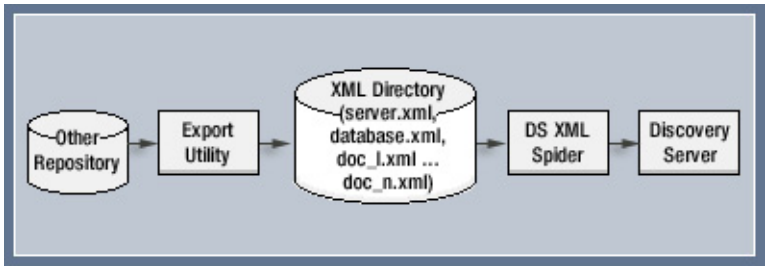
Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

Level: Intermediate
Works with: Discovery Server
Updated: 02-Jan-2003

by
Wendi
Pohs

Fans of Lotus Discovery Server use it to collect, organize, and search for data from a variety of data sources. Discovery Server gathers data from these sources with a set of collectors, called spiders; administrators enable one spider for each source type. Spiders collect both unstructured and structured data from the documents they crawl. Unstructured data is usually the body or text of a document. Structured data is information contained in fields and tags associated with a document, which includes data such as a document's title, author, creation date, URL, and descriptive keywords.

Discovery Server ships with spiders for Notes databases, Lotus QuickPlaces, Domino.Doc applications, Web sites, files on a file system, email, and Microsoft Exchange folders. But there's also a general purpose spider, the XML spider, that can (with a little programming effort) access data stored in a variety of legacy and Web applications. As the following illustration shows, the XML spider gathers XML files from directories on the file system. As long as these XML files conform to the published Discovery Server Document Type Definition (DTD), the XML spider can use them to collect data for Discovery Server.



This article offers advice on how to use the XML spider that ships with Discovery Server. We explain the details of the XML spider and provide two sample ways to use it. (One of these samples includes code you can download from the **Sandbox**.)

This article assumes some familiarity with XML and with the Discovery Server spiders and how they work. For more information about Discovery Server, see the *LDD Today* article **"A preview of Lotus Discovery Server 2.0**." For detailed information on the Discovery Server spiders, see the **Discovery Server Control Center Guide**, which you can download from the Documentation Library. And for more information on how to integrate with a content management system, see the IBM Redbook **Lotus Discovery Server 2.0 Deployment, Planning and Integration**.

Why XML?

XML has become the *lingua franca* of data exchange. Application designers create elements, formats, and rules that describe the data in their applications so that non-native applications can use them. Programmers work from Document Type Definitions (DTDs) to understand these elements and to create XML files that can be consumed by external applications. They can also create stylesheets (XSL files) that contain the rules for transforming XML data into another format.

Why use the XML spider?
When designing Discovery Server, we intended it to access information from many sources. Based on our experience accessing our own data, and by using Discovery Server internally at Lotus, we realized that while we could develop individual spiders for each data source, we would still have less information and knowledge of these sources than the actual application developers themselves. So we designed and built the XML spider as a generic way for other developers to add application knowledge to a Discovery Server implementation.

We also realized that data from multiple sources tends to be inconsistent. Title fields are not always called Title and fields containing author information are called anything from Creator to Contributor to From. Many documents contain extemporaneous fields that describe workflow and other housekeeping information that is not necessarily relevant to users. We imagined that content owners would use the XML spider to standardize their data as best they could and to make it useful for Discovery Server searches.

Discovery Server XML DTD
The Discovery Server XML DTD is the published specification of the content elements, format, and rules used by Discovery Server spider extraction processes. The XML DTD is used with the Discovery Server XML spider to extract XML documents that are stored in a network-accessible file system. The Discovery Server XML spider requires three types of XML files (Document, Server, and Database) and includes a DTD (Document Type Definition) for each type. All documents from the same data repository should be outputted and grouped in a separate directory containing the two special files called Server.xml and Database.xml.

Validating published XML against the Document Type definitions

While all Discovery Server spiders write XML to the Discovery Server work queues, only the Discovery Server XML spider can process XML published from proprietary, or external, sources. The XML spider consumes the Published XML files, validates them against the Discovery Server DTDs, then uses XSL transformations to produce XML to be processed by the K-map Building, Metrics, and K-map Indexing services.

Document, Server, and Database DTDs are located in document.dtd, server.dtd, and database.dtd files respectively and are installed in the Discovery Server's \Data\discovery spider xml directory. Externally developed XML applications are responsible both for specifying a correct file path to the corresponding DTD and for generating a DOCTYPE declaration for each XML file they produce, for example:

<!DOCTYPE DOCUMENT SYSTEM "\\myServer\fdrive\Lotus\DS\Data\discovery spider xml\document.dtd">

Note: For Discovery Server 2.0, HTTP URLs such as

<!DOCTYPE DOCUMENT SYSTEM "http://myServer.lotus.com/xml/document.dtd">

are not supported.

Entity Document specifications
The following table identifies which elements are required and cannot be empty, which are optional, and whether or not multiple entries are acceptable. An element marked as mandatory should be included in each document. The only exception from this rule is a delete request, which is a special case (see <DELETE> and <DOCUMENT> elements for details). If information associated with a required tag is unknown or not applicable, an empty tag should still be included to speed up processing of the document.

Element name	Description	Mandatory/ optional	Can be empty?	Multiple entries?
DOCUMENT	The root element that contains an <IDENTIFIER> element followed by either a <DELETE/> tag or a set of metadata elements.	Mandatory	No	No
IDENTIFIER	Contains a document ID, a unique alphanumeric string that identifies the	Mandatory	No	No

	document. This tag has to be present for each document and appears only once. It cannot be empty. The <HTTPURL> or <NATIVEURL> elements may be good candidates for the <IDENTIFIER>. **Note:** A semicolon is not a valid character for the <IDENTIFIER>.			
AUTHOR	Contains the name of the person who created the document. If the author is unknown, the tag can be left empty. Each <AUTHOR> element can contain only one name, but multiple occurrences of the element are allowed. For example, if the authors are Paul White and Jake Black, then the <DOCUMENT> element contains: ● <AUTHOR>Paul White</AUTHOR> ● <AUTHOR>Jake Black</AUTHOR> Authors are used to generate affinities.	Mandatory	Yes	Yes
UPDATEDBY	Contains the name of the person who last modified/saved the document. This tag follows the same rules as <AUTHOR>. In addition, it has the added requirement that it cannot be empty when the <AUTHOR> tag is filled. If people who modified the document are unknown, copy authors into <UPDATEDBY> elements.	Mandatory	Yes	Yes
CREATED	Contains the date/time when the document was first created. All time/dates must be GMT, which means the XML spider performs no conversion. The date/time should be presented in the YYYY-MM-DD-HH.MM.SS format, for example: 1999-03-30-21.19.26 Each <DOCUMENT> has to contain exactly one <CREATED> tag, unless the document is being deleted. This element cannot be empty. If the creation date is unknown, default to the current time.	Mandatory	No	No
LASTREAD	Contains the date/time when the document was last accessed. This tag follows the same rules as the <CREATED> element. If the time of the last access is unknown, copy the <CREATED> time into this tag.	Mandatory	No	No
MODIFIED	Contains the date/time when the document was last modified. This tag follows the same rules as the <CREATED> element. If the time of the last access is unknown, copy the <CREATED> time into this tag.	Mandatory	No	No
REVISIONS	Contains a set of <REVISION> elements. Only one <REVISIONS> per document is permitted.	Mandatory	No	No
REVISION	This child element of <REVISIONS> contains a date/time when the document was modified. This tag follows the same rules as the <CREATED> element, but multiple <REVISION> elements are permitted. If revision information is unknown, copy the <CREATED> time into	Mandatory	No	Yes

	this tag.			
FIELD	Optional element. Reserved for future use.	Optional	Yes	Yes
TITLE	Contains the title of the document. Same requirements as for the <SUMMARY> element below. If omitted, Discovery Server attempts to generate a title using document subject, file name derived from the <HTTPURL>/<NATIVEURL> elements, or the first line of the document body. If unsuccessful, Discovery Server defaults to "[Untitled]."	Optional	No	No
SUBJECT	Contains the subject of the document. Same requirements as for the <SUMMARY> element below. If subject is omitted, Discovery Server attempts to generate one.	Optional	No	No
SUMMARY	Provides a short description of the document. If present, the element should not be empty. Only one <SUMMARY> per document is permitted. If this tag is omitted, Discovery Server tries to create a summary, providing that the document is written in a supported language. See the list of supported languages later in this article. Summary should be 256 characters or shorter.	Optional	No	No
KEYWORDS	Contains a set of <KEYWORD> elements. This element is optional, but if present, it should contain at least one <KEYWORD>. Only one <KEYWORDS> element per document is allowed. All keywords combined should be MAX_LEN characters or shorter, where MAX_LEN is computed according to the formula: MAX_LEN = 256 - (number of <KEYWORD> elements).	Optional	Yes	No
KEYWORD	A child element of <KEYWORDS>. Same requirements as for <SUMMARY>, except that multiple <KEYWORD> elements are allowed.	Optional	No	Yes
BODY	Contains the document body. This tag can be empty. Only one <BODY> element per document is allowed.	Mandatory	Yes	No
APPLICATION	Provides the name of the application that created the document. If present, this tag should not be empty. **Note:** This setting does not affect how the K-Map and the K-Map Editor display the document. Non-Domino files are viewed in the application registered by the operating system for the document's file extension.	Mandatory	No	No
LANGUAGE	Contains the default language of the document represented in the two-letter **ISO 639** language abbreviation followed by an optional ISO 3166 country code. In the ISO 639/ISO 3166 convention, language names are written in lowercase, while country codes are written in uppercase, for example, en-US.	Mandatory	Yes	No

	If this tag is empty, Discovery Server guesses the correct language by examining the document body. This information is used to create a document summary and a list of keywords. Only one <LANGUAGE> element per document is allowed. In addition to the two-letter ISO 639 language abbreviations, use *bk* for Bokmal and *ny* for Nynorsk.			
NATIVEURL	Contains the Notes or File URL of the document, for example: ● NOTES://epr.acme.com/Wsj.nsf/0/00E5CFF068B33B1A852569760021DDDF?Open ● file:////epracmepr/fdrive/Testfiles/fsspr/93sales.xls. <NATIVEURL> is used by the K-Map and the K-Map Editor to display the document. If the document cannot be retrieved, the <HTTPURL> tag is used instead. For Notes documents, native URL is used when the <USENOTES/> tag is present.	Optional		No
HTTPURL	Contains the HTTP URL of the document, for example: HTTP://epr.acme.com/Wsj.nsf/0/00E5CFF068B33B1A852569760021DDDF?OpenDocument URLs should be represented in absolute form and be consistent with **Uniform Resource Identifiers (URI): Generic Syntax and Semantics, RFC 2396**.	Mandatory		No
ACL	Optional tag that contains a document's access control list. Contains a collection of ALLOW and DENY elements. If empty or missing, it is assumed that everyone with repository access can view the document. Only one access control list per document is permitted. In addition to the document ACL, Discovery Server takes into account the repository ACL supplied in the Database.xml file. The document is exposed to a user only if the user is included in the repository and document ALLOW lists and is not included in the DENY list. Ensure that user identities defined in external repositories map to user identities Discovery Server uses to grant access to the K-map (via HTTP authentication and user identity and password contained within the DS Directory - Person record)	Mandatory	Yes	No
ALLOW	This child element of <ACL> contains a group or a user who can read the document. This element is optional, but if	Optional	No	Yes

	present, it should not be empty. Each <ALLOW> element can contain only one name or group, but multiple occurrences of the element are allowed. If <ALLOW> tags are not present, Discovery Server assumes that all users with repository access can view the document, except for those explicitly stated in the DENY list. NT users and groups should be listed in the DOMAIN/USER_NAME format. Currently, access checking of users in external NT domains is not supported. Names are case-insensitive.			
DENY	This child element of <ACL> contains a group or a user who is denied reader access. Same requirements as for <ALLOW>.	Optional	No	Yes
LINKS	Contains a collection of <LINK> elements. This element is optional, but if present, it should contain at least one <LINK>. Only one <LINKS> element per document is allowed.	Optional	No	No
LINK	This child element of <LINKS> describes a link contained in the document. The URL attribute is an address that points to the destination anchor, that should be represented in absolute form, and that should be consistent with **Uniform Resource Identifiers (URI): Generic Syntax and Semantics, RFC 2396**. Multiple occurrences of the element are allowed.	Optional	Yes	Yes
USENOTES	Specifies a preferred viewer for a Lotus Notes document. If this tag is present, the document opens in a new Lotus Notes window. If the Notes client is not available, the default Internet browser becomes the fallback viewer. If the tag is omitted, the document is displayed in a new browser window. This element is always empty.	Optional	Yes	No
INCLUDE	Optional tag. Only one <INCLUDE> tag per document is allowed. If this element is present, Discovery Server merges the included file and the container. The FILEPATH attribute should not be empty. The resulting document is registered with the Discovery Server. The merge happens according to the following rules: ● Always use text (document body), keywords, and summary from the included file. ● Use container's metadata. If container is missing an author or title, then get omitted information from the included file. ● Use container's ACLs. Attribute: ● FILEPATH - location of the attached file.	Optional	Yes	No

ATTACHMENT	Optional tag. If this element is present, the attributes should not be empty. For multiple attachments in a document, Discovery Server processes the attached file in a manner similar to how attachments are processed for Domino. Attachments are registered with Discovery Server independently from the container. Rules used to process attachment metadata: ● If the attachment is missing an author, title, summary, or keywords, then get omitted metadata from the container. ● Combine the title/subject of the container with the attachment name. For example, if container's title is Container1, and attachment is named file1.doc, the new title is Container1(attachment - file1.doc). ● Use container's ACLs. Attributes: ● FILEPATH - Specifies the location of the attached file ● IDENTIFIER - Contains a unique ID for the attachment ● NATIVEURL - Specifies the Notes or File URL of the attachment ● HTTPURL - Specifies the HTTP URL of the attachment ● USENOTES - Specifies a preferred viewer for the attachment	Optional	Yes	Yes
DELETE	Used to remove previously processed documents from the data repository. If this tag is present, the document is deleted. This element is always empty. **Note:** If the document is new and has not been registered during a previous run of the spider, a message is logged.			

Database Document specifications
The following table identifies which elements are required and cannot be empty, which are optional, and whether or not multiple entries are acceptable. An element marked as mandatory always should be included. The DATABASE element must be stored in a special file called Database.xml. If Database.xml is missing, the XML spider logs an error and terminates spidering.

Element name	Description	Optional/ mandatory	Can be empty?	Multiple entries?
DATABASE	The root element that contains <DATABASEID> and <ACL> elements.	Mandatory	No	No
DATABASEID	Contains a data repository ID as an alphanumeric string that identifies the repository. This tag is required, can appear only once, and cannot be empty. On the initial traversal, the ID should be new and unique. In other words, there should be no other data repository registered with this ID. On the following traversals, the ID should not be changed.	Mandatory	No	No

	Note: The semicolon is not a valid character for the <DATABASEID>.			
NATIVEURL	Contains the Notes or File URL for the repository. <NATIVEURL> is used by the K-map and the K-map Editor to display the repository. If the repository cannot be retrieved, <HTTPURL> is utilized instead. For Notes databases, <NATIVEURL> is used when the <USENOTES/> tag is present.	Mandatory	No	No
HTTPURL	Contains the HTTP URL for the repository. URLs should be represented in absolute form and be consistent with **Uniform Resource Identifiers (URI):Generic Syntax and Semantics, RFC 2396**.	Mandatory	No	No
SUMMARY	Provides a short description of the repository. If present, the element should not be empty. Only one <SUMMARY> tag per database is permitted. The summary should be 256 characters or shorter.	Optional	No	No
OWNER	Contains the repository owner's name. If present, the element should not be empty. Only one <OWNER> tag per database is permitted.	Optional	No	No
ACL	Contains the repository access control list. Contains a collection of <ALLOW> and <DENY> elements. If not present or if empty, it is assumed that everyone has repository access rights. Only one access control list per repository is permitted. In addition to the repository ACL, Discovery Server takes into account ACLs provided for each individual document. You need to ensure that user identities defined in external repositories map to user identities that Discovery Server uses to grant access to the K-map (via HTTP authentication and user identity and password contained within the Discovery Server Directory - Person record). Attributes: ● DOMAINTYPE - Set to WindowsNT or Domino ● DEFAULTDOMAIN - This default name is used if <ALLOW> or <DENY> elements lack domain names and only contain user names (for example, <ALLOW>User1</ALLOW>). The Authentication part of Discovery Server must know about this domain to handle it.	Optional	Yes	No
ALLOW	This child element of <ACL> contains a group or a user who can read documents in the repository. This element is optional, but if present, it should not be empty. Each <ALLOW> element can contain only	Optional	No	Yes

	one name or group, but multiple occurrences of the element are allowed. If <ALLOW> tags are not present, we assume that all users are granted repository access rights, except for those explicitly stated in the <DENY> list. NT users and groups should be listed in the DOMAIN/USER_NAME format. Currently, access checking of users in external NT domains is not supported. Names are case-insensitive.			
DENY	This child element of <ACL> contains a group or a user who is denied reader access. This element is optional. Same requirements as for <ALLOW>.	Optional	No	Yes

Server Document Specifications
The following table identifies which elements are required and cannot be empty, which are optional, and whether or not multiple entries are acceptable. An element marked as mandatory always should be included.

Element name	Description	Optional/ mandatory	Can be empty?	Multiple entries?
SERVER	The root element that contains a <SERVERID>. <SERVER> must be stored in a special file called Server.xml.	Mandatory	No	No
SERVERID	Contains a server ID as a unique alphanumeric string. This tag is required, can appear only once, and cannot be empty. On the following traversals, the ID should not be changed.	Mandatory	No	No

Sample applications for XML spider integration

Two examples of the kinds of applications that you can integrate with the XML spider are:

- Content management systems
- Web sites

This section discusses these applications and how you can integrate them with the XML spider.

Content management systems
Because the Discovery Server is optimized for finding text documents, it's often a good idea to integrate it with an existing content management system. Content management systems often use relational database structures to maintain metadata, document security, and version control. They can also use the server's file system to store the content files (for example, MS Word documents). Content management systems also provide their own user interface, optimized for the task of creating and maintaining documents. It is unlikely that a Discovery Server native spider will have direct access to the files managed by the document management system. If Discovery Server directly spidered the file system for the document files, it would never see the metadata stored in the relational database. To access the relevant metadata, you can transform the document file and the metadata into a single XML file that can then be spidered by the Discovery Server XML spider.

In this scenario, you can write an extract program that connects to the content management system, generally using a published API from the content management system vendor. This program reads the relevant metadata and creates a new XML file for each document. In addition to including any required metadata in the XML file, you can reference the document file in the XML file, and the XML spider will collect the document file as well as use the included metadata. In situations where it is not possible for the Discovery Server spider to access the document file directly in its original location (usually due to security policies) the extraction program has to place a copy of the document file in a spider-accessible directory as a temporary copy for the spider to use. (Note that this temporary copy is only used by the spider and that K-map users will access the document file in its original location, using any required authentication.) For more detailed information on how to integrate with a content management system, see

the IBM Redbook **Lotus Discovery Server 2.0 Deployment, Planning and Integration.**

Web sites
The XML spider can also be used with existing data on the Web. Indexing and abstracting Web sites like the National Library of Medicine's **PubMed** provide utilities that allow developers to select certain fields to download data in XML format. Developers can use this data in a Discovery Server implementation by writing an XSL transformation that conforms to the published DTD. After the transformation, they place these files in a directory on the file system for the XML spider to use. In this scenario, Discovery Server users can access the relevant medical citations along with internal information.

Below is a sample XSL transformation of PubMed data. (You can download this sample from the **Sandbox**.) In this sample, the developer knew the format of the XML he downloaded from the PubMed Web site. Using the Discovery Server published DTD as a guide, he created the following XSL transformation. In it, he created a series of templates that map the PubMed data to the elements specified in the Discovery Server's document.dtd file.

The process initiates here:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" doctype-system="document.dtd"/>

    <xsl:template match="@*|node()">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()"/>
        </xsl:copy>
    </xsl:template>

    <!-- The root of the article is PubmedArticle -->
    <xsl:template match="PubmedArticle">
        <xsl:element name="DOCUMENT">
            <xsl:apply-templates select="@*|node()"/>
        </xsl:element>
    </xsl:template>
```

This next section performs the transformation, using the templates defined later on in the routine. The templates map the PubMed XML elements to the XML elements in the Discovery Server's document.dtd file:

```
    <!-- Main line routine calls all required templates, completes default sections
         as required by the lotus dtd. -->
    <xsl:template match="MedlineCitation">
        <xsl:element name="IDENTIFIER">
            <xsl:text>MedlineID:</xsl:text>
            <xsl:value-of select="MedlineID"/>
            <xsl:text> PMID:</xsl:text>
            <xsl:value-of select="PMID"/>
        </xsl:element>
        <xsl:apply-templates select="Article/AuthorList"/>
        <UPDATEDBY></UPDATEDBY>
        <xsl:apply-templates select="DateCreated"/>
        <LASTREAD></LASTREAD>
        <xsl:choose>
            <xsl:when test="DateRevised">
                <xsl:apply-templates select="DateRevised"/>
            </xsl:when>
            <xsl:otherwise>
                <MODIFIED></MODIFIED>
            </xsl:otherwise>
        </xsl:choose>
        <REVISIONS></REVISIONS>
        <xsl:apply-templates select="Article/ArticleTitle"/>
        <KEYWORDS>
        <xsl:apply-templates select="MeshHeadingList"/>
        <xsl:apply-templates select="ChemicalList"/>
```



```
        </KEYWORDS>
        <xsl:apply-templates select="Article/Abstract"/>
        <APPLICATION></APPLICATION>
        <LANGUAGE></LANGUAGE>
        <NATIVEURL></NATIVEURL>
        <HTTPURL></HTTPURL>
        <ACL></ACL>
    </xsl:template>
```

This next section defines the templates. This developer and his users have searched the external PubMed database before, so they know which fields are relevant to their work and which fields they would like to see when they search both internal and external sources using the Discovery Server. The developer used the PubMed DTD, also available on the PubMed Web site, as a guide:

```
    <!-- Following are templates for handling specfic transformations -->

    <xsl:template match="Article">
        <xsl:apply-templates select="AuthorList"/>
        <xsl:apply-templates select="ArticleTitle"/>
        <xsl:apply-templates select="Abstract"/>
    </xsl:template>

    <xsl:template match="AuthorList">
        <xsl:apply-templates select="Author"/>
    </xsl:template>
    <xsl:template match="Author">
        <xsl:element name="AUTHOR">
            <xsl:value-of select="LastName"/>
            <xsl:text>,</xsl:text>
            <xsl:value-of select="ForeName"/>
            <xsl:text> </xsl:text>
            <xsl:value-of select="Initials"/>
        </xsl:element>
        <!--<Author name="{LastName},{ForeName} {Initials}"/> -->
    </xsl:template>

    <xsl:template match="Abstract">
        <xsl:apply-templates select="AbstractText"/>
    </xsl:template>

    <xsl:template match="DateCreated">
        <xsl:element name="CREATED">
            <xsl:value-of select="Year"/>
            <xsl:text>-</xsl:text>
            <xsl:value-of select="Month"/>
            <xsl:text>-</xsl:text>
            <xsl:value-of select="Day"/>
        </xsl:element>
    </xsl:template>
    <xsl:template match="DateRevised">
        <xsl:element name="MODIFIED">
            <xsl:value-of select="Year"/>
            <xsl:text>-</xsl:text>
            <xsl:value-of select="Month"/>
            <xsl:text>-</xsl:text>
            <xsl:value-of select="Day"/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="AbstractText">
        <BODY TYPE="TEXT">
            <xsl:value-of select="." />
        </BODY>
```



```
        </xsl:template>

        <xsl:template match="ArticleTitle">
            <xsl:element name="TITLE">
                <xsl:value-of select="." />
            </xsl:element>
        </xsl:template>

        <xsl:template match="MeshHeadingList">
            <xsl:apply-templates select="MeshHeading"/>
        </xsl:template>
        <xsl:template match="MeshHeading">
            <xsl:apply-templates select="DescriptorName"/>
            <xsl:apply-templates select="QualiferName"/>
        </xsl:template>

        <xsl:template match="DescriptorName">
            <xsl:element name="KEYWORD">
                <xsl:value-of select="." />
            </xsl:element>
        </xsl:template>
        <xsl:template match="QualifierName">
            <xsl:element name="KEYWORD">
                <xsl:value-of select="." />
            </xsl:element>
        </xsl:template>

        <xsl:template match="ChemicalList">
            <xsl:apply-templates select="Chemical"/>
        </xsl:template>
        <xsl:template match="Chemical">
            <xsl:apply-templates select="NameOfSubstance"/>
        </xsl:template>

        <xsl:template match="NameOfSubstance">
            <xsl:element name="KEYWORD">
                <xsl:value-of select="." />
            </xsl:element>
        </xsl:template>

        <!-- default templates no transfformatio required -->
        <xsl:template match="PubmedData"/>
        <xsl:template match="CitationSubset"/>
        <xsl:template match="MedlineJournalInfo"/>
        <xsl:template match="DateCompleted"/>
        <xsl:template match="MedlineID"/>
        <xsl:template match="PMID"/>
    </xsl:stylesheet>
```

The following is a sample of the contents of an output XML file this program writes to the local file system:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DOCUMENT SYSTEM "document.dtd">
<DOCUMENT>
<IDENTIFIER>MedlineID:21200508 PMID:11304842</IDENTIFIER>
<AUTHOR>Baglin,T P TP</AUTHOR>
<UPDATEDBY/>
<CREATED>2001-04-17</CREATED>
<LASTREAD/>
<MODIFIED>2001-11-28</MODIFIED>
<REVISIONS/>
<TITLE>Heparin induced thrombocytopenia thrombosis (HIT/T) syndrome: diagnosis and treatment.</TITLE>
<KEYWORDS>
```



```
<KEYWORD>Anticoagulants</KEYWORD>
<KEYWORD>Heparin</KEYWORD>
<KEYWORD>Hirudin</KEYWORD>
<KEYWORD>Hirudin Therapy</KEYWORD>
<KEYWORD>Human</KEYWORD>
<KEYWORD>Pipecolic Acids</KEYWORD>
<KEYWORD>Platelet Count</KEYWORD>
<KEYWORD>Recombinant Proteins</KEYWORD>
<KEYWORD>Thrombocytopenia</KEYWORD>
<KEYWORD>Thrombosis</KEYWORD>
<KEYWORD>Anticoagulants</KEYWORD>
<KEYWORD>Pipecolic Acids</KEYWORD>
<KEYWORD>Recombinant Proteins</KEYWORD>
<KEYWORD>lepirudin</KEYWORD>
<KEYWORD>Argatroban</KEYWORD>
<KEYWORD>Hirudin</KEYWORD>
<KEYWORD>Heparin</KEYWORD>
</KEYWORDS>
<BODY TYPE="TEXT">Heparin induced thrombocytopenia thrombosis (HIT/T) is associated with a high morbidity and mortality. Diagnosis is essentially clinical and negative results of laboratory assays do not exclude the diagnosis. Treatment involves stopping all heparin immediately and giving an alternative thrombin inhibitor. The adoption of low molecular weight heparins is one reason for the reduced incidence of this disease in recent years.</BODY>
<APPLICATION/>
<LANGUAGE/>
<NATIVEURL/>
<HTTPURL/>
<ACL/>

</DOCUMENT>
```

After all the XML files have been extracted and transformed, a Discovery Server administrator can specify a data repository definition form that instructs the XML spider where to find the particular XML content. Unlike other spiders, there's no need to specify field mapping for XML repositories because the mapping is already defined in the XML. As with other repository types, the data repository form enables the administrator to schedule when the XML files should be queued for processing by the Discovery Server services. Unlike the File System spider, which always leaves repository files intact, the XML spider manages data according to an After Processing setting that appears on the Data Repository form. Selecting either the Truncate or Remove option saves space by compressing or removing the data after it's been processed by the XML spider.

Access considerations for the XML spider

During its initial spider run, the XML spider looks for two special files: Server.xml and Database.xml, which help it distinguish between published and unpublished XML files. These files are processed prior to all other documents. Server and database IDs extracted from these files are written to the repository's context. This is the only time when database and server IDs are processed. During the subsequent processing, the XML spider picks up ACL changes from Database.xml, but does not update database or server IDs.

The XML spider attempts to read files of any size, as if the No Maximum option were selected, so no special file size parameters need to be specified on the Spider Settings form.

Note: The XML spider terminates spidering of a repository if it can't access DTD or XSL files located in the Data\discovery spider xml directory.

Unlike the File System spider, the XML spider does not use the WindowsNT ACLs associated with the spidered directory. It uses the ACLs supplied in the XML file instead. To use ACLs in the XML spider:

- Include an <ACL> element in the Database.xml file.
- Include at least one <ALLOW> element for database ACL.
- Use an empty <ALLOW> tag to allow full access for all users.

Conclusion

By using the XML spider, an organization can open up data content that might not normally be available to Discovery Server end users. An application developer's simple investment of time can open up valuable and useful

content that would otherwise remain inaccessible to the people who need it.

ABOUT THE AUTHOR

Wendi Pohs is a principal taxonomy specialist on the Discovery Server team and the author of a book about knowledge management methodologies, *Practical Knowledge Management: The Lotus Knowledge Discovery System*, published by IBM Press. Wendi joined Lotus Development Corporation in 1996 and has worked on various projects as a spec writer, online help designer, and user assistance manager. Prior to joining Lotus, Wendi worked at the American Mathematical Society and at Digital Equipment Corporation. Wendi received her BA and MILS degrees from the University of Michigan.

