generators produce C, C++ or Java code.
   - Generate RESQML proxy classes that can deserialize the Property Kind loader file.

2. Integrate the proxy classes into your software package.

**Step 3: Implement HDF5 in Your Software Package**
Implementing HDF5 is the same whether you use a proxy generator or a parser to implement XML.

Integrate the HDF5 libraries that you downloaded as appropriate for your software package. Libraries consist of:

- File that ships with your product
- Static files that you link to

HDF5 offers libraries for C, C++, C# and Java. For more information on how to implement HDF5, available tools and tutorials, or to download the libraries, see the HDF Group website at: http://www.hdfgroup.org/HDF5/.

For more information specific to RESQML, see chapter 4, Implementing HDF5, page 11.

### 3.3 Reading and Writing RESQML Documents

The basic process of reading and writing RESQML documents in RESQML-enabled software packages is deserializing (reading) the content from your software package's internal representation and serializing (writing) the software's internal data representation.

**3.3.1 Basic Read Process with Proxy Classes and HDF5**

1. Deserialize the RESQML document using the proxy classes to create an in-memory representation of the file.

2. Create your internal objects from objects in the in-memory representation.

3. If the RESQML file indicates that data is stored in the HDF5 file, read from the HDF5 file as needed to populate your internal objects.
3.3.2 Basic Write Process with Proxy Classes and HDF5
1. Create an in-memory representation of the XML file using the proxy classes from your internal objects.
2. Write the heavy data to the HDF5 file and update the in-memory representation to indicate the location of the data in the HDF5 file.
3. Serialize the in-memory representation to an XML file.

3.3.3 Basic Read Process with XML Parser
1. It is recommended that you first validate a RESQML document with an XML validating tool. Validating the XML file before reading ensures that you have a data file that is compliant with the RESQML schema. If the file is not compliant, then a RESQML-enabled application may not be able to read it.
2. Read the RESQML document.
   The parser transforms the content to the appropriate format, depending on the parser used, for example, a Document Object Model (DOM) hierarchy for QtDom or simple API for XML (SAX).
   The software package code then parses the hierarchy/format, extracts the data, and transforms it into the software package's internal representation.

3.3.4 Basic Write Process with XML Parser
1. Transform the software's internal representation into the parser format, such as DOM hierarchy.
2. Write the content in the parser format to the file.

Note: If an HDF5 file is included, it is handled separately, similarly as described above.


4 Implementing HDF5
HDF5 is a set of open file formats and libraries designed to store and organize large amounts of numerical data, and improve speed and efficiency of data processing. Specifically, it provides: machine/architecture-independent “binary” format (supported on Windows, Linux, etc. APIs available in C++, Java, .NET, etc); hyper-slabbing of array data so that IJK sub-arrays may be extracted from the RESQML document without reading the entire data file; and built-in data compression.

HDF5 is considered optional for a RESQML implementation. However, with the volume of data involved in reservoir modeling and simulation, use of HDF5 is highly recommended.

- For a high-level workflow on how to implement RESQML including HDF5, see chapter 3, Overview of RESQML HDF5
- For more information on HDF5, available tools and tutorials, or to download the libraries, see the HDF Group website at: http://www.hdfgroup.org/HDF5/.

4.1 Overview of RESQML HDF5
For detailed HDF5 conventions and formats used with RESQML, see section 4.6, HDF5 File Conventions and Formats, page 14.

The “heavy” data associated with a RESQML object may be stored in an HDF file in one or more “datasets.” Datasets are collected hierarchically in the file in groups.

The location in the file of the Datasets for a particular object is provided by an element for the object in the XML. This element is of type cs_resqmlHdfGroup, and it provides the name of the group containing the Dataset. The name of the Dataset is fixed depending on the type of data being stored (see 4.6.1 Group/Dataset Hierarchy and Name Guidelines page 14).

The hierarchy of data within a RESQML HDF5 file is by convention as follows:

- A top-level group named RESQML contains all RESQML data.
- Each object that is stored in the file has its own group. This group is named with the global unique ID (GUID) of the object.
- Within the object’s group, one or more Datasets contain the data.

Referencing a Dataset can be done using a path-like naming convention. For example, the following references the explicitNodeList dataset for a grid:

/RESQML/32428d9d-6d3a-4860-8fcb-a6fa9022d043/explicitNodeList

The contents of a RESQML HDF5 file may be inspected using tools provided by the HDF Group or other third-party vendors. For example, the H5DUMP application lists the attributes, groups, Datasets, and data found in an HDF5 file.

4.2 Adding HDF5 to your Code
HDF5 is provided as dynamic link libraries with a C-like application interface. You may add references to these DLLs directly using the C headers provided with the libraries. You may similarly link to these libraries in a .NET application using P/Invoke, or in a Java application using JNI.

A number of language-specific implementation layers are available on the HDF Group website. These implementations “wrap” the C interface, and they may provide a more intuitive way of accessing HDF5. These implementations vary in terms of the amount of the HDF5 API that is covered, and they may not be appropriate for advanced use of HDF5. However, RESQML requires only a subset of the HDF5 API, so you may find one of these implementations suitable for your development needs.

The rest of this chapter illustrates a number of typical RESQML HDF5 scenarios using the C API.
4.3 HDF5 Handles
The HDF5 API uses a handle-based pattern. This means that the typical programming scenario is as follows:

- Create or open an HDF5 element, such as a file or Dataset. HDF5 returns a handle of type hid_t.
- Interact with the HDF5 element using the handle.
- Close the handle using the appropriate “close” method. For example, you can close a file handle using the H5Fclose function.

In the examples below, the closing of handles is omitted for clarity.

4.4 Reading from a RESQML HDF5 File
Reading data from a RESQML HDF5 file involves a number of steps. This brief overview shows the key steps in this process, omitting details, such as error handling and handle closing, for clarity.

4.4.1 Create the HDF5 Types Required for RESQML
HDF5 has a number of built-in atomic types available. RESQML, however, makes use of “compound” data types for some of its information. Before these can be used, your application must tell HDF5 what these types look like so that it can read and write the data properly.

The creation of types is done through the H5T family of API functions. One example is shown here: the Point3D compound type, which is used to store x/y/z coordinates in various other RESQML types.

Creation of a compound type involves creation of the type identifier and then addition of the elements. The addition of the elements tells HDF5 what will be in the file for the type, and how to map this information with the structure’s location in memory.

```c
hid_t Point3DID = H5Tcreate(H5T_COMPOUND, sizeof(Point3DImplementation));
herr_t e = H5Tinsert(Point3DID, "x", HOFFSET(Point3D, x), H5T_NATIVE_DOUBLE);
e = H5Tinsert(Point3DID, "y", HOFFSET(Point3D, y), H5T_NATIVE_DOUBLE);
e = H5Tinsert(Point3DID, "z", HOFFSET(Point3D, z), H5T_NATIVE_DOUBLE);
```

4.4.2 Reading from a RESQML HDF5 File
Reading data from an HDF5 file involves these steps:

(i) Open the file
(ii) Open the dataset and read the data
(iii) Close the file

4.4.2.1 Open a RESQML HDF5 File
To open a RESQML HDF5 file, use the H5Fopen function from the H5F family of functions. Many options are available, but this example shows how to open a file for reading only.

```c
hid_t FileID = H5Fopen(_FileNameH5F_ACC_RDONLY, H5P_DEFAULT);
```

4.4.2.2 Open a Dataset and Read Data
Reading data from a Dataset involves a number of steps: (i) open the hierarchical groups containing the Dataset, (ii) open the Dataset, (iii) get the Dataspace information to determine the rank and dimensions of the data in the Dataset, (iv) read the data, and (v) close all the handles when done.

The path to the Dataset—including the groups it is contained in—is provided in the XML file. This path may be broken into group components, and each of these groups may be opened in turn. Alternatively, the complete path information may be passed to the dataset-opening function. This sample opens the dataset directly using the complete path information.

```c
hid_t DatasetID = H5Dopen2(FileID, DatasetName, H5P_DEFAULT);
```
After the Dataset is open, you may query the Dataspace to get information on the rank and dimensions of the data. This information is usually also contained in the RESQML XML file, but you may still want to read it from the HDF5 file to verify the integrity of the file.

```c
hid_t DataspaceID = H5Dget_space(DatasetID);
int ndims = H5Sget_simple_extent_ndims(DataspaceID);
if (ndims == ExpectedDimensionsRank)
    ndims = H5Sget_simple_extentDims(DataspaceID, Dimensions, NULL);
```

Reading the data may be done all at once or in pieces called *hyperslabs*. Reading using hyperslabs can be more efficient in terms of memory usage, but for clarity the sample below reads all data into a memory buffer at once.

```c
herr_t e = H5Dread(DatasetID, TypeID, H5S_ALL, H5S_ALL, H5P_DEFAULT, Buffer);
```

### 4.5 Writing Data to a RESQML HDF5 File

Writing data to a RESQML HDF5 file is similar to reading data from a file. You must create the appropriate types as before, and all handles must be closed when they are no longer required.

#### 4.5.1 Enabling Compression

RESQML is designed to store large quantities of data, and HDF5 helps with this by compressing the data as it is saved to a file.

HDF5’s implementation of compression provides the writer with a trade-off between compression efficiency and access speed for readers. It does this by splitting the data into fixed-sized “chunks” and compressing each chunk individually.

The larger a chunk is, the more efficient the compression will be. However, this efficient compression is at the cost of reading efficiency, because HDF5 needs to read and decompress into memory larger parts of the file to get at a particular part of the data.

Smaller chunks improve the reading efficiency of HDF5, because less data must be read and decompressed into memory to get at a particular part of the data. The downside to this is that the compression is less efficient and resulting file sizes will be larger.

When determining a chunk size to use, consider how the data will be accessed. For example, a likely use case for applications reading explicit grids is to extract only particular layers out of the data using hyperslabbing. A chunking setup that supports this scenario is to make a chunk for each layer. This approach makes accessing the data a layer at a time maximally efficient, while still giving HDF5 a large enough block of data to compress.

Enabling compression involves creating an HDF5 Parameter that is used when creating a Dataset. This parameter contains the size of the chunk and the compression properties.

```c
hid_t DatasetParameterID = H5Pcreate(H5P_DATASET_CREATE);
if (DatasetParameterID > 0)
{
    herr_t e = H5Pset_chunk(DatasetParameterID, ChunkRank, ChunkDimensions);
    e = H5Pset_deflate(DatasetParameterID, 1);
}
```

#### 4.5.2 Creating the Groups that Contain the Dataset

Groups are handled slightly differently when writing to a file: HDF5 does not create groups automatically when a Dataset path is supplied. Your application must do one of the following:

- If a group doesn’t yet exist, explicitly create each group.
- If a group does exist, open it before creating the dataset.

With the current HDF5 format, only two groups are required:

- The top-level RESQML group.
• A group for the object being stored, which has the object’s GUID as its name.
The top-level group is created using the FileID:

hid_t RESQMLGroup = H5Gcreate2(FileID, "RESQML", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

The object group is created using the RESQMLGroupID:

hid_t ObjectGroupID = H5Gcreate2(RESQMLGroup, ObjectGuidString, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

Remember to close these IDs when you are finished with them.

4.5.3 Creating the Dataspace
The Dataspace tells HDF5 how much data is to be stored and how it is configured in terms of rank and dimensions.

hsizes_t Dimensions[Rank];
Dimensions[0] = Length0;
//Etc.
hid_t DataspaceID = H5Screate_simple(Rank, Dimensions, NULL);

4.5.4 Creating the Dataset
The Dataset is created using the ObjectGroupID, the DataspaceID, the DatasetCompressionParameter, and the Dataset name.

hid_t DatasetID = H5Dcreate2(ObjectGroupID, DatasetName, TypeID, DataspaceID, H5P_DEFAULT, DatasetParameterID, H5P_DEFAULT);

4.5.5 Writing the Data
Writing of the data to the Dataset is done using the DatasetID the TypeID of the data being stored, and a pointer to the data buffer. Hyperslabbing may be used to write the data a portion at a time.

herr_t e = H5Dwrite(DatasetID, TypeID, H5S_ALL, H5S_ALL, H5P_DEFAULT, Buffer);

4.6 HDF5 File Conventions and Formats
For HDF5, RESQML does not have a machine-readable schema. This section describes formats and conventions for implementing HDF5 with RESQML.
The DDL form of the type is explained at http://www.hdfgroup.org/HDF5/doc/ddl.html.
The data for a given RESQML object may be contained in more than one Dataset. These Datasets will be contained within one HDF Group, which is referenced in the XML file.

4.6.1 Group/Dataset Hierarchy and Name Guidelines
RESQML data is contained in an HDF5 file in the following hierarchy:
• The first level is RESQML. All RESQML data is contained in groups or datasets beneath this group.
• Data for individual RESQML objects is contained within a Group under the RESQML group. The Group name is the GUID of the RESQML object.
• Within the RESQML object’s group, the data is contained in one or more Datasets. Each Dataset is named explicitly for the data it contains.
For example, the topologyFlags data for a Corner Point Grid may be contained in the following Dataset:
/RESQML/c95da709-9262-45f0-a738-e3a7f5855f35/topologyFlags
4.6.2 Common Types
A point3d is defined as a compound type containing one X value, one Y value and one Z value.

```plaintext
DATATYPE "point3d" H5T_COMPOUND
{
    H5T_NATIVE_DOUBLE "x";
    H5T_NATIVE_DOUBLE "y";
    H5T_NATIVE_DOUBLE "z";
}
```

4.6.3 Structural Types
An edge is defined as a compound type containing a pair of integer values. Each of these values corresponds to the index of one point in a list of points. Both of these points must be vertices of the same triangle. The order in which the indices are given to define a split edge is not important.

```plaintext
DATATYPE "edge" H5T_COMPOUND
{
    H5T_NATIVE_UINT "ix1";
    H5T_NATIVE_UINT "ix2";
}
```

A triangle is a compound type containing a triplet of integer values. Each of these values corresponds to the index of one point in a list of points. The order in which the indices are given to define a triangle is not important.

```plaintext
DATATYPE "triangle" H5T_COMPOUND
{
    H5T_NATIVE_UINT "ix1";
    H5T_NATIVE_UINT "ix2";
    H5T_NATIVE_UINT "ix3";
}
```

4.6.4 Structural Representations
- **Point set patch.** A point set patch is basically a list of points that are defined in a 3D space. The order of the points in this list is not important. If point properties are present, they must be ordered to coincide with the order of the points in this Dataset.

```
GROUP guidPointSetPatch
{
    DATASET "point3dSet"
    {
        DATATYPE "point3d"
        DATASPACE SIMPLE { nPoints }
    }
}
```

- **Grid 2D patch.** A grid 2D patch is a 2D array of Z values. These Z values represent the vertical coordinates of the nodes of the grid. The data is stored in a 2D Dataspace that is NJ by NI in size. If any properties are present, they must be ordered to coincide with the order of Z values in this Dataset.

```
GROUP guidGrid2DPatch
{
    DATASET "zValueSet"
    {
        DATATYPE H5T_NATIVE_DOUBLE
        DATASPACE SIMPLE { NJ, NI }
    }
}
```
Triangulated Patch. A triangulated patch is a list of points, defined in a 3D space, which represent the vertices of triangles. The point3dSet Dataset contains the point3d values that represent the vertices of the triangles. If any vertex properties are present, they must be ordered to coincide with the order of the vertices in this Dataset.

The triangleNodeIndexSet Dataset contains the information that defines the triangles in the patch. If any triangle properties are present, they must be ordered to coincide with the order of the triangles in this Dataset.

The splitEdgeNodeIndexSet Dataset contains the information describing the split edges in the patch (if any).

GROUP guidTriangulatedPatch
{
  DATASET "point3dSet"
  {
    DATATYPE "point3d"
    DATASPACE SIMPLE { nVertices }
  }
  DATASET "triangleNodeIndexSet"
  {
    DATATYPE "triangle"
    DATASPACE SIMPLE { nTriangles }
  }
  DATASET "splitEdgeNodeIndexSet"
  {
    DATATYPE "edge"
    DATASPACE SIMPLE { nSplitEdges }
  }
}

Pillar Set Patch. A pillar set patch is intended to represent the picking of a fault on several slices of the seismic block (inline, crossline, time slice or any other slices, even non-planar). On each slice, the user picks a fault by means of one or more polylines, which are called pillars.

The pillarSet Dataset contains a 2D Dataspace with the point3d values for the patch. The first dimension in the Dataspace is the number of pillars. The second dimension is the number of points that the largest pillar holds. For pillars that have fewer than this maximum number of points, the extra locations are filled with { NaN, NaN, NaN }. If any point properties are defined, they must be ordered to coincide with the order of the points in this dataset.

GROUP guidPillarSetPatch
{
  DATASET "pillarSet"
  {
    DATATYPE "point3d"
    DATASPACE SIMPLE { nPillars, nMaxNodes }
  }
}

Hybrid Patch. A hybrid grid representation is composed of one grid 2D patch and one to many triangulated patches. Thus, a hybrid patch is either a grid 2D patch or a triangulated patch and must be define as such.

4.6.5 Grid Types

• A splitLineReferenceCorner is an enumeration defining the values for iCorn and jCorn in the splitNodeReference and splitLineReference classes.
DATATYPE "splitLineReferenceCorner" H5T_ENUM
{
    H5T_NATIVE_CHAR;
    "zero" 0;
    "one" 1;
}

• A splitNodeReference is a compound type containing the information for a single splitNodeReference.

DATATYPE "splitNodeReference" H5T_COMPOUND
{
    H5T_NATIVE_INT i;
    H5T_NATIVE_INT j;
    H5T_NATIVE_INT k;
    splitLineReferenceCorner iCorn;
    splitLineReferenceCorner jCorn;
    splitLineReferenceCorner kCorn;
    H5T_NATIVE_UINT reference;
}

• A splitLineReference is a compound type containing the information for a single splitLineReference.

DATATYPE "splitLineReference" H5T_COMPOUND
{
    H5T_NATIVE_INT i;
    H5T_NATIVE_INT j;
    splitLineReferenceCorner iCorn;
    splitLineReferenceCorner jCorn;
    H5T_NATIVE_UINT reference;
}

• Non-standard adjacency IJK cell face pairs can now be stored in HDF. Here is the HDF schema for IJK cell face pairs:

    //HDF Representation of a grid cell
    DATATYPE "gridIJKCell" H5T_COMPOUND {
        H5T_NATIVE_INT "gridID";
        H5T_NATIVE_UINT "i";
        H5T_NATIVE_UINT "j";
        H5T_NATIVE_UINT "k";
    }

    //HDF Representation of a face direction inside a cell, letter indicates the axis the face is perpendicular to, minus indicated if the face is in contact with the previous cell along the axis or the next cell along the axis.
    DATATYPE "gridIJKCellFaceDirection" H5T_ENUM {
        H5T_NATIVE_CHAR "I-" 0 "J-" 1 "K-" 2 "I+" 3 "J+" 4 "K+" 5
    }

    //HDF representation of an individual face given by the pair (cell,direction)
    DATATYPE "gridIJKCellFace" H5T_COMPOUND {
        gridIJKCell "cell"
        gridIJKCellFaceDirection "face"
    }

    //HDF representation of a pair of faces
    DATATYPE "gridIJKCellFacePair" H5T_COMPOUND {
        gridIJKCellFace "face1";
        gridIJKCellFace "face2";
4.6.6 Grid Representations

A CornerPointGrid Group contains the Datasets required to describe a corner point grid. It contains a Dataset containing the explicit node values, a Dataset containing the grid’s topologyFlags, a Dataset containing the splitLineReferences (if any), a Dataset containing the splitNodeReferences (if any), and if the splitNodeCount is greater than zero, a Dataset containing the splitNodes.

- The **explicitNodeList** Dataset contains a 2D Dataspace describing the explicit point3d values in the grid. The first dimension in the Dataspace is \([NK+\text{gapCount}+1]\) in order of increasing K. The second dimension of the Dataspace is \([(NI+1)(NJ+1)+\text{splitLineCount}] with the values ordered such that the \([(NI+1)(NJ+1)] values are first, ordered with I moving fastest, followed by the splitLineCount values. **REQUIRED.**

- The **splitLineReferences** Dataset contains a 1D Dataspace of splitLineReference classes. Its length is splitLineReferenceCount (which is not defined in the XML). Required only if splitLineCount is greater than zero.

- The **topologyFlags** Dataset contains a 3D Dataspace in [NK, NJ, NI] with the topologyFlags for the grid. The type of the data is H5T_NATIVE_SHORT. **REQUIRED.**

- The **splitNodes** Dataset contains a 1D Dataspace containing the point3d values representing the splitNodes. Its length is splitNodeCount (which is not defined in the XML). Required only if splitNodeCount is greater than zero.

- The **splitNodeReferences** Dataset contains a 1D Dataspace of splitNodeReference classes. Its length is splitNodeReferenceCount (which is not defined in the XML). Required only if there are splitNodeReferences in the model.

GROUP guidCornerPointGrid
{
  DATASET "explicitNodeList"
  {
    DATATYPE "point3d"
    DATASPACE SIMPLE { NK+\text{gapCount}+1, (NI+1)(NJ+1)+\text{splitLineCount} }
  }
  DATASET "splitLineReferences"
  {
    DATATYPE "splitLineReference"
    DATASPACE SIMPLE { \text{splitLineReferenceCount} }
  }
  DATASET "topologyFlags"
  {
    DATATYPE H5T_NATIVE_UCHAR
    DATASPACE SIMPLE { NK, NJ, NI }
  }
  DATASET "splitNodeReferences"
  {
    DATATYPE "splitNodeReference"
    DATASPACE SIMPLE { \text{splitNodeReferenceCount} }
  }
  DATASET "splitNodes"
  {
    DATATYPE "point3d"
    DATASPACE SIMPLE { \text{splitNodeCount} }
  }
}
4.6.7 Property Representations

A property Group contains one Dataset named "values" which contain the property values. The dimension of the Dataspase in the Dataset corresponds to the dimension of the property data array.

The data type can be one of the following datatypes:

- H5T_NATIVE_SCHAR (8 bits, xml "byte")
- H5T_NATIVE_SHORT (16 bits, xml "short")
- H5T_NATIVE_INT (32 bits, xml "int")
- H5T_NATIVE_LLONG (64 bits, xml "long")
- H5T_NATIVE_FLOAT (xml "float")
- H5T_NATIVE_DOUBLE (xml "double")

If the data type is not either H5T_NATIVE_FLOAT or H5T_NATIVE_DOUBLE, and if the data contains null values, the Dataset must have an attribute called "nullValue" of the same type that contains the null value.

If the data type is either H5T_NATIVE_FLOAT or H5T_NATIVE_DOUBLE, the null value must be NaN ("not a number") and the "nullValue" attribute is not required.

- 1D Property Values. A 1D property Group contains one Dataset named "values" which contains the property values. The Dataspase for the Dataset is 1D.

```xml
GROUP guidProperty
{
  DATASET "values"
  {
    DATATYPE <type>
    DATASPACE SIMPLE { NUM_VALUES }
    ATTRIBUTE "nullValue"
    {
      DATATYPE <type>
      DATASPACE SCALAR
    }
  }
}
```

- 2D Property Values. A 2D property Group contains one Dataset named “values” which contains the property values. The Dataspase for the Dataset is 2D: the first dimension is NJ, and the second dimension is NI.

```xml
GROUP guidProperty
{
  DATASET "values"
  {
    DATATYPE <type>
    DATASPACE SIMPLE { NJ, NI }
    ATTRIBUTE "nullValue"
    {
      DATATYPE <type>
      DATASPACE SCALAR
    }
  }
}
```
• **3D Property Values.** A 3D property Group contains one Dataset named "values" which contains the property values. The Dataspace for the Dataset is 3D; the first dimension is NK, the second dimension is NJ, and the third dimension is NI.

GROUP guidProperty
{
    DATASET "values"
    {
        DATATYPE <type>
        DATASPACE SIMPLE { NK, NJ, NI }
        ATTRIBUTE "nullValue"
        {
            DATATYPE <type>
            DATASPACE SCALAR
        }
    }
}
5 Using RESQML Property Kinds

RESQML includes a list of standard property names that represent the basis for the commonly used properties in the E&P subsurface workflow. Use of this list allows programmers implementing RESQML to map their software property names to the RESQML standard property software names, which makes it possible for RESQML-enabled software packages to translate property names between each other.

For example, if Package A names “porosity” as POR and Package B names it PORO, and both packages are RESQML enabled, the packages can recognize that these properties are equivalent because they refer to the same RESQML parent property, e.g., “porosity”. They can also exchange without ambiguity the numerical values that are recorded because those values will be in the same unit of measure.

This chapter lists and describes the available files to help you with this mapping process.

5.1 Available Files

To help you implement RESQML property name translation capabilities in your software package, these files are available, which can be downloaded at [http://www.energistics.org/current-resqml-standards](http://www.energistics.org/current-resqml-standards).

<table>
<thead>
<tr>
<th>File and Type</th>
<th>Description/Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema: ./ancillary/cs_enumValuesResqml.xsd</td>
<td>Defines (or constrains) the data in the RESQML loader file.</td>
</tr>
<tr>
<td>Loader File: ./ancillary/enumValuesResqml.xml</td>
<td>Machine-readable file containing actual property name data included in RESQML.</td>
</tr>
<tr>
<td></td>
<td>Programmers implementing RESQML can use this file to help them tailor the Property Kind section of the RESQML schema.</td>
</tr>
<tr>
<td></td>
<td>Note: Implementation of property name translation requires that someone in your organization reviews the RESQML property names and maps them to the corresponding property names name in your software package.</td>
</tr>
</tbody>
</table>

5.1.1 Overview of the Files

5.1.1.1 The schema File: ./ancillary/cs_enumValuesResqml.xsd

This schema constrains the content of the enumValuesResqml.xml file. The following hierarchy of elements matches the pattern in the equivalent files from WITSML and PRODML (Energistics’ drilling and production data exchange standards).

```
enumListSet
    enumList
        name
        description
        namingSystem
        value
            name
            description
            version
            deprecated
            replacedBy
```

These values are of relevance to all lists. Note, however, that the version, deprecated and replacedBy elements are not used in this initial version of the data for RESQML. Additional elements that are only relevant to specific lists have been defined within a choice. Within a list, the same choice will be exercised for all values in that list.
5.1.1.2 The loader file: ./ancillary/enumValuesResqml.xml
The list named “ResqmlPropertyKind” defines the list of property kinds that are the basis for all of the more detailed properties. Every kind (except the root) defines a parent kind. The child kind represents a specialization of the semantics defined by the parent kind. The following base hierarchy is represented in the list:

- **RESQML root property** represents the root of all property kinds.
- **categorical** represents a property with enumerated string values.
- **continuous** represents a property with floating point values.
- **quantity** represents a property whose floating point values have a unit of measure.
- **unitless** represents a property with unitless floating point values.
  - **discrete** represents a property with integer values.

The following table lists and describes how the information is presented in the loader file (for each "well known" ResqmlPropertyKind). ("Well known" refers to specific items within a particular naming system (dictionary) that are generally known and understood by the user community. A company may have a proprietary list that is well known by its customers.)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description/Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>RESQML’s name of the property (formalized string).</td>
</tr>
<tr>
<td>description</td>
<td>A brief description or definition of the property.</td>
</tr>
<tr>
<td>isAbstract</td>
<td>True (&quot;1&quot; or &quot;true&quot;) indicates that the property is abstract and cannot be used except as a parent of a concrete property. False (&quot;0&quot; or &quot;false&quot;) or not given indicates a concrete property that can be instantiated.</td>
</tr>
<tr>
<td>parentKind</td>
<td>Parent element of the property, if the property is derived from another property. A child property must have the same unitOfMeasure and dimensionalClass values as its parent.</td>
</tr>
<tr>
<td>unitOfMeasure</td>
<td>Unit of measure of the property. Applies to specializations of the “quantity” kind. RESQML has chosen to limit all quantities to only allow one unit of measure for all values of the same kind. The loader file also contains a list named “MeasureClass”, which associates a single unit of measure with a class of quantity. This list has been included in the property list as specializations of the “quantity” property kind, along with the associated unit of measure. All specializations of a kind must use the same unit of measure as the parent. To allow a continuous property to have a mandatory unit of measure element in the schema, a special unit of “NONE” has been defined for the “unitless” kind.</td>
</tr>
<tr>
<td>Element</td>
<td>Description/Purpose</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| dimensionalClass   | The dimensional analysis of the unit of measure. Applies only to specializations of the “quantity” kind. Its purpose is to provide additional clarification about the underlying nature of the property and unit. For example, length per time would be indicated by "L/T". The following nomenclature is used:  
A = angle (radian)  
B = light amount (cd)  
C = electrical current  
F = force  
K = temperature (Kelvin)  
L = length  
M = mass  
N = amount of substance (mole)  
S = solid angle (sr)  
T = time  
1 = dimensionless  
2 = squared  
3 = cubed  
4 = 4th power  
5 = 5th power  
6 = 6th power  
7 = 7th power  
8 = 8th power  
For consistency, the values are broken into numerator and denominator, separated by a slash (/), and in alphabetical order (LM, not ML). To allow a continuous property to have a mandatory dimensional class element in the schema, a special class of “0” has been defined for the “unitless” kind. |

5.1.1.3 Type ResqmlPropertyKind in typ_catalogResqml.xsd
The type ResqmlPropertyKind defines an enumerated list of names from the loader file list of the same name. That is, there is a one-to-one correspondence between the values in this list and the names in the loader file. The type constrains some values allowed for elements resqmlKind and parentResqmlKind in a RESQML message, and the loader file provides additional metadata about the name.

5.1.1.4 propertyKind structure
In a RESQML message, a propertyKind structure must be created to represent property values. The propertyKind may represent a “standard” kind from the loader file or it may represent a local property kind, which specializes a standard kind.

For example, Package A may add a property, porosity, while Package B may add a property that is a specialization of porosity.

```xml
<propertyKindSet>
  <propertyKind uid="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa">
    <dc:title>POR</dc:title>
    <namingSystem>urn:resqml:this-company.com:package-a</namingSystem>
    <isAbstract>false</isAbstract>
    <resqmlKind>porosity</resqmlKind>
    <unitOfMeasure>Euc</unitOfMeasure>
    <dimensionalClass>ratio(L^3)</dimensionalClass>
  </propertyKind>
</propertyKindSet>
```
Note that RESQML intends to publish a dictionary of more specialized property names with a specific namingSystem assigned to it. If this dictionary is used, then the above dc:title values would be from the RESQML dictionary when the RESQML namingSystem is specified.

5.1.1.5 Property Values
After you have found or created your property kind in the propertyKindSet enumeration list, you can create/read a propertyValues element to store/read the numerical data of one property.

For the current version of RESQML, the definition of these propertyValues has to be based on three subclasses depending on the parentClass of the corresponding propertyKind:

- **continuous.** Values of associated property can be any floating point number.
- **discrete.** Values of associated property can only be an integer number.
- **categorical.** Values of associated property belong only to a specific enumeration of values associated to the property kind. Then each property can associate the value inside the enumeration (by reference to a code) to a string indicating the actual meaning.

So for example, a user must ensure that a “resqml property discrete values” corresponds to a “resqml property discrete kind”.

Note that for future RESQML versions, more "dimensional" representation types (like continuous vector or continuous tensor) could be added.

5.2 Process Overview
There are many possible ways to implement RESQML standard property names into your software package. The following steps comprise the high-level process.

1. Download files (see Section 5.1 above).
2. Map your property names to RESQML standard property names. How you do this mapping will vary depending on whether you use RESQML well known property kinds or proprietary property kinds:
   - **If you want to reuse the Resqml well known Property kinds:**
     Mapping requires that your programmers and/or domain experts review the resqmlUnitOfMeasure and ResqmlPropertyKind lists, paying close attention to the property description. That is, you are looking to map properties based on the same (or more specialized) meaning, not necessarily the same names.
     Then map the corresponding RESQML property kind name to Proprietary name in your software package and realize the Unit of measure transformation between your Unit System and the Resqml Unit System (SI), if needed.
   - **If you want to add a other proprietary property kinds:**
     Either add your own proprietary kinds as local property kinds into the cs_resqmlPropertyKindSet.xsd of your resqml document by referencing some resqml standard property kinds.
     Or, if the property kind you are adding could be a standard one, ask Energistics to update the enumValuesResqml.xml file to add your property kind.
3. Implement property name mapping into your software package.

   Use the information in the loader file to determine how best to populate property kinds in a RESQML document (as defined in the Property Kind section of the RESQML schema) for your software package.

   There are many possible ways that property kinds may be implemented based on an individual software package’s design.
6 Implementing Coordinate Reference Systems and Coordinate Systems

This chapter describes how coordinate reference system (CRS) information is used in a RESQML document.

6.1 Introduction

Use of a global CRS allows models to be accurately located on the Earth. Each geometric representation is specified with respect to a local coordinate system (LCS), which is embedded in the global CRS. That LCS must then be located and oriented (proper rotation) with respect to the parent CRS. This ability to properly locate and rotate an LCS within a CRS helps ensure consistency and coherency when data are brought together from different packages and sources.

The CRS is specified using Geographic Markup Language (GML), which includes EPSG codes to identify coordinates for specific, well known global locations.

6.1.1 Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPSG Code</td>
<td>The RESQML schema can use EPSG codes for geolocalization as part of defining a coordinate reference system (CRS). The European Petroleum Survey Group (EPSG) publishes a database that includes codes for easy reference to well known global locations, such as the elevation of the mean sea level of a specific body of water or of the summit of a specific mountain. The EPSG (<a href="http://www.epsg.org/">http://www.epsg.org/</a>), the globally recognized experts on geodetic issues, has been absorbed into the Surveying and Position Committee of the International Association of Oil &amp; Gas Producers (OGP), which is now the owner of the EPSG database of Geodetic Parameters and assigned codes.</td>
</tr>
</tbody>
</table>

6.2 Local Coordinate Reference System

A local coordinate system (LCS) has these characteristics:

- Always uses SI units of measure
- Its z-axis is always oriented down (towards the center of the Earth)
- The axes form a left-handed system.

For each LCS, these items can be specified within its parent CRS:

- Its areal rotation
- Its origin

To locate and rotate the LCS in its parent CRS, the minimum information required is:

- Units of measure of the parent global 2D projected CRS system.
- Units of measure of the parent 1D vertical CRS.
- Axes names and their order in the parent global 2D projected CRS.
- The specification of whether the parent vertical CRS axis is increasing up or down.

This information is always specified in a de-normalized form, and is also available from the full CRS definitions. A full explanation of how the rotation and location of the local coordinate system LCS is specified is described below.
6.3 Global Coordinate Reference System

6.3.1 Defining the Global 2D CRS
Each RESQML document defines one and only one global 2D CRS, which is a projected Cartesian CRS. The global 2D CRS can be defined explicitly using one of the following methods:

- Using the GML projected CRS XML construct
- In the case where a well-known projected CRS is being used, using a URN containing the appropriate EPSG code, for example: `urn:ogc:def:crs:EPSG:7.1:32056`

As well as supplying either an EPSG code or a full GML definition of a projected CRS, we must also provide the areal unit of measure of the projected CRS and an enum, which captures both the well known axes names and their order (see below). Although this leads to de-normalization (namely this information can be recovered from the GML or from the EPSG code) it allows non-geospatially aware applications to interpret a RESQML document.

In some circumstances either the geospatial location of a model is unknown or, for confidentiality reasons it cannot be used. To handle this case, use the following urn:

`urn:ogc:def:crs:RESQML:1.0:AREAL`

In this case, the areal unit of measure and axes names and order are only available through the de-normalized parameters.

6.3.2 Defining the Global Vertical CRS
Each RESQML document defines one and only one global vertical CRS, which can be defined using either of the following:

- A URN for a well-known vertical CRS
- The GML vertical CRS construct.

In addition to providing a URN or full GML definition, you must supply:

- The unit of measure
- The orientation of coordinate system; whether it is orientated up (pointing away from the center of the Earth) or down (pointing towards the center of the Earth).

In some circumstances either the geospatial location of a model is unknown or, for confidentiality reasons it cannot be used. To handle this case, use the following urn:

`urn:ogc:def:crs:RESQML:1.0:VERTICAL`

In this case, the vertical unit of measure and whether the coordinate system is oriented up or down are only available through the de-normalized parameters.

6.3.3 Defining Global Time CRS
If a RESQML document contains any feature in a time domain, then one and only one global time CRS must be defined.

The global time CRS is always orientated downwards (towards the center of the Earth), with a unit of measure seconds, however its origin needs to be defined. The origin must be with respect to the global vertical CRS and is referred to as the seismic reference datum (SRD) of the document.

Figure 3 shows a global time CRS, defined above mean sea level (MSL).
6.4 Defining the Local Coordinate Reference System

Each RESQML data object (for example, horizon or fault) is defined with respect to a local 3D Cartesian CRS, which must be defined in the RESQML document (after the Global CRS) and are always defined with respect to the global 2D CRS and either the global vertical CRS or global Time CRS.

The RESQML development team considered using a GML definition for the local Cartesian CRS, however, decided to define a schema for the construct based on the following definitions.

6.4.1 Areal Local Coordinate Reference System

The areal definition of a local CRS is defined the same way for both depth and time local 3D CRSs.

An areal local coordinate system can be translated and/or rotated with respect to its parent global 2D CRS; however, its unit of measure are always meters.

The orientation of the axis of an areal local coordinate system is such that the x-axis is always 90 deg clockwise from the y-axis (Figure 4).

Figure 4 For an areal local coordinate system, the x-axis is always 90 deg clockwise from the y-axis.

The definition of the rotation angle and translation is dependent on the axis names and order within the parent global projected CRS.
The EPSG dataset (version 7.1) defines the following axis name combinations for Projected CRS's.

<table>
<thead>
<tr>
<th>Axis 1 – Name</th>
<th>Axis 2 – Name</th>
<th>RESQML enum</th>
<th>Count of Projected CRSs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easting</td>
<td>Northing</td>
<td>easting northing</td>
<td>2238</td>
</tr>
<tr>
<td>Northing</td>
<td>Easting</td>
<td>northing easting</td>
<td>816</td>
</tr>
<tr>
<td>Westing</td>
<td>Southing</td>
<td>westing southing</td>
<td>28</td>
</tr>
<tr>
<td>Southing</td>
<td>Westing</td>
<td>southing westing</td>
<td>2</td>
</tr>
<tr>
<td>Westing</td>
<td>Northing</td>
<td>westing northing</td>
<td>2</td>
</tr>
<tr>
<td>Northing</td>
<td>Westing</td>
<td>northing westing</td>
<td>13</td>
</tr>
<tr>
<td>First local axis</td>
<td>Second local axis</td>
<td>N/A</td>
<td>2</td>
</tr>
<tr>
<td>Topocentric East</td>
<td>Topocentric North</td>
<td>N/A</td>
<td>3</td>
</tr>
</tbody>
</table>

RESQML only supports global projected CRSs that use 1 of the first 6 name combinations. Specifically RESQML excludes the 5 global projected CRSs whose axis names are "First local axis" and "Second local axis" or "Topocentric East" and "Topocentric North".

The location of a point within a global projected CRS is defined by a pair of coordinates, where the first value is with respect the first axis of the global projected CRS and the second value is with respect to the second axis of the global CRS.

To unambiguously define the rotation bearing, we specify that under a 0 rotation bearing and a (0, 0) origin translation the axis of the areal local coordinate system aligns with the global 2D projected CRS.

Then for the 6 combinations above, define the rotation bearing and translation as described in the following sections.
6.4.1.1 **Axis1 – Easting, Axis 2 – Northing**

![Diagram](image1)

**Fig. 1**

**No Translation – No Rotation**
- The y-axis aligns with the Northing Axis (Axis 2), the x-axis aligns with the Easting Axis (Axis 1).
- The origin of the local coordinate system is at the origin of the global 2D projected CRS.

![Diagram](image2)

**Fig. 2**

**Translation**
- The y-axis is parallel with the Northing Axis (Axis 2), the x-axis is parallel with the Easting Axis (Axis 1).
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Easting) and the second value is with respect to Axis 2 (Northing). Both values are in the unit of measure of the parent global 2D projected CRS.

![Diagram](image3)

**Fig. 3**

**Rotation**
- The grid is rotated by giving the bearing of the y-axis of the local coordinate system, measured clockwise from the Northing Axis of the global 2D projected CRS.

![Diagram](image4)

**Fig. 4**

**Translation and Rotation**
- The grid is rotated by giving the bearing of the y-axis of the local coordinate system, measured clockwise from the Northing Axis of the global 2D projected CRS.
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Easting) and the second value is with respect to Axis 2 (Northing). Both values are in the unit of measure of the parent global 2D projected CRS.
6.4.1.6 Axis1 – Northing, Axis 2 – Easting

- The y-axis aligns with the Northing Axis (Axis 1), the x-axis aligns with the Easting Axis (Axis 2).
- The origin of the local coordinate system is at the origin of the global 2D projected CRS.

6.4.1.7 No Translation – No Rotation - Fig. 5.
- The y-axis aligns with the Northing Axis (Axis 1), the x-axis aligns with the Easting Axis (Axis 2).
- The origin of the local coordinate system is at the origin of the global 2D projected CRS.

6.4.1.8 Translation – Fig. 6.
- The y-axis is parallel with the Northing Axis (Axis 1), the x-axis is parallel with the Easting Axis (Axis 2).
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Northing) and the second value is with respect to Axis 2 (Easting). Both values are in the unit of measure of the parent global 2D projected CRS.

6.4.1.9 Rotation – Fig. 7.
- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Northing Axis of the global 2D projected CRS.

6.4.1.10 Translation and Rotation – Fig 8
- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Northing Axis of the global 2D projected CRS.
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Northing) and the second value is with respect to Axis 2 (Easting). Both values are in the unit of measure of the parent global 2D projected CRS.
6.4.1.11 **Axis 1 – Westing, Axis 2 – Southing**

- The y-axis aligns with the Southing Axis (Axis 2), the x-axis aligns with the Westing Axis (Axis 1).

6.4.1.12 **No Translation – No Rotation - Fig. 9.**

- The y-axis aligns with the Southing Axis (Axis 2), the x-axis aligns with the Westing Axis (Axis 1).
- The origin of the local coordinate system is at the origin of the global 2D projected CRS.

6.4.1.13 **Translation – Fig. 10.**

- The y-axis is parallel with the Southing Axis (Axis 2), the x-axis is parallel with the Westing Axis (Axis 1).
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Westing) and the second value is with respect to Axis 2 (Southing). Both values are in the unit of measure of the parent global 2D projected CRS.

6.4.1.14 **Rotation – Fig. 11.**

- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Southing Axis of the global 2D projected CRS.

6.4.1.15 **Translation and Rotation – Fig 12.**

- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Southing Axis of the global 2D projected CRS.
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Westing) and the second value is with respect to Axis 2 (Southing). Both values are in the unit of measure of the parent global 2D projected CRS.
### 6.4.1.16 Axis 1 – Southing, Axis 2 – Westing

**Fig. 13**

- The y-axis aligns with the Southing Axis (Axis 1), the x-axis aligns with the Westing Axis (Axis 2).

**No Translation – No Rotation**

- The origin of the local coordinate system is at the origin of the global 2D projected CRS.

### 6.4.1.17 No Translation – No Rotation - Fig. 13.
- The y-axis aligns with the Southing Axis (Axis 1), the x-axis aligns with the Westing Axis (Axis 2).
- The origin of the local coordinate system is at the origin of the global 2D projected CRS.

### 6.4.1.18 Translation – Fig. 14.
- The y-axis is parallel with the Southing Axis (Axis 1), the x-axis is parallel with the Westing Axis (Axis 2).
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Southing) and the second value is with respect to Axis 2 (Westing). Both values are in the unit of measure of the parent global 2D projected CRS.

### 6.4.1.19 Rotation – Fig. 15.
- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Southing Axis of the global 2D projected CRS.

### 6.4.1.20 Translation and Rotation – Fig 16.
- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Southing Axis of the global 2D projected CRS.
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Southing) and the second value is with respect to Axis 2 (Westing). Both values are in the unit of measure of the parent global 2D projected CRS.
6.4.1.21 *Axis 1 – Westing, Axis 2 – Northing*

- Fig. 17

- No Translation
  - No Rotation
  - The y-axis aligns with the Westing Axis (Axis 1), the x-axis aligns with the Northing Axis (Axis 2).
  - The origin of the local coordinate system is at the origin of the global 2D projected CRS

6.4.1.22 *No Translation – No Rotation - Fig. 17.*

- The y-axis aligns with the Westing Axis (Axis 1), the x-axis aligns with the Northing Axis (Axis 2).
- The origin of the local coordinate system is at the origin of the global 2D projected CRS

6.4.1.23 *Translation – Fig. 18.*

- The y-axis is parallel with the Westing Axis (Axis 1), the x-axis is parallel with the Northing Axis (Axis 2).
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Westing) and the second value is with respect to Axis 2 (Northing). Both values are in the unit of measure of the parent global 2D projected CRS.

6.4.1.24 *Rotation – Fig. 19.*

- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Westing Axis of the global 2D projected CRS.

6.4.1.25 *Translation and Rotation – Fig 20.*

- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Westing Axis of the global 2D projected CRS.
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Westing) and the second value is with respect to Axis 2 (Northing). Both values are in the unit of measure of the parent global 2D projected CRS.
6.4.1.26 *Axis 1 – Northing, Axis 2 – Westing*

- **Fig. 21**
  - **No Translation No Rotation**
    - The y-axis aligns with the Westing Axis (Axis 2), the x-axis aligns with the Northing Axis (Axis 1).
    - The origin of the local coordinate system is at the origin of the global 2D projected CRS.

6.4.1.27 *No Translation – No Rotation - Fig. 21.*

- The y-axis aligns with the Westing Axis (Axis 2), the x-axis aligns with the Northing Axis (Axis 1).
- The origin of the local coordinate system is at the origin of the global 2D projected CRS.

6.4.1.28 *Translation – Fig. 22.*

- The y-axis is parallel with the Westing Axis (Axis 2), the x-axis is parallel with the Northing Axis (Axis 1).
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Northing) and the second value is with respect to Axis 2 (Westing). Both values are in the unit of measure of the parent global 2D projected CRS.

6.4.1.29 *Rotation – Fig. 23.*

- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Westing Axis of the global 2D projected CRS.

6.4.1.30 *Translation and Rotation – Fig 24.*

- The grid is rotated by given the bearing of the y-axis of the local coordinate system, measured clockwise from the Westing Axis of the global 2D projected CRS.
- The origin of the local coordinate system is given as a location within the parent global 2D projected CRS. It is specified as a pair of values, where the first value is with respect to Axis 1 (Northing) and the second value is with respect to Axis 2 (Westing). Both values are in the unit of measure of the parent global 2D projected CRS.

6.4.2 *Local Depth Coordinate System*

A local depth coordinate system first has its areal component defined as above. To define the vertical component, define the vertical origin (with respect to the global vertical CRS). The local coordinate system is always meters.
Figure 5 shows a vertical local coordinate reference system; its origin is above MSL and, because the global vertical CRS is orientated down, is a negative number defined in m (unit of measure of CRS 5715).

![Local Vertical Coordinate System](image)

**Figure 5 Vertical local coordinate reference system**

### 6.4.3 Local Time Coordinate System

A local time coordinate system first has its areal component defined as above. To define the vertical component, define the vertical origin, which is defined with respect to both the global time CRS and the global vertical CRS. The vertical origin is always oriented downwards; its unit of measure is always seconds.

Figure 6 shows the vertical component of the local time coordinate system (in orange). Its unit of measure is seconds (s) and its origin is defined with respect to both the global time CRS (unit of measure s) and the global vertical CRS (unit of measure m).

![Global Time Coordinate System](image)

**Figure 6 Vertical components of the local time coordinate system.**
7 Bin Grids

Horizons and faults can be represented with 2D grid representations. Most of the time, these grids have a strong meaning: they correspond to the seismic bin grid where the interfaces have been picked.

Because these bin grids can be used to support several 2D grid representations of several interfaces, they are defined independently of any interface. That’s why they are not contained within one particular interface but within the interface feature set of the RESQML document. Thus, any 2D grid representation of any interface can reference any bin grid.

7.1 Definition

Each bin grid must have orthogonal axes and equally spaced nodes. The first axis of the bin grid is always parallel to the first axis of the local 3D CRS where it is defined. The second axis is always parallel to the second axis of this same local 3D CRS.

Figure 7 Bin grids can be used in RESQML to represent faults and horizons.

Bin grids are defined using the following elements:

- **Guid.** To identify and to reference the bin grid.
- **Dublin Core metadata.** For traceability.
- **nI** and **nJ.** Respectively, the number of nodes in I and J dimension.
- **local3dCRS.** A pointer to the local 3D CRS which represents the coordinates.
- **originOrdinal1** and **originOrdinal2.** Respectively, the value at the origin of the bin grid on the first and second axis of the local 3d CRS.
- **deltaI** and **deltaJ.** Respectively the constant distance between nodes along the I and J axis of the grid. The unit of measure is defined by means of the global CRS.
If the bin grid defines a seismic bin grid (usual case), then some other attributes should be defined to better characterize the bin grid; these include:

- **inlineParallelToIAxis**. True indicates that the inlines are parallel to the i-axis which means that each inline represents a constant value of j and each crossline represents a constant value of i. False indicates that the inlines are parallel to the j-axis which means that each inline represents a constant value of i and each crossline represents a constant value of j.

- **originInline** and **originCrossline**. Respectively the index of the first inline and crossline beginning at i=0, j=0.

- **inlinIncrement** and **crosslineIncrement**. The inline (and crossline) increment.
  - If inlines are parallel to the i-axis, then the inline increment will be the difference in the inline index from node i=0, j=0 to node i=1, j=0.
  - If inlines are parallel to the j-axis, then the inline increment will be the difference in the inline index from node i=0, j=0 to node i=0, j=1. The increment can be a positive or negative integer, but not zero.

### 7.2 Use of Bin Grids

Bin grids are essentially used for 2D grid representation patches definition. Each 2D grid patch must reference a bin grid. Furthermore, each 2D grid patch must indicate where it is located on this bin grid and if it is defined at the same resolution or a coarser resolution than the bin grid. Figure 8 shows two 2D grid patches lying on a same bin grid.

![Figure 8 Two grid patches lying on the same bin grid.](image)

2D grid patch attributes to define its location and its resolution according to the bin grid:

- **binGrid**. Points to the bin grid for which this patch lies on.
- **nl** and **nJ**. Respectively, the number of nodes in I and J dimension of the bin grid.
- **iStart** and **jStart**. Respectively, the location of the patch grid origin on the I and J dimension of the bin grid.
- **iIncrement** and **jIncrement**. Respectively, the constant increment between nodes along the I and J axis of the bin grid. An increment that is not equal to one represents a grid coarsening.
## Appendix A. Reference Standards

The following table lists other standards consulted, used, or incorporated in RESQML.

<table>
<thead>
<tr>
<th>Standards/Organization</th>
<th>Description of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Schema 1.1</td>
<td>Used to define the schema that constrains the content of a RESQML XML document.</td>
</tr>
<tr>
<td>Hierarchical Data Format 5 (HDF5)</td>
<td>Optional set of open file formats and libraries that can be used with the RESQML schema.</td>
</tr>
<tr>
<td>EPSG Codes</td>
<td>The European Petroleum Survey Group (EPSG), the globally recognized experts on geodetic issues, has been absorbed into the Surveying and Position Committee of the International Association of Oil &amp; Gas Producers (OGP), which is now the owner of the EPSG database of Geodetic Parameters and assigned codes. RESQML implementations can use EPSG codes as part of defining a coordinate reference system (CRS).</td>
</tr>
<tr>
<td>Dublin Core Metadata Elements</td>
<td>A metadata standard that has been adapted for use with RESQML.</td>
</tr>
<tr>
<td>Documentation Standard</td>
<td>The key words &quot;MUST&quot;, &quot;MUST NOT&quot;, &quot;REQUIRED&quot;, &quot;SHALL&quot;, &quot;SHALL NOT&quot;, &quot;SHOULD&quot;, &quot;SHOULD NOT&quot;, &quot;RECOMMENDED&quot;, &quot;MAY&quot; and &quot;OPTIONAL&quot; in this guide are to be interpreted as described in RFC 2119. (<a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a>)</td>
</tr>
<tr>
<td>RESQML predecessor standard.</td>
<td>RESQML Version 1 replaces RESCUE functionality and addresses some of the key user issues with RESCUE. An E&amp;P industry data exchange used since the 1990s for 3D gridded reservoir models, horizons, faults and structural models, and associated well data.</td>
</tr>
</tbody>
</table>
## Appendix B. Dublin Core Elements

The following tables list the Dublin Core elements for a RESQML document and for data objects (anything with a GUID) within a document. Between documents and data objects, the elements are used as consistently as possible, though some minor differences exist with respect to creation and modified dates, as explained below.

### Dublin Core Elements for a RESQML Document (ResqmlDocumentInfo)

<table>
<thead>
<tr>
<th>Element Count</th>
<th>Dublin Core Name</th>
<th>Description of RESQML Use</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>title</td>
<td>One line description/name of the RESQML model.</td>
<td>User entered</td>
</tr>
<tr>
<td>1,1</td>
<td>creator</td>
<td>Name (or user ID) of the person who initially creates the RESQML document. The creator remains the same if you write the same file.</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0,1</td>
<td>subject</td>
<td>Key words to describe an activity, for example, history match or volumetric calculations. Used as a search string.</td>
<td>User entered</td>
</tr>
<tr>
<td>0,1</td>
<td>description</td>
<td>User comments. For end-user use (human readable); not used by software.</td>
<td>User entered</td>
</tr>
<tr>
<td>1,1</td>
<td>publisher</td>
<td>Software or service that was used to initially create the document. Must be human and machine readable and unambiguously identify the software by including the company name, software name and software version. Format: string or URI</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0,1</td>
<td>contributor</td>
<td>Name (or user ID) of the last person who makes updates/changes to the document after it has been created. Must be human-readable. This information can also be found in the child object that was last changed.</td>
<td>Vendor software</td>
</tr>
<tr>
<td>1,1</td>
<td>created</td>
<td>Date and time the document was created (saved to the RESQML format). Format: YYYY-MM-DD hh:mm:ss Time zone: Mandatory (to provide the correct absolute time relative to the earth)</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0,1</td>
<td>modified</td>
<td>Last date and time the document was modified. If you modify anything in the document, “modified” is updated.</td>
<td>Vendor software</td>
</tr>
</tbody>
</table>
### Dublin Core Elements for a RESQML Document (ResqmlDocumentInfo)

<table>
<thead>
<tr>
<th>YYYY-MM-DD</th>
<th>hh:mm:ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>This information can also be found in the child object that was last changed.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1,1 format</th>
<th>Used so the reading software can identify quickly identify the type of RESQML document it is reading. Possible values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• RESQML XML only</td>
<td></td>
</tr>
<tr>
<td>• RESQML XML/HDF</td>
<td></td>
</tr>
</tbody>
</table>

<p>| Vendor software |</p>
<table>
<thead>
<tr>
<th>Element Count</th>
<th>Dublin Core Name</th>
<th>Description of RESQML Use</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>title</td>
<td>Name of the feature or object</td>
<td>User entered</td>
</tr>
<tr>
<td>1,1</td>
<td>creator</td>
<td>Name (or user ID) of the person who creates the item—the semantic content in the application, not the XML file written, the native file.</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0,1</td>
<td>description</td>
<td>User comments. For end-user use (human readable); not used by software.</td>
<td>User entered</td>
</tr>
<tr>
<td>1,1</td>
<td>publisher</td>
<td>Software or service that was used to initially create the document. Must be human and machine readable and include the company name, software name and software version. Format: string or URI</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0, 1</td>
<td>contributor</td>
<td>Name (user ID) of a contributor, the last person who updates/edits the item after it was created, the semantic content in the application, not the XML file written, the native file.</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0,1</td>
<td>created</td>
<td>Date and time the item was created by the software (not when it was written to the RESQML document). Format: YYYY-MM-DD hh:mm:ss</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0,1</td>
<td>modified</td>
<td>Date and time the item was last modified by the software (not when it was written to the RESQML document). Format: YYYY-MM-DD hh:mm:ss</td>
<td>Vendor software</td>
</tr>
<tr>
<td>0, N</td>
<td>BibliographicCitation</td>
<td>If company wants to provide a source for additional information about something contained in a model, for example, a stratigraphic unit. Format: most useful to use a URL/URI but may also be a string. For end-user use (human readable); not used by software.</td>
<td>Company/user controlled</td>
</tr>
</tbody>
</table>