ARL-TR-7956 • Feb 2017

US Army Research Laboratory

Producing a Data Dictionary from an Extensible Markup Language (XML) Schema in the Global Force Management Data Initiative

by Frederick S Brundick

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-7956 ● FEB 2017

US Army Research Laboratory

Producing a Data Dictionary from an Extensible Markup Language (XML) Schema in the Global Force Management Data Initiative

by Frederick S Brundick
Computing and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
Feb 2017	Technical Report	May 2012–October 2012

4. TITLE AND SUBTITLE
Producing a Data Dictionary from an Extensible Markup Language (XML) Schema in the Global Force Management Data Initiative

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Frederick S Brundick

5d. PROJECT NUMBER
R.0010376.8

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CII-T
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7956

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 is unlimited.

13. SUPPLEMENTARY NOTES
primary author's email: <frederick.s.brundick.civ@mail.mil>.

14. ABSTRACT
A data dictionary is a document that describes the schema of a database—its tables, fields, datatypes, and allowable values—plus additional instructions regarding its use. Traditionally the data dictionary is used to construct the database. However, when the Global Force Management (GFM) Data Initiative (DI) information exchange data model (IEDM) was migrated from Structured Query Language (SQL) to Extensible Markup Language (XML), the new XML Schema Definition (XSD) files were maintained in parallel with the data dictionary. Over time the XSD and data dictionary diverged so an application was written to construct the data dictionary in Hypertext Markup Language (HTML) from the XSD files.
This report documents the transformation script that was written, how it works, and the elements that were added to the XSD to facilitate the generation of the data dictionary. It concludes with suggestions for additional applications of this technique to extract information for use by other programs.

15. SUBJECT TERMS
XML, XSD, XSLT, HTML, SQL, database, data dictionary

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Frederick S Brundick
Unclassified	Unclassified	Unclassified	UU	38	19b. TELEPHONE NUMBER (Include area code) 410-278-8943

Contents

List of Figures

INTENTIONALLY LEFT BLANK.

1. Introduction

The Global Force Management (GFM) Data Initiative (DI)[1] uses an information exchange data model (IEDM) to ensure that producers and consumers may share force structure data. The original GFMIEDM was designed using ERwin,[2] a tool for creating entity-relationship (E-R) diagrams. ERwin was used to produce a data dictionary in the form of an Excel spreadsheet from the E-R data. This spreadsheet was then manually edited to make it easier to read and to add more information such as business rules.

Since ERwin was designed to produce Structured Query Language (SQL) code, it does not have any Extensible Markup Language (XML) features. When the GFMIEDM was manually converted to an XML Schema Definition (XSD)[3,4] replacing the SQL schema, the data dictionary had to be maintained by hand. As the XSD evolved over time, the schema and data dictionary began to diverge.

The decision was made to write an eXtensible Stylesheet Language: Transformations (XSLT)[5] script to produce a data dictionary in Hypertext Markup Language (HTML) directly from the XSD files. This effort would have 2 major benefits:

1) The XSD and data dictionary would remain synchronized, and
2) The XSD would contain additional information to aid a GFM DI developer who had access to only the XSD files.

Some of the information displayed in the spreadsheet could not be derived from the XSD, so additional, machine-readable elements had to be added.

This report contains a discussion of the XSLT script that was written, how it works, and the elements that were added to the XSD to facilitate generating the data dictionary.

2. Background

2.1 Documenting XML

All programming languages, including XML, allow comments to be inserted in the code to provide explanations to the person who is reading the code. The hierarchical design of XML enabled the designers of XSD to include both human- and machine-readable documentation elements directly inside the schema elements that they describe. Figure 1 shows how an enumerated value was defined prior to the new GFM DI additions.

```
<xs:enumeration value="NTF">
  <xs:annotation>
    <xs:documentation>Naval Task Force: A multi-mission maritime
    force that is a temporary grouping of units, under one
    commander, formed for the purpose of carrying out a specific
    operation or mission.</xs:documentation>
  </xs:annotation>
</xs:enumeration>
```

Fig. 1 Enumerated value with documentation element

The critical information is the enumerated value NTF. The optional `annotation` element may be placed inside of any XSD element, and it in turn may contain 2 standard elements: `documentation` and `appinfo`.

The former is intended to hold human-readable documentation; applications such as XML editors generally extract this element and display it separately. The latter may contain any valid XML, and it is the responsibility of the schema's designer to provide details on how its contents are to be processed. Some of the custom `appinfo` elements of the GFMIEDM XSD are shown in bold in Fig. 2.

Notice that even though the `appinfo` contents are designed to be processed by applications, they are still readable by humans. All of the GFM DI `appinfo` elements are described in Sections 3.3, 3.4, and 3.6.

```
<xs:enumeration value="NTF">
  <xs:annotation>
    <xs:documentation>Naval Task Force: A multi-mission maritime
    force that is a temporary grouping of units, under one
    commander, formed for the purpose of carrying out a specific
    operation or mission.</xs:documentation>
    <xs:appinfo>
      <app:change-request>11-0025</app:change-request>
      <app:package>
        <app:number>2</app:number>
        <app:date>2012-10-01</app:date>
      </app:package>
    </xs:appinfo>
  </xs:annotation>
</xs:enumeration>
```

Fig. 2 Enumerated value with appinfo element added

2.2 Transforming XML

An elegant feature of XML is that all related languages are written in XML. Any application that processes XML data may also manipulate XSD and XSLT files. XSLT scripts are often used to transform XML data from one schema to another or to make them easier to read by producing HTML versions of the data. The XSLT script described in Section 3 generates an HTML file from a set of XSD files. This data dictionary contains all of the information that was produced by ERwin as an Excel spreadsheet. Unlike the spreadsheet, it requires no manual editing.

3. Processing the XSD

3.1 Desired Output

Part of the original spreadsheet is shown in Fig. 3. The next sections describe the contents of each cell and how it is obtained from the XSD.

The first column displays an attribute counter for easy reference. The second and third columns (ELEMENT and ATTRIBUTE) both contain the logical (English) name, the physical name(s) used by XSD, and a definition. The fourth column (Business Rules) contains guidance for the data developer; these comments were manually added to the spreadsheet.

#	ELEMENT Name / Abbreviation / Definition	ATTRIBUTE Name / Abbreviation / Definition (JC3IEDM / GFM-specific / FMIDS)	Business Rules and Guidance for GFM DI	Valid Value Display Value	Valid Value Data Value	ATTRIBUTE Valid Value Definition MANDATORY / OPTIONAL / IDENTIFIER	VALIDATION Names Used in XSD
6	**AIRCRAFT TYPE** **(ACFT_TYPE)** **An EQUIPMENT-TYPE that is designed to fly.**	**Aircraft Type ID** **(ACFT_TYPE.acft_type_id)** The equipment-type-id of a specific AIRCRAFT-TYPE (a role name for objecttype-id).		**MANDATORY IDENTIFIER CODE: NUMERIC (20)** UID Value inherited from **FMID OBJ_TYPE.obj_type_id:** ACFT_TYPE.acft_type_id = EQPT_TYPE.eqpt_type_id = MAT_TYPE.mat_type_id = OBJ_TYPE.obj_type_id			
7		**Aircraft Type Category Code** (ACFT_TYPE.acft_type_cat_code) The specific value that represents the class of AIRCRAFT-TYPE.		**MANDATORY CODE: VARCHAR (6)**			
				Rotary wing	AIRRW	A machine or device capable of atmospheric flight and dependent on rotating blades for lift.	DS336_acft_type_cat_code
				Fixed wing	FIXWNG	A manned machine or device capable of atmospheric flight and dependent on wings for lift.	DS336_acft_type_cat_code

Fig. 3 Spreadsheet data dictionary

The next 3 columns are collapsed into a single column with an indication if the attribute is mandatory or optional, the SQL datatype, and the type's definition. All 3 columns are used for enumerated values where they contain the logical name, enumerated value, and definition. The final column contains the rule (datatype) name for enumerations. (This column was incorporated into the previous 3 columns in the new data dictionary.)

3.2 Terminology

Terms used in the data dictionary have different meanings than the same words used in XML. The GFMIEDM XSD stores all data values in XML elements. Only security markings (which are ignored in this report) are stored in XML attributes. The ELEMENT column in the data dictionary corresponds to a table in SQL, and the ATTRIBUTE column contains SQL field names. SQL terms are used in reference to the XSD in the following sections.

The globally unique values in GFM DI are called Enterprise-wide Identifiers (EwIDs). A subset of these values are Force Management Identifiers (FMIDs). The data model is object-oriented, with EwIDs joining the elements that make up the generalization hierarchy. The EwID at the "top" of a hierarchy is an FMID. Simple tables also have FMIDs as their primary key to uniquely identify each record.

3.3 Data Sources

The XSD is the implementation of the GFMIEDM physical model. Data dictionary cells that contain logical elements of the model were derived when possible; otherwise, additional information had to be added to the XSD.

Element. The XSD names each table type instead of defining each table in place. The table type names are in mixed case, and the XSLT script adds a space before each capital letter to make the names more readable. (The name in the spreadsheet is in all capitals.) For example "AircraftType" becomes "Aircraft Type." The Element Abbreviation is the physical table name in parentheses, and the definition is taken from the `documentation` element.

Attribute. The logical name was not in the XSD and had to be added via an `appinfo` element. The Abbreviation is the table name, a period, and the field name, all surrounded by parentheses. (The field names in the XSD are in all capitals and converted into lower case for readability.)

Business Rules. This column is an example of new information that was added to the XSD, making it easier to understand. While the Data Implementation Business Rules (DIBR) document[6] describes the proper ways to produce GFM DI data, the data dictionary contains reminders of some of the main rules.

Datatype. Every datatype cell spans 3 columns. The first line indicates if the field is mandatory or optional along with a term which describes the datatype (Text, Number, Identifier, etc.). The next line names the XSD datatype, replacing both the Oracle[7] datatype and the validation column in the spreadsheet data dictionary. The third line contains the datatype definition. Fields that are EwIDs also contain the full names of fields that they reference in other tables. When a field contains enumerated values, additional lines display the display name of the value, its XSD value, and a description. All of these values are obtained directly from the XSD.

3.4 New Elements

Many of the changes to the XSD involved adding new text to existing `documentation` elements. However, logical names, FMID flags, EwIDs, references, and business rules had to be inserted in new `appinfo` elements.

Attribute. Every field required the addition of an `appinfo` element inside of its existing `annotation` element. There are GFM DI guidelines for how to create a field name, but there are exceptions and it would have been harder to write a comprehensive set of conversion rules than to add the logical names to the XSD.

Figure 4 shows the ACFT_TYPE_ID field. An `appinfo` element with a child `app:name` element has been added to hold the logical field name. (All new elements that were added for the data dictionary belong to the `app` namespace.)

```
<xs:element name="CAT_CODE" type="DS336_acft_type_cat_code">
  <xs:annotation>
    <xs:documentation>The specific value that represents the
    class of AIRCRAFT-TYPE.</xs:documentation>
    <xs:appinfo>
      <app:name>Aircraft Type Category Code</app:name>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Fig. 4 Addition of `name` element

The data dictionary denotes fields that are FMIDs. All EwID fields, whether they are primary (i.e., FMIDs) or refer to other attributes, use the datatype identifier20 or index20. Since there is nothing unique about FMIDs in the XSD, an empty element named `app:fmid` was added to mark these fields as shown in Fig. 5.

Other identifiers are foreign keys that refer to other elements. They are denoted with `app:ewid` elements that contain the table and field names of the primary key. The aircraft type identifier in Fig. 6 refers to the object type identifier field in the object type table.

Business Rules. The new `dibr` element, shown in Fig. 7, contains additional notes that are displayed in the Business Rules column of the data dictionary.

```
<xs:element name="ADDR_ID" type="identifier20">
  <xs:annotation>
    <xs:documentation>The unique value, or set of characters,
    assigned to represent a specific ADDRESS and to distinguish
    it from all other ADDRESSs.</xs:documentation>
    <xs:appinfo>
      <app:fmid/>
      <app:name>Address Identifier</app:name>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Fig. 5 Addition of `fmid` element

```
<xs:element name="ACFT_TYPE_ID" type="identifier20">
  <xs:annotation>
    <xs:documentation>The equipment-type-id of a specific
    AIRCRAFT-TYPE (a role name for object-type-id).
    </xs:documentation>
    <xs:appinfo>
      <app:name>Aircraft Type Identifier</app:name>
      <app:ewid>OBJ_TYPE.obj_type_id</app:ewid>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Fig. 6 Addition of `ewid` element

```
<xs:element name="CAT_CODE" type="DS4138_addr_cat_code">
  <xs:annotation>
    <xs:documentation>The specific value that represents
    the class of ADDRESS. It serves as a discriminator that
    partitions ADDRESS into subtypes.</xs:documentation>
    <xs:appinfo>
      <app:name>Address Category Code</app:name>
      <app:dibr>This is for the optional PHYADR of the facility
      that serves as the homestation of the organisation or the
      organisation's homestation mailing address.</app:dibr>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Fig. 7 Addition of `dibr` element

3.5 Missing Documentation

The Joint Consultation, Command and Control IEDM (JC3IEDM),[8] which was the basis for the GFMIEDM, contains the logical name of each enumerated value at the beginning of the `documentation` element as shown in italics in Fig. 2. The logical name was manually copied from the data dictionary into the XSD for every enumerated value that was missing the name.

3.6 New Documentation

Modifications to the GFMIEDM XSD are approved by the Configuration Control Board (CCB), and each update is given a Change Request (CR) number. In late 2011 the CRs were combined into "packages" with an implementation date for each package. New elements `app:change-request` and `app:package` were added to the XSD. As shown in Fig. 2, `app:package` contains child elements `app:number` and `app:date`. These elements are provided for the GFM DI developers and are currently ignored by the XSLT script.

3.7 Dictionary Changes

During the development of the XSLT script, the spreadsheet version of the data dictionary underwent a lot of scrutiny and some changes were submitted to the CCB. (The addition of `appinfo` elements was also a CR.) Spelling mistakes and other errors were not fixed in the spreadsheet, but the changes were applied to the XSD files and incorporated into the XSLT script.

The JC3IEDM was written using ERwin with Oracle as the target relational database management system, and the definitions explicitly named the Oracle datatypes. These definitions were copied into the JC3IEDM XSD. The GFMIEDM has specified XML as the target, and the XSD datatypes should be sufficient. As part of the trend to move away from SQL and provide all documentation in XML terms, the Oracle datatype was deleted from the fifth column. It was replaced with the XML datatype as defined by the GFM DI XSD. The VALIDATION column of the spreadsheet was deleted because it was redundant; the enumeration rule's name is the same as the datatype.

3.8 XSLT Script

3.8.1 Language Versions

XPath 1.0[9] started as a sublanguage of XSLT 1.0.[10] While both languages have evolved into version 2.0, many tools support only the earlier version, and the XSLT script was written using XSLT 1.0 and XPath 1.0. However, the version 2 features are used in this report because they are shorter, clearer, and easier to understand. The version 2.0 code is typeset in san serif in the program listings.

3.8.2 Basic Approach

The GFMIEDM XSD consists of 7 XSD files, 5 of which describe the data schema. Only 3 of these are required to produce the data dictionary.

GFMIEDM353tables.xsd defines each table element which consists of the table name and its type. The element for the `Aircraft Type` table is shown in Fig. 8.

GFMIEDM353relatTableTypes.xsd defines the `documentation` and all of the fields inside of each table. As shown in Fig. 9, each field consists of a name, type, definition, and `app:name` with the logical name.

GFMIEDM353simpleTypes.xsd defines all of the datatypes. The only information needed in a simple type are the name and definition as highlighted in Fig. 10. The enumerated type in Fig. 11 adds a value and definition for each enumeration.

```
<xs:element name="ACFT_TYPE_TBL">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ACFT_TYPE" type="AircraftType"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fig. 8 Definition of `ACFT_TYPE_TBL` element

```
<xs:complexType name="AircraftType">
  <xs:annotation>
    <xs:documentation>An EQUIPMENT-TYPE that is designed
    to fly.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="ACFT_TYPE_ID" type="identifier20">
      <xs:annotation>
        <xs:documentation>The equipment-type-id of a specific
        AIRCRAFT-TYPE (a role name for object-type-id).
        </xs:documentation>
        <xs:appinfo>
          <app:ewid>OBJ_TYPE.obj_type_id</app:ewid>
          <app:name>Aircraft Type Identifier</app:name>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="CAT_CODE" type="DS336_acft_type_cat_code">
      <xs:annotation>
        <xs:documentation>The specific value that represents
        the class of AIRCRAFT-TYPE.</xs:documentation>
        <xs:appinfo>
          <app:name>Aircraft Type Category Code</app:name>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    ...
  </xs:sequence>
</xs:complexType>
```

Fig. 9 Definition of Aircraft Type fields

```
<xs:simpleType name="txt_optional_100">
  <xs:annotation>
    <xs:documentation>100-character string.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="ascii_string">
    <xs:maxLength value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Fig. 10 Definition of txt_optional_100 datatype

```
<xs:simpleType name="DS336_acft_type_cat_code">
  <xs:annotation>
    <xs:documentation>Datatype for the validation rule
    DS336_acft_type_cat_code</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="AIRRW">
      <xs:annotation>
        <xs:documentation>Rotary wing, manned: A machine or
        device capable of atmospheric flight and dependent on
        rotating blades for lift.</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="FIXWNG">
      <xs:annotation>
        <xs:documentation>Fixed wing, manned: A manned machine or
        device capable of atmospheric flight and dependent on wings
        for lift.</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    ...
  </xs:restriction>
</xs:simpleType>
```

Fig. 11 Definition of DS336_acft_type_cat_code

3.8.3 Documented Code

The transformation engines used to run the script on the XSD files are the free version of Saxon[11] and the commercial product XMLSpy[12] which was also used to locate stray portions of XPath 2.0 code. Both products operate on a single input file so the XPath **document** function was used in the script to load the other 2 files.

Initialize. The script begins with the standard `xsl:stylesheet` root element. The highlighted text shows the addition of the `app:` namespace. Since the output will be an HTML file, the output method is declared to be HTML 4.0. (Technically, XSLT uses XHTML because the tags must have matching end tags.)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:app="urn:us:gov:dod:gfmdi:appinfo">
  <xsl:output method="html" version="4.0"
    doctype-public="-//W3C//DTD HTML 4.0 Transitional//EN"
    doctype-system="http://www.w3.org/TR/REC-html40/loose.dtd"
    indent="yes"/>
```

The other 2 XSD files are loaded into variables. This is practical because they are relatively small.

```
<!-- load all required XSD files into variables -->
<!-- the file GFMIEDM353tables.xsd has already been loaded -->
  <xsl:variable name="relat"
              select="document('GFMIEDM353relatTableTypes.xsd')"/>
  <xsl:variable name="simple"
                select="document('GFMIEDM353simpleTypes.xsd')"/>
```

As is the case in most XSLT scripts, the template that matches the root element is the main driver. Any elements that are not XSLT elements are assumed to be HTML tags and are written to the output file verbatim.

```
<!-- root writes general HTML wrapper and processes all data -->
  <xsl:template match="/">
    <html>
      <head>
        <title>Data Dictionary for GFMIEDM 3.5.3</title>
```

Color coding is used in the data dictionary to denote GFM DI tables and fields, FMID and EwID fields, and mandatory fields. Cascading Style Sheets (CSS)[13] are used to provide this functionality. The comments in the style definitions explain the color specifications.

```
<!-- colors and other details are defined in CSS styles -->
  <style type="text/css">
    body, th, td {
      font-family: verdana, arial, helvetica, sans-serif;
      font-size: small;
      color: black;
    }
  <!-- align all table elements vertically for multiple-line
        descriptions and collapse all borders to eliminate
        white lines
  -->
    table { border-collapse: collapse; }
    table, th, td { border: 1px solid black; }
    th, td { padding: 5px; vertical-align: top; }
  <!-- header has grey background -->
    th { background-color: #C0C0C0; text-align: left; }
  <!-- table separator has black background -->
    th.sep { background-color: black; height: 8px; }
  <!-- GFM table names are white text on green -->
    td.gfm-table { background-color: green; color: white; }
  <!-- FMID field names are blue text on white -->
    td.fmid-field,
    span.fmid-field { color: blue; }
  <!-- GFM field names are green text on white -->
    td.gfm-field,
    span.gfm-field { color: green; }
  <!-- FMID field types are black text on light blue -->
    td.fmid-type { background-color: #CCFFFF; }
    span.fmid-type { color: #CCFFFF; }
  <!-- Mandatory field types are black text on light green
  --> td.man-type { background-color: #CCFFCC; }
    span.man-type { color: #CCFFCC; }
  <!-- Optional field types are black text on light yellow
  -->
    td.opt-type { background-color: #FFFF99; }
    span.opt-type { color: #FFFF99; }
  </style>
```

The final part of the HTML header is a block of JavaScript[14] code that declares a variable and defines a simple function. It is used to generate the attribute numbers in column one when the file is viewed in a browser.

```
<!-- JavaScript to implement a row counter -->
  <script type="text/javascript">
    rowCount = 0;
    function inc()
    {
      return ++rowCount;
    }
  </script>
</head>
```

The initialization ends with the start of the body of the HTML file. A big HTML table is used to construct the rows and columns of the data dictionary. The first row (`tr` = table row) builds the table headers (`th`); the `span` elements show how the CSS styles are invoked.

```
<body>
  <h1>Data Dictionary for GFMIEDM 3.5.3</h1>
<!-- the entire HTML file is one giant table -->
  <table>
  <tr>
  <!-- Row Number -->
    <th>#</th>
  <!-- Element = Table -->
    <th>
      ELEMENT
      <br />Name / Abbreviation / Definition
    </th>
  <!-- Attribute = Field -->
    <th>
      ATTRIBUTE
      <br />Name / Abbreviation / Definition
      <br />(JC3IEDM /
      <span class="gfm-field"> GFM-specific</span> /
      <span class="fmid-field"> FMID</span>)
    </th>
  <!-- Business Rules from the DIBR -->
    <th>Business Rules and Guidance for GFM DI</th>
    <!-- datatype spans 3 columns for enumerated values -->
```

```
      <th>Valid Value Display Value</th>
      <th>Valid Value Data Value</th>
      <th>
        ATTRIBUTE Valid Value Definition
        <br />
        <span class="man-type"> Mandatory</span> /
        <span class="opt-type"> Optional</span> /
        <span class="fmid-type"> Identifier</span>
      </th>
    </tr>
```

A `for-each` loop examines each element that corresponds to a table type definition. The element name is saved as the full table name, while the short table name is extracted from the nested sequence element. Elements which define hierarchical tables (i.e., full table names that contain the string "_OO_") are ignored.

The file GFMIEDM353relatTableTypes.xsd, now stored in the variable relat, defines each table type as an XSD complexType. The next template is invoked with the element that defines the current table type and the short table name.

A black row to visually separate the tables is written before the loop ends.

```
<!-- each table -->
<!-- the pattern matches the tables in the file
GFMIEDM353tables.xsd -->
  <xsl:for-each select="/*/xs:element">
  <!-- remember the table name with and w/o the suffix -->
    <xsl:variable name="full-table-name"
                  select="@name"/>
    <xsl:variable name="short-table-name"
                  select="*/*/xs:element/@name"/>
    <xsl:choose>
      <xsl:when test="contains($full-table-name, '_OO_')">
        <!-- ignore object-oriented (hierarchical) tables -->
      </xsl:when>
      <xsl:otherwise>
      <!-- remember the table type -->
        <xsl:variable name="table-type"
                      select="*/*/xs:element/@type"/>
      <!-- process the table type from the "relat" XSD file -->
        <xsl:apply-templates
          select="$relat//xs:complexType[@name=$table-type]">
```

```
            <xsl:with-param name="table-name"
                            select="$short-table-name"/>
          </xsl:apply-templates>
        <!-- separate Elements with a black row -->
          <tr>
            <th class="sep" colspan="8"/>
          </tr>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
```

The template ends by terminating the open elements in the HTML file.

```
      </table>
    </body>
  </html>
</xsl:template>
```

Table Processing. The table processing template defines several variables to simplify the rest of the code. To properly nest the table cells, the number of rows required to display each datatype used by the table must be computed. The variable enum-counts is a temporary tree that has an element for each table type with the number of enumerated values (plus one for the type itself). The counts for the Aircraft Type table are shown in Fig. 12. The total number of rows is stored in the variable num-rows. The variables table-class, prefix-count, and suffix are explained in the following when they are used by the script.

```
<!-- this template is invoked with each table type -->
  <xsl:template match="//xs:complexType">
    <xsl:param name="table-name"/>
  <!-- count number of enumerations in each field type plus 1
       and store them in a tree fragment
  -->
    <xsl:variable name="enum-counts">
      <blk>
        <xsl:for-each select=".//xs:element/@type">
          <xsl:variable name="type-name" select="."/>
          <val>
            <xsl:attribute name="type">
              <xsl:value-of select="$type-name"/>
            </xsl:attribute>
```

```
            <xsl:value-of
        select="count($simple//xs:simpleType[@name=$type-name]/*/
        xs:enumeration) + 1"/>
            </val>
          </xsl:for-each>
        </blk>
      </xsl:variable>
    <!-- compute number of rows needed by all types and their
         enumerations -->
      <xsl:variable name="num-rows">
          <xsl:value-of
          select="sum($enum-counts/blk/val)"/>
      </xsl:variable>
      <xsl:variable name="table-class"
                  select="if (starts-with(@name,'GFM')) then 'gfm-table'"/>
    <!-- prepare to break table name apart at each capital letter -->
    <!-- how many initial letters must be skipped? -->
      <xsl:variable name="prefix-count">
        select="if (starts-with(@name,'GFM')) then 3 else 0"/>
      <xsl:variable name="suffix" select="substring(@name,
                    $prefix-count + 1)"/>

<blk>
   <val type="identifier20">1</val>
   <val type="DS336_acft_type_cat_code">7</val>
   <val type="DS4365_acft_type_arfrm_dsgn_cd">12</val>
   <val type="DS4366_acft_type_manning_code">6</val>
   <val type="DS4367_acft_type_mil_civ_code">4</val>
   <val type="DS4368_acft_type_main_purp_cd">62</val>
   <val type="DS4372_acft_type_train_cat_cd">4</val>
   <val type="DS4204_acft_type_toff_land_cd">6</val>
   <val type="GF001_acft_type_eng_ind_code">3</val>
</blk>
```

Fig. 12 Row counts for Aircraft Type

Field Processing. The template loops through the fields in the table, performing the same kinds of actions that were done for the table. The field-class and fmid-man-opt-class variables are assigned CSS class names to apply the proper styles to the text of the field name and type, respectively. The field contains an FMID if the `fmid` element is present, or it is defined for GFM DI if the name starts with `GFM`. The field type is an FMID if the `fmid` element is present; it is optional if the XSD attribute `minOccurs='0'` is present; it is an FMID if the datatype is identifier20

or index20; otherwise it is mandatory. The number of enumeration rows is extracted from the variable enum-counts as before.

To minimize duplicate code, the last 2 variables store HTML text. The text for the field type cell is stored in td-type where the contents are the logical name, the physical table and field names, and the field definition. (If the logical name is absent, the physical name is used. The HTML tag `td` = table detail and `br` produces a line break.) The variable td-dibr contains the optional business rules.

```
<!-- each field -->
  <xsl:for-each select=".//xs:element">
    <xsl:variable name="type-name" select="@type"/>
  <!-- is this an FMID or GFM field? -->
    <xsl:variable name="field-class"
      select="if (xs:annotation/xs:appinfo[app:fmid]) then 'fmid-field'
              else if (matches(@name,'^GFM')) then 'gfm-field'"/>
  <!-- is this field an FMID, optional, or mandatory? -->
    <xsl:variable name="fmid-man-opt-class" select="
        if (xs:annotation/xs:appinfo[app:fmid]) then 'fmid'
        else if (@minOccurs='0') then 'opt'
        else if (matches(@type,'^identifier20$')) then 'fmid'
        else if (matches(@type,'^index20$')) then 'fmid'
        else 'man'"/>
  <!-- we need to span multiple enumerations -->
    <xsl:variable name="num-enums">
      <xsl:value-of
      select="$enum-counts/blk/val[@type=$type-name]"/>
    </xsl:variable>
  <!-- save TD element with field and definition contents -->
    <xsl:variable name="td-type">
      <td class="$field-class" rowspan="$num-enums">
        <b>
          <xsl:value-of select="
            if (xs:annotation/xs:appinfo[app:name])
            then xs:annotation/xs:appinfo/app:name
            else @name"/>
        </b>
        <br />
        <xsl:text>(</xsl:text>
        <xsl:value-of select="$table-name"/>
        <xsl:text>.</xsl:text>
        <xsl:value-of select="lower-case(@name)"/>
```

```
        <xsl:text>)</xsl:text>
        <br /><br />
        <xsl:value-of select="xs:annotation/xs:documentation"/>
      </td>
    </xsl:variable>
  <!-- save TD element with DIBR instruction -->
    <xsl:variable name="td-dibr">
      <td class="$field-class" rowspan="$num-enums">
        <xsl:value-of
        select="xs:annotation/xs:appinfo/app:dibr"/>
      </td>
    </xsl:variable>
```

The rest of the loop produces output for the HTML file. The initial cell contains a snippet of JavaScript code to number each field for future reference. The first row in the HTML table is special because the schema table cell must span all of the field rows. The number of cells has already been computed and stored in num-rows, while the table-class variable contains the class name to apply the GFM DI formatting style (if appropriate). (An empty argument is ignored by the web browser.)

The first prefix-count characters of the `name` are capital letters, and they are placed into the cell. The remaining characters have a blank inserted in front of each remaining capital letter. (For example, GFMCrewPlatformType becomes GFM␣Crew␣Platform␣Type.) The next line of the cell has the table name in parentheses, while a blank line precedes the table's definition.

```
<tr>
<!-- insert row counter -->
  <td>
    <script type="text/javascript">
      document.write(inc());
    </script>
  </td>
<!-- insert table's logical name, physical name, and definition
-->
  <td rowspan="$num-rows" class="$table-class">
    <b>
      <xsl:value-of select="substring(@name, 1,
      $prefix-count)"/>
      <xsl:value-of select="replace($suffix, '([A-Z])', '␣$1')"/>
    </b>
```

```
    <br />
    <xsl:text>(</xsl:text>
    <xsl:value-of select="$table-name"/>
    <xsl:text>)</xsl:text>
    <br /><br />
    <xsl:value-of select="xs:annotation/xs:documentation"/>
  </td>
```

The first field in the table is treated differently than the rest because its row has already been created, and the first 2 cells have been filled. For the remaining rows, a new row is created and the JavaScript counter is inserted in the first cell. The second cell may be ignored because the `rowspan` attribute was used. In both cases, the type and business rule cells are printed from their variables, and a template is invoked for the field's datatype using the contents of the file GFMIEDM353simpleTypes.xsd.

```
<!-- first row is different than other rows in table -->
  <xsl:choose>
    <xsl:when test="position() = 1">
      <xsl:copy-of select="$td-type"/>
      <xsl:copy-of select="$td-dibr"/>
      <xsl:apply-templates
        select="$simple//xs:simpleType
        [@name=current()/@type]">
        <xsl:with-param name="css-name"
                        select="$fmid-man-opt-class"/>
        <xsl:with-param name="field-class"
                        select="$field-class"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <tr>
        <td rowspan="$num-enums">
          <script type="text/javascript">
            document.write(inc());
          </script>
        </td>
        <xsl:copy-of select="$td-type"/>
        <xsl:copy-of select="$td-dibr"/>
        <xsl:apply-templates
          select="$simple//xs:simpleType[@name=current()/
```

```
                @type]">
                <xsl:with-param name="css-name"
                                select="$fmid-man-opt-class"/>
                <xsl:with-param name="field-class"
                                select="$field-class"/>
              </xsl:apply-templates>
            </tr>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
```

The template ends by terminating the row.

```
    </tr>
  </xsl:template>
```

Type Processing. This template produces the contents of the cell that spans all 3 cells of field's datatype. The text `Mandatory` or `Optional` is printed based on the CSS class name. To avoid adding more metadata to the XSD, the prefix of each datatype (or the entire type name) is examined to produce the corresponding text. If the field is not an enumerated type, the type's definition is printed. The next template is invoked to generate the text for the enumerated values of the field.

```
<!-- this template is invoked with each field -->
  <xsl:template match="//xs:simpleType">
    <xsl:param name="css-name"/>
    <xsl:param name="field-class"/>
    <td colspan="3" class="$css-name-type">
      <xsl:choose>
        <xsl:when test="$css-name = 'man' or $css-name = 'fmid'">
          <xsl:text>Mandatory </xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>Optional </xsl:text>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:choose>
        <xsl:when test="starts-with(@name, 'cnt_')">
          <xsl:text>Number</xsl:text>
        </xsl:when>
        <xsl:when test="starts-with(@name, 'dttm_')">
```

```
        <xsl:text>Date-Time-Group</xsl:text>
      </xsl:when>
      <xsl:when test="starts-with(@name, 'qty_')">
        <xsl:text>Number</xsl:text>
      </xsl:when>
      <xsl:when test="@name = 'identifier20' or
                      @name = 'index20'">
        <xsl:text>Identifier</xsl:text>
      </xsl:when>
      <xsl:when test="starts-with(@name, 'txt_')">
        <xsl:text>Text</xsl:text>
      </xsl:when>
      <xsl:when test=".//xs:enumeration">
        <xsl:text>Code</xsl:text>
      </xsl:when>
    </xsl:choose>
    <br />
    <xsl:value-of select="@name"/>
    <xsl:if test="not(.//xs:enumeration)">
      <br />
      <xsl:value-of select="xs:annotation/xs:documentation"/>
      <xsl:if test="string-length($ewid-text) > 0">
        <br />
        <xsl:text>UID value inherited from </xsl:text>
        <b>
          <xsl:value-of select="$ewid-text"/>
        </b>
      </xsl:if>
    </xsl:if>
  </td>
</xsl:template>
```

Enumerated Values. The last template uses variables to make the code more legible. The writers of the JC3IEDM XSD put the logical name in front of the definition for each enumerated value. Logical names were manually inserted into each definition in the XSD that did not already have its name.

A row is produced with 4 cells consisting of the logical (display) name, enumerated value, definition, and the datatype name. The preceding cells in the row are ignored because the `rowspan` attribute was used. This section of code clearly shows how CSS classes are used to apply a particular style to the cells.

```
<!-- this template is invoked with each enumerated value -->
  <xsl:template match="//xs:enumeration">
    <xsl:param name="field-class"/>
    <xsl:variable name="display"
                  select="
      if (matches(xs:annotation/xs:documentation, ': '))
        then substring-before(xs:annotation/xs:documentation, ': ')
      else '***MISSING***'"/>
    <xsl:variable name="definition"
                  select="
      4 (matches(xs:annotation/xs:documentation, ': '))
        then substring-after(xs:annotation/xs:documentation, ': ')
        else xs:annotation/xs:documentation"/>
    <tr>
      <td class="$field-class">
        <xsl:value-of select="$display"/>
      </td>
      <td class="$field-class">
        <xsl:value-of select="@value"/>
      </td>
      <td class="$field-class">
        <xsl:value-of select="$definition"/>
      </td>
      <td class="$field-class">
        <xsl:value-of select="$type-name"/>
      </td>
    </tr>
  </xsl:template>
```

3.9 HTML Output

A fragment of the output produced by the script, as displayed by a web browser, is shown in Fig. 13.

#	ELEMENT Name / Abbreviation / Definition	ATTRIBUTE Name / Abbreviation / Definition (JC3IEDM / GFM-specific / FMID)	Business Rules and Guidance for GFM DI	Valid Value Display Value	Valid Value Data Value	ATTRIBUTE Valid Value Definition Mandatory / Optional / Identifier
6	**Aircraft Type** (ACFT_TYPE) An EQUIPMENT-TYPE that is designed to fly.	**Aircraft Type Identifier** (ACFT_TYPE.acft_type_id) The equipment-type-id of a specific AIRCRAFT-TYPE (a role name for object-type-id).		Mandatory Identifier identifier20 Specification for identifiers. UID value inherited from **OBJ_TYPE.obj_type_id**		
7		**Aircraft Type Category Code** (ACFT_TYPE.cat_code) The specific value that represents the class of AIRCRAFT-TYPE.		Mandatory Code DS336_acft_type_cat_code		
				Rotary wing, manned	AIRRW	A machine or device capable of atmospheric flight and dependent on rotating blades for lift.
				Fixed wing, manned	FIXWNG	A manned machine or device capable of atmospheric flight and dependent on wings for lift.

Fig. 13 HTML data dictionary fragment

4. Discussion

4.1 Data Dictionary Differences

A primary goal for creating the data dictionary from the XML schema was to use existing information from the XSD files when possible. The definitions in the datatypes are more detailed than the corresponding cells in the Excel spreadsheet. In the case of date/time groups, the required format is described in the field's definition in the spreadsheet and in the field's type definition in the HTML file. Figure 14 shows how the format has moved from the field definition (upper text) to the field type definition (lower text). The highlighted text is the same in both dictionaries.

```
GFM Address Termination DTG
(ADDR.gfm_addr_t_dtg)
End DTG defining the termination of the viable time interval
of this data.
Use XML dateTime option requiring 20 characters:
YYYY-MM-DDTHH:MM:SSZ.
Example: 2007-09-12T14:58:59Z

Mandatory Date-Time-Group
The designation of a specific year, month, day, hour, minute,
second and milliseconds. Format is YYYY-MM-DDTHH:MM:SSZ. This
is based on ISO-8601.
```

Fig. 14 Date/time group definitions

When an element is part of a generalization hierarchy (i.e., it is a component of an Object Type or Object Item) the identifiers that make up the chain are listed in the field type's definition. This is shown in Fig. 3. The text was manually added to both the spreadsheet and the XSD. While it may be possible for the script to derive the text, it is unlikely that the values will change.

4.2 Documentation Revisions

A thorough review of both the Excel spreadsheet and the XSD was required to ensure that all of the necessary information was present in the XSD files. The parts of the XSD that were written for JC3IEDM had never been modified (a basic tenet of GFM DI), and some mistakes were discovered. All of the XSD `documentation` elements were spell checked and errors were corrected. The JC3IEDM writers prefixed some of the `documentation` elements with "Definition: ". This text is redundant and it was deleted from the XSD files.

A minor change was expanding ID to Identifier for each field's logical name.

4.3 Future Enhancements

The Excel spreadsheet has an important feature that is currently missing from the HTML data dictionary—the column headers stay at the top when the user scrolls through the data. This ability will be added to the next version of the HTML data dictionary along with hardwired column widths.

A table of contents may be added at the top of the HTML file with hyperlinks to the tables. Additional links will allow the user to easily explore the data dictionary. For example, FMIDs or CR packages could be chained together.

A second script could be written to extract information about desired CRs or packages because a developer may need to know how the current XSD differs from the previous schema. As an alternative, the script could highlight changes in the data dictionary for a given package number.

A condensed dictionary may be desired without all of the enumerated values. The users of the new data dictionary will probably request new features as they become accustomed to the HTML version.

The long-term goal is to define the GFMIEDM in Unified Modeling Language (UML)[15] in place of the ERwin E-R diagram. The UML model will be used to produce both the XSD schema and the data dictionary.

5. Conclusion

Generating a data dictionary from an XML schema is possible with minor additions to the schema files. The documentation added to the schema enhances the understanding of a developer who reads the schema files. The HTML file produced by the XSLT script contains all of the information that is in the current Excel spreadsheet, and it may be viewed in any web browser. When the schema is modified, the script may be run to produce an updated data dictionary, preventing the divergence that occurs when distinct files must be maintained manually.

This work has also demonstrated how an XSD may be parsed to supply information in a different form. For example, the IChart force structure viewer and editor application[16] could extract the definition of each enumerated value and present it to the user. When users are prompted to choose a weapon type they will not have to remember what `ATGRLC` stands for and how an `ATGRLH` differs from an `ATGRLL`.

Additional information could be added to the XSD to be extracted for other purposes.

6. References

1. Chamberlain SC, Boller M, Sprung G, Badami V. Establishing a community of interest (COI) for global force management. In: Proceedings of the 10th International Command and Control Research and Technology Symposium; 2005 June; McLean, VA.

2. Computer Associates. ERwin. 2012 April [accessed 2012 Jun 11]. http://erwin.com/.

3. W3C. XML schema part 1: Structures. 2004 October [accessed 2009 Aug 20]. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

4. W3C. XML schema part 2: Datatypes. 2004 October [accessed 2009 Aug 20]. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

5. W3C. XSL Transformations (XSLT) version 1.0. 1999 November [accessed 2009 Aug 20]. http://www.w3.org/TR/1999/REC-xslt-19991116/.

6. Global force management data initiative (GFM DI) data implementation business rules (DIBR) for GFM XSD v3.5.2. 2011 December [accessed 2015 Oct 19]. https://inteldocs.intelink.gov/inteldocs/page/document-details?nodeRef=workspace://SpacesStore/b5dc19f1-9cab-4afa-9893-4a8b7fd969f4.

7. Oracle. Oracle database documentation library. [accessed 2009 Aug 20]. http://www.oracle.com/pls/db111/homepage.

8. MIP. JC3IEDM browse representation. 2007 December [accessed 2009 Aug 20]. http://www.mip-site.org/publicsite/04-Baseline_3.0/JC3IEDM-Joint_C3_Information_ Exchange_Data_Model/HTML-Browser/index.html.

9. W3C. XML path language (XPath) version 1.0. 1999 November [accessed 2009 Aug 20]. http://www.w3.org/TR/1999/REC-xpath-19991116/.

10. Kay M. XSLT 2.0 programmer's reference (programmer to programmer). Hoboken (NJ): Wrox; 2004.

11. Kay M. Saxon-B 9.0.0.4J. 2008 March [accessed 2009 Aug 20]. http://www.saxonica.com/.

12. Altova. XMLSpy. 2008 [accessed 2009 Aug 20]. http://www.altova.com/products/xmlspy/xml_editor.html.

13. W3C. Cascading style sheets level 2 revision 1 (CSS 2.1) specification. 2011 June [accessed 2012 May 7]. http://www.w3.org/TR/2011/REC-CSS2-20110607/.

14. Mozilla. JavaScript Reference. 2011 October [accessed 2012 May 15]. https://developer.mozilla.org/en/JavaScript/Reference.

15. Object Management Group. UML resource page. 2011 February [accessed 2009 Aug 20]. http://www.uml.org.

16. Brundick FS, Hartwig GW Jr, Chamberlain SC. IChart: a graphical tool to view and manipulate force management structure databases. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2008 Sep. Report No.: ARL-TR-4610.

List of Symbols, Abbreviations, and Acronyms

CCB	Configuration Control Board
CR	Change Request
CSS	Cascading Style Sheet
DI	Data Initiative
E-R	entity relationship (diagram)
EwID	Enterprise-wide Identifier
FMID	Force Management Identifier
GFM	Global Force Management
HTML	Hypertext Markup Language
IEDM	information exchange data model
JC3	Joint Consultation, Command and Control
SQL	Structured Query Language
UML	Unified Modeling Language
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language: Transformations

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO L IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
4 (PDF)	DIR USARL RDRL CII T R HOBBS F BRUNDICK T HANRATTY M MITTRICK