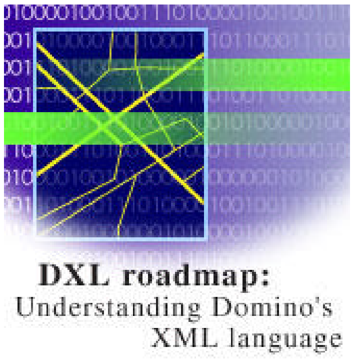
Notes.net

Iris Today



by
Russ
Lipton

Level: Intermediate
Works with: Domino 5.0
Updated: 10/01/2001

XML (eXtended Markup Language) has become an industry-standard meta-language for data modeling and presentation. DXL (Domino XML Language) is Domino data expressed as XML. With it, you can conveniently import, export, work with, and transform the data in your Notes databases. Today's Lotus XML Toolkit Release 1.0 and tomorrow's Domino Rnext capabilities solidify Domino's role as a strategic data integration point within your organization.

This article offers an overview of XML and DXL in the context of R5, the Lotus XML Toolkit, and Domino Rnext. While XML and DXL will be moving targets for the next few years, both are stable enough to support a wide range of prototyping as well as production application development.

This article assumes you are an experienced Lotus Notes/Domino developer with a basic working knowledge of XML. Proficiency in Java and C++ will be a definite help. Also, a several **online resources** are provided at the end of the article to help you ramp up your XML skills so that you can take advantage of DXL.

Why XML and DXL?

The Web—and especially HTML—has illustrated the amazing benefits of using a consistent data model. Do you want to jump from Notes.net to Google.com, to Lotus.com, or to another site to solve a problem? Click. Click. Click. Click.

Consider what it would have taken to exchange and display the same data with older protocols in those far-off days of, say, 1991. It might not have been possible at all. Convenient collection and efficient distribution of data to a billion users around the globe in near real-time would definitely have been impossible.

The Web has demonstrated that an always-almost-broken network dependent on a limited, semi-standard presentation language is just the ticket for reasonably static documents. However, this won't cut it for mission-critical applications. While Java, Javascript, and a host of proprietary sub-languages fill vital niches, they don't address the Internet's two most critical requirements:

- A simple, readable language to define and validate well-structured data through universally accessible data exchange protocols
- An equally consistent mechanism to transform and present that data

XML and its sibling protocols promise to meet these requirements for the industry as a whole. DXL, XML for Domino, fulfills these requirements for Notes users. Because DXL is XML, it exposes the Domino architecture and data stores for data import, export, and transformation to anyone or any tool that can process XML.

Domino is already a superb environment for data integration. Using DXL, you can rely on Domino to manage data that has been too difficult or costly to integrate programmatically in the past. You can also move your Domino data outside Notes if you prefer to use tools other than Designer for crafting your applications. So, most likely you'll use DXL in four ways:

- To import XML data from external databases or applications into Domino databases.
- To export XML data from Domino databases into other applications or databases.
- To modify data in a Domino database by exporting DXL, making changes, and then reimporting the data back to Domino.
- To use Domino data for processing in an external XML tool, either as an end-point or as an intermediate stage before reimporting the data back into Domino. Or, you can create DXL data directly in a third-party tool and import it into Domino.

Simply stated, any data in any environment anywhere in your corporation is now a candidate for management within Domino and Notes.

DXL is not stored in an NSF (using the on-disk storage format) nor can it be displayed directly in a browser—that is, retaining presentation semantics. (Limited display of raw XML data is feasible with some browsers.)

That said, beyond the uses described above, Domino is ideal for storing XML texts, including DTDs, DXL data, and XSL stylesheets. By storing these texts within a Domino database, you gain Domino's security, replication, and view capabilities for organizing your XML. You must weigh these benefits against a small performance penalty compared to retrieving XML directly from the file system.

The foundation: XML

SGML (Structured Generalized Markup Language) is a robust, mature, and nearly impenetrable standard for defining document structure in a presentation-independent form. While the government has often mandated its use, private industry has found the costs of widespread adoption prohibitive.

HTML is an example of a sub-language defined with a SGML-compliant DTD (data type definition). HTML's direct costs are minimal and its adoption has been universal. HTML was designed originally to support simple presentation of documents over the Internet. It has become overloaded with features that undermine its chief virtue (simplicity) while failing to provide enough power and flexibility for serious data manipulation.

XML has emerged as a reasonable compromise between the complexity of SGML and the ad hoc nature of HTML. By design, XML defines a surprisingly spare set of tags and rules. XML serves as the base for defining other meta-languages that adhere to its rules.

The classic examples are still relevant. Where you use <P> and only <P> to specify a paragraph break in HTML, you may now also define <PRODUCT> or <SOFTWARE> or <RNEXT> to structure your XML data so that it can be processed or transformed at a higher semantic level. XML separates data from presentation more rigorously than HTML does—and in a way that is consistent with the Notes architecture, which applies forms to display underlying data.

Here is a simple XML file:

```
<?xml version=”1.0”?>
<DOMINO_PRODUCTS>
    <OS type=”NT”>
        <VERSION>Server</VERSION>
        <RELEASE>5.0</RELEASE>
    </OS>
    <OS type=”NT”>
        <VERSION>Client</VERSION>
        <RELEASE>5.0</RELEASE>
    </OS>
    <OS type="Linux">
        <VERSION>Server</VERSION>
        <RELEASE>5.0</RELEASE>
    </OS>
</DOMINO_PRODUCTS>
```

This code defines two different types of operating system products, for NT and Linux, respectively. Each product type is assigned a version (for instance, server or client) and a release number.

The key point here is that while HTML tags have been defined by a handful of parties, XML tags can be defined by anyone. That is, while Microsoft's Internet Explorer browser only reads HTML tags conformant to its model, any standard XML processor should be able to read any XML standard tags—whether they were defined by Microsoft, Lotus, or you. And that means you have a great deal of flexibility. For example, you might choose this alternative approach to defining products:

```
<PRODUCT type = "Server">
    <OS>NT</OS>
    <RELEASE>5.0</RELEASE>
    <MEDIA>CD</MEDIA>
    <PRICE>995.00</PRICE>
</PRODUCT>
```

XML is verbose by design (although you may be as cryptic with your tags as you'd like). Because XML is text, it can be written or read in any text editor and is easily modifiable.

As XML matures, editing, validation, and schema design tools are making it more convenient to ensure that XML texts are well-formed (with matching tags) and valid (they meet their data type or schema definition, if one is provided). See the last section of this article, **Internet resources to support you DXL work**, for a list of XML resources and links.

The Domino DTD and DXL

XML documents can be created with or without document type definitions (DTD). Without a DTD, XML processors can still determine whether a document is well-formed syntactically. However, a DTD makes it possible to decide whether a document is also valid—that is, whether the tags in a document conform to a specification.

In fact, DTDs are not standard XML, a key reason why XML schemas are now vying for their role. However, DTDs are relatively straightforward. Basically, you specify the elements within your XML documents in a hierarchical structure that resolves at its root to parsed (PCDATA) or unparsed (CDATA) data.

The following simple example shows what a DTD might look like for the sample XML file in the previous section. (Note that it is purely coincidental that DOMINO_PRODUCTS is both the document type and an element within the document.)

```
<?xml version ="1.0" standalone="yes"?>
<!DOCTYPE DOMINO_PRODUCTS [
<!ELEMENT DOMINO_PRODUCTS (OS)*>
<!ELEMENT OS (VERSION, RELEASE)>
<!ELEMENT VERSION (#PCDATA) >
<!ELEMENT RELEASE (#PCDATA) >
<!ATTLIST OS type (Linux | NT | Macintosh)>
]>
```

What is crucial is that elements contain other elements in hierarchical relationship; in this example, a Domino product is defined by its operating system. which in turn, contains a version and release. Notice also that the OS element is associated with a list of attributes. This is why the actual document could specify OS type = "Linux".

You can add tags to a DTD at any time. Tags that are not used or recognized within your XML document are ignored. If you fail to describe tags properly within the DTD, most XML processors will complain appropriately.

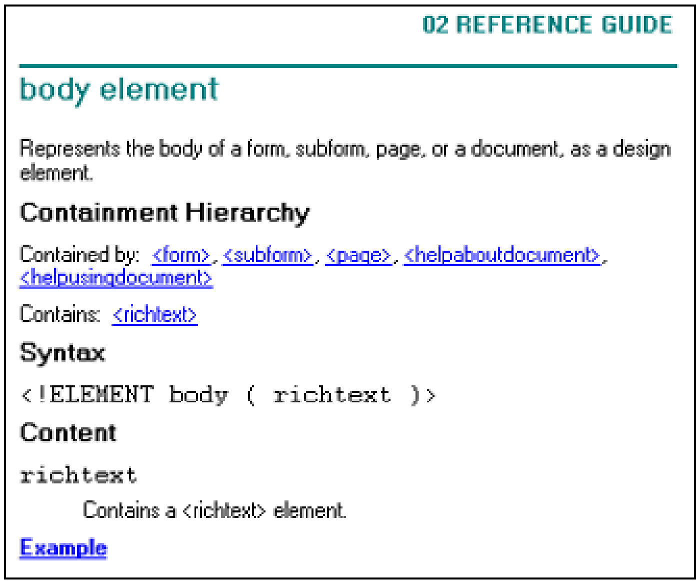
The Domino DTD, which is supplied with the **Lotus XML Toolkit**, describes the rules for creating valid XML files for Notes database elements. You will need to understand the DXL tags within the Domino DTD well enough so that you can choose them for your own documents.

Ultimately, DXL will represent every meaningful data element managed by Notes databases in XML—from rich text documents through forms and views to fields and design elements. Most elements are already represented. More are being added every month. Internal work has focused first on Notes data representation. It is moving outward by stages (and in parallel with data representation) to provide increasingly sophisticated support for parsing and transforming that data.

You will also be able to choose between using either the Domino DTD or a Domino XML Schema for your DXL work. You can download a **Lotus-developed schema** for evaluation.

If you already understand Notes databases and classes, you will find the DTD reasonably self-documenting. Copious comments within the DTD further explain the relationship of elements to one another for use as DXL.

The Domino DTD is understandably extensive. The documentation chunks it into usable pieces, complete with examples. This screen from the **Lotus XML Toolkit Guide** shows the body element help topic:


02 REFERENCE GUIDE

body element

Represents the body of a form, subform, page, or a document, as a design element.

Containment Hierarchy

Contained by: <form>, <subform>, <page>, <helpaboutdocument>, <helpusingdocument>

Contains: <richtext>

Syntax

```
<!ELEMENT body ( richtext )>
```

Content

`richtext`

Contains a <richtext> element.

Example

In the DTD documentation, you can follow links to drill for more detail. Not only will a knowledge of Notes databases make it easy to read the DTD, but the DTD is itself a tremendous training tool for understanding the structure of Notes databases.

Here is a fragment that partially demonstrates how rich text described in the Domino DTD can be specified in an actual DXL text:

```
<document form='Memo'> <noteinfo
unid='9C93469B4BFC2081852567AE00559882'>
            <created>

<datetime>19991205T091500,00-04</datetime>
            </created>
        </noteinfo>
        <item name='Subject'>
            <text>DXL Article</text>
        </item>
        <item name='Sent'>
            <datetime>19991205T091500,00-04</datetime>
        </item>
        <item name='From'>
            <text>Russ Lipton</text>
        </item>
        <item  name='Body'>
            <richtext>
                <par>
                    <run>
                        <font  style='italic'/>Please
                    </run>
                         use Domino DXL!
                </par>
            </richtext>
        </item>
</document>
```

As you can see, this is "just" standard XML using the specially defined Domino tags. Note these points:

- The document is represented as a tree composed of nodes and subnodes. I have added white space and indentation here to bring this point out. You may or may not want to do this yourself.
- Most (though not all) XML processors will ignore the white space within an XML text. They will retain the white space that is created within tags themselves (for instance, <text>Russ Lipton</text>). (White space is a specialized subject all to itself).
- "Item" is the primary element of this XML text.
- Attributes specialize the meaning of defined database components (in this case, the document's form attribute is "Memo"). Like all XML, each tag is enclosed (that is, <item>some text goes here</item>). The "name" attribute of the Item element includes the following possible values: "subject," "sent," "from," and "body".
- The "richtext" tag enables us to decompose a note into most of its detailed components. A strong goal for Rnext is complete representation of all rich text components.

This next fragment shows another key Domino data type (forms), illustrating that you can make use of design elements, formulas, buttons, and scripts within DXL just as you can with other database components:

```
<form name='Mini  Doc' default='true'  mailable='true'>
    <noteinfo>...
    </noteinfo>
    <code   event='windowtitle'>
        <formula>@If(@IsNewDoc;   "New   Memo";  Subject)
        </formula>
    </code>
```

```
        rest of XML text follows
</form>
```

Since new releases introduce new components, the Domino DTD will change periodically. However, nearly everything you work with in Domino is now (or will soon be) at your disposal within DXL. You will probably find that the supplied DXL tags best meet your needs, but you may certainly create your own DXL tags if you require a custom mapping of Domino data.

The Lotus XML Toolkit and Rnext

The Lotus XML Toolkit, originally released in the summer of 2000, for preview, works with Domino 5.0 and later releases and is readily available as a **free download**.

The Lotus XML Toolkit Release 1.0 offers a set of Java and C++ classes for DXL programming written over the Notes C API for all Win32 platforms. (New DXL features, including support for other platforms, will appear in Rnext.)

You can also make considerable use of the Lotus XML Toolkit at the command line. Here are two typical command lines for exporting and importing DXL (don't worry about the flags for now; this is just for illustration):

```
dxlexport db.nsf -o db.xml -a
```

```
dxlimport -i db.xml -d newdb.nsf -co
```

The toolkit offers a series of examples that you can use as a tutorial to learning DXL programming.

The Importer and Exporter classes in the Toolkit give you the core functionality you need to work with data. The DXLImporter class inputs DXL data into a Notes database. Import option properties are set before calling the methods. They include:

DXLIMPORTOPTION_CREATE (always create new data)
DXLIMPORTOPTION_IGNORE (ignore the DXL data)
DXLIMPORTOPTION_REPLACE (replace existing data with DXL data)
DXLIMPORTOPTION_UPDATE (update existing data with DXL data)

The DXLExporter class generates DXL data from a Domino database.

At the other end of the spectrum, with the new NotesNoteCollection class in Rnext, you can create a NotesNoteCollection object to specify a subset of notes for export. Based on the NSF search service in the C API, you can specify type, formula, and/or a "since" time to select the notes of interest.

So, for instance, you can select Notes documents, design elements, admin notes, or anything that has been represented in the Domino DXL DTD. To select all design elements:

```
Dim nc as NotesNoteCollection
Set nc = db.CreateNoteCollection(false)
Call nc.SelectAllDesignElements(true)
```

That is, you either retrieve the entire database (db.CreateNoteCollection(true)) or pick out the specific notes that you want to use.

And don't worry: all standard Notes/Domino security is available to ensure that only authorized persons are permitted to export or import XML data.

While I have emphasized data exchange in this article, the *Toolkit* samples also include retrieval of Domino design elements, creation of a Domino application without the use of Designer, and the use of DXL to summarize design synopsis information. *You can even use* DXL to build tools to complement Designer itself.

The Rnext roadmap

The Notes/Domino Rnext **beta version** is available for testing now on Notes.net.

Plans call for Rnext to maintain compatibility with the Lotus XML Toolkit. The two projects share much of the underlying code. Rnext will also extend Domino XML support in several important respects.

First, the Rnext development team has factored in important XML classes from the initial DXL implementation. The original R5 implementation coupled methods to existing Domino classes. As XML support increases, this would proliferate methods across all classes. The XML class design delivers an elegant simplification of the architecture that will decrease code maintenance and enhance performance.

Second, LotusScript support will be available.

Third, NotesStream and NotesNoteCollection classes will be added.These can pipe XML import and export or select granular elements of Notes databases for data exchange.

Piping means you can save data for interim transformation before further import-export or, for instance, specify that data should be streamed directly through a transformation to an export, discarding the interim results.

Rnext class support can be divided into three types:

- Standard XML Processors (XMLTransformer, DOMParser, SAXParser)
- DXL Specific Processors (DXLExporter, DXLImporter)
- Helper Classes (NotesStream, NotesNoteCollection)

Support is planned for both LotusScript as well as Java implementations.

XML processing relies on parsers that traverse the ordered nodes of XML texts and transform XML data for further use by the receiver of the data. Richtext items as well as objects created through NotesStream or NotesNoteCollection classes are all available to the XML processors.

XML processors rely on a Document Object Model (DOM) tree representation or an event-based model (SAX). The former is more memory-intensive since the entire document must be loaded; the latter requires more up-front programming but can query selected nodes more efficiently. Each is useful in different situations. A third model (XSLT) is gaining major traction (more on XSLT later in this article).

Rnext's implementation of the DOMParser will enable you to create a tree of node objects (elements, attributes, entities, and so on), query those nodes for the value of their properties (for instance, is the queried node null?), and specify methods against the nodes.

The process method parses the incoming data into a DOM tree, while the postParse event handler contains the processing logic you want to apply against the tree.

The SAXParser follows the event-driven model of the standard XML model. You can process the input XML as a series of events (StartDocument, StartElement, EndElement, and so on) and use the Output method to write strings to the Output object.

The XSLTransformer class applies an XSL stylesheet against the input to generate the desired output. This may become the preferred model for most XML processing as XSLT matures.

XML processors can get their input from, or direct their output to, a NotesStream or Richtext item, so you can retain intermediate results for use elsewhere. Alternatively, the output from one processor can be piped automatically as the input to another processor.

While the NotesStream class has been designed to support Notes XML applications, it is a general-purpose class suitable for use across Notes. Basically a FIFO (First In First Out) bucket, a NotesStream object can be "filled" with data from a back-end class (or several classes) and "emptied" of that same data for use by other classes.

NotesStream might, for instance, be filled with data exported from a Notes database using the DXLExporter class and then be emptied into an XSL Transformer class. Data can be streamed either to a memory buffer or a file.

Introducing XSLT

If you are already competent with DOM or SAX parsers, you will continue to use them to parse and transform DXL data. If, on the other hand, you are just now diving into XML, we recommend that you climb the steep but short XSL/XSLT learning curve. XSL stands for Extensible Stylesheet Language, and XSLT is the part of the XSL language for transforming documents. XSLT combines many of the functional characteristics of earlier parsers and has a longer life cycle ahead.

Ideally, we would like Web browsers to display XML directly. Internet Explorer 5.0 and higher provides limited support for this. However, direct browser support for XML lags.

XSL has been evolving for some time as a stylesheet language for transforming XML to HTML, since all browsers display HTML. Because XSL is "just" XML, it benefits from its own DTD, ensuring consistency. Individual stylesheets can be processed by all industry-standard XML tools.

The XSLT stylesheet shown below matches the entire document submitted to it as XML and counts the number of fields retrieved:

```
<xsl:stylesheet>
      xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
      version='1.0'
      xmlns:dxl='http://www.lotus.com/dxl'
      exclude-result-prefixes='dxl' >
      <xsl:output method='html'/>
      <xsl:template match='/'>
      <H2>Field count</H2>
      <xsl:for-each select='//dxl:form'>
      <xsl:sort select='@name'/>
<BR/><B><xsl:value-of select='@name'/></B>:
<xsl:value-of select='count(dxl:body//dxl:field)'/> fields
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Notice that the XSLT stylesheet not only describes what it is we want to match within an XML text but also how we want to present it—the embedded HTML tags are processed normally.

The partial output document will generate this HTML code. It can, of course, be displayed in a browser:

```
<H2>Field count</H2>
<BR/>
<B>_Calendar Entry</B>:
38 fields
<BR/>
<B>_Document Memo</B>:
49 fields
<BR/>
<B>_Special\Link Message</B>:
23 fields
<BR/>
<B>_Special\Phone Message</B>:
22 fields
<BR/>
<B>_Special\Send Memo To Database Manager</B>:
39 fields
```

XSLT relies on template-specification, pattern-matching, and associated rules for manipulating data based on the results of matches.

Why might you choose XSLT over DOM or SAX even if you are proficient in all three? With DOM and SAX, you must code a custom program to traverse the nodes or query the events within XML documents. Each new type of XML document requires a new program. By contrast, XSLT lets you describe the transformation you are seeking against the data and lets the XSL processor determine the best way to go about it. You don't mess with specifying tedious node manipulation logic.

XSLT's rule-based model may seem confusing at first for those of us comfortable with procedural languages. It is not really that different from a database query language like SQL. (XSLT isn't the same as SQL but the underlying declarative model is somewhat similar). This also reminds us that many programmers are already writing rule-like queries.

Strategies for real-world use

As we have described, beta versions of Domino Rnext features will be available over the coming months, providing an opportunity for hands-on exploration. However, your best bet for XML work right now is the Lotus XML Toolkit.

We also recommend that you climb the XSLT learning curve. This is well within the reach of any moderately competent Notes programmer. Don’t let the need to master XSLT become the barrier that delays real-world usage of XML and DXL.

Certainly, emerging industry tools for processing DXL, XML, and XSLT will hide implementation details from view that make current XML applications awkward at times. Still, remember that once the XML application is generated by Domino, any XML tool~~s~~ available in the industry can be applied against the data—like all XML data, it is "just" text.

These things said, how useful is DXL today? Any application that uses data can benefit from DXL. This covers every application in your organization.

It pertains especially to applications whose enhancement has been stalled by costly or time-consuming data exchange requirements. It also touches applications that have been too complex for integration within Notes. Thanks to DXL, Domino now becomes a data integration environment for vast data stores across your entire organization.

Internet resources to support your DXL work

General XML
W3C Extensible Markup Language (XML) page
XML specifications, guidelines, software, and tools from the World Wide Web Consortium (W3C).

A Technical Introduction to XML
A short but comprehensive tutorial to XML from XML.Com from 1998 but still eminently usable.

The XML FAQ
Peter Flynn's definitive FAQ is stable but was updated as recently as June, 2001.

O'Reilly's XML.Com
This online magazine is probably the foremost editorial source for up-to-date information on XML developments in the industry.

IBM developerWorks XML Zone
A superb collection of articles, tools, and links covering the gamut of XML and related technologies.

Microsoft's MSDN XML Core
This is a good place from which to explore Microsoft's involvement with this technology.

The Apache XML Project
A watering hole for open-source development of XML technologies and tools.

The XML Cover Pages
Robin Cover's near-daily coverage of XML news has long been a favorite of XML aficionados.

Cafe con Leche XML News and Resources
Another great source for daily XML news coverage plus lots of more persistent articles.

DevX XML Zone
Yet another comprehensive hub for XML links and resources.

Lotus Software strategy and tools for XML
Lotus XML page
A collection of articles that focus on using XML with current and emerging Lotus products.

Lotus XML Toolkit page
Download the Toolkit here.

The DXL Resources page on Notes.net
DXL presentations from Lotusphere.

DTDs and Schemas
XML DTD Tutorial
You won't find elaborate explanations, but this will serve as a ready reference for using DTD syntax correctly.

XML Schemas
The W3C's schema page, with multiple links to ongoing developments around the Internet.

XML Parsers and Protocols

Guide to XML Parsers
This not only explains parsers but offers numerous links around the Web to parsing tools.

XML-RPC
The spec and needed help for using XML for remote procedure calls over the Internet based around HTTP.

Java
Java and XML
APIs, downloads, and white papers from Sun.

XSLT
What Kind of Language is XSLT?
An IBM developerWorks article by Michael Kay, who authored the Sax parser and is widely considered the foremost authority on XSLT.