CHAPTER 1

Introducing DirXML
Do you use groupware applications such as Lotus Notes or GroupWise? Does your company use Human Resource Management or payroll software such as PeopleSoft? In any organization, various databases and applications are used to store diverse, company-related information. These diverse databases are likely disconnected as a corporate whole and isolated within singular departments with divided ownership. This means that within the same organization there are likely multiple databases that contain much of the same information (such as user ids necessary to access various applications) arranged within different data definitions. Each time a new application or database is added to the enterprise, so is a new set of data definitions. The larger the organization, the more significant this issue is.

This creates a problem since for each new system that is added, a good portion of the data that this system needs may be redundant information that may already exist in one form or another in one or more of the other systems already in the environment. Consider the following scenario.

A company uses NetWare 6 as their network operating system and Lotus Notes running on Windows NT Servers for e-mail. NetWare stores user information in the Novell Directory Services (NDS) database while Lotus Notes keep information about employees who use the e-mail system in a Notes-specific database, separate from NDS. That's two separate databases the system administrator has to maintain, yet both databases contain duplicate information: user id, password, and user's first name and last name—in the interest of simplifying access for users, they are often assigned the same user id and password for different applications whenever possible. At some point, the company adds an Oracle database system to track sales and inventory data. Information about the employees that are authorized to access the databases is stored in Oracle's own user database, which is separate from NDS and the Notes database. All three databases contain certain information (such as user id, user's first and last name) that is already present in the other databases. What's more, whenever a new employee is hired, the system administrator has to create the corresponding user id three times, one for each database.

So what happens if the information common to these systems changes? The common data must be changed wherever the data is stored; in this example the information in NDS, the Notes e-mail system, and the Oracle user database must be changed. This update requires multiple operations and in some cases is performed by different system administrators—as it is not uncommon to have the NetWare network administrator separate from the Notes e-mail administrator separate from the Oracle database administrator. All these separate operations add to the potential for error or oversight.
Introducing DirXML

Having an enterprise-wide (information) directory is the solution to the problem of redundant data. Ideally, each application in an enterprise retrieves information from a (single) central directory and stores its shared information in the directory. Unfortunately, few of today’s applications and even fewer legacy applications are “directory-enabled.” And if an application is directory-enabled (meaning it uses the directory for access to authentication information and for storing and accessing data associated with directory objects), the data structure or the directory technology used is not necessary compatible with those used by other applications. This is where Novell’s DirXML technology comes in.

This chapter provides you with information on these topics:

- What is DirXML
- What can DirXML do for your company
- An overview on how DirXML works
- DirXML basics: Terms, concepts and architecture

What Is DirXML

Combining the Novell Directory Services (NDS) eDirectory technology and the capabilities of eXtensible Markup Language (XML), Novell’s DirXML is a technology that enables enterprise-wide management of data, dramatically reducing the cost of managing different databases and directories. In other words, DirXML allows you to connect multiple, disparate directory and database systems into an enterprise system in which each application shares and synchronizes common data with every other system. As a result, because the data is centrally managed, this reduces the cost of managing the various different databases and directories used in a company. In addition, DirXML enables much more accurate and up-to-date data to exist in the enterprise network (as there is a single source from which all systems can obtain their data).

Unless specified otherwise, this book uses the term “directory” to specifically refer to the database that supports a given directory service (such as NDS). It should not be confused with a file system’s directory (folder) structure.
**What DirXML Can Do for You**

In today's information age, data equates to money. However, for the data to be worth anything, it must be accurate and up-to-date data. Therefore, data management is vital for any business. One of the many business challenges that many organizations face today is the extremely daunting task of integrating their different network platforms, directories, databases, and countless applications. It is not uncommon for an organization to have many different “silos of data” created by different business divisions or due to projects initiated by different groups within the same business division (see Figure 1.1). Other times the data silos are created by various applications (such as e-mail and database systems) used by the enterprise.

![Silos of corporate data](image)

**FIGURE 1.1** Many types of (unconnected) databases, applications, and directories exist in a typical company environment.

Consider the following two examples. In the first example, a bank's Savings Accounts division has a database to track its customer records, while the Loans division would have its own separate database for their own (loan) customers. Since chances are that the loan customers also have savings accounts at the bank, both databases would contain redundant records of customers and other related information. Now add Credit Card division's database to this equation, and so forth. Scaling this duplication of information up to include thousands of redundant records and tens or hundreds of databases you easily can appreciate the idea that the ability to properly manage an organization's data quickly becomes very cost prohibitive.
In the second example, a company uses NetWare 6 as their network operating system and Lotus Notes for e-mail. NetWare stores user information in NDS whereas Lotus Notes keeps information about employees who use the e-mail system in a Notes-specific database, separate from NDS. That's two separate databases the system administrator has to maintain yet both databases contain duplicate information: user id, password, and user's first name and last name, just to name a few. The company then added an Oracle database system to track sales and inventory data. Information about employees authorized to access the databases is stored in Oracle's own user database, which is separate from NDS and from the Notes database. All three databases contain certain information (such as user id, user's first and last name) that is already present in the other databases. What's more, whenever a new employee is hired or an employee leaves the company the system administrator has to update the corresponding user record three times, one for each database. This task of accurately updating multiple databases manually and in a timely fashion can easily become daunting as the number of databases involved increases.

To facilitate easier access to different applications for users, network administrators often assign users the same user id and password for different applications whenever possible.

In light of the above two examples the following points raise some key issues (relating to the flow and management of vital information) that a business needs to closely examine:

- What is the people-cost involved in creating new user accounts for accessing various applications within the organization?
- How much time and money are expended in updating and synchronizing a single user record in every information system across the enterprise?
- What are the risks of human error and costs associated with incorrectly or inconsistently entering user information in the various systems within the company?
- How long does it take to remove a user record and access rights from every system in the enterprise, and what aspects of the business are at risk while those access rights remain in force?

How much is having multiple sets of redundant yet possibly inaccurate data and unsynchronized data silos costing your business every year? A simple calculation shows that a company with 10,000 users and 5 independent directories
can expect to spend $22,500 or more per year to keep all five directories up-to-date. On the other hand, using DirXML to synchronize changes among the various directories would cost the same company just $4,500 per year, a savings of 80 percent! Table 1.1 shows how these costs are derived.

<table>
<thead>
<tr>
<th>DIRECTORY MANAGEMENT TASKS</th>
<th>NUMBER OF CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of employees</td>
<td>10,000</td>
</tr>
<tr>
<td>Annual personnel turnover</td>
<td>20%</td>
</tr>
<tr>
<td>Number of directories</td>
<td>5</td>
</tr>
<tr>
<td>Average number of directory record changes</td>
<td>2,500</td>
</tr>
<tr>
<td>(1,000 additions, 1,000 deletions, and 500 modifications)</td>
<td></td>
</tr>
<tr>
<td>Total number of directory record changes (for five directories)</td>
<td>12,500</td>
</tr>
<tr>
<td>Average time to update one directory</td>
<td>5 minutes</td>
</tr>
<tr>
<td>Total time needed to manage (five) directories every year</td>
<td>25,000 minutes (417 hours, or 52 8-hour work days—20% of one person’s time)</td>
</tr>
<tr>
<td>Average administrative cost per hour (assuming a salary of $45,000, working roughly 260 days per year, 8 hours per day)</td>
<td>$21.60</td>
</tr>
<tr>
<td>Administrative cost per change</td>
<td>$1.80</td>
</tr>
<tr>
<td>Total cost of directory changes</td>
<td>$22,500</td>
</tr>
<tr>
<td>Cost of managing all directories via an integrated directory (using a directory synchronization solution such as DirXML)</td>
<td>$4,500</td>
</tr>
</tbody>
</table>

In general, a typical enterprise environment that has more than 5,000 employees will have many more than 50 isolated data stores or directories. Applying the above calculation for a company that does not have automated information management results in a yearly cost of over a quarter of a million dollars. The integration and automation of this same corporation’s different information data stores could reduce its yearly information management costs by 98 percent! The more separate data silos you can integrate, the more cost effective data management becomes.
Using eDirectory as the central data store, DirXML automatically distributes new and updated information bi-directionally across every designated directory and application in the enterprise network (see Figure 1.2). You can even configure DirXML to securely synchronize outside your firewall, to include those databases and directories located on your business partners’ systems. By using a set of business rules defined by you and stored in eDirectory, DirXML enables your customers, business partners, and employees to access consistent and up-to-date information from any connected application or directory.

DirXML can be leveraged as a B2B (Business-to-Business) procurement and fulfillment solution to monitor internal transaction status within an organization and its partners’ Enterprise Resource Planning (ERP) systems. For instance if your customer generates a purchase order, through the combined efforts of DirXML and eDirectory that order “event” can immediately trigger your company’s ERP system to initiate steps to fill the order. As your system responds to the order, associated status and events (i.e., backorder, partial shipments and so on) can likewise be immediately reported to your customer’s ERP system. As a result DirXML enables highly efficient, bi-directional interaction between an organization’s business partners and customers.

DirXML updates all directories on any application for which DirXML-compatible drivers have been created. The current shipping version of DirXML,
1.1 at the time of this writing, has the following drivers that are available “out-of-the-box”:

- Delimited Text (flat file) DirXML driver
- LDAP DirXML driver, supports LDAP v3-compliant directories and servers such as the iPlanet Directory Server, IBM SecureWay Directory, Innosoft Directory Services, and Critical Path InJoin Directory
- Lotus Notes DirXML driver
- Microsoft Active Directory DirXML driver
- Microsoft Exchange DirXML driver
- Microsoft NT Domain DirXML driver
- Novell eDirectory DirXML driver
- Novell GroupWise DirXML driver

Also available for purchase, separate from the DirXML base product, are the following drivers:

- DirXML driver for PeopleSoft, supports PeopleSoft 7.0, 7.5, and 8.1
- DirXML driver for JDBC, supports Oracle 8.x, Microsoft SQL Server 7.x, and IBM DB2 Universal Database (UDB) 7.x
- DirXML driver for SAP HR, supports SAP 4.5B and 4.5C HR systems

These DirXML drivers are sold for a one-time master software download fee and a per-user license fee.

The term “support” in the context of software applications and versions (such as Oracle 8.x) refers to the fact that Novell has only tested the given DirXML driver against the versions listed. Chances are good that the driver will work with other versions of the application but since it hasn’t been tested, it is designated “not supported.”

For up-to-date pricing information about DirXML and additional DirXML drivers visit the following Novell Web sites:

INTRODUCING DIRXML


Should you need a DirXML driver (which are written using C, C++, or Java) for a particular need not supported by one of the existing drivers, Novell offers DirXML deployment training and driver development support so customers can build custom drivers to support to their specific business requirements. Novell also partners with consultants and system integrators who are available to develop custom drivers for you.

For details on these resources, visit the following sites:
- DirXML deployment training: http://www.novell.com/education
- DirXML driver development: http://developer.novell.com

Discussions about coding your own DirXML driver can be found in Chapter 16, “DirXML Driver Development.”

How DirXML Works

DirXML has powerful advantages over previous integration technologies: the use of NDS eDirectory information infrastructure as its data repository, the DirXML association mechanism, and the use of industry standard eXtensible Markup Language (XML) as the data representation language to communicate content from one application to another. Additionally, DirXML has the capability to permit any application to be configured as the authoritative data source for a particular piece of data.

The following sections provide a high-level overview of the workings of DirXML.

An in-depth “under the hood” look of DirXML components can be found in Chapter 6, “Understanding DirXML Components.”

eDirectory

An enterprise directory needs to provide secure storage, management, and access to data in networked environments. Novell’s eDirectory is a full-service
directory that provides for flexible and extensible discovery, rich security tools, an extremely scalable storage engine, and the ability to manage relationships—whether internal or external to your organization.

A full-service directory, as defined by Novell, is a general-purpose database that manages and provides the following features:

- **Discovery.** Discovery enables you to browse, search, and retrieve specific information from the directory. For example, you should be able to search for specific object types such as Users and Groups, or search for their specific properties such as a user name and telephone number of a User object. Discovery must support the industry standard access methods, for example the Lightweight Directory Access Protocol (LDAP).

- **Security.** Security controls access to all the information that is stored in the directory. This means you should be able to establish access rules and grant rights to the users for the information in the directory. In addition you should be able to control the flow of information within your company across networks of partners, and even to your customers.

- **Storage.** Storage is the database structure for the directory. It should give you the ability to save information in the database for future reference as a minimum. The database should be indexed, cached, and guarded from data corruption by a transaction system. Besides merely storing data, the database should enable you to automatically control the type of data by applying classifications to the data structures. The classifications should be flexible and extensible to provide future representations in the database. The database should be capable of being split into physical pieces and distributed or placed on multiple servers regardless of their operating system. These features would enable you to keep a portion of the relevant data close to the users and resources that need them and make multiple copies of the data for redundancy.

- **Relationships.** Relationship provides you with the ability to build associations between the people, network devices, network applications, and information on the network. For example instead of storing the user’s desktop configuration profile information on the local machine, it can be stored in the directory.

This would result in the profile data becoming global within the scope of the directory. Consequently the user will be able to access the profile information from anywhere on the network. This means the user can receive the same desktop configuration profile regardless of where he or she
attaches and logs in to the network. In addition, the access to the profile information can be tightly controlled (as per the Security feature discussion above). The only user that can gain access is the one with proper credentials. Thus, the integrity of the profile data is protected and secure, and the user can access it globally, and easily.

To date, eDirectory is the only enterprise-wide full directory service that has a proven and documented track record of supporting networks of all sizes, ranging from a simple five-user setup to networks consisting of millions of users. Of particular importance is the NetWare-independence and native LDAP support of eDirectory:

- **NetWare-independence.** Modern enterprise networks often consist of servers running different operating systems. Therefore a true enterprise directory must run on various platforms. Originally for NetWare servers only, eDirectory is now a truly cross-platform global directory that operates natively on NetWare, Windows NT/2000, Solaris, and Linux.

You can download evaluation copies of eDirectory for the various platforms from [http://www.novell.com/download](http://www.novell.com/download).

- **LDAP support.** NDS eDirectory features a native implementation of LDAP v3 providing fast searches, auxiliary classes, referrals, and controls. Such native implementation provides an open structure for applications and developers and simple integration with applications that are written to this Internet standard. To this end Novell has released an LDAP Library for C Software Developer Kit (SDK) that makes developing to LDAP and eDirectory easy and practical. This LDAP Library for C from Novell is based on the OpenLDAP Foundation libraries and is derived under and subject to the terms of the OpenLDAP Public License, version 2.0.1.

For more information about the LDAP Libraries for C from Novell, visit [http://sourceplus.novell.com/c_lib.html](http://sourceplus.novell.com/c_lib.html).

Using the event system of eDirectory, data changes are easily detected and consequently synchronized to other data stores.

**DirXML Association**

Older integration technologies, including meta-directory solutions, generally require that every application or directory that is to be linked together in a
common system maintain a common key that uniquely identifies an object across all systems. This generally requires that most applications be extended in some fashion in order to provide this support. If extension is not feasible then the application must either use an existing field in an unsupported fashion (such as storing this common key value in the memo field of a database) or the application cannot be integrated into the enterprise system.

**Meta-directories are applications that often work alongside directories. They provide facilities to manage information about people, resources, and policies that is spread, and possibly duplicated, across more than one directory or database. In essence, a meta-directory appears to applications and end-users as a single seamless service, but really constitutes a central broker that directs queries to the appropriate distributed server. For these reasons, DirXML can be considered a meta-directory.**

DirXML removes this limitation by the use of *DirXML Association*. The DirXML Association is a list of identifiers contained by an object in the central directory. Each identifier contains two values: A reference to a connected application, and a value that identifies the object or record in the connected application that is associated with the directory object by the predefined identifier used in that connected application (see Figure 1.3). The list-nature of the DirXML Association attribute means that a directory object can be associated with a corresponding object in any number of connected applications—much like a lookup table.

**DirXML does not pose a limitation on the types of objects that can be associated. For example you can associate a User object in NDS with a database record in Oracle.**

The DirXML driver provides a unique key value for each application object, and the DirXML engine manages the storage of those key values in NDS. A driver's responsibility with respect to associations is to provide a unique key value for each object and to notify DirXML whenever something happens that affects that unique key. In practice this means the driver will:

- Provide the unique key value for the object whenever communicating any information about the object to DirXML (events and queries).
- Report to DirXML whenever the unique key value of a relevant object changes, by using the `<modify-association>` XML tag.
FIGURE 1.3 The value in the DirXML Association is unique to the application and is supplied by the application. No modification of the application is required, and no foreign key values are introduced into the application.

- Report to DirXML the unique key value of an object created as a result of processing `<add>` in the SubscriptionShim interface using `<add-association>` in the result document returned from the add processing.
- Optionally tells DirXML to forget any key that is no longer valid and for which the driver has reason to believe that DirXML is still holding using the `<remove-association>` tag. The most common occurrence of this is when DirXML sends a command using a key value that no longer exists in the application.

You will read more about XML and the various terms and concepts used by DirXML in later chapters.

**XML Advantage**

DirXML uses the industry standard eXtensible Markup Language (XML) as the data representation language to communicate content from one application to another, while preserving the native language of the different applications. There are two main advantages of using XML for data synchronization. Firstly XML is a meta-language, which can be used to describe other languages (or information,
such as data). Secondly XML has the ability to prevent data corruption due to badly formatted XML “instructions.”

If you are familiar with Network File System (NFS), one of the protocols in the TCP/IP protocol suite commonly used in UNIX environment for seamlessly sharing files between different hosts, you can draw some parallel between DirXML’s use of XML and NFS’ use of XDR (eXternal Data Representation) protocol.

XDR defines a format by which various data types (integers, floating-point numbers, characters, and so on) can be represented in a uniform manner across the network regardless of the operating system and machine platform. Similarly XML is a World Wide Web Consortium (http://www.w3.org) specification that provides a method for representing both data and the context of application data that is unambiguous and avoids common pitfalls such as lack of extensibility, lack of support for internationalization or localization, and is platform-independent.

The XML specification provides a method to create vocabularies that are used to describe various classes of data. For example, MathML (Mathematical Markup Language; see http://www.w3.org/TR/REC-MathML) is an XML vocabulary used to describe mathematical data; and the major goal of VoiceXML (Voice eXtensible Markup Language; see http://www.w3.org/TR/voicexml/) is to bring the advantages of Web-based development and content delivery to interactive voice response applications. Similarly Novell’s DirXML specification defines a vocabulary used to describe directory and application data, and directory and application change events.

Novell has submitted much of its significant directory-related research and intellectual property related to DirXML to the DSML (Directory Service Markup Language; see http://www.dsml.org) working group to be included as part of the DSML 2.0 specification. This group is part of the Organization for the Advancement of Structured Information Standards (OASIS; see http://www.oasis-open.org), an international consortium that creates interoperable specifications based on standards such as XML.

XML files (generally referred to as XML documents) are text files. The files are intended to be read directly by applications and are not meant to be read by
humans. However, they are in text file format to allow programmers to more easily debug applications. In cases where required, programmers can use a simple text editor to quickly fix a broken XML file. But the rules for XML files are much stricter than for HTML: any error, even a small one, such as a forgotten end-tag or an attribute value without quotes, makes the file unusable.

Tags

Like HTML (HyperText Markup Language), XML makes use of “tags”—words enclosed by angle brackets (“<” and “>”)—and attributes (of the form name = “value”). Although tags are used to describe the data elements, being a meta-language XML uses arbitrary tags (i.e., not predefined, unlike those used in HTML). Any label can be used. The power of XML is its ability to define tags and then use the tags to organize, define, and structure the data.

You can easily use XML to describe a dog. To do this you can create a tagged data element called “dog,” and within that element describe sub-elements that enrich the description of the dog. The following code snippet shows a possible set of defined attributes for a dog:

```xml
<dog>
  <breed>Retriever (Golden)</breed>
  <name>Tarwathie's Golden Mercedes</name>
  <registrationNum>JJ592381</registrationNum>
  <sex>Female</sex>
  <birthday>16 May, 1999</birthday>
  <color>Golden</color>
  <weight>75 lbs.</weight>
</dog>
```

While HTML predefines what each tag and attribute means (and often how the text between them will be displayed in a browser) XML uses the tags only to delimit pieces of data and leaves the interpretation of the data completely to the application that reads it. In other words, if you see “<P>” in an XML file, it does not necessarily mean it is a paragraph, as is the case for HTML. Depending on the context, it may be a price, a parameter, a person, or whatever the creator of that tag wants it to represent (while the name of this object represented by “<P>” does not need to start with the letter “P”).
In the HTML world you are often allowed to omit the end-tag for brevity. For example, a paragraph “should” be surrounded by <P> and </P> but they are not absolutely required. On the other hand it is specified in the official XML specification that:

“All XML documents must be well-formed. A well-formed XML document requires start-tags and end-tags on every normal element, and any EMPTY elements must be made unambiguous, either by using normal start-tags and end-tags, or by affixing a slash to the start-tag before the closing “>” (such as <BR/>) as a sign that there will be no end-tag.”

Should an application detect any violation of well-formedness or validity it is not allowed to second-guess the creator of a broken XML file but must stop processing at the offending location in the XML document and issue an error.

The structured approach that XML brings to data markup means that data of arbitrary complexity can be easily and accurately described. In addition, the structure of XML means that applications can much more easily determine the context of data items with much less possibility of error. The fact that XML has been widely adopted in the industry means that there are many powerful tools for dealing with XML. There are industry-standard specifications for representing XML data in programming languages. There are industry-standard languages for processing XML. The fact that DirXML makes use of these industry standards means that DirXML development is much easier than for older technologies that required the use of proprietary data representations and processing languages.

**Authoritative Data Sources**

It is rare to use DirXML to simply connect a single application with the enterprise directory. DirXML is more commonly used to connect multiple disparate systems together into an enterprise system in which each application shares some common data with every other system. Sharing common data among applications is good, but what happens if multiple applications change the same data? Which application’s change should be considered “correct”? You can apply the concept of an authoritative data source.

If one application should “own” a particular piece of data that no other application should change, DirXML can designate the owner of this data as the authoritative data source. That means only this application may update the item, and all other applications, including the enterprise directory, are mere consumers of that data. The granularity of this configuration allows objects to consist of data from multiple authoritative sources. For example the PBX system is configured to
be the authoritative data source for an employee’s office telephone and FAX number. Changes to these two fields initiated from the PBX application will be accepted. If changes to these two fields are initiated from the e-mail system, for instance, the changes will be blocked.

**DirXML Architecture**

The goal of the DirXML technology is to provide easy and accurate movement of data between eDirectory and any application, directory, or database that requires directory information. To accomplish this DirXML uses a well-defined interface that takes NDS data and events and translates them into XML format. This interface allows the data to flow in and out of NDS as shown in Figure 1.4.

![Figure 1.4](image)

**FIGURE 1.4** DirXML is composed of several components that operate together to transfer information and subsequent changes from eDirectory to the DirXML driver (connected to an external data store) and vice versa.

The data that flows between NDS and the target application is managed and processed by two “channels”: the Subscriber channel and the Publisher channel. These channels are the means of linking event systems (or changes in the data) of eDirectory and the target application together so that data flow is determined by its dynamic characteristics. The Subscriber and Publisher channels act as filters for
deciding which events are received from NDS and which are published to NDS. These filters also determine which object classes and attributes are accepted into the channels. They are configured through NDS, giving you control of how the data flows throughout the enterprise.

The Subscriber and Publisher channels communicate with the target application through an Application Shim. An Application Shim is an application-specific piece of code (provided as part of the driver, such as the Delimited Text DirXML driver), which knows how to communicate with the target application. The shim passes data to the application once the data has been processed by DirXML and receives data from the application that is translated before it goes to eDirectory.

The DirXML engine is a collection of interfaces and data manipulation technologies that allow disparate data systems to connect and share data. The DirXML engine provides an interface to NDS that exposes NDS data and NDS events using an XML format. The DirXML engine employs a rules processor and a data transformation (XSL processor) engine to manipulate the data as it flows between two systems. It also provides an interface for application-specific programs to attach through the DirXML engine to capture the data streams that flow from NDS.

XML Stylesheet Language (XSL) is a language for expressing style sheets. An XSL style sheet is a file that describes how to display an XML document of a given type. XSL also adds a transformation language for XML documents: XSLT. Originally intended to perform complex styling operations like the generation of tables of contents and indexes, XSLT is now used as a general-purpose XML processing language.

You will read more about XML style sheets, XSL and XSLT in Chapter 5, “XML Style Sheets.”

DirXML Concepts

When DirXML operates on top of eDirectory, change events are passed between NDS and the DirXML engine. NDS manages the drivers that are loaded and configured to consume these events. NDS also provides some basic management services related to the events themselves so those events are always stored until they’ve been successfully consumed.
The DirXML engine first converts the NDS events into XML documents (in DOM format, see Note below). Rules are then applied to the XML document, and data transformations are performed to convert the data into the target application's native data format. Once the XML document has been completely processed the event data is handed to an Application Shim that delivers the formatted data to the target application. If the application can natively publish and consume XML then the shim does nothing but pass XML through.

**Document Object Model (DOM)** is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. In the DirXML product, the DOM is a structure in memory that XML manipulates. It is not visible to humans in any way.

You can visit the W3C DOM Working Group's Frequently Asked Questions (FAQ) page at http://www.w3.org/DOM/faq for more information.

Each DirXML-related component except the DirXML Engine is represented as a DirXML object in the NDS tree. The eDirectory schema will be extended to support these objects during your DirXML installation. The new object types are:

- **DirXML-DriverSet.** A container object that defines a collection of drivers.
- **DirXML-Driver.** A container object that defines all the components of a driver including rules, style sheets, and the Application Shim.
- **DirXML-Publisher** and **DirXML-Subscriber.** These are container objects corresponding to the Publisher and the Subscriber channels of each DirXML driver. These objects contain DirXML-Rule objects.
- **DirXML-Rule.** A leaf object containing an XML document that defines a rule that will be applied to the NDS event stream as it flows through the DirXML engine. Examples of these rules are Event Rule, Schema Mapping Rule, Matching Rule, Create Rule, and Placement Rule.
- **DirXML-StyleSheet.** A leaf object containing an XSLT document that defines a transformation of the NDS XML code into some other data format, or other complex data manipulations not possible in a standard DirXML-Rule.
Although not specifically represented as an object in NDS, the Application Shim is worth noting as it completes the driver set/driver definition. The Application Shim is an executable piece of code (.dll or .jar file, depending on the OS the application runs on) that the DirXML Engine uses to interface with the target application. The shim is actually stored as an attribute of the DirXML-Driver object in NDS.

Rules and style sheets are discussed in more detail in Chapter 6, “Understanding DirXML Components.”

DirXML-DriverSet Object

A DirXML-DriverSet object is a container in the NDS tree that holds DirXML driver objects. You can create as many driver sets as you like but only one driver set may be active on an NDS tree at a time. As a result, all active drivers have to be grouped in the same driver set. It is not necessary to activate all the drivers in a driver set on every server that is using that driver set. The current implementation of DirXML requires the DirXML-DriverSet object to exist in a writeable (i.e., Master or Read/Write) replica on any server that is using it. Therefore it is recommended that you partition the tree at the DirXML-DriverSet container to make replication easier.

You start and stop drivers via the Properties dialog of the DirXML-DriverSet object.

DirXML-Driver Object

A DirXML-Driver object represents an application being synchronized with eDirectory. The following are the components and configuration information found in the DirXML-Driver object:

- A DirXML-Subscriber object.
- A DirXML-Publisher object.
- One or more DirXML-Rule objects that are referenced by the driver, DirXML-Subscriber and DirXML-Publisher objects.
- An executable driver shim that is referenced by the driver object.
- Shim-specific parameters that are configured by the administrator.
- A password for the driver object (stored in NDS), which may be used by the shim to authenticate a remote part of the shim.
CHAPTER 1
INTRODUCING DIRXML

- Authentication parameters used to connect and authenticate to the supported directory or application.
- The startup option for the driver that includes these options:
  - Disabled. The driver will not run.
  - Manual. The driver must be started manually through ConsoleOne.
  - Auto start. The driver will start automatically when eDirectory starts.
- A reference to a Schema Mapping Rule.
- An XML representation of the supported application or directory’s schema. This is typically obtained automatically from the application or directory through the application shim.

The application driver must be granted sufficient NDS rights to any object it reads or writes. This NDS access right is enforced through the DirXML-Driver object. You can make the assignments by making the driver object a trustee of the NDS objects the driver will access.

**It is generally more efficient to grant the DirXML-Driver object NDS rights to the container in which the target objects reside than to each individual object: one single assignment versus many.**

**DirXML-Subscriber Object**
A DirXML-Subscriber object contains filter information for the Subscriber channel of the DirXML driver. This includes the following:

- A list of object classes and associated attributes to be synchronized. This is referred to as the Subscriber filter.
- References to DirXML-Rule objects that contain the Create Rule, the Placement Rule, Matching Rule, and style sheets for the Subscriber channel.

**DirXML-Publisher Object**
A DirXML-Publisher object contains filter information for the Publisher channel of the DirXML driver. This includes the following:

- A list of object classes and associated attributes to be synchronized. This is referred to as the Publisher filter.

**Application Shims subscribe to the DirXML Engine to receive NDS event data.**
References to DirXML-Rule objects that contain the Create Rule, the Placement Rule, Matching Rule, and style sheets for the Publisher channel.

Data changes in the connected application are published by Application Shims to the DirXML Engine for consumption by NDS.

DirXML-Rule Object

There are two classes of rules within DirXML. The administrator configures the way each driver synchronizes data and events through a series of rules. These rules fall into the class of rules called Object Mapping Rules. There are three types of rules that may be associated with the Subscriber and Publisher channels of a DirXML driver:

- Create Rule
- Matching Rule
- Placement Rule

Typically, each DirXML driver will come with a set of pre-configured rules that can be modified to suit the particular installation. However all the rules are optional.

These Object Mapping Rules are applied to the XML documents that are used for communication between DirXML and the driver shim. The result of applying the rules may be a modification of the document, or a rejection of the document. For example, if a Create Rule specifies that a user object must have a value for the Given Name attribute, any attempt to create a User object without a Given Name value will be rejected.

Another class of rules defines data transformations of the NDS event stream. These rules are called Transformational Rules. This class of rules includes the following:

- Schema Mapping Rule
- Event Transformation Rule
- Rules that govern the application of XSLT style sheets for data transformation

To provide for flexibility, DirXML 1.1 allows you to “chain” different rules together. Instead of needing to duplicate the simple logic of a rule in a style sheet, you can keep the simple logic in the rule and place the complex logic in a style sheet that you chain to the rule. The style sheet
you chain will then be called directly after the rule. In addition you can chain rules of the same type together. More information on setting up Rule Chaining is presented in Chapter 6, “Understanding DirXML Components.”

A DirXML rule is an XML document that defines a set of actions or restrictions to be applied to the NDS data stream as it transverses the DirXML Engine. Rules are stored as DirXML-Rule objects in NDS and are created as leaf objects in the DirXML-Driver, DirXML-Publisher, or DirXML-Subscriber (container) objects.

**DirXML-StyleSheet Object**

Each DirXML driver can employ one or more style sheets that describe the handling and transformation of information passed between NDS and an application. A style sheet is an XSLT document that defines a transformation of the NDS XML code into some other data format, or other complex data manipulations not possible in a standard DirXML-Rule. Style sheets are stored as DirXML-StyleSheet objects in NDS and are created as leaf objects in the DirXML-Driver, DirXML-Publisher, or DirXML-Subscriber (container) objects.

**Application Shim**

The Application Shim (also referred to as driver shim) serves as a conduit for information between the application, directory, or database and NDS. The shim is written in Java, C, or C++. The communication between the DirXML Engine and the driver shim is in the form of XML documents that describe events, queries, and results. The following object events are supported by the shim:

- Add (create)
- Modify
- Delete
- Rename
- Move

In addition, the shim must provide a query capability so that DirXML may query the synchronized application, directory, or database.

When an event occurs in NDS that causes an action in the synchronized application or directory, DirXML creates an XML document that describes the NDS event and submits it through the Subscriber channel to the driver shim for processing. When an event occurs in the synchronized application, directory, or database, DirXML creates a corresponding XML document and submits it through the Driver or Publisher channel to the driver shim for processing.
database the driver shim generates an XML document that describes the application event and then submits the XML document to the DirXML Engine for processing through the Publisher channel. After processing the event through any applicable rules, DirXML instructs NDS to take the appropriate action.

## What's New in DirXML 1.1

DirXML 1.0 first started shipping in late September of 2000. At the time of writing the shipping version is DirXML 1.1 and has the following new features:

- **Remote Loader.** The Remote Loader allows you to load and run DirXML drivers on remote systems not running with eDirectory. For example, the DirXML Driver for Active Directory must run on Windows 2000. Using the Remote Loader, you can synchronize data between NDS and Active Directory *without* requiring eDirectory and DirXML on the Windows 2000 machine; you only need to run the DirXML Driver for Active Directory on the Windows 2000 Server.

- **XML Editor.** To improve the readability of the XML code, the XML Editor in ConsoleOne now inserts a tab into the XML text instead of moving focus to the next control when you press the Tab key on the keyboard while working in the XML Edit Field.

- **Templates Files System Support.** This is one of the most requested features—if a user is added on the Publisher channel with the Home Directory attribute populated, DirXML will automatically create the home directory. Similarly, if a user is created from a template with a home directory specified, DirXML will automatically generate the Home Directory name for the User object, and any file system rights and volume space restrictions from the template are applied.

  This feature may be used on any platform, but currently it only manages NetWare file systems. Appropriate rights must be granted to the Driver object for the feature to work.

- **Rule and Style Sheet Enhancements:**
  - Allows for separate rules (Command Transformation Style Sheet) on each of the Subscriber and Publisher channels.
  - Rule Chaining allows each rule to be implemented by a series of Rule objects.
• Query enhancements allow you to query from Input and Output Transformations and the Schema Mapping Rule. Queries issued from the style sheet no longer need to be enclosed in <nds> and <input> tags.

• A channel writeback feature permits sending of commands from a rule on either the Subscriber or Publisher channel to both eDirectory and the application.

➤ **Automatic Auxiliary Classes.** On the Publisher channel, auxiliary classes are now automatically added to objects as needed. A Command Transformation rule may be used to modify or disable this feature. Automatic auxiliary classes are not supported on the Subscriber channel.

➤ **Remote Contact for Publisher Channel Moves.** If you perform object moves on the Publisher channel, DirXML 1.1 will remotely contact the Master replica, if necessary.

➤ **Dynamic Class Loader.** New .jar files are automatically detected each time a driver is started. However, if you update a .jar file, you still need to restart the dhost or ndsd process for the change to be recognized.

➤ **Modify Password Support.** The modify-password command is now supported on the Publisher channel. However, this command can only be used to synchronize passwords from an application that stores passwords as clear text. Support for modify-password varies per driver.

➤ **Advanced Logging.** An advanced logging feature has been added that allows you to determine the health of your DirXML drivers and identify any errors and warnings from the DirXML Engine and drivers.

For new features and enhancements specific to the drivers, refer to the appropriate driver chapters in this book.

**Summary**

In this chapter, you learned about the following topics:

➤ What is DirXML
➤ What DirXML can do for your company
➤ An overview on how DirXML works
➤ DirXML basic terms, concepts, and architecture
You learned Novell's DirXML is a meta-directory technology that enables enterprise management of data, which has the goal of dramatically reducing the costs of managing different databases and directories. It covers these requirements in a way that makes the system easy to extend and customize. Because the data is expressed using industry-standard XML, the data and events can be easily shared between disparate systems and modified to meet the individual needs of each. Furthermore, unlike other meta-directory implementations, DirXML doesn't enforce a hierarchy, or a common naming scheme—DirXML allows for disparate schemas, and even completely different data representations.