

Check Out That DirXML Engine

Linda Kennard

Assuming you've read the related article "Too Many Directories? Synch 'Em With DirXML," you already know that DirXML can synchronize shared data among the different directories on your company's network. You also know that by synchronizing these directories, DirXML saves you time, saves your company money, and opens e-business doors. What you don't know is how DirXML works. Interested? Then read on.

THE INFO FLOW: COMING OR GOING?

The DirXML engine logically sits between NDS eDirectory and any other directory you choose to synchronize with NDS eDirectory (hereafter referred to as NDS). (See Figure 1 on p. 18.) Between NDS and any other directory (hereafter referred to as Other Directory), the DirXML engine enables bidirectional communications through two channels:

- The subscriber channel
- The publisher channel

Information between NDS and Other Directory is either coming from or going to NDS. The subscriber channel carries information coming from NDS to Other Directory. Conversely, the publisher channel carries information from Other Directory going to NDS.

Using the subscriber and publisher channels between NDS and Other Directory, DirXML can synchronize any attribute of any object class you choose. The specific information that crosses these channels depends upon the filters you create. For each channel, you create a filter that identifies the object classes and attributes you want DirXML to synchronize between NDS and Other Directory.

For example, when you create a filter on the subscriber channel, you may indicate that you want DirXML to push through to Other Directory changes made to NDS User objects' names and telephone numbers. When you create a filter on the publisher channel, you may indicate that you want DirXML to push through to NDS changes to the employee ID numbers stored in Other Directory's Person objects.

Filters for the subscriber and publisher channels enable you not only to control which object classes and attributes you synchronize but also to control which directory will act as the authority on a particular object class and attribute.

For example, suppose you want to make the human resource (HR) database the authority on employee ID numbers. To ensure that the HR database is the source for any employee's ID number field in all DirXML-connected directories, you create a



filter for the HR database publisher channel. This filter indicates (among other things) that you want DirXML to push changes to employee ID numbers through to NDS. As long as the publisher channels for Other Directory do not include such a filter, NDS will only accept changes to employee ID numbers that come from the HR database.

The subscriber and publisher channels span the entire DirXML engine, which includes two main components. (See Figure 1 on p. 18.)

- The Join Engine
- The Application Shim

THE JOIN ENGINE TRANSLATING FORMATS

For the sake of simplicity, this article focuses on the process of passing information from NDS to Other Directory. When information passes through the subscriber channel en route to Other Directory, the DirXML Join Engine uses Extensible Markup Language (XML) to convert the information it receives. A W3C standard since 1998, XML is a simplified dialect and arguably best-parts version of the Standard Generalized Markup Language (SGML).

Like its SGML parent, XML defines how to prepare platform-independent, unambiguous, text-formatted versions of structured data. Structured data includes documents such as spreadsheets, financial transactions, technical drawings, and any other data you create using any program. In a DirXML context, the structured data that XML converts is directory objects, attributes, and their respective values. (For more information about XML, visit <http://www.w3.org/XML/1999/XML-in-10-points>.)

After converting the information received, the Join Engine loads the resulting XML-formatted data into what is called the *Document Object Model (DOM)*, a representation of the XML-formatted data in memory. The DOM is a W3C specification that provides a standard application programming interface (API) for XML (and HTML) documents. The DOM represents the logical structure of the documents it models in a hierarchical or tree-like fashion. (For more information about the DOM, see

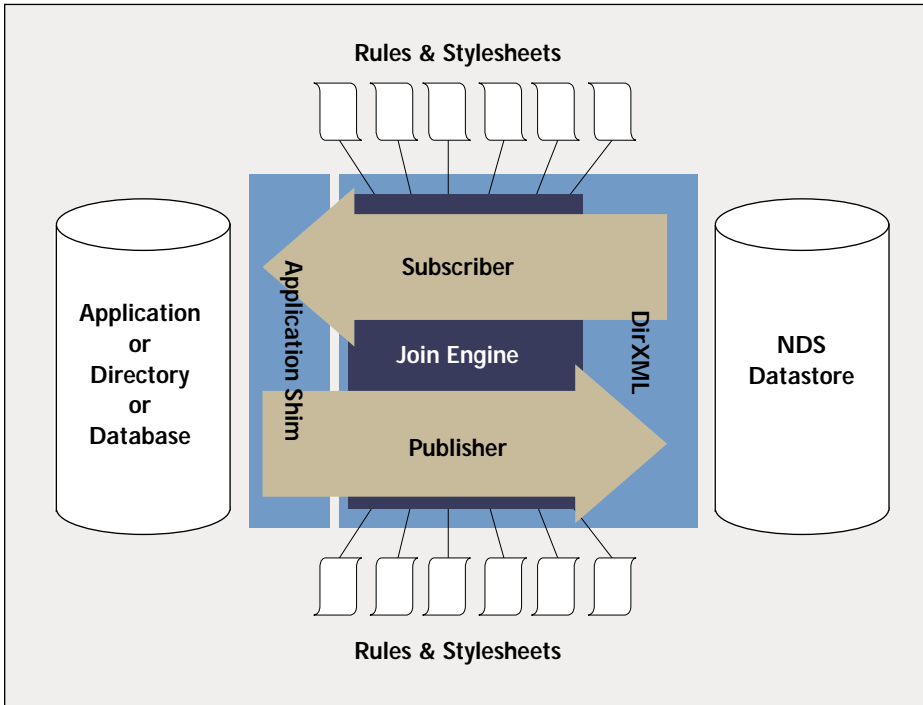


Figure 1. The DirXML Architecture

<http://www.w3.org/tr/2000/cr-dom-Level-2-20000307/introduction.html>.)

Because the XML-formatted data is represented in memory and is never serialized—that is, never converted into an actual XML text file—it is easy to make changes to the information the XML-formatted data references. “In memory,” explains Nick Nikols, DirXML development manager and architect, “the XML[-formatted data] is represented as a hierarchical structure, from which we can grab any leaf element and change it however necessary.” In contrast, Nikols continues, if DirXML serialized the XML-formatted data, “we’d have to use a parser to operate on the information.” In other words, because the XML-formatted data is represented in memory as a DOM, the DirXML Join Engine can easily access, change, add to, or delete any portion of the information referenced in this XML-formatted data.

The ease with which the Join Engine can modify the XML-formatted data (represented in memory as a DOM) is fortunate because changing this data is essentially the Join Engine’s job. The DirXML Join Engine modifies the XML-formatted data to ensure that it is application-ready. The Join Engine converts this data until it no longer references NDS naming or the NDS schema but instead references the naming and schema of Other Directory.

To render the XML-formatted data application-ready, the DirXML Join Engine applies several rules, which are stored within NDS. You configure these rules using either the DirXML snap-in module for ConsoleOne or, if you prefer, using XML.

To apply these rules, the Join Engine uses two processors:

- The Rules Processor
- The XSLT Processor

The Rules Processor

The Rules Processor is responsible for applying two types of rules:

- Schema-mapping rules
- Object-mapping rules

As the name suggests, the schema-mapping rules map the NDS schema to the Other Directory schema. The Rules Processor applies the schema-mapping rules to match NDS object classes and attributes to the Other Directory’s object classes and attributes. For example, the Rules Processor may determine that the NDS User object equals Other Directory’s Person object and that the NDS Surname attribute matches Other Directory’s Last Name attribute.

The object-mapping rules identify matches between specific objects. For ex-

ample, by applying the object-mapping rules, the Rules Processor may determine that the NDS User object once named Linda Boyer matches Other Directory’s Person object of the same name. When the Rules Processor finds a match, the Rules Processor creates an Association, which is stored in NDS. After the Rules Processor has created an Association, DirXML can consult this Association to learn where to place changes associated with this object.

If the Rules Processor finds that it has not yet created an Association, it uses the following three object-mapping rules to create one:

- Matching rules
- Create rules
- Placement rules

As the name suggests, matching rules are the rules you create to help the Rules Processor determine which NDS and Other Directory objects match. For example, your matching rules may say that if the social security number (SSN) for the suspected matching objects is the same, the Rules Processor should assume the objects match and create an Association. You may also create a second matching rule that says if one or both objects’ SSN values are missing, the Rules Processor should compare the first name, last name, and telephone numbers. If these values are the same, your rule may conclude, the Rules Processor should assume the objects match and create an Association.

If after applying the matching rules, the Rules Processor does not find a match, it assumes Other Directory does not yet have this object. Depending upon the rules you create, the Rules Processor then uses the create rules, which describe the criteria for creating new objects in Other Directory.

The create rules simply answer the question, “What attributes do I need before I can create an object?” For example, a create rule may specify that you do not want the Rules Processor to create a new object in Other Directory until it has values for the First Name, Last Name, SSN, and Telephone attributes.

After determining that it has the attributes required before creating an object, the Rules Processor applies the placement rules to determine where in Other Directory to place the new object.

For example, a placement rule may stipulate that if the NDS User object is in the NDS Engineering container and its Locality attribute is Provo, then the Rules Processor will place the matching, new object in the Provo Engineering space in Other Directory.

The XSLT Processor

The DirXML Join Engine also uses the Extensible Stylesheet Language Transformation (XSLT) Processor. In general, XSLT is a stylesheet transformational language that describes how to convert an XML document in one format to another XML format. (For more information, visit <http://www.w3.org/TR/xslt>.)

DirXML uses XSLT to do event transformations and data transformations. To convert events, the XSLT Processor uses event transformation stylesheets, which describe what the new event should be for any given event. For example, you may specify that the XSLT Processor should convert a "mark as inactive" event in PeopleSoft to a "delete" event in NDS.

The XSLT Processor can also convert data, or more specifically the presentation of information, to suit differing storage patterns. To do so, the XSLT Processor uses output and input transformation stylesheets. For example, in NDS, you store the date of birth according to the month, day, and year. However, suppose the date of birth field in Other Directory reads day, month, and year. In this case, if the DirXML Join Engine received an NDS birth date of 060873, the XSLT Processor would use an output transformation style sheet to convert that format so the birth date reads 080673 for Other Directory.

THE APPLICATION SHIM

After converting the XML-formatted data to an application-ready format, the DirXML Join Engine passes the representation of that data in memory—the DOM—to the Application Shim (commonly known as the DirXML driver). (See Figure 1.)

Like the DirXML Join Engine, the Application Shim uses the DOM to manipulate the XML-formatted data as necessary. Nikols points out that "there isn't any XML parsing necessary to operate on the XML-formatted data," because the Application Shim (and the DirXML Join Engine) use the DOM to manipulate that data.

In fact, little is left for the Application Shim to do. Basically, the Application Shim receives the XML-formatted data, "breaks it up using the DOM, pulls out the information it needs, and then calls the APIs to the application," Nikols explains. For example, if Other Directory is a Netscape LDAP directory, the Application Shim makes LDAP calls to the directory with all of the information in the appropriate Netscape format. If the information comes from Other Directory to NDS, the Application Shim receives the information from the application through the application's APIs, builds an XML document using DOM methods, and then submits the DOM to the DirXML Join Engine. In the end, Nikols summarizes, "the Shim packs and unpacks XML documents and calls native APIs."

FROM THE TOP

The overall DirXML process goes something like this: Suppose you change the Surname of NDS User object Linda from Boyer to Kennard. Further suppose that on the subscriber channel for your Netscape LDAP directory, you have created a filter that indicates that this directory should receive all changes regarding the names of NDS User objects.

DirXML receives the change on Netscape's subscriber channel and uses XML to convert the received information, which it represents in memory as a DOM. Next, DirXML finds an existing Association or creates a new Association between NDS User object Linda and the associated Netscape object, such as LBoyer. DirXML then changes Boyer to Kennard on this object and passes the change via the DOM to the Application Shim.

The Netscape Application Shim breaks up the DOM, extracts the information it needs, and makes the appropriate LDAP calls to the Netscape directory. And that's it. Although the process may sound complicated when described in detail, the process is very fast. How fast? This question is difficult to answer, and Nikols is understandably reluctant to answer it. However, Nikols says, assuming an Association already exists for the object on which there is a change, the process of making the change in Other Directory is as fast as regular event replication.

Linda Kennard works for Niche Associates, which is located in Sandy, Utah. ●

Please visit our advertiser

Tobit Software

<http://www.tobitsoftware.com>

For more information, visit
<http://www.nwconnection.com/advertise.html>.