MXA: a customizable HDF5-based data format for multi-dimensional data sets

2010 Modelling Simul. Mater. Sci. Eng. 18 065008

(http://iopscience.iop.org/0965-0393/18/6/065008)

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
22 APR 2010		**00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
MXA: a customizable HDF5-based data format for multi-dimensional data sets	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **AFRL/MLLMD,Materials and Manufacturing Directorate,Wright-Patterson AFB,OH,45433**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A new digital file format is proposed for the long-term archival storage of experimental data sets generated by serial sectioning instruments. The format is known as the multi-dimensional eXtensible Archive (MXA) format and is based on the public domain Hierarchical Data Format (HDF5). The MXA data model, its description by means of an eXtensible Markup Language (XML) file with associated Document Type Definition (DTD) are described in detail. The public domain MXA package is available through a dedicated web site (mxa.web.cmu.edu), along with implementation details and example data files.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**14**	

Modelling Simul. Mater. Sci. Eng. **18** (2010) 065008 (13pp) doi:10.1088/0965-0393/18/6/065008

MXA: a customizable HDF5-based data format for multi-dimensional data sets

M Jackson[1], **J P Simmons**[2] **and M De Graef**[3,4]

[1] Bluequartz Software, Dayton, OH, USA
[2] Materials and Manufacturing Directorate, AFRL/MLLMD, Wright-Patterson AFB, OH 45433, USA
[3] Department of Materials Science and Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3980, USA

E-mail: mike.jackson@bluequartz.net, Jeff.Simmons@wpafb.af.mil and degraef@cmu.edu

Received 7 December 2009, in final form 22 April 2010
Published 30 July 2010
Online at stacks.iop.org/MSMSE/18/065008

Abstract
A new digital file format is proposed for the long-term archival storage of experimental data sets generated by serial sectioning instruments. The format is known as the multi-dimensional eXtensible Archive (MXA) format and is based on the public domain Hierarchical Data Format (HDF5). The MXA data model, its description by means of an eXtensible Markup Language (XML) file with associated Document Type Definition (DTD) are described in detail. The public domain MXA package is available through a dedicated web site (mxa.web.cmu.edu), along with implementation details and example data files.

1. Introduction

A new trend in Materials Science is 'data in the large', that is, large volumes of data that can be generated by experimental or numerical computerized techniques. The expansion of data set sizes enabled by these technological advancements also leads to large amounts of time and money invested in the production of quality data sets. The potential for significant advancements in Materials Science exists if technologies can be developed that facilitate reuse of these high quality/high production cost data sets. This paper describes developments of one technology toward this end: the *Multi-dimensional eXtensible Archive* (MXA) format for preservation of multi-dimensional experimental or numerical data, that our group is developing specifically to facilitate data reuse.

Data storage for reuse is a non-trival problem. As any graduate advisor can attest to, simply transferring data from one graduate student to another one, in particular after a multi-year

[4] Author to whom any correspondence should be addressed.

hiatus, can pose insurmountable barriers: perhaps the file format used was non-standard and the detailed format description was lost; perhaps the filename convention was insufficient to clearly identify the files; perhaps some crucial parameters describing the experiment are missing; and so on. It is obviously possible to collect all files for a given experiment in an archive format, such as *.tar* [1] or *.zip* [2], but this does not really solve the problem; after extracting the contents of the archive, the same issues mentioned above might still be encountered. There is, hence, a need for a flexible file format for the storage of large volume data sets, in such a way that the data become self-contained, with no possibility for loss of crucial information. In particular, the long-term survivability of a data set should not depend on the memory of the researcher(s) who obtained or created it. There have been attempts in the past to create file formats that are specific to the materials community. For instance, the *MatML* format [3] is an eXtensible Markup Language (XML) developed specifically for the standardized exchange of material property data values. Often, however, these special formats are too limited in their scope, and widespread adoption of the format is unlikely.

File naming conventions are, to some extent, a matter of personal preference. Since most computer operating systems now allow for long filenames, it is often a practice to rely on filenames, alone, as the documentation of the data, where as many descriptors as possible are concatenated in the filename. Often, abbreviations and shorthand are used whose meaning is clear to the researcher while working with the data, but information documented in this fashion fades with time and may become incomprehensible to other researchers attempting to reuse the data.

In developing the MXA specification, we have identified the following characteristics to be critical for reuse and archiving. Firstly, data, whether it be numerical or experimental, nearly always consist of the actual data and, what is commonly called, *meta-data*. Meta-data can encompass both data that are specific to a single image, such as a slice number or a polarizer setting, and data that are common to the entire data set, such as sample chemistry, sample preparation, data ownership and experimental conditions. To guarantee the integrity of the file set, both data and meta-data must always be kept together. Secondly, it is all too easy to open an image file using any of a series of image processing programs, modify the image and then save it back to the same file. In a sense, this would corrupt the data set, since the original image is now no longer available. Obviously, one could give the original data files 'read-only' protection, but this is easily circumvented. The long-term storage and preservation of the original data thus become a significant issue. Thirdly, data sets can have many different dimensionalities; for instance, the data could consist of 2D images taken as a function of one or more external parameters (e.g. applied magnetic field, sample temperature, etc), and a flexible data format should be capable of accommodating all these different cases. Fourthly, the data acquired in an experiment or generated in a simulation are often *hierarchical*; for instance, an optical microscope image may be a mosaic of multiple smaller images, and each of these might, in turn, consist of multiple components, either red–green–blue or perhaps multiple images acquired with different polarizer orientations. The new data format will have to be flexible enough to accommodate a wide variety of data hierarchies. Finally, the methods used to store data in and extract data from the archive must be simple and independent of the complexity of the data.

The MXA format was designed to eliminate these common problems with data storage. The MXA format provides the following platform-independent features:

(i) the ability to store multi-dimensional data sets in a single file;
(ii) a rigorously defined but very flexible hierarchical data structure;
(iii) transparent methods to store and extract both data and meta-data.

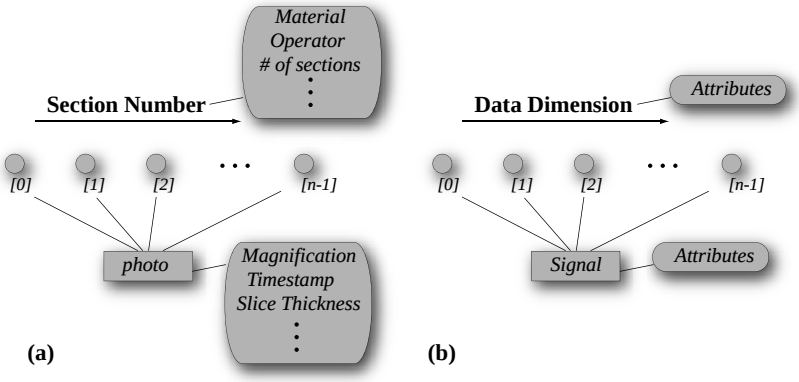


Figure 1. (*a*) Schematic representation of the data acquired during a serial sectioning experiment, in which for each slice a single optical micrograph is recorded; for each micrograph, several items of meta-data are stored, and there is additional meta-data for the entire experiment (i.e. items common to all micrographs). In (*b*), the logical structure of (*a*) is represented in abstracted form, introducing the terminology used to describe the MXA data model.

A recent report from the National Research Council on the relatively new field of *Integrated Computational Materials Engineering* (ICME) calls for the creation of pre-competitive data repositories as part of a larger scale cyberinfrastructure 'that permits multidisciplinary analysis, collaborative model development and design optimization with materials as a key optimization parameter' [10]. These data repositories should contain both experimental data and the tools and modules needed to analyze the data. The file format presented in this paper represents a significant step in the direction of a standardized data format for such ICME microstructural databases.

The structure of this paper is as follows: first, in section 2, we define the basic MXA data model by means of a few examples. In section 3, we provide a more formal description of the data format in terms of a Document Type Definition (DTD) and an XML approach. We discuss the implementation and the structure of the web site from which the public domain MXA package can be downloaded. We conclude this paper in section 4 with the description of a series of three example data sets that are made available as part of the MXA package.

2. The MXA Data Model

Consider a serial sectioning metallography experiment, in which one repeatedly removes a thin layer of material from a sample and records an image of the new surface by means of an optical microscope. Such an experiment could be performed manually [11] or by means of a robot-controlled setup [12]. The result of the experiment (i.e. the data) is a series of optical micrographs, representing consecutive sections through the sample. As meta-data, one could record the date and time of the experiment, the name of the operator, the sample label, the thickness of each removed slice, the type and magnification of the optical microscope and so on. Schematically, the data resulting from this experiment are represented in figure 1(*a*): the micrographs are labeled according to slice number (0 to $n-1$, with n the number of slices), and for each micrograph a number of meta-data items are to be stored, such as the time-stamp and the slice thickness. There is also meta-data that all micrographs have in common, such as the operator, the material and so on; these types of meta-data are associated with the entire

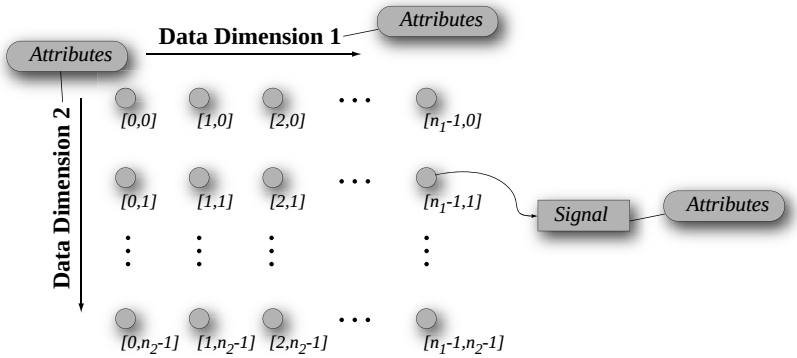


Figure 2. Schematic representation of the data model for a two-dimensional data set; each entry of the type $[i, j]$ is associated with a single signal (with possible attributes).

series of images, instead of with any particular image. The logical structure of the data and meta-data in figure 1(*a*) is represented in more abstract form in figure 1(*b*). The data are one-dimensional (i.e. the thickness series), and for each data node a single signal is stored (i.e., the optical micrograph). The data series has a number of attributes, and each individual signal may also have attributes. What emerges from this representation is that the data consist of *signals* which correspond to *data nodes* along a *data dimension*; both the data dimension and the individual signals may have one or more *attributes*.

Next, assume that, in addition to the optical micrograph, we also record a polarized light image for each slice. This second image can be regarded as a second signal, so that with each entry along the data dimension, we associate two signals, each with its own attributes. If we record a series of polarized light images for each slice, for instance, at eight 45° increments of the polarizer orientation angle, then we could regard each of these images as a new signal, or, alternatively, we could regard the orientation angle as a new data dimension. This is shown schematically in figure 2. Each data entry is labeled with two indices $[i, j]$, where i labels the slice number and j the polarizer orientation, and is associated with a single signal (the polarized light image). It is obvious that many data sets can be mapped onto this particular abstract data model.

Consider next the case of a two-dimensional data set with, for each dimension i, a number n_i of data nodes. Each node is characterized by two coordinates ($[0, 0] \leqslant [i, j] \leqslant [n_1 - 1, n_2 - 1]$); each node has associated with it a set of signals. In figure 3, there are two signals (1 and 5) and a signal group which itself consists of three signals (2, 3 and 4). Both signals and data dimensions may have multiple attributes, i.e. descriptors that are specific to each item. To make the example more concrete, consider the following experiment: a binary alloy sample has been prepared for optical microscopy by conventional metallographic techniques. An optical microscope is used to examine the polished surface as a function of the sample temperature; for each temperature, several locations on the sample surface are imaged using three different detectors. The first detector produces an infrared image, the second one a visible light image, separated into its red, green and blue component images and the third detector produces an ultraviolet image. The sample is mounted on an automated x–y translation stage, so that the same sample locations can be imaged at different temperatures. This example is readily mapped onto the schematic of figure 3. Data dimension 1 is selected to be the location on the sample; if there are 20 different locations at which images are acquired, then $n_1 = 20$. The second data dimension is the sample temperature; perhaps the sample is

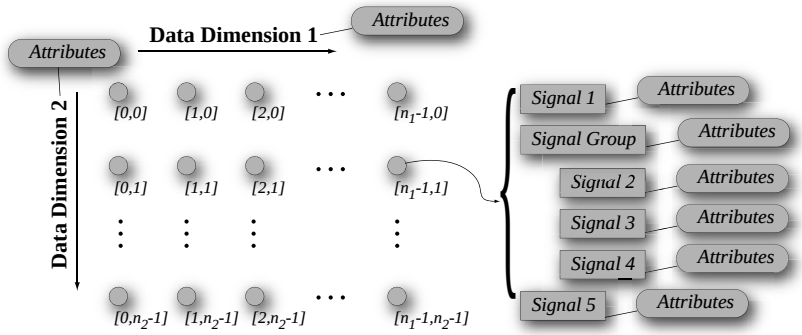


Figure 3. Schematic representation of a multi-dimensional data set (two-dimensional in this case) with several signals (two signals and a signal group consisting of three signals) for each data node.

heated from 300 to 800 K, in steps of 50 K, so that $n_2 = 11$. Since the same locations are used at all temperatures, the list of coordinates of all locations can be stored as an attribute to data dimension 1, whereas the list of all temperatures can be an attribute to data dimension 2. The infrared image would be signal 1, the ultraviolet image signal 5 and the components of the visible light image would be grouped as three signals in a signal group. For each signal, there may be instrumental or detector settings that need to be stored as attributes.

In the general multi-dimensional case with N_d dimensions, each data node is labeled by a vector of integer components, $[p_0, p_1, p_2, \ldots, p_{N_d-1}]$, with $0 \leqslant p_0 \leqslant n_0-1, 0 \leqslant p_1 \leqslant n_1-1$, $\ldots, 0 \leqslant p_{N_d-1} \leqslant n_{N_d-1} - 1$. The first allowed index of each dimension is taken to be 0. In the example above, the temperature T (dimension 2) corresponding to data node j can be determined as

$$T_j = s_2 + j \times \Delta s_2,$$

where s_2 is the starting value (300 K in this case) and Δs_2 the increment (50 K). This simple linear scaling is used for all data dimensions, so that the parameters s_i, Δs_i and n_i must be stored as attributes to each dimension i.

Obviously, there are many possible variations on this schematic, with a different number of dimensions, and different combinations of signals and signal groups. The MXA format is designed to accommodate all possible variations of this basic schematic, which we will refer to as the *MXA Data Model*.

3. Implementation

Because the MXA Data Model is a specification that describes the organization of data and meta-data, the actual storage can be accomplished in any number of ways. During the search for a fast, stable, open, flexible and portable storage format, a number of established file formats were evaluated for use as the basis for the MXA Data Model. The selection of Hierarchical Data Format (HDF5) as the underlying file format for the MXA format was made based on a comparison of six different commonly available scientific data formats: (HDF4) [5], HDF5 [4], netCDF (network Common Data Form, for storage of array-oriented scientific data) [6], PDB (Protein Data Bank format) [7], FITS (Flexible Image Transport System, used predominantly in astronomy) [8] and DICOM (Digital Imaging and Communications in Medicine) [9]. Out of these six formats, HDF5 was selected because it is widely supported on platforms ranging

from small personal computers to High Performance Computing machines, it is accessible from C, C++, Fortran and Java, it is scalable (no size limits), it is searchable (random access), it has a vendor friendly Open Source License (BSD Like) and it is constantly updated and enhanced. While other formats also satisfy several of these criteria, HDF5 presents the best balance of all the above criteria. From the HDF5 web site at http://www.hdfgroup.org we quote:

"HDF5 is particularly good at dealing with data where complexity and scalability are important. Data of virtually any type or size can be stored in HDF5, including complex data structures and data types. HDF5 is portable, running on most operating systems and machines. HDF5 is scalable—it works well in high end computing environments, and can accommodate data objects of almost any size or multiplicity. It is also efficient, providing fast access to data, including parallel I/O. It also can store large amounts of data efficiently—it has built-in compression, or applications can also provide their own special-purpose compression."

The current MXA specification is implemented using both C++ and IDL (Interactive Data Language, [13]). The MXA C++ library contains a robust Application Programmer Interface (API) that allows the scientist to import, export and query data stored in the MXA data file. The current version of the MXA library implements APIs that allow for the integration of the library into any existing simulation or experiment where data storage is performed. The library also exposes APIs to import existing data sets based on tiff images or the use of APIs and base classes to create computer code to import custom data files in formats not already included in the library. Examples that show how to import various types of data are included in the MXA source code distribution. Additional examples show usages for many of the APIs available in the MXA Library. The current C++ code base is portable across many different compilers, including the Visual Studio suite from Microsoft, the GNU C/C++ compilers and the Intel C++ compilers in both 32 and 64 bit forms. The MXA code base leverages several open source projects to streamline the development. The most important among them is the HDF5 (www.hdfgroup.org) file format in which all MXA data are stored. In addition, the Expat software library is utilized for XML processing and the Tiff (www.remotesensing.org) library is used for basic image import and export. The MXA project uses the CMake meta-build system to generate appropriate build systems for each type of compiler suite being used and all the MXA APIs can be converted into HTML code using the Doxygen (http://www.stack.nl/dimitri/doxygen/) utility. Using the MXA library, researchers can easily integrate the import, processing and archive of their data sets into existing code bases.

The source code for both C++ and IDL versions is available at the following URL: http://mxa.web.cmu.edu/

This web site provides the following components: background information on the MXA format; a detailed (formal) specification of the MXA format, in terms of HDF5 variable types; compilation notes for Microsoft Windows, UNIX and UNIX like (including Apple Inc.'s Mac OS X) based operating systems, as well as system requirements; and a series of 'How To' pages, describing how to include MXA in a project, how to write a data import delegate and how to import data into HDF5 using the MXA application programming interface (API). A detailed API, describing the implementation details of all the C++ classes and namespaces, along with their hierarchy, is made available. Finally, the site provides detailed documentation and a series of examples (described in the next section). To facilitate the generation of new importer routines, the web site also provides an 'import generator', a simple utility which generates all the skeleton code needed for a new importer. The three example MXA files discussed in the next section along with the corresponding XML files can also be downloaded from the MXA web site. These files can be viewed with

the generic QHDFViewer application as well as the more MXA specific 'DataViewer' application.

The IDL implementation of the MXA model consists of three libraries:

(i) *hdf_lite.pro* contains the complete HDF5 Lite API converted into IDL function calls with identical names and arguments (with the exception of the H5LTrepack routine, for which there is no easily implementable IDL equivalent code). The HDF5 Lite API provides a series of higher level routines to facilitate the creation and reading of data sets and attributes.

(ii) *hdf_im.pro* implements the HDF5 IM API (a collection of routines for reading and writing grayscale and color images as well as color palettes) as a series of IDL function calls, again with identical names and arguments as in the original interface.

(iii) *mxa_library.pro* contains all the MXA routines for reading, writing and parsing of both MXA and XML files, as well as a few wrapper routines to catch and gracefully handle errors.

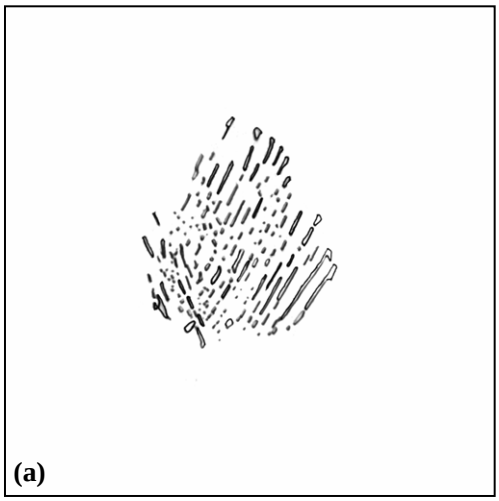
Since IDL is an interpreted language, all three libraries must be compiled at the beginning of an IDL session before they can be used; a simple routine *mxa_init.pro* is provided for this purpose.

While the API approach provides maximum flexibility to the programmer, it also requires an intimate familiarity with the inner workings of the library. In order to ease the transition of existing data sets to the MXA file format a few enhancements have been added to the MXA library. One of these enhancements allows the scientist to describe the simulation or experimental data using the XML in terms of the MXA Data Model components. The XML file format is written in plain text and is 'human readable'. The XML file contains the MXA Data Model description, the Required Meta-Data, any user meta-data and links to the data files that will be imported into the MXA archive file. Numerous XML tools and editors can be employed to help the generation of the XML file. The availability of a DTD on the internet facilitates error checking of the XML document before its use. All three examples in the next section make use of this XML-based approach.

The XML DTD for the higher level interface to the MXA data model can be found on the web site under the heading 'XML Usage'. This page shows the complete DTD with a brief explanation of all its components, as well as an example XML file that makes use of all the required fields and attributes. Using XML to import and export the MXA data model allows the researcher to reuse models from one experiment to the next. In the current usage scenario, it is expected that the XML file is used as a template for a particular type of experiment or simulation. For this reason, certain entries in the XML file are optional; if they are omitted, it is assumed that their values will be gathered through another mechanism, such as a Graphical User Interface (GUI) or a configuration/initialization file for the experiment. The DTD file is available over the internet at the following URL: http://mxa.web.cmu.edu/mxa_0.4.dtd

4. Examples

In this section, we describe three examples of the use of the XML approach; two examples are taken from experiments, one from simulations, to emphasize that the data format is not exclusively designed for one or the other. The data sets are taken from real experiments and simulations. The MXA web site provides both the MXA-formatted data files and the complete XML files used to define their internal structure.



(a)

(b)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE File_Root SYSTEM "http://mxa.web.cmu.edu/mxa_0.4.dtd">
<File_Root>
  <Data_Model File_Type="MXA" File_Version="0.4">
    <Data_Root Name="Pearlite" />
    <Data_Dimensions>
      <Dimension Name="Slice" Alt_Name="Slice #" Start_Value="0" Increment="1"
                 Count="50"  End_Value="49" />
    </Data_Dimensions>
    <Data_Records>
      <Signal Name="SliceImage" Alt_Name="Cropped Optical Microscopy Image" />
    </Data_Records>
  </Data_Model>
  <Meta_Data>
    <Required_MD Creator="M. Hillert" Date="1959" Name="Pearlite Colony"
    Description="Manual serial sectioning of pearlite colony" Pedigree="Derived"
    Original_Source_File="Not Applicable" Distribution_Rights="Unlimited"
    Release_Limitation="Not Applicable"  Release_Number="Not Applicable" />
    <UserDefined_MD Pixels_Per_Micron="6.3" />
  </Meta_Data>
</File_Root>
```

Figure 4. (*a*) Slice #40 of the example pearlite MXA file; the XML file used to create the example MXA file is shown in (*b*).

4.1. Serial sectioning of a pearlite colony

This data set, courtesy of M Hillert, was created in the late 1950s by means of manual polishing and optical microscopy on a pearlite colony in a carburized electrolytic iron alloy. The complete data set consists of 242 images with a 1 μm slice thickness; a 3D visualization of the pearlite colony was recently published in the *Journal of Metals* [11]. The MXA file available through the web site contains the first 50 images of this experimental stack; an example image (for slice #40) is shown in figure 4(*a*).

The XML file describing the data structure of the pearlite MXA file is shown in figure 4(*b*). The file defines two elements: the Data_Model and the Meta_Data. There is only one Data_Dimensions element corresponding to the stack of slices. Several optional attributes are also listed (in light gray) for this Dimension: Start_Value $= 0$ (number of the first slice); Increment $= 1$ (slice images are numbered sequentially); Count $= 50$ (number of slice images);

Table 1. Image and file size data for the example MXA files available from the MXA web site (http://mxa.web.cmu.edu/).

File	Number of slices	Image size (pixels)	*.mxa.zip file size (bytes)
pearlite.mxa	40	1024×1024	1, 901, 215
in100.mxa	20	1024×954	47, 135, 202
phasefield.mxa	4×64	256×256	185, 496, 486

End_Value = 49 (index of last slice image). The explicit inclusion of optional attributes in the XML file provides for a more detailed description of the data model, but it makes the XML file less generally useful in the sense that this file can then only be used for image series consisting of 50 images. Omission of these optional attributes results in a more generally useful XML file.

The Data_Records element in figure 4(*b*) corresponds to a single Signal, which is the actual image for each slice. For the present data set, the original images were made available in the jpeg format; they were then manually adjusted to remove unwanted ferrite/ferrite grain boundaries and allotriomorphs and stored in the MXA format. Each image has dimensions 1024×1024 pixels, with a scale factor of 6.3 pixels μm^{-1}; this information is entered in the user-defined meta-data. To indicate that the images are derived from the original microstructural images, the required Meta_Data element Pedigree is set to 'Derived' instead of 'Original'.

The resulting MXA file, *pearlite.mxa*, and the corresponding XML file, *pearlite.xml*, are available from the MXA web site. Selected file information is listed in table 1.

4.2. Focused ion beam (FIB) serial sectioning of an IN100 superalloy

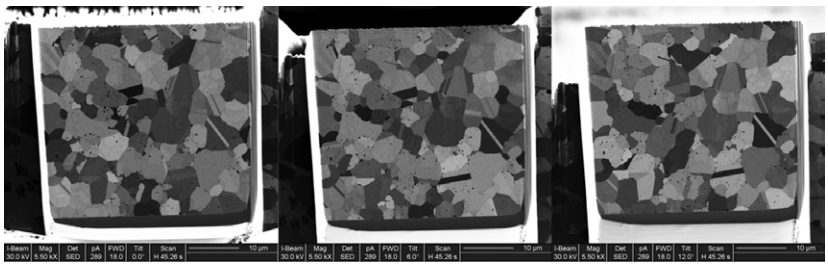
A second, somewhat more complex, experimental example involves a data series acquired using a FIB electron microscope. The data are obtained by serial sectioning through a sample; for each section, an electron back-scatter diffraction (EBSD) map is recorded, as well as four ion-induced secondary electron (ISE) images at different sample tilt angles. These different sample orientations give rise to different contrasts due to the changing channeling orientations of individual grains, and a series of images rather than a single one will aid in the subsequent segmentation of the data set. The EBSD data can be broken down into multiple arrays and ordered with a hierarchical grouping within the Data Records section, as shown in figure 5(*d*).

Using a custom written parser, the data are imported from the files produced by the orientation imaging microscopy (OIM) system and the scanning electron microscope, and stored in the MXA data file. Example images for slice #15 are shown in figure 5(*a*)–(*c*); in (*d*), the Data_Model section of the corresponding XML file is shown. The MXA file contains 20 slices, with multiple signal groups and signals for each slice (table 1).

Note that one can include information about the sample tilt angles as XML attributes to the individual signals. Doing so will restrict the use of the corresponding XML file to data sets for which the tilt angles have the same values. Alternatively, one could define these user-defined attributes when the MXA file is created, without specifying them in the XML file. This would make the XML file more generally useful. For maximum portability and generality, the XML file should only contain the required attributes, as defined in the DTD.

4.3. Three-dimensional phase field simulation

The third and final example is a computational one. This example uses results from a 3D phase field simulation of a NiAl two-phase γ–γ' alloy with a γ' volume faction of 54%



(a) **(b)** **(c)**

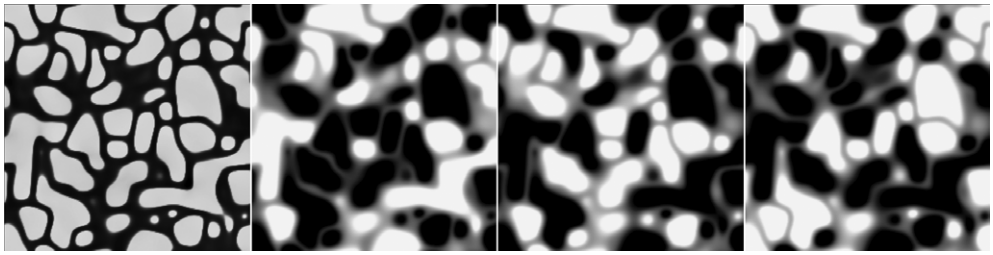
```
<Data_Model Model_Type="MXA" Model_Version="0.4">
  <Data_Root Name="OIMFIB/Data/" />
  <Data_Dimensions>
      <Dimension Name="FIB Slice" Alt_Name="FIB Slice" Start_Value="1" End_Value="20"
       Count="20" Increment="1" Index="1"  Uniform="1" />
  </Data_Dimensions>
  <Data_Records>
    <Signal Name="Secondary Electron Image 1" Alt_Name="SE1" />
    <Signal Name="Secondary Electron Image 2" Alt_Name="SE2" />
    <Signal_Group Name="Ion Induced SEI Tilt Series" Alt_Name="ISE Tilt Series">
      <Signal Name="Tilt-0" Alt_Name="Tilt-0" />
      <Signal Name="Tilt-1" Alt_Name="Tilt-1" />
      <Signal Name="Tilt-2" Alt_Name="Tilt-2" />
      <Signal Name="Tilt-3" Alt_Name="Tilt-3" />
    </Signal_Group>
    <Signal_Group Name="OIM" Alt_Name="OIM">
      <Signal Alt_Name="Image Quality" Name="Image Quality" />
      <Signal Alt_Name="Confidence Index" Name="Confidence Index" />
      <Signal Alt_Name="Phase" Name="Phase" />
      <Signal_Group Alt_Name="Euler Angles" Name="Euler Angles">
        <Signal Alt_Name="Phi1" Name="Phi1" />
        <Signal Alt_Name="Phi" Name="Phi" />
        <Signal Alt_Name="Phi2" Name="Phi2" />
      </Signal_Group>
    </Signal_Group>
  </Data_Records>
</Data_Model>
```

(d)

Figure 5. (*a*)–(*c*) ISE electron images of a powder-processed IN100 alloy, for three of the four experimental sample tilt angles (0°, 6° and 12°); a portion of the XML file used to create the example MXA file is shown in (*d*).

(VF54). The phase field program produces large ASCII text files as output. These data were organized into the MXA Data Model using a pair of Data Dimensions (Time and XYSlice) and 4 Data Records (Composition and 3 Order Parameters). The Data Records were once again put into a hierarchical grouping, as shown in figure 6(*e*). A composition map and the accompanying three order parameter maps for slice # 0 at timestep 28 are shown in figure 6(*a*)–(*d*), respectively. The data were parsed and stored into an MXA file, available from the MXA web site (see table 1). The data set contains time steps 1, 10, 19, 28 and slices 0 through 63 (Inclusive). Each slice is a 256 × 256 sample area stored as 32-bit floating point values.

As an example of how data can be extracted from an MXA file, we produce here a short section of relevant C++ source code for processing all slices of data for a



(a) **(b)** **(c)** **(d)**

```
<Data_Model>
  <Data_Root Name="Phasefield/VF54_Set-1/" />
  <Data_Dimensions>
    <Dimension Alt_Name="Timestep" Count="4" End_Value="28" Increment="9" Index="0"
               Name="Timestep" Start_Value="1" Uniform="1"/>
    <Dimension Alt_Name="XY Slice" Count="64" End_Value="63" Increment="1" Index="1"
               Name="Slice" Start_Value="0" Uniform="1"/>
  </Data_Dimensions>
  <Data_Records>
    <Signal_Group Alt_Name="NiAl" Name="NiAl">
      <Signal  Alt_Name="Composition"  Name="Composition"/>
      <Signal_Group  Alt_Name="Order Parameters" Name="Order Parameters">
        <Signal Alt_Name="Eta-1" Name="Eta-1"/>
        <Signal Alt_Name="Eta-2" Name="Eta-2"/>
        <Signal Alt_Name="Eta-3" Name="Eta-3"/>
      </Signal_Group>
    </Signal_Group>
  </Data_Records>
</Data_Model>
```

(e)

Figure 6. (*a*) Composition map and (*b*)–(*d*) three order parameter maps for a 3D phase field simulation of a NiAl alloy with a γ' volume fraction of 54%; the images were taken from slice #0 at time step 28. A portion of the XML file defining the structure of the corresponding MXA file is shown in (*e*).

specific time index:

/ Use typedefs to make code more readable */*

```
typedef IDataFile::Pointer      DataFile;
typedef IDataModel::Pointer     DataModel;
typedef IDataDimension::Pointer Dimension;
typedef IDataRecord::Pointer    Record;
```

/ Load the MXA file and gather the needed Data Dimension and Record variables */*

```
DataFile dataFile = H5MXADataFile::OpenFile(filename, READ_ONLY);
DataModel model = dataFile->getDataModel();
Dimension slice = dataModel->getDataDimension("Slice");
Record compositionRec = model->getDataRecordByNamedPath("Nickel/Composition");
```

/ Create some variables to hold the data and error codes */*

```
std::vector<float> data;  // Use an STL Vector to store data
int indices[2] = {10, 0}; // Time step 10, Slice number zero
herr_t error = 0;
```

```
/* Loop over all slices in the data set */
for (int i = slice->getStartValue(); i <= slice->getEndValue(); ++i)
{
  indices[1] = i;   // Set the index to the current slice value

/* Generate an internal HDF5 path to the data that is to be read */
  std::string datasetPath
            = H5MXAUtilities::generateH5PathToDataset(model, indices,
              compositionRec);

/* Read the data into the std::vector object 'data'*/
  error = H5Lite::readVectorDataset(dataFile->getFileId(), datasetPath, data);
  if (error < 0) {
    process_error();  // Handle any errors reading the data from the HDF5 file
  }

/* Process the data through a custom filter*/
  ProcessData(data);
} // End Loop on Slice
```

5. Conclusions

We have created the MXA file format to address the important problem of long-term archival storage of data acquired using a variety of experimental or numerical simulation techniques. The format is based on the HDF5 public domain format and is available for downloading from a dedicated web site (mxa.web.cmu.edu). The file format can be described rigorously via an XML file, which, in turn, relies on the MXA DTD file. Two implementations of the MXA format are available: C++ and the Interactive Data Language. While the file format was originally designed to address a need in the experimental materials science community, it is conceivable that other science and/or engineering areas may also benefit from this standardized file format.

Acknowledgments

The authors would like to acknowledge Dr M Hillert for the use of the pearlite data set and Dr M Uchic for the IN100 data set. The phase field data were provided by Dr Y-H Wen. Financial support for MJ and MDG was provided by the ONR/DARPA Dynamic 3D Digital Structures Program (# 2526753).

References

[1] The Open Group www.unix.org/version3/ieee_std.html
[2] PKware www.pkware.com/documents/casestudies/appnote.txt
[3] MatML www.matml.org/
[4] The HDF Group www.hdfgroup.org/hdf5/
[5] The HDF Group www.hdfgroup.org/products/hdf4/
[6] University Corporation for Atmospheric Research www.unidata.ucar.edu/software/netcdf/index.html

[7] Worldwide Protein Data Bank www.pdb.org/pdb/home/home.do
[8] FITS Support Office (NASA/Goddard Space Flight Center) http://fits.gsfc.nasa.gov/
[9] Medical Imaging and Technology Alliance http://medical.nema.org/
[10] Committee on Integrated Computational Materials Engineering 2008 Integrated computational materials engineering: A transformational discipline for improved competitiveness and national security *Technical Report* National Research Council of the National Academies
[11] De Graef M, Kral M V and Hillert M 2006 A modern view of an 'old' pearlite colony *JOM* **58** 25–8
[12] Spowart J E, Mullens H M and Puchala B T 2003 Collecting and analyzing microstructures in three dimensions: a fully automated approach *JOM* **55** 35–7
[13] ITT Visual Information Solutions 2007 *The Interactive Data Language* Boulder CO www.ittvis.com/ProductServices/IDL.aspx