

**XML USE IN TECHNICAL DOCUMENTATION
THE SOCIETY OF BROADCAST ENGINEERS
EFD PROJECT**

David M. Baden
Chief Technology Officer
Radio Free Asia
2025 M Street NW
Washington, DC 20036

ABSTRACT

Engineer Friendly Documentation (EFD) is an XML based “technical documentation” standard proposed by the Society of Broadcast Engineers. The purpose of EFD is to facilitate the migration of technical information to an efficient and standardized documentation platform. EFD is an effort to define a common technical documentation content model in order to make it easier for broadcast engineers to find the information they need. Use of the EFD XML standard will allow “compliant” documentation to have consistency in data content structure. With this consistency benefits such as simplifying specification comparisons across multiple manufacturer’s product lines can be realized.

EFD XML BASICS

Extensible Markup Language (XML) is a subset of SGML (Standardized General Markup Language). The World Wide Web Consortium (W3C) has released the first version of the Extensible Markup Language (XML) specification to broad support. XML was designed to fill a deficiency identified in other Internet and markup standards by supporting data definition and information processing requirements. XML provides a middle ground between the restrictions and general inflexibility of HTML, and the overall complexity of SGML.

The usability of the XML data content files in the EFD are depend on two key components, the DTDs (Document Type Definitions), and XSL (Extensible Stylesheet Language) files.

The DTD is used to formally define and validate the overall structure and content of all EFD informational data files. Simply put, the DTD acts as a template used to define the data structure and relationships between the various data elements.

XSD XML Schema files are also available for the EFD Format. This is done in recognition of the eventuality of DTD replacement with the more XML specific XML Schema as the conforming document. In the interim the DTD is to be used as the primary validation document for the EFD.

XSL is a language for creating Stylesheets for XML documents. An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of that class is transformed onto various platforms with the use of the XSL formatting vocabulary. The XSL vocabulary uses XML like notation. In XSL, the formatting object tree can be radically different from the source data tree and inheritance of formatting properties is on the formatting object tree.

XML BASICS

XML is a self-describing domain-specific markup language syntax. By construction, XML documents as a subset of SGML (Standardized General Markup Language) are conforming SGML documents.

The W3C classifies XML as a common syntax for expressing structure in data. XML describes a class of data objects or XML documents. XML also servers to describe the behavior of XML processors. XML processors (or Parsers) are software modules, generally working on behalf of an application, that are used to interpret XML documents in order to provide data content and data structure access.

XML structured data refers to data that is labeled or tagged for its content, meaning, or use. XML separates data structure and content from the presentation of that data. The ability to separate data from presentation allows for the possibility of XML becoming the standardized mechanism for the exchange of data as well as a universal document translation platform.

XML documents are made up of storage units called entities that contain data in “Elements”. Elements can contain either parsed or unparsed data. Parsed data (designated as PCDATA for Parsed Character Data) is made up of characters, some of which form text data, and some of which can form markup. The XML Processor parses PCDATA as the document is used by an application. As parsed data the use of reserved escape character data is prohibited.

If the content of data elements does contain XML escape characters then the data can be designated as unparsed data. Unparsed data, known as CDATA (Character Data), passes straight through the XML Processor without parsing. CDATA allows XML escape and command characters within the data to pass through the XML Processor without acting on the commands.

XML is not in itself a language. XML is a standard that allows the creation of document unique language that conforms to the XML criteria. XML is a self-describing domain-specific markup language syntax. XML syntax encodes a description of the XML document's storage layout and logical structure. XML syntax provides a mechanism to impose constraints on the data, data storage layout and document logical structure.

Well-Formed and Valid XML Documents

There are two levels of conformance for XML documents, “Well-Formed” and “Valid”. To be Well-Formed an XML document must adhere to the basic syntax rules of XML as specified by the W3C XML Standard. To be Well-Formed an XML document must have exactly one root element, and every sub-element (and recursive sub-elements) have delimiting start and end tags which are properly nested within each other.

A Valid XML documents must also adhere to the basic syntax rules of XML additionally they must also conform to the allowed document layout and structure constraints specified in an external document. External documents that provide these content models for XML documents may be either a Document Type Definition file (DTD designated with a “.dtd” file extension) or an XML Schema file (designated with a “.xsd” file extension). A Well-Formed XML document can be correct with or without a separate DTD or Schema file. All Valid XML documents must first be Well-Formed. Not all Well-Formed XML documents are Valid. To be Valid an XML Parser must verify that the document

conforms to the allowed document structure and syntax as defined in a DTD or Schema file.

DTDs and XML Schemas and Validation

A DTD (Document Type Definition) or an XML Schema is a document that is a set of syntax rules used to define the content model for XML documents. A DTD or an XML Schema describes the valid order and nesting of allowed data elements, the data types for those elements, designates the relationship between the multiple elements and defines attributes of the elements used in the XML document.

Validation compares a particular XML document structure and data types against a DTD or XML Schema. To check validity XML documents pass through a validating processor, which reports any errors found. XML processors (i.e. parsers in XML enabled Browsers) test the XML document against the basic syntax rules of XML and the specific DTD or Schema file. A document that passes such a consistency and validation test is Well-Formed and Valid.

XML Processors can test for conformance within the general requirements of XML for an XML document that does not have a DTD or Schema. A document that passes this lesser standard is Well-Formed. XML processors will fail to render an XML document that is not Well-Formed or fails to conform to a DTD or Schema file (if one exist).

Validation ensures that data is correct before passing it into an application by confirming that the data structure format, as defined in the DTD or Schema, is adhered to. As XML is not a language itself, but rather a system for defining languages, it does not have a universal DTD or Schema. Therefore any industry or organization that wishes to utilize XML for data exchange must define its own DTD or Schema.

The DTD

A Document Type Definitions or DTD file lists the data structure and type of the elements, attributes, and entities contained in an XML document. It defines the order of and the relationships between the different elements and attributes. Document Type Definition can be defined internally from inside an XML document or externally as a separate DTD file. External DTD files must be called from within the XML document.

The DTD format is inherited from and is the schema mechanism for SGML. As with SGML, the elements of an XML document can be formally specified using a DTD

The use of DTD files to define XML document structure is limited in several ways; DTDs are written SGML syntax, which differs from XML syntax. DTDs can only express the data type of attributes in terms of explicit enumerations and a few string formats, there is no facility for describing numbers, dates or currency values. DTDs do not have the ability to express the data type of character data within elements. More importantly DTD files do not fully support XML namespaces.

XML Schema

Schema definition language offers facilities for describing the structure and constraining the contents of XML documents. A Schema is a model for describing the structure of information. Schema is a term borrowed from the database community and is used to describe the structure of data in relational tables. An XML Schema describes a model for a whole class of documents. The XML Schema model describes the possible arrangement of data tags and text in a valid document. An XML Schema can also be described as an agreement on a common vocabulary for a particular application that involves exchanging documents or data.

XML Schema language resembles XML in syntax. XML Schemas allow for a broader definition of allowed data types. The expanded data type definitions in XML Schemas allow for more robust portability between XML documents and databases. XML Schema files have a major advantage over DTD files when defining XML documents as they can process namespaces more efficiently. Namespaces allow the validating rules for data elements in XML documents to be defined in multiple Schema files, possibly residing in physically different locations.

DTD or Schemas

While the use of an XML Schema to define XML documents overcomes many of the limitation of the DTD, the DTD is still the dominant validation file format in use for XML. This is primarily due to the comparative age of the two formats. The DTD has been in uses since the 1970s in SGML. The XML Schema was originally proposed by Microsoft, and

became an official W3C recommendation in May 2001. It is a standard that is still under development by the W3C.

In the long term the use of XML Schemas will eventually replace the DTD, in the short term DTDs still have many advantages. All SGML tools and many XML tools and browsers can process DTDs for XML documents. A large number of XML document types are already defined using the DTD model. A large community of SGML and XML programmers understands DTD language.

BASIC XML SYNTAX

XML provides a formal syntax that defines elements, attributes and text. XML data is contained in entities, which consist of multiple (or singular) elements and their attributes. All XML documents must contain a primary or “Root” Element that all of the data used in the document is nested in.

XML elements can have either text or data content. Elements may contain attributes, be pre-defined entities themselves or contain additional elements (called child elements). Data contained in elements maybe any of the various allowed data types (i.e. Parsed, Unparsed, string, integer, etc). Elements may also be designated as “empty”. Empty elements can be used as place markers in an XML document to reserve space where an event may occur.

XML is a self-describing domain-specific markup language. As self-describing, names (as well as order, data type and attributes) for data elements are defined within the XML document itself or in a DTD or Schema. Wherever the unique data structure for an XML document is defined the specifics for that data structure must exist.

Tags and Elements

Elements are uniquely named. The self-defined name for an element is denoted in angle brackets or “<>”. This bracket set in an XML document is referred to as a “Tag”. All element tags used in XML must use nested data and have both a begin tag and an end tag. End tag sets are designated with a “/” in front of the element name.

Element Tag Set With Nested Data Example:

<Element_1>this is the data content</Element_1>

An element tag set in XML with no data association can be properly expressed in a single tag set terminated with “/” or as an empty tag pair.

Element No Data Examples:

No Data Tag Set-

```
<Element_1></Element_1>
```

No Data Single Tag-

```
<Element_1/ >
```

Tags in XML are used to designate a hierarchical set of entities. An entity is one or more elements. Elements maybe of various data types, consist of multiple elements and can contain one or more attributes. The elements that are contained within elements are referred to as “Child” Elements. Child elements must be nested in their parent element's tag set. All XML documents must contain a primary or “Root” Element that all the data elements used in the XML document are nested in. Elements that contain only other elements are said to have element content.

Root and Child Element Example:

```
<Root Element>
<Element_1>
    <Child_1/>
    <Child_2/>
    <Child_3/>
</Element_1>
</Root Element>
```

The general rules of XML syntax are more rigorous than those of HTML or SGML. In XML character case and white space is significant in all uses of text.

Invalid XML Examples:

Case not consistent-

```
<Element_1></element_1>
```

White space used-

```
<Element_1 ></Element 1 >
```

In an XML document additional markup language appears inside of angle brackets. A few of the basic data declaration common in XML markup are as follows:

Comments are contained within a single tag set and not nested between a pair tags. All comments begin with “<!--” and end with “-->”. Comments can contain any data (except the literal string “--”). Comments can be used anywhere in an XML

document and serve as non-processed instructions to others reading the XML raw data file.

Comment Example:

```
<!--This is a comment that contains information on the XML document-->
```

Processing Instructions (PIs) provide information to independent applications. Like comments PIs are not part of the XML document. PIs are contained within a single tag set and not nested between a pair tags. All PIs must begin with “<?” and end with “?>”.

The text inside the PI tag set is the PI target and identifies the PI to the application. Applications should process only the targets they recognize and ignore all other PIs. The XML processor is required to pass PIs to applications.

All XML documents begin with a processing instruction <?xml...?>. This is the XML declaration. While not required it serves to explicitly identify the document as an XML document, indicates the version of XML to which it was authored under, the text encoding format and if it is a “standalone” document. A standalone XML document is one that contains the DTD information internally (i.e. there is not an external DTD or Schema for the file).

PI Document Declaration example:

```
<?xml version='1.0' encoding="UTF-8"
standalone="no"?>
```

PI Stylesheet Association example:

```
<?xml:stylesheet type="text/xsl"
href="ftp://techweb.rfa.org/pub/SBE-EDF/DTD
Resource/efd_1.xsl"?>
```

Document Type Declarations define the document type internal to the XML file or as a call to an external DTD or Schema file. This declaration defines the rules imposed on the XML data structure. The purpose of a Document Type Definition is to define the legal building blocks of any SGML-based document. It defines the document structure with a list of legal elements. This is a required declaration in any file that must conform and validate to a external DTD or XML Schema file. Like a comment and PI the Document Type Declaration is contained within a tag set and not nested between a pair of tags. The

Document Type Declaration begins with “<!” and ends with a simple closed “>”.

Document Type Declaration example:

```
<!DOCTYPE vendor SYSTEM
"ftp://techweb.rfa.org/pub/SBE-EDF/DTD
Resource/efd_1.dtd">
```

Namespace Declarations- An XML namespace is a self-defined name association that links data within an XML document with a URI reference. A URI is a Uniform Resource Identifier, which is a string of characters that identifies an Internet Resource. A common URI used when declaring namespaces is a URL or Uniform Resource Locator. URL's identify an Internet domain address.

Namespaces are declared within a tag set using the XML declaration “xmlns” followed by a colon. The text that follows the colon becomes the reference name in the XML document for the namespace. The referenced namespace name is followed by “=” after which the text that follows contains the associated URI in quotation marks.

Namespace Declaration Example:

```
<document xmlns:refName="http://anyuri">
```

To create an element association that is defined by a declared namespace, the reference name for the namespace is used as a prefix for the element tag followed by a colon.

Namespace Use Example:

```
<refName:Element_1>Element Data</Element_1>
```

Namespaces

Namespaces became a W3C recommendation in January 1999. An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. In practice, namespaces let you match a tag you are using with a particular set of tags. XML namespaces differ from the "namespaces" conventionally used in other computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set.

An XML namespace is an identification by URI, which is used in XML documents to be associated with the processing of element types and attribute names. The URI reference that identifies namespaces are considered to be identical if they are exactly the same character-for-character.

A single XML document may contain markup vocabulary that is defined for and used by multiple software modules. This modularity allows the re-use of markup language rather than be required to re-invent it for each XML application. XML documents that contain multiple markup vocabularies pose a problem for applications with recognition and the possibility of collision. The use of namespaces in XML give these software modules the ability to recognize by URI the tags and attributes which they are designed to process.

The combination of a universally managed URI namespace and the document's own namespace produces identifiers that are universally unique. An attribute-based syntax is used to declare the association of the namespace prefix with a URI reference. Names for XML namespaces appear as qualified names in the XML document. Namespace names contain a single colon, separating the name into a namespace prefix. The prefix is used throughout the XML document and provides mapping to specific URI references.

Attributes

Elements in an XML document are permitted to have “Attributes”. An attribute is meta-data for elements. Meta-data is data about data. Attributes are used in XML to modify information contained in elements. The difference between elements and attributes is subtle, elements should be considered nouns while attributes should be considered adjectives. Elements are "kind of" and Attributes are "type of". Attributes are name-value pairs that occur inside element start-tags after the element name.

There are three XML attribute data types allowed, string, tokenized and enumerated. A string type attribute may be any literal string. Tokenized attribute types have varying lexical and semantic constraints. Enumerated attribute types are an allowed value list of attributes for an element.

If more than one value is provided for the same attribute of an element the first declaration is binding and later declarations are ignored. All element attribute values must be quoted in the XML

document. Elements are not allowed to have two attributes that share the same name.

Attributes and their allowable values must be defined in the DTD or Schema files. Attributes with multiple values are specified in attribute list. Attribute values when used in XML documents must always be contained inside quotations.

Attribute Example:

```
<Element_1 size="big">This element has the size attribute of big</Element_1>
```

Entities

Elements can also be defined with allowed data values or "Entities". Entities are defined in the DTD or Schema and are assigned an entity name. Entities are used to make entering and managing of recurring or restricted data simple. An entity in XML is a short cut to a predefined set of information.

Entity values can also be stored in a separate file (designated with an ".ent" file extension). Entities that associate a name in an XML document with the content of another file are called external entities. Internal entities associate a name with a string of literal text within the XML file or the conforming file. Every entity must have a unique name.

When passed through an XML Processor the entity in an XML document is "expanded" to its full value. Entities are referenced in an XML document nested in the element tag set as regular text data is. The entity call begins with an ampersand, followed by the entity name and ending with a semicolon (&entity-name;).

Entities are also used in XML to represent special XML characters (&, ", <, etc) in parsed (PCDATA) data. XML specifies a set of general entities (amp, lt, gt, apos, quot) to represent these characters.

XSL Stylesheets

XSL is a language that follows the same rules as XML. An XSL stylesheet capable application accepts an XML document and an XSL stylesheet and transforms the output to a usable predefined presentation. The presentation appears on the platform as intended by the designer of the XSL stylesheet. The presentation may vary depending on the desired platform the data is to be presented (i.e.

browser, WAP device, print format, etc.). The use of XSL Stylesheets highlights the advantage XML gains by separating content from design. The same XML data content can be transformed into multiple output formats by linking to different Stylesheets.

Including formatting semantics in the result tree enables XSL formatting. XSL formatting semantics are expressed in terms of a catalog of classes of formatting objects. The XSL classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth. Finer control over the presentation of typographic abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and hyphenation control. In XSL, the classes of formatting objects and formatting properties provide the vocabulary for expressing presentation intent.

In XSL there is no underlying semantic to augment for XML, XSL must specify how each element should be presented and what the element is. XSL defines not only a language for expressing Stylesheets, but also a vocabulary for "formatting objects" that have the necessary base semantics.

XSL style sheets are linked to XML documents by a PI (Processing Instructions) in the XML file.

XSL Link Example:

```
<?xml:stylesheet type="text/xsl" href="ftp://any.org/Stylesheet.xsl"?>
```

The XSL language is a transformation language. The use of XSL in Stylesheets is only a part of XSL (also known as XSLT). XSL also describes a language for formatting objects (also known as XSL FO). For example, XSL is also capable of transforming XML documents into more complicated formats such as PDF and PostScript files.

EFD FUNCTIONAL OVERVIEW

The Society of Broadcast Engineers EFD XML specification is divided into two XML files, EFD_1.XML and EFD_2.XML. Templates of these XML files and validating DTD files can be found at the Society of Broadcast Engineers EFD Web Site located at <http://www.sbe-efd.org>.

Specific vendor contact information is stored in the EFD_1.XML file. "Vendor" is defined in the EFD as the organization, individual or company using the

EFD as a technical documentation standard. Vendor contact information includes street address, phone numbers, e-mail address and Web URL for the main office as well as for multiple sales and technical support offices.

The “EFD_1.DTD” file validates the vendor contact informational EFD_1.XML file. The EFD_1.DTD file specifies the allowed data elements for vendor contact information that may be included in an EFD_1.XML file. Most data elements in EFD_1 DTD are optional. Vendors are allowed the choice of providing the minimum required entries to complete a valid EFD XML file. Each EFD_1.XML file has a unique file name assigned by the vendor. To be EFD compliant vendors must provide at the minimum one EFD_1 XML informational file. This file must validate to EFD_1.DTD.

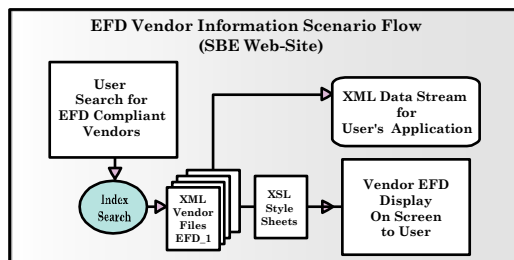


Figure #1

The vendor informational EFD_1 XML files can be rendered into a browser viewable format by the EDF_1 XSL stylesheet. The EFD XSL Stylesheet files are also available on the Society of Broadcast Engineers EFD Web Site (<http://www.sbe-efd.org>).

Compliant vendors that supply EFD_1 contact information XML files to the Society of Broadcast Engineers can be publicly accessed by indexed search on the EFD web site. The EFD Web Site is currently developing native XML database tools and other transformation applications under the Open Source software license. It is hoped that these tools will help facilitate broader industry adoption of the EFD. XML tools will be available on the EFD Web Site as they become available.

Indexing of vendors is accomplished in the EFD_1 XML file by the inclusion of mandatory classifications. The Vendor Classification element consists of three data child elements. These child elements are “Vendor Function” or primary business (i.e. Manufacturer, Representative, etc), “Vendor Discipline” or primary business sector served (i.e. Radio, Television, etc) and “Vendor Product” or primary product offered (i.e. Equipment, Software,

Consulting, etc). The Vendor classification entities are restricted data elements in the EFD_1.XML file. The allowed data entries are described in the EFD_1.DTD file.

Information on specific products and/or services offered by the vendor are stored in the EFD_2.XML file. One EFD_2.XML file must be completed for each product or service the vendor offers. Each product EFD_2.XML file also should have a unique file name assigned by the vendor. The EFD_2 product information file contains specific data on products inclusive of model names, inventory identifiers, pricing, revision histories, technical specifications and physical specifications.

The “EFD_2.DTD” file validates the product informational EFD_2.XML file. The EFD_2.DTD file specifies the allowed data fields that the vendors can include in the product informational XML file. Most data elements of the EFD_2.XML file are optional and vendors are allowed the choice to fill in the minimum required entries to validate to the EFD standard.

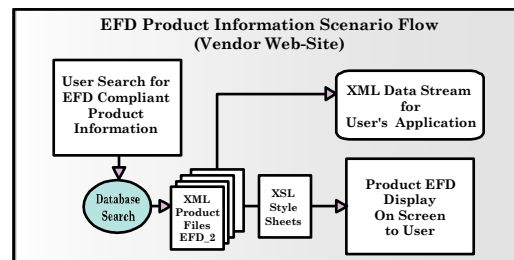


Figure #2

Product informational EFD_2 XML files can also be transformed into a browser viewable format by the EFD_2.XSL stylesheet file located on <http://www.sbe-efd.org>. EFD_2 product informational files are envisioned as residing on vendor maintained web sites. The Society of Broadcast Engineers is in the process of developing XML database tools to allow vendors to post EFD_2 files on the EFD Web-site if they wish to.

The use of different XML files to separate Vendor contact information from product and service information allows for minimum effort on the part of an EFD compliant vendor to update technical documentation. Used in tandem an entire catalog or Web Site can be rendered using a single EFD_1 vendor informational file and multiple EFD_2 product informational files, in conjunction with the proper XSL stylesheet. If a change is required in a specific product XML file or if the

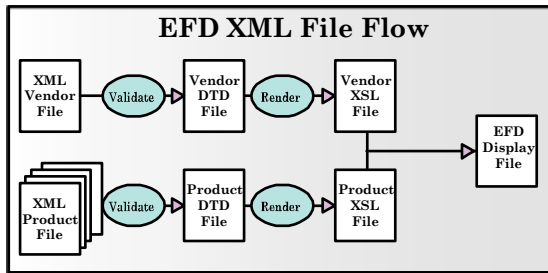


Figure #3

vendor relocates and requires an address update, data would only need to be updated in one EFD XML file. This would eliminate the need to make multiple editorial updates across multiple documents in multiple formats.

EFD IMPLEMENTATION

The Society of Broadcast Engineers maintains a Web Site dedicated to the EFD Project at <http://www.sbe-efd.org>. SBE-EFD.Org is the authoritative Web Site for the EFD standard, all current information and files related to the EFD can be found on this site. The site also hosts an EFD discussion mailing list. The URL to subscribe to the EFD mailing list is: <http://techweb.rfa.org/mailman/listinfo/sbe-efd>

The EFD Web Site also serves as a public indexed listing of all vendors offering EFD compliant technical documentation. Vendor informational EFD_1.XML files maintained on the EFD Web Site are linked to the vendor maintained EFD compliant web sites. The links to vendor maintained EFD compliant product informational web sites exist internal in each vendor informational EFD_1.XML File.

Vendors interested in becoming compliant with the EFD are invited to complete an EFD_1.XML file online via a web-based interactive form (at www.sbe.efd.org) or by ftp to the EFD site. All files received by the EFD site are validated by the Society of Broadcast Engineers to the EFD-1.DTD before posting.

All EFD XML files submitted to the Society of Broadcast Engineer EFD Web Site will remain the copyright protected property of the vendor of origin.

It is encouraged that search engines for users utilizing vendor EFD compliant web sites be developed and maintained by the vendor. While The Society of Broadcast Engineers plans to offer XML tools, search

engines and programming advice no warranty for their use will be available.

EFD INTERGRATION SENERIOS

End users of EFD conduct an initial “compliant vendor” search at the Society of Broadcast Engineers EFD Web-site. Users search for vendors by classification based on their informational needs. The results of vendor searches at Society of Broadcast Engineers EFD Web-site will be rendered on a HTML page with links to the vendor’s site.

Once linked to the selected vendor’s compliant EFD site XML product information can be searched on the database engine the vendor is utilizing. All product matches found by the vendor’s database can be returned to the user rendered as an HTML page

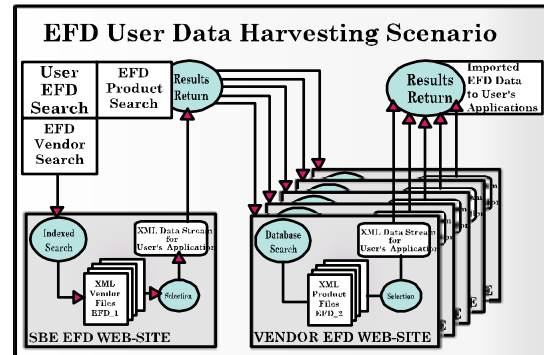


Figure #4

or, as a complete product brochure PDF file or, as an XML data stream. The actual transformations of EFD XML files are dependent on the available XSL and XSLT files on the vendor’s site.

It is hoped that third party developers will offer search tools that are capable of multiple site searches and product comparisons. Potentially EFD XML files can be “XML data streaming” download directly from the vendor’s web site to the users CAD or project management program.

VENDOR INFORMATIONAL EFD_1 ELEMENTS AND IDENTIFIER DESCRIPTIONS

Vendor (must be entered once)

Main XML Element

Consist of elements “v_class”, “v_header”, “v_banner”, “v_copy”, “address”, “contact” and “efdlink”.

v_class (must be entered once)
Classifies what business the vendor is in.
Contains elements "v_function", "v_discipline" and "v_product".

v_function (must be entered once)
Describes primary business function.
Internal entity allowed choices only, choices are:
"Manufacturer"
"Representative"
"Value Added Reseller"
"Distributor"
"Retailer"
"Organization"
"Systems Integrator"
"Contractor Engineer"
"Consultant"

v_discipline (must be entered once)
Describes the vendor's primary market target segment.
Internal entity allowed choices only, choices are:
"Broadcast"
"Radio"
"Television"
"Production"
"Electronics"
"Education / Museums"
"Security"
"Corporate Systems"

v_product (must be entered once)
Describes vendor's primary product.
Internal entity allowed choices only, choices are:
"Equipment"
"Software"
"Services"

v_header (must be entered once)
The vendor's name and logo.
Contains elements "v_name" and "v_logo".

v_name (must be entered once)
Vendor's used company name.

v_logo (optional may be used once or none)
Vendor corporate logo.

v_banner (optional may be used once or none)
Company motto.

v_copy (optional may be used once or none)
Text copy that describes the company.

address (must be entered once)
The primary business location address and contact information.
Contains elements "add_name", "street", "city", "region", "country", "zip", "phone1", "phone2", "phone3", "email" and "web".

street (optional may be used once or none)
Primary location street number and name and/or PO box.

city (optional may be used once or none)
Primary location, name of the city.

region (optional may be used once or none)
Primary location, name of the state or region.

country (optional may be used once or none)
Primary location, name of the country.

zip (optional may be used once or none)
Primary location, zip or postal code.

phone1 (optional may be used once or none)
Primary location main phone number.
Required Attribute = type

Allowed phone types "voice", "fax" or "toll"
phone2 (optional may be used once or none)
Primary location, second phone number.

Required Attribute = type
Allowed phone types "voice", "fax" or "toll"
phone3 (optional may be used once or none)
Primary location, third phone number.

Required Attribute = type
Allowed phone types "voice", "fax" or "toll"

email (optional may be used once or none)
Primary location, main e-mail contact address.

web (optional may be used once or none)
Primary location, main web page URL.

Contact (optional may be used once or none)
Secondary business addresses for technical and sales support.

Contains elements "sales" and "tech_support".

sales (can be entered 0 or many times)
A secondary business location or sales office address and contact information.

Contains elements "s_name" and "s_address"

s_name (must be entered once)
The name of secondary or sales office.

s_address (must be entered once)
The secondary or sales office location address and contact information

Contains elements "s_street", "s_city", "s_region", "s_country", "s_zip", "s_phone1", "s_phone2", "s_phone3", "s_email" and "s_web".

s_street (optional may be used once or none)
Sales office location street number and name and/or PO box.

s_city (optional may be used once or none)
Secondary or sales office location name of the city.

s_region (optional may be used once or none)
Secondary or sales office location name of the state or region.

s_country (optional may be used once or none)
Secondary or sales office location name of the country.

s_zip (optional may be used once or none)
Secondary or sales office location zip or postal code.

s_phone1 (optional may be used once or none)
Secondary or sales office location main phone number.

Required Attribute = type
Allowed phone types "voice", "fax" or "toll"

s_phone2 (optional may be used once or none)
Secondary or sales office location second phone number.

Required Attribute = type

Allowed phone types "voice", "fax" or "toll"

s_phone3 (optional may be used once or none)
Secondary or sales office location third phone number.

Required Attribute = type

Allowed phone types "voice", "fax" or "toll"

s_email (optional may be used once or none)
Secondary or sales office location main e-mail contact address.

s_web (optional may be used once or none)
Secondary or sales office location main web page URL.

tech_support (can be entered 0 or many times)
Technical support address and contact information.

Contains elements "ts_name" and "ts_address"

ts_name (must be entered once)

The name of technical support location.

ts_address (must be entered once)

The technical support location address and contact information

Contains elements "ts_street", "ts_city", "ts_region", "ts_country", "ts_zip", "ts_phone1", "ts_phone2", "ts_phone3", "ts_email" and "ts_web".

ts_street (optional may be used once or none)
Technical support location street number and name and/or PO box.

ts_city (optional may be used once or none)
Technical support location name of the city.

ts_region (optional may be used once or none)
Technical support location name of the state or region.

ts_country (optional may be used once or none)
Technical support location name of the country.

ts_zip (optional may be used once or none)
Technical support zip or postal code.

ts_phone1 (optional may be used once or none)
Technical support main phone number.

Required Attribute = type

Allowed phone types "voice", "fax" or "toll"

ts_phone2 (optional may be used once or none)
Technical support second phone number.

Required Attribute = type

Allowed phone types "voice", "fax" or "toll"

ts_phone3 (optional may be used once or none)
Technical support third phone number.

Required Attribute = type

Allowed phone types "voice", "fax" or "toll"

ts_email (optional may be used once or none)
Technical support main e-mail contact address.

ts_web (optional may be used once or none)
Technical support main web page URL.

efdlink (must be entered once)

URL Link to EFD compliant product documentation.

EFD_1.DTD

```
<!--Society of Broadcast Engineers EFD Project \
V1.20 \ EFD Committee (dmb) March 2002-->
<!--EFD_1 Vendor Informational-->
<!ELEMENT vendor
(v_class,v_header,v_banner?,v_copy?,address,contact?,efdlink)>
<!ELEMENT v_class
(v_function,v_discipline,v_product)>
<!ELEMENT v_function (#PCDATA)>
<!ENTITY vf1 "Manufacturer">
<!ENTITY vf2 "Representative">
<!ENTITY vf3 "Value Added Reseller">
<!ENTITY vf4 "Distributor">
<!ENTITY vf5 "Retailer">
<!ENTITY vf6 "Organization">
<!ENTITY vf7 "Systems Integrator">
<!ENTITY vf8 "Contractor Engineer">
<!ENTITY vf9 "Consultant">
<!ELEMENT v_discipline (#PCDATA)>
<!ENTITY vd1 "Broadcast">
<!ENTITY vd2 "Radio">
<!ENTITY vd3 "Television">
<!ENTITY vd4 "Production">
<!ENTITY vd5 "Electronics">
<!ENTITY vd6 "Education / Museums">
<!ENTITY vd7 "Security">
<!ENTITY vd8 "Corporate Systems">
<!ELEMENT v_product (#PCDATA)>
<!ENTITY vp1 "Equipment">
<!ENTITY vp2 "Software">
<!ENTITY vp3 "Services">
<!ELEMENT v_header (v_name,v_logo)>
<!ELEMENT v_name (#PCDATA)>
<!ELEMENT v_logo (#PCDATA)>
<!ELEMENT v_banner (#PCDATA)>
<!ELEMENT v_copy (#PCDATA)>
<!ELEMENT address
(add_name,street?,city?,region?,country?,zip?,phone1?,phone2?,phone3?,email?,web)>
<!ELEMENT add_name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT region (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone1 (#PCDATA)>
<!ATTLIST phone1 type (voice|fax|toll) "voice">
<!ELEMENT phone2 (#PCDATA)>
<!ATTLIST phone2 type (voice|fax|toll) "voice">
<!ELEMENT phone3 (#PCDATA)>
```

```

<!ATTLIST phone3 type (voice|fax|toll) "voice">
<!ELEMENT email (#PCDATA)>
<!ELEMENT web (#PCDATA)>
<!ELEMENT contact (sales*,tech_support*)>
<!ELEMENT sales (s_name,s_address)>
<!ELEMENT s_name (#PCDATA)>
<!ELEMENT s_address
(s_street?,s_city?,s_region?,s_country?,s_zip?,s_ph
one1?,s_phone2?,s_phone3?,s_email?,s_web?)>
<!ELEMENT s_street (#PCDATA)>
<!ELEMENT s_city (#PCDATA)>
<!ELEMENT s_region (#PCDATA)>
<!ELEMENT s_country (#PCDATA)>
<!ELEMENT s_zip (#PCDATA)>
<!ELEMENT s_phone1 (#PCDATA)>
<!ATTLIST s_phone1 type (voice|fax|toll) "voice">
<!ELEMENT s_phone2 (#PCDATA)>
<!ATTLIST s_phone2 type (voice|fax|toll) "voice">
<!ELEMENT s_phone3 (#PCDATA)>
<!ATTLIST s_phone3 type (voice|fax|toll) "voice">
<!ELEMENT s_email (#PCDATA)>
<!ELEMENT s_web (#PCDATA)>
<!ELEMENT tech_support (ts_name,ts_address)>
<!ELEMENT ts_name (#PCDATA)>
<!ELEMENT ts_address
(ts_street?,ts_city?,ts_region?,ts_country?,ts_zip?,ts
phone1?,ts_phone2?,ts_phone3?,ts_email?,ts_web?)>
<!ELEMENT ts_street (#PCDATA)>
<!ELEMENT ts_city (#PCDATA)>
<!ELEMENT ts_region (#PCDATA)>
<!ELEMENT ts_country (#PCDATA)>
<!ELEMENT ts_zip (#PCDATA)>
<!ELEMENT ts_phone1 (#PCDATA)>
<!ATTLIST ts_phone1 type (voice|fax|toll) "voice">
<!ELEMENT ts_phone2 (#PCDATA)>
<!ATTLIST ts_phone2 type (voice|fax|toll) "voice">
<!ELEMENT ts_phone3 (#PCDATA)>
<!ATTLIST ts_phone3 type (voice|fax|toll) "voice">
<!ELEMENT ts_email (#PCDATA)>
<!ELEMENT ts_web (#PCDATA)>
<!ELEMENT efdlink (#PCDATA)>

```

PRODUCT INFORMATIONAL EFD_2.DTD AND IDENTIFIER DESCRIPTIONS

efd (must Element be entered once)

Main XML Element.

Consist of elements "vendor_call", "product",
"model", "vendid", "price", "tech_spec",
"phy_spec", "io_spec", "rev_info", "copy",
"copyright" and "xinfo".

vendor_call (must be entered once)

File name of corresponding EFD_1.XML file. Links
product to vendor information.

product (must be entered once)

Name of the product, used to describe product type.

model (must be entered once)

Product model name as described by the vendor.

vendid (must be entered once)

Vendor product identifying, item, part, order number
or UPC.

price (optional may be used once or none)

Price schedule for the product.

*Contains the elements "price_schd" and
"price_value".*

price_schd (can be entered 0 or many times)

Describes price structure used (i.e. retail, distributor,
etc).

price_value (can be entered 0 or many times)

Actual cost for the product matching to "price_schd".

tech_spec (optional may be used once or none)

Product overall technical specifications.

Contains elements "tspec_name" and "tspec_value"

tspec_name (can be entered 0 or many times)

Vendor described technical specification name.

tspec_value (can be entered 0 or many times)

Value of specification, matches to "tspec_name".

phy_spec (optional may be used once or none)

Product overall physical specification.

*Contains elements "pspec_name" and
"pspec_value"*

pspec_name (can be entered 0 or many times)

Vendor described physical specification name.

pspec_value (can be entered 0 or many times)

Value of specification, matches to "pspec_name".

io_info (optional may be used once or none)

Product Input / Output (I/O) information.

*Contains elements "io_direction", "io_name",
"io_abbv", "io_type", "io_signal", "io_connector"
and "io_xref".*

io_direction (can be entered 0 or many times)

Product I/O Information

*Internal DTD entity allowed data input only, choices
are:*

"Input"

"Output"

"Bi-Directional"

"Loop"

io_name (can be entered 0 or many times)

I/O name as labeled on the product by the vendor.

io_abbv (can be entered 0 or many times)

I/O name abbreviation for external export (i.e. CAD
program) limited to eight characters.

io_type (can be entered 0 or many times)

I/O Signal type.

Internal entity allowed choices only:

"Analog Audio"

"Analog Video"

"Digital Audio"

"Digital Video"

"Data"

io_signal (can be entered 0 or many times)
Signal type by standard (i.e. AES/EBU audio, balanced line level etc).

io_connector (can be entered 0 or many times)
I/O actual connector used as recommended by the vendor.

io_xref (can be entered 0 or many times)
URL to additional vendor supplied I/O information (i.e. Pin out sheets).

rev_info (optional may be used once or none)
Product revision information.
Contains elements "market_date", "hrev_ver", "hrev_date", "srev_ver", "srev_date", "frev_ver", "frev_date", "pkill_date".

market_date (optional may be used once or none)
Date product was available on the market.

hrev_ver (optional may be used once or none)
Current hardware version number.

hrev_date (optional may be used once or none)
Current hardware version release date.

srev_ver (optional may be used once or none)
Current software version number.

srev_date (optional may be used once or none)
Current software version release date.

frev_ver (optional may be used once or none)
Current firmware version number.

frev_date (optional may be used once or none)
Current firmware version release date.

pkill_date (optional may be used once or none)
Product kill date, date product was taken off the market.

copy (must be entered once)
Product text descriptive information.
Contains element "text".

text (must be entered once)
Product text descriptive information subsections.
Contains elements "sec_title", "section", "sec2_title", "section2" and "graphics".

sec_title (can be entered 0 or many times)
Product descriptive information text title style 1.

section (can be entered 0 or many times)
Product descriptive information text body style 1.

sec2_title (can be entered 0 or many times)
Product descriptive information text title style 2.

section2 (can be entered 0 or many times)
Product descriptive information text body style 2.

graphics (can be entered 0 or many times)
Pictures and graphics used in product descriptive.

copyright (must be entered once)
Vendor copyright information.

xinfo (must be entered once)
URL to additional vendor supplied product information
Contains elements "x_cad", "x_doc" and "x_efd".

x_cad (optional may be used once or none)
Vendor product Cad file URL.

x_doc (optional may be used once or none)
Other product document file URL (pdf, etc).

x_efd (optional may be used once or none)
Other product EFD document URL.

EFD_2.DTD

```

<!--Society of Broadcast Engineers EFD Project \
V1.00 Draft \ EFD Committee (dmb) March 2001-->
<!--EFD_2 Product Informational-->
<!ELEMENT efd
 (vendor_call,product,model,vendid,price,tech_spec?,
 phy_spec?,io_spec?,rev_info?,copy,copyright,xinfo)>
<!ELEMENT vendor_call (#PCDATA)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT vendid (#PCDATA)>
<!ELEMENT price (price_schd,price_value )>
<!ELEMENT price_schd (#PCDATA)>
<!ELEMENT price_value (#PCDATA)>
<!ATTLIST price_value cur (USD | PCDATA)
 "USD">
<!ELEMENT tech_spec (tspec_name, tspec_value)>
<!ELEMENT tspec_name (#PCDATA)>
<!ELEMENT tspec_value (#PCDATA)>
<!ELEMENT phy_spec (pspec_name, pspec_value)>
<!ELEMENT pspec_name (#PCDATA)>
<!ELEMENT pspec_value (#PCDATA)>
<!ELEMENT io_spec
 (io_direction,io_name,io_abbrev,io_type,io_signal,io_c
 onnector,io_xref)>
<!ELEMENT io_direction (#PCDATA)>
<!ENTITY in "Input">
<!ENTITY out "Output">
<!ENTITY bi "Bi-Directional">
<!ENTITY lp "Loop">
<!ELEMENT io_name (#PCDATA)>
<!ELEMENT io_abbrev (#PCDATA)>
<!ELEMENT io_type (#PCDATA)>
<!ENTITY aa "Analog Audio">
<!ENTITY av "Analog Video">
<!ENTITY da "Digital Audio">
<!ENTITY dv "Digital Video">
<!ENTITY data "Data">
<!ELEMENT io_signal (#PCDATA)>
<!ELEMENT io_connector (#PCDATA)>
<!ELEMENT io_xref (#PCDATA)>
<!ELEMENT rev_info
 (market_date,hrev_ver,hrev_date,srev_ver,srev_date,
 frev_ver,frev_date,pkill_date)>
<!ELEMENT market_date (#PCDATA)>
<!ELEMENT hrev_ver (#PCDATA)>
<!ELEMENT hrev_date (#PCDATA)>
<!ELEMENT srev_ver (#PCDATA)>

```

```
<!ELEMENT srev_date (#PCDATA)>
<!ELEMENT frev_ver (#PCDATA)>
<!ELEMENT frev_date (#PCDATA)>
<!ELEMENT pkill_date (#PCDATA)>
<!ELEMENT copy (text*)>
<!ELEMENT text
(sec_title*|section*|sec2_title*|section2*|graphics*)*
>
<!ELEMENT sec_title (#PCDATA)>
<!ELEMENT section (#PCDATA)>
<!ELEMENT sec2_title (#PCDATA)>
<!ELEMENT section2 (#PCDATA)>
<!ELEMENT graphics (#PCDATA)>
<!ELEMENT copyright (#PCDATA)>
<!ELEMENT xinfo (x_cad,x_doc,x_efd)>
<!ELEMENT x_cad (#PCDATA)>
<!ELEMENT x_doc (#PCDATA)>
<!ELEMENT x_efd (#PCDATA)>
```