By now, everyone should know that XML is all about data. Unfortunately, that’s about all most people know about XML. Depending on whom you talk to, XML is projected to be the framework for replacing all software currently in existence or is seen as an interesting technology that should be watched over the next year or two.
The future of XML is certainly debatable, but it’s clear XML is a hot new standards-based technology for defining the interchange of data. But questions arise:

- Aren’t there already standards-based technologies for accessing data, such as Open Database Connectivity (ODBC)?
- Will XML replace ActiveX Data Objects/Object Linking and Embedding (ADO/OLE) databases?
- Do I use XML in conjunction with or instead of Java Database Connectivity (JDBC)?

This article provides a clear understanding of how XML fits with the existing data access standards such as ODBC, JDBC, and ADO/OLE. Additionally, it explores the components that can be used to build an XML-enabled Business-to-Business (B2B) infrastructure.

The Data Access Standards

Let’s review the existing standards-based data access specifications: ODBC, JDBC, and ADO/OLE.

**ODBC** is an Application Programming Interface (API) for accessing tabular data using Structured Query Language (SQL). It’s based on Call-Level Interface (CLI) specifications for database APIs from the X/Open SQL Access Group, the International Standards Organization (ISO) and International Electrotechnical Commission (IEC). ODBC is a mature, cross-platform solution that is widely accepted in the industry and is both robust and flexible. There are hundreds of commercial applications that can use ODBC for accessing data, including the Microsoft Office suite, Lotus SmartSuite, and several Web servers such as Allaire’s Cold Fusion.

**JDBC**, like ODBC, is an API that’s based on specifications from X/Open and ISO/IEC. It accesses tabular data using SQL. JDBC is tailored specifically for the Java programming language and is a natural fit for applications written in Java that access data. Unlike ODBC, JDBC is object-oriented in nature. JDBC also provides some capabilities that fit well into the Internet paradigm, such as providing disconnected access to data. That is, a JDBC thin client can request and receive a set of records over the Internet, make updates to the data from the thin client, and return the data back across the Internet for resynchronization with the database server. Almost all commercial Java-based applications that access data do so using JDBC.

Microsoft ActiveX Data Objects (ADO) is a high-level, object-oriented API that applications can use to access all types of data on the Microsoft Windows platform. ADO provides an interface to OLE, a low-level API that provides the underlying access to back-end database servers. **ADO/OLE** does not require SQL to access data. Instead, ADO/OLE defines interface definitions for a variety of data types, including tabular, non-structured, semi-structured, and Online Analytical Processing (OLAP) data types. Like JDBC, ADO/OLE provides some Internet-friendly capabilities, including read/write access to disconnected data and the ability to translate record sets into Internet-ready formats, including XML! ADO/OLE can be used to access data from Web servers, such as Microsoft’s IIS, and from any application built using tools like Microsoft’s Visual Basic.

### How Does XML Fit?

XML was built on a set of guidelines that are parallel with the best attributes of all these technologies. Namely, XML was built to be platform-independent and work over the Internet with a wide variety of applications. But, XML is not an API like the other data access standards discussed above. XML, by definition, is a markup specification language. You must write a program to do something with an XML document because a document doesn’t do anything inherently.

The Worldwide Web Consortium (W3C) architecture domain for XML (http://www.w3.org/xml) shows that XML is not a single technology; it’s a family of technologies. Within this ever-growing family are different XML APIs, including:

- **Document Object Model (DOM)**, which was produced from a W3C working group
- **Simple API for XML (SAX)**, which was developed from discussions that originated on the XML-DEV mailing list.

These APIs are used to access and manipulate XML documents, not to interface with back-end databases. Clearly, the greatest value is gained by using XML with live data. Corporate data will continue to originate from database management systems and packaged applications, not from an XML document.

Several database vendors have discussed plans to provide ways to use XML to interface with their database engines. These solutions, while built on top of various XML standards, can only provide access in a proprietary way—at least until these vendors agree on a common solution. This is an area to watch, but no two vendors are currently using the same approach. The only data...
access methods now based on industry standards are ODBC, JDBC, and ADO/OLE. If using standards is important to your organization, leveraging XML with these existing data access standards may be the best approach to building applications.

Data Access Standards and XML

Let's examine how the benefits of XML can be used to provide new and robust capabilities for businesses. Leveraging the Internet is at the top of the list for most corporate IT departments. The meaning of this varies by business, but clearly, leveraging the Internet involves exchanging data over the Web. As the Internet grows, more data will be exchanged with consumers and other businesses. For example, a consumer ordering business supplies online might trigger a low-inventory event in the wholesaler's online commerce system or ERP system. This event, in turn, could trigger an immediate re-order of the item from the wholesaler to a supplier. Before the order was actually placed from the wholesaler, the supplier's inventory levels could be checked to ensure available stock. If the item isn't in stock, the wholesaler could choose an alternate supplier.

Traditionally, the wholesaler would have chosen to provide an online catalog of merchandise in Hypertext Markup Language (HTML) format over the Web. Using XML, the wholesaler could now provide their online catalog in XML format. XML provides semantics to the catalog information that let the consumer do business with the wholesaler in many ways. For example, large-volume consumers could now use scripts or programs to query the wholesaler's XML catalog and, similarly, could place orders more efficiently. If the catalog were in HTML format, the catalog information would have no context. Writing scripts or programs to interact with the catalog pages would be difficult. Additionally, scripting to HTML catalogs would require maintenance every time the wholesaler changed the catalog.

There are several ways to expose data as XML. Let's examine how to do it using existing data access standards. Most likely, the wholesaler's catalog information is stored natively in a relational database inside the wholesaler's Internet firewall. Through either the JDBC or ADO/OLE APIs, the wholesaler could read the catalog information from the relational database and submit that information to the Web server as an XML document. Using JDBC, the catalog information would be requested from the database through an SQL SELECT command (for example: SELECT ITEM_NO, PRICE, DESCRIPTION, PHOTO FROM ALL_STOCK ORDER BY 3). The rowset interface could then be used to retrieve the data and to save the data as an XML document, which could be exposed to consumers on the Web. Similarly, using ADO, the resulting data could be sent directly to an Active Server Pages (ASP) request object. The ADO code fragment in Figure 1 demonstrates how simple it can be to expose data to the Web using XML.

Once the wholesaler has exposed the catalog on the Web as an XML document, the catalog can be viewed using an XML-enabled browser. More important, customers can interact with the wholesaler more efficiently by using automated application programs. These programs, or scripts, could interact with the catalog as if they had direct access to the wholesaler's relational database.

Software components that implement a data access API for a specific data format are called drivers. Today, there are drivers available to XML documents for the ODBC, JDBC, and ADO/OLE data access standards. A consumer application could "connect" to a Universal Resource Locator (URL) that specified the wholesaler's XML catalog using an XML driver. The application could then use the catalog data in any way available through the data access API. For example, using an XML ODBC driver, an automated customer application could be built to check the price and availability of any office item from a pool of three different suppliers. Based on business rules, a supplier could be chosen and an order could be placed to the supplier by posting an XML message to the wholesaler's Website. Placing the order could be accomplished, using SQL, with a simple "INSERT" statement.

We can see from this example that the combination of standards-based APIs and XML provides a powerful B2B framework. XML is a hot new standards-based technology for defining data interchange. It's similar to other standards-based technologies, such as ODBC or JDBC, in that it uses a common programming interface for accessing and manipulating data. It's different in that the XML programming interfaces are somewhat immature — with weaknesses in the areas of security, query languages, updating, and access to back-end data stores. XML extends the typical data access API, however, by adding two key features: Web enablement and a common data format. Using XML with the existing data access standards leverages the best of both worlds by providing a powerful nucleus for building Web-enabled business applications. 

These APIs are used to access and manipulate XML documents, not to interface with back-end databases. Clearly, the greatest value is gained by using XML with live data.

PHOTO FROM ALL_STOCK ORDER BY 3)

About the Author

John Goodson is vice president of research and development at MERANT. He has spent his entire career focusing on databases and data access APIs. E-mail: John.Goodson@merant.com.