CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

FEATURE ARTICLE

Edward Steinfeld

Embedded XML

Make Your Customer's IT Department Happy

The popularity of XML is growing. As a result, engineers must work closely with IT departments to meet data management requirements. The task can be difficult, but there are ways you can help enhance data flow without having to customize each device you design.

Instrument designers seldom talk to IT departments. But, this will change with the increased use of extensible markup language (XML) coded data. Most IT departments have decided to implement XML throughout their companies, which means that XML must extend to the requests sent to instruments and the data generated by them.

Consequently, instrument manufacturers must start polling their potential customers and speaking to IT department database designers. In many cases, database designers have already defined a schema for their XML coded data. A manufacturer cannot economically customize every instrument, but it's possible to devise a common set of XML tags that are easily mapped to the existing database schema. This collaboration must broaden to competitors and other manufacturers supplying instrumentation to the same industry. To date, only a few industries have created industry-wide XML standards like this.

The information coded with XML doesn't have to be data; it can include actions and related parameters similar to remote procedure calls (RPC). Now, IT departments have the ability to control and manage devices on the plant floor.

XML BASICS

XML provides a processor-independent way of encoding data for interchange between diverse systems. Like hypertext markup language (HTML), XML is derived from standard generalized markup language (SGML); however, XML is simplified for easier machine parsing. Unlike HTML, XML has no predefined meanings associated with its elements or tags. Instead, it's a set of rules (a syntax) for constructing tag-delimited information. Individual XML documents can use different element definitions to encode information from dissimilar data environments. A set of element definitions to encode data from a particular data environment is called a "schema."

Different schemas or element definitions that use XML syntax are being developed for diverse application environments. Some of these schemas include vocabularies for chemical engineering, vector graphics, electronic invoicing, weather information, and spreadsheet formulas. A specific XML schema (tag set definition) can be defined by one organization and published for others. For example, Microsoft has defined a set of XML tags for exchanging spreadsheet and word processing documents with the Office 2000 and Office XP product suites. Industry standards groups sometimes get together to define ele-

Incorrect (acceptable as HTML)	Correct (required by XML)
<tag>data	<tag>data</tag>
<tag1><tag2>data</tag1></tag2>	<tag1><tag2>data</tag2></tag1>
<tag1 arg1=abc>data</tag1>	<tag1 arg1="abc">data</tag1>

Table 1—*It's important to note the differences between XML and HTML. When comparing the strict XML formatting to HTML, remember that XML is case sensitive and attributes within tags require quotation marks.*

ment sets. Vector graphics and chemical engineering groups are currently developing XML schema standards.

Two functions, or programs, are required to convert a schema-defined document into internal data and vice versa. A parser is a program that reads an XML document and provides access to the data inside the XML tags. On the other hand, a framer is a program that takes internal data and formats it into an XML document.

HOSPITAL DATA FLOW

From the time a doctor decides to test a patient's blood to the time that data is actually provided, the requests for and results concerning the tests pass through a number of steps. Telling the appropriate instruments what tests to perform is one of the important steps in the process.

After a set of tests is chosen, the doctor either enters the requests into a computer or has a nurse do so (see Figure 1). The requests are XML framed and sent to the hospital's IT computer. From there, the requests are sent to the nurse's station on the floor where the patient is located. A pick-up list is printed so the nurses know which patients are having tests done. In addition, they're provided with a bar-coded labels to attach to the vials of blood.

The IT computer also sends the requests to the appropriate laboratories. The pathology laboratory receives the information for blood tests. The data that's sent to the lab includes patient identification material and specific test information.

The IT computer does not specify which instruments will perform the tests, and the same information is sent to all of the instruments. Most embedded XML parsers will ignore data framed by tags that are not identified for them. On the other hand, they'll accept all of the data within the tags they understand. This means the same set of data records can be sent to every test instrument in the pathology lab (see Listing 1). The blood chemistry tester will accept all tests enclosed by the `ChemTest` tag (i.e., blood urea nitrogen, or BUN test), but will ignore test requests enclosed by the `CellTest` tag. The blood cell analyzer will accept both the `leukocyte_count` (i.e., total white blood cell count test) and `diff` (i.e., the percentage difference of each type of white cell) test requests.

All instruments will accept the patient identification data. In the meantime, the nurse collects the fluids and carries them to the lab. After the tests are completed, the results and patient identification information are sent to the hospital's IT computer. A copy of the results is delivered to the nurse's station to be printed or displayed. Another copy of the results is forwarded to the requesting doctor's computer. Finally, a copy of the tests is sent to the billing and insurance department's computer.

Generalized XML parser/framers are used in the computers for the doctor nurse's station, billing department, and the IT department. Special embedded XML parser/framers are used in the testing instruments.

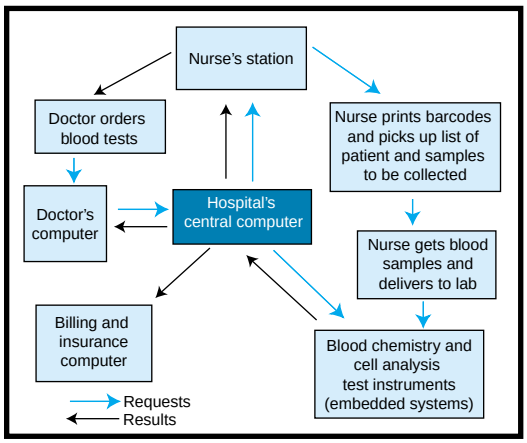

Figure 1—*Something as simple as a routine blood test involves a lot of communication between numerous people and devices. In this example, it's XML framed data that's flowing from doctors to instruments and back.*

XML/HTML COMPARISON

As I explained earlier, HTML and XML were derived from the tag description language SGML. There are differences, though. Both SGML and HTML describe the format of the data; alternatively, XML describes the content of the data. Unlike XML tags, whose definitions are up to the programmer (and sometimes the industry), all SGML and HTML tags are predefined.

Before I continue with this comparison, I want to familiarize you with XML's key features: all elements must be balanced with a start tag and end tag, or they must use a special self-terminating format; nesting is strictly enforced; and attributes within tags always use quotes. Finally, remember that XML is case-sensitive; therefore, for example, `<TAG>`, `<tag>`, and `<Tag>` are different and could have different meanings (see Table 1).

XML syntax provides a free-format interchange of data. The definitions of the schema can be documented in a number of ways, allowing communities of users other than the original definer to understand what the designer of the schema intended. You can document schemas with SGML DTD language, W3C XML schema definition language, or other techniques using XML primitives. The XMLSPY suite is an inexpensive program that can help with XML schemas. It's an

Listing 1—*XML code is sent to a number of devices. Each device will recognize only the XML tags that are defined in its `ObjectDefs.o`. The other tags are ignored.*

```
<PatientTest>
    <Patient>
        <FirstName>John</FirstName>
        <MiddleName>Wesley</MiddleName>
        <LastName>Jones</LastName>
        <idNumber>572-89-2387</idNumber>
    </Patient>
    <PathologyLab>
        <ChemTest>BUN</ChemTest>
        <CellTest>leukocyte_count</CellTest>
        <CellTest>diff</CellTest>
    </PathologyLab>
</PatientTest>
```

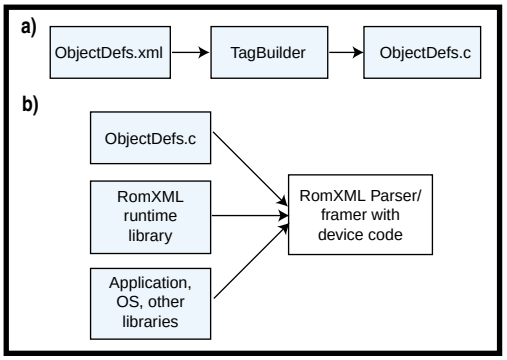

Figure 2a—*Using the TagBuilder utility, you can create XML object definitions outside the device.* **b—***XML object definitions are compiled and linked with the application, RomXML runtime library, and real-time operating system (RTOS).*

XML editor, schema designer, and more. You may download a 30-day evaluation copy from the XMLSPY web site (www.xmlspy.com).

With some schema definition techniques, a general-purpose parser can read a DTD or schema file that contains the definitions it needs to parse a particular XML document. A parser that checks an XML document for correct XML syntax is said to be checking for well-formed XML documents. A parser that checks XML documents against an external DTD definition is said to be a validating parser. Because embedded systems (as well as devices and instruments) and their XML tags are predefined, the schema definitions can be handled outside the embedded system, consequently reducing the system requirements in the device.

XML documents are both human and machine-readable. The same schema can be defined using multiple techniques. Allegro Software Development's RomXML parser and framer use a special schema definition language called RxSchema. This definition language is similar to the W3C XML schema definition that uses XML elements to define the XML schema. RxSchema provides control of the internal storage definition, which is useful in limited-resource embedded environments.

EMBEDDED PARSER/FRAMER

It's useful for an embedded device to exchange XML-based information with other embedded devices, general-purpose desktop workstations, and mainframe computers. However, the processing and memory requirements for a general-purpose XML parser are prohibitive. An embedded device doesn't need a dynamic tag set or vocabulary, because each embedded tool is typically a dedicated, single-purpose device. Allegro provides the RomXML parser/framer toolkit for such devices.

An embedded XML parser efficiently translates data between the XML syntax and an internal format (e.g., a C structure). Allegro's RomXML parser/framer provides a lightweight translation between predefined C language structures and XML-based representations.

By defining the rules for data translation external to the embedded device, you can build a small-footprint, special-purpose XML translator that uses dedicated vocabulary definitions to reduce the size of code and data storage. This permits the embedded device to move data to other machines in an XML-based standard format without carrying the overhead of general-purpose XML tools. The RomXML parser/framer fits in about 10 KB of memory in the device.

Additionally, the RomXML toolkit uses a set of XML tags to define an XML object. The XML object is defined with the C internal storage structures and the XML tag set that's used for the XML-based data interchange.

The RomXML TagBuilder utility program analyzes the RomXML object definitions and produces an object definition file in C language (see Figure 2a). This is compiled with the RomXML parser/framer code (see Figure 2b). The object definition file contains the transformation tables that the runtime RomXML code uses to perform specific translations for each XML object between the defined C structures and XML tag set.

The RomXML TagBuilder program uses standard C library calls for read-

Listing 2—*You can transmit different types of information with XML. This portion of an XML framed data transfer provides a few examples.*

```
<?xml version="1.0"?>
<datarecord>
  <Patient OutPatient="false">
    <idNumber>572892387</idNumber>
    <name>
      <firstname>John</firstname>
      <middle>Wesley</middle>
      <lastname>Jones</lastname>
    </name>
  </Patient>
  <field1>23</field1>
  <field2>1026</field2>
  <switch>192.23.45.67</switch>
</datarecord>
*********************************************************************
You can store the XML framed data in an embedded device using C.
*********************************************************************
typedef struct {
       Patient sPatient;
       Unsigned8 Field1;
       Unsigned32 Field2;
       char SwitchAddress[4];
} myRecord;
typedef struct {
       PatientName sName;
       Unsigned32 Number;
       Unsigned8 outpatient;
} Patient;
typedef struct {
       char FirstName[20];
       char MiddleName[20];
       char LastName[32];
} PatientName;
```

ing and writing the XML object definition files. It's delivered in executable form as a Windows program or in source format so you can host it in a particular UNIX environment.

The RomXML runtime code is a series of calls that you can use to handle the XML objects. Therefore, you can pass an incoming XML datastream and XML object definition to the RomXML program, and it will parse the XML datastream into a C structure (see Figure 3a).

Furthermore, you can pass a C structure and the XML object definition to the RomXML program and it will prepare an XML datastream by framing the data from the C language structure with the appropriate XML tags (see Figure 3b).

RomXML software works directly with a stand-alone embedded application or other Allegro products. XML datastreams are transmitted to and from other systems as an HTTP object with the RomPager embedded web server or RomWebClient embedded HTTP client. The XML datastreams are also sent and received as e-mail attachments via the RomMailer or RomPOP embedded e-mail programs.

XML TRANSLATION

An XML document contains XML syntax elements that describe the structure of the data and underlying data. The XML document itself does not contain any information about how to store the data in a given device.

Listing 2 shows an example of an XML document that contains some information about a customer. The XML schema or tag set used in this particular document provides a framing of the data that can be clearly understood by both you and a machine. In order for an embedded device to use the data in this XML document, it might decide to store the underlying data in a series of C structures.

The RomXML parser/framer provides an efficient mechanism for translating data between XML and internal C structure representations of the data. The RxSchema language is used to describe the relationships between the XML document and C structures. Because XML is a string language that expresses structure (not data types), the RxSchema language is also used to tell the embedded device that the <field2> tag frames an unsigned 32-bit integer value, and the <switch> tag frames a value expressed in dotted-decimal notation.

TagBuilder UTILITY

The RxSchema language is expressed in XML via a notation similar to the

Listing 3—*RomXML embedded XML uses an XML-based schema to define the relationships between the XML data tags and the embedded C program data structure. The schema is used to predefine these relationships; therefore, resources aren't used during runtime. I compiled this schema using the TagBuilder utility, and then linked it to the embedded application.*

```
<?xml version="1.0"?>
<objectList>
<object name="myObject">
<header><?xml version="1.0"?></header>
<struct sname="myRecord" fname="myRecord" xname="datarecord">
    <struct sname="Cust" fname="sCust" xname="Patient">
<attribute fname="outpatient" xname="OutPatient" type="boolstr"/>
  <element fname="Number" xname="idNumber" type="Unsigned32"/>
      <struct sname="CustName" fname="sName" xname="name">
<element fname="FirstName" xname="firstname" type="string.ansi" size="20"/>
<element fname="MiddleName" xname="middle" type="string.ansi" size="20"/>
<element fname="LastName" xname="lastname" type="string.ansi" size="32"/>
      </struct>
    </struct>
  <element fname="Field1" xname="field1" type="Unsigned8"/>
<element fname="Field2" xname="field2" type="Unsigned32" default="51"/>
<element fname="SwitchAddress" xname="switch" type="dotform" size="4"/>
</struct>
</object>
</objectList>
```

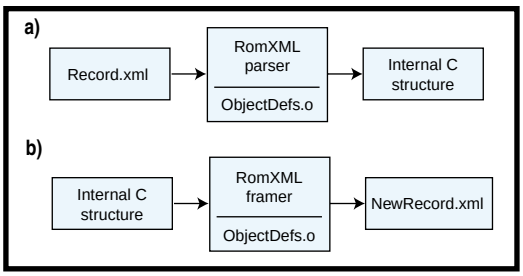

Figure 3a—*Incoming XML-framed records are scanned by the RomXML parser program, predefined object definitions, and the data inserted in the appropriate C data structures.* **b—***Data in the device is XML-framed by the RomXML framer and the predefined XML object definitions.*

W3C XML schema definition language. The definitions of the XML document that will be parsed and framed along with the definitions of the internal C structure are compiled by the TagBuilder utility. This utility provides a compact internal representation of the translation definitions.

After linking the compiled definitions with the RomXML runtime library and the rest of the embedded device software, the device can read an XML document and call the RomXML parser to translate the document into the internal format. When the device is ready to create an XML document, it then calls the RomXML framer to translate the internal format into the document.

The XML document that contains the set of RxSchema definitions for the sample C language structures and XML document is depicted in Listing 3. This example is a modified subset of an example taken from the RomXML programming reference manual.

AVAILABLE NOW

Network-enabled devices become easier to deploy and interoperate when IT departments use embedded XML parsers and framers. When you're designing a device, you can implement the more general XML products, but the additional required resources can make the device too expensive to produce. And, because embedded or stand-alone products are usually single-purpose devices, there's no need for dynamic tag resolution.

Results tend to vary when searching the 'Net for "embedded XML." You'll get hits for embedded XML databases, parsers, documentation, and the like. Few companies provide an embedded XML parser/framer product. Allegro and NetSilicon are two. Many RTOS vendors, such as Green Hills Software and OSE, provide embedded XML from third parties. Others, like Wind River Systems, provide pointers to compatible offerings. I suggest that you go to your favorite RTOS provider to search for XML products and information.

Allegro's embedded XML toolkit products are provided as ANSI C sources. The toolkit sells for $15,000, but it's royalty free for runtime code sold on a single platform or product. This is typical of many network toolkits meant for embedded products. So, make your customer's IT department happy. Implement XML in the next product you design.

Edward Steinfeld has more than 25 years of experience in real-time and embedded computing. He began his career as a programmer, writing code and designing hardware to test hybrid circuit boards. His international experience includes a stint in Hong Kong as a Far East channels manager for international OEM sales in the Pacific Rim and Europe. Currently, Ed provides market research, planning, and services to the embedded computing industry. You may reach him at edward@go-embedded.com.

RESOURCE

XML information
www.xml.com

SOURCES

RomXML Embedded XML toolkit
Allegro Software Development Corp.
(978) 264-6600
www.allegrosoft.com

XMLSPY
Altova, Inc.
(978) 816-1600
www.xmlspy.com