IEEE Software
www.computer.org/software

Portable C/C++ Code for Portable XML Data

Zhaoqing Wang, *Zhejiang Sci-Tech University*
Harry H. Cheng, *University of California, Davis*

Vol. 23, No. 1
January/February 2006

Portable C/C++ Code for Portable XML Data

Zhaoqing Wang, *Zhejiang Sci-Tech University*
Harry H. Cheng, *University of California, Davis*

Ch XML packages integrate an embeddable C-compatible interpreter with XML C/C++ toolkits—giving developers the option of using portable C/C++ scripts to process portable XML data.

XML is changing the world of information sharing and exchange by letting users clearly define their data and documents for specific tasks, such as electronic data interchange, content management, or publishing.[1] XML uses context encapsulation to separate content from presentation and to support a hierarchical structure among data from various sources. XML data is reusable, easily derivable, and reconfigurable. However, XML-based applications need a programming technology to perform processing-related tasks such as parsing, generating, manipulating, and validating the data.

Application developers commonly use toolkits based on C/C++, such as Gnome's XML C parser and toolkit, Oracle's XML Developer's Kit for C/C++ (XDK), and Microsoft's XML Parser. These toolkits reflect the rich set of facilities, data types, operators, control structures, and runtime library functions that make C/C++ such a popular programming language. In fact, these tools, as well as the large number of skilled C/C++ programmers, often make C/C++ the preferred language for building portable systems.

However, C/C++ presents challenges in handling XML data. Even though C standard-conforming programs are portable, the compilation process is platform dependent. A C/C++ code needs different compilers on different platforms. The platform compilers handle portability issues such as different-sized data types, byte ordering, and file specification syntax, so truly portable C/C++ code isn't typically generated and executed dynamically. For portability, it needs a virtual machine or interpreter. This is why Java and a Java virtual machine are commonly used to process XML data.[2]

Ch is an embeddable C/C++ interpreter for cross-platform scripting, shell programming, numerical computing, network computing, and embedded scripting (see www.softintegration.com).[3,4] It lets applications written in C/C++ run dynamically across multiplatforms, such as Windows, Mac OS X, Linux, and Unix, without tedious compilation and linking. We used Ch to develop Ch XML, an open source Ch package based on Gnome libxml2 (http://chlibxml2.sourceforge.net) and Oracle XDK for C/C++ (http://iel.ucdavis.edu/projects/chxml).[5]

The Ch XML for these toolkits is designed to integrate portable XML data with portable C/C++ scripting code. Here we use Gnome libxml2 to discuss the integration, but the ideas presented also apply to the Ch XML package for Oracle's XDK. Our packages also let many existing applications and technologies based on C/C++ work seamlessly with XML documents through a network. We illustrate Ch XML's application potential in network computing.

Script computing for XML

XML application designers often use a two- or three-tiered architecture to facilitate development. They implement the application logic using XSL stylesheets and the lower-level components using a scripting language such as Python, Tcl, PHP, or Perl. Scripting languages are ideal for portable application development and system integration.[6] The lowest-level components, including an XML parser, are implemented using a system language such as C or C++.

Gnome libxml2 is a high-performance, widely used XML C parser and toolkit for the Gnome project. It can also perform WXS (W3C XML Schema) schema validation and Relax-NG validation, and most scripting languages include modules to interface with it. For example, the Libxml2-Python package contains a module that lets applications written in the Python scripting language use the Gnome libxml2 library to manipulate XML files (see http://rpmfind.net/linux/rpm2html/search.php?query=libxml2-python). TclXML (http://sourceforge.net/projects/tclxml) is a package for parsing XML documents using the Tcl scripting language; it provides a wrapper for libxml2. PHP-libxml2 is an implementation of a PHP binding to libxml2 (see www.zend.com/php5/articles/php5-xmlphp.php#Heading4), and libXML-Perl is a Perl binding to it (see http://perl-xml.sourceforge.net).

The Ch interpreter provides the basis for portable processing of XML data in C/C++ and easy integration with legacy applications. The Ch interpreter supports all features in the ISO 1990 C standard and most new features added in the ISO C99, such as complex numbers, variable-length array, binary constants, and IEEE 754 floating-point arithmetic. Ch also supports classes, objects, and encapsulation in C++ for object-based programming. As a superset of C, all existing C libraries and modules can be part of the Ch libraries. So, Ch libraries' potential is almost unlimited. Ch supports Posix, TCP/IP socket, Winsock, Win32, X11/Motif, GTK+, OpenGL, open database connectivity, Lapack, the Lightweight Directory Access Protocol, the Numerical Algorithms Group's statistics library, Intel OpenCV for computer vision, ImageMagick for image processing, and National Instrument's NI-DAQ and NI-Motion.

Ch is especially suitable for Web-based applications. With development modules, such as classes for a common gateway interface (CGI) to Web servers, Ch allows rapid development and deployment of Web-based applications and services. For example, Harry Cheng and his colleagues have developed an open source Web-based system for control system design and analysis in Ch, Ch CGI, and the open source Ch Control System Toolkit.[7]

The Ch interpreter provides the basis for portable processing of XML data in C/C++ and easy integration with legacy applications.

Integrating Gnome libxml2 with Ch

A key Ch feature is the Ch Software Developer's Kit (www.softintegration.com/docs) included in the distribution. Ch SDK makes it easy to integrate binary static or dynamic C/C++ libraries with the language environment without recompilation. The SDK creates an interface between scripting code and the binary static or dynamic C/C++ libraries. The interface interprets the scripting application code and calls the corresponding library functions.

The Ch libxml2 package consists of mainly two parts—the *chf* (Ch function) and a dynamically loaded library. Applications call the chf files directly using libxml2 library functions. The dynamically loaded library is developed in C/C++ and contains interface functions that the Ch functions call. It's also linked to the binary libraries with libxml2 functions. The Ch space refers to the code for a user application program in scripting mode. The C space refers to the code in the dynamically loaded library and binary libraries in binary mode.

The representation of code and parameters differs between the Ch and C spaces. The Ch wrapper, which consists of components in both Ch and C spaces, acts as the broker between the Ch space and C space, transferring the code and argument value. In the following sections, we describe how to create a Ch wrapper to make the barrier transparent between the user's application in the Ch space and the dynamic library in the C space.

Based on the function argument type, three types of functions exist in libxml2. The first one is a regular function without an argument

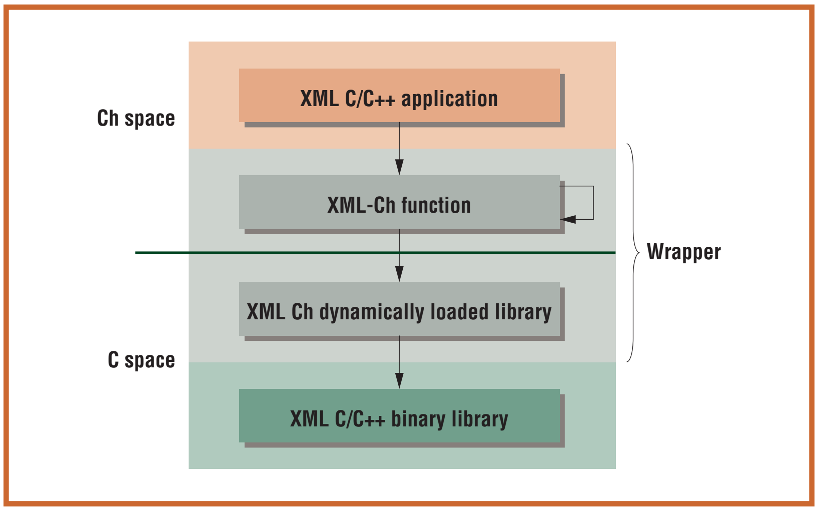


Figure 1. Integration of a general API with Ch.

type of pointer to function. The second one is a function with an argument type of pointer to the user-defined callback function. A function in the libxml2 library in the C space might invoke the callback function in the Ch space. The third one is a function with an argument type of pointer to the system's predefined callback function. A function in an application program in the Ch space might invoke the predefined callback function in the C space.

Integration without callback functions

The XML Document Object Model API creates a tree structure in memory to store the XML document's data. Typically, a DOM-based XML C/C++ application has no callback functions. It's relatively simple to create a Ch binding to those libxml2 APIs that have no callback functions.

Figure 1 illustrates the basic execution procedures of an application through the Ch wrapper. The architecture consists of three layers:

- the user's applications, which are the existing applications using the XML C functions;
- the Ch wrapper, which is the broker between the binary functions and text-based interpretive functions; and
- the original C/C++ binary libraries provided by Gnome libxml2.

In figure 1, the modules in the upper part are in the Ch space, and those in the lower part are in the C space.

With the Ch binding for Gnome libxml2, we can interpretively execute the user applications without compilation across platforms. When an XML application calls an XML function, the program calls the XML-Ch function. This function typically invokes the XML Ch interface function (chdl) in a dynamically loaded library, which in turn calls the corresponding binary function in the libxml2 library.

For example, an application program gjobread.c uses function `xmlParseFile()` with an argument of the file name to be passed. When gjobread.c launches, it loads the Ch function file xmlParseFile.chf for function `xmlParserFile()`. The function in the Ch function file invokes the binary interface function `xmlParseFile_chdl()` through the dynamically loaded library, which in turn calls the existing binary function `xmlParseFile()` provided in libxml2. In this case, xmlParseFile.chf is an element in the intersection of the Ch space and Ch wrapper, and `xmlParseFile_chdl()` in the dynamically loaded library is in the intersection of the C space and Ch wrapper. Function `dlrunfun()` in a Ch function file invokes the interface function in the C space. The Ch wrapper functions act as a broker between the Ch and C spaces.

Integration with registered callback functions

The Simple API for XML (SAX) uses an event-based model to process XML documents. A SAX-based parser invokes functions in C when it encounters a markup, such as a start or end tag. The SAX application defines the callback functions for the XML document. The Ch wrapper provides the registration for this kind of callback function, as figure 2a shows.

When an application calls a function with an argument of pointer to a user-defined function (callback function) in the Ch space, the application passes the callback function's address to the Ch function as an argument of the function. However, the callback function should be registered in the Ch wrapper. The Ch function passes this address to the interface function in the C space to register this callback function in the Ch wrapper. When an event is encountered, the XML binary library function will call this callback function using the registered function we added to the C space. This function invokes the callback function in the Ch space using the Ch SDK API `Ch_CallFuncByName()` or `Ch_CallFuncByAddr()`.

For example, a SAX program *testSAX.c* contains the function `xmlSAXUserParseFile()`

with the following function prototype:

```
int  xmlSAXUserParseFile
(xmlSAXHandlerPtr sax, void
*user_data, const char *filename)
```

The first argument of type `xmlSAXHandlerPtr`, pointer to structure, has member fields of pointer to function, such as `startDocument()`, in the Ch space. The callback function `startDocument_chdl_funarg()` in the C space in this example invokes the callback function `startDocument()` using SDK function `Ch_CallFuncBy Addr()` when it encounters the "start a document" event. The interface function `xmlSAX UserParseFile_chdl()`, which function `dlrunfun()` invokes in the Ch space, calls function `xmlSAXUserParseFile()` in the libxml2 binary library.

Integration with system default registered functions

The encoding module inibxml2 has APIs such as `xmlFindCharEncodingHandler()` to search for a registered handler to read and write the corresponding encoding. The registered handler can be a system default function or user-defined callback function. After the module obtains the handler, an application program in the Ch space calls the handler. If the handler is in the Ch space, it's easy to use, because the user-defined function handler will run in the same space as the user application.

However, if the handler is in the C space, the system default callback function in the C space, shown as `default_callback_func()` in figure 2b, must run in the Ch space. To invoke `default_callback_func()` from the Ch space, the applications will call function `default_callback_ch()` using function `dlrunfun()`. This is similar to the situation described earlier (without callback functions), when a typical function in the C space ran in the Ch space. The only difference is that the return value of a function with a default system callback function, such as `xmlFindCharEncodingHandler()`, should be the pointer to `default_callback_ch()` in the Ch space instead of the pointer to `default_callback_func()` in the C space.

For example, the testWriter.c demo program first uses function call `xmlFindCharEncodingHandler(encoding)` to get the encoding handler. The handler is then called in the Ch space. The handler can be either in the Ch or C space. If the handler is in the C space, a callback function, similar to `default_callback_ch()`, is preloaded through a header file in the Ch space. It then invokes the system's predefined callback function similar to `default_callback_func()` in the C space. Function call `xmlFindCharEncodingHandler(encoding)` returns the callback function's address in the Ch space. Similar to the case without the callback function, an interface function `default_callback_chdl()` invokes the binary function `default_callback_func()` provided in libxml2 for the corresponding Ch function `default_callback_ch()`, as figure 2b shows.

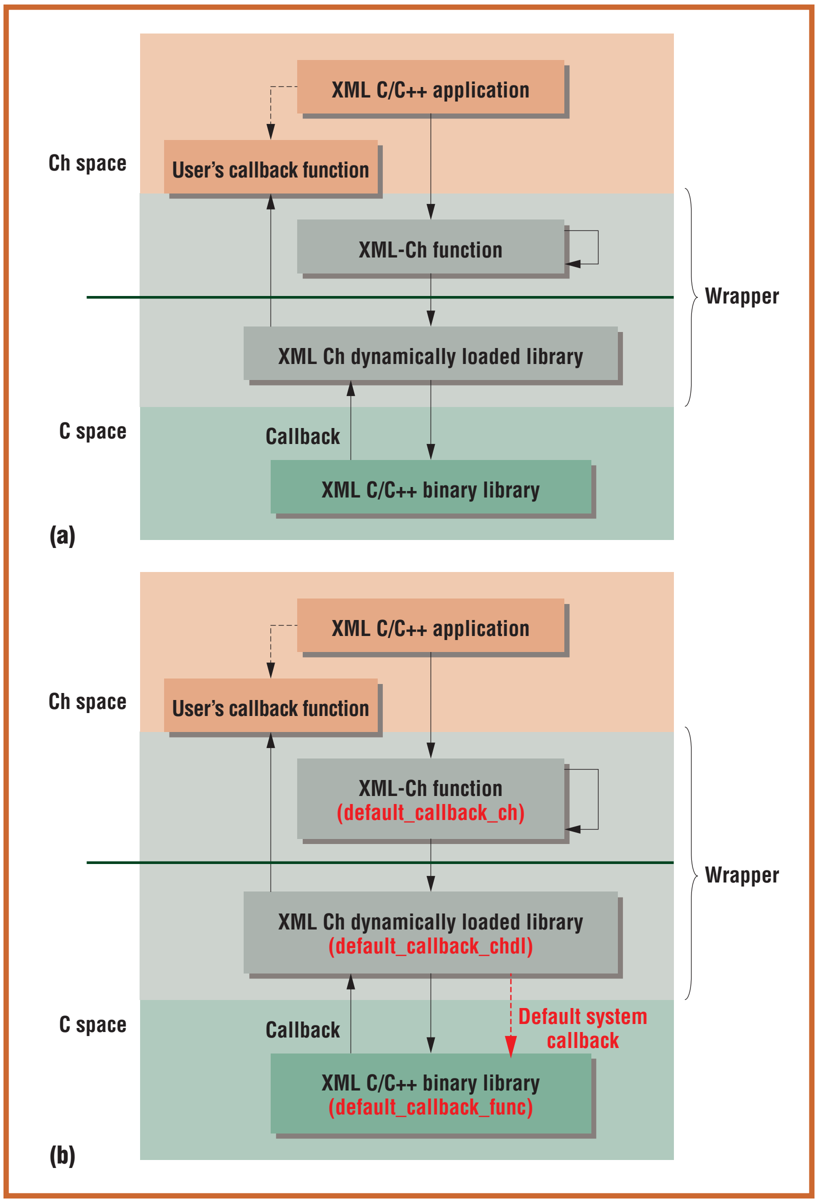
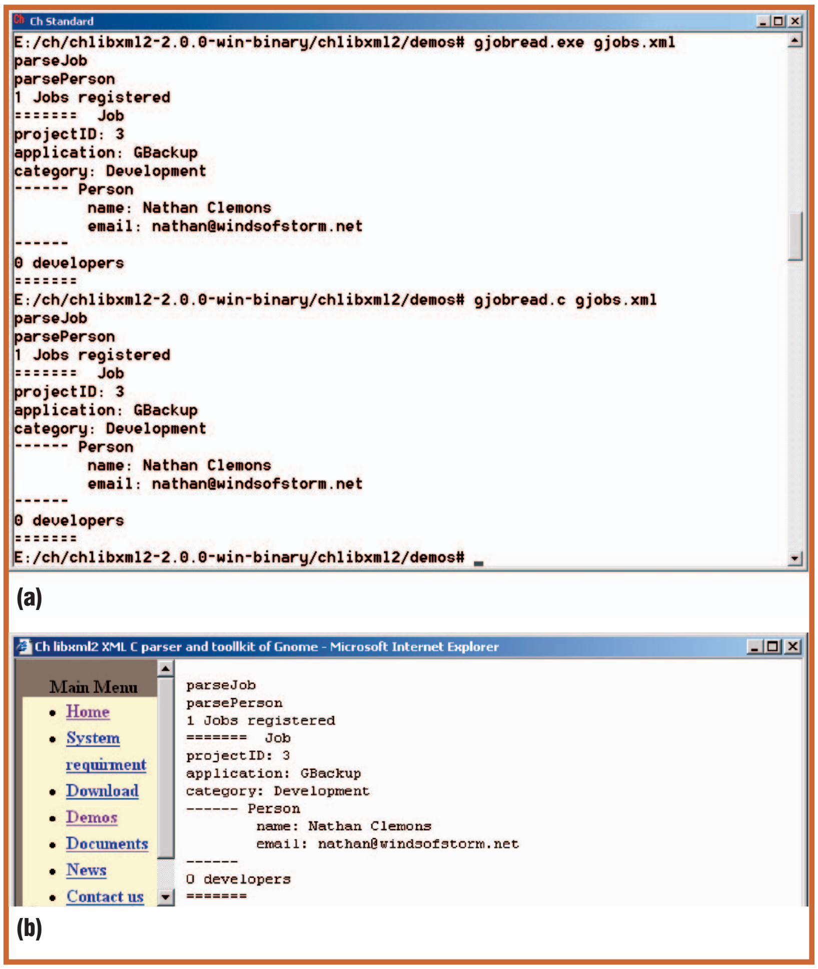


Figure 2. (a) The architecture for integrating the Simple API for XML with callback functions in Ch; (b) the architecture for integrating an API with a system default callback function.



```
E:/ch/chlibxml2-2.0.0-win-binary/chlibxml2/demos# gjobread.exe gjobs.xml
parseJob
parsePerson
1 Jobs registered
=======  Job
projectID: 3
application: GBackup
category: Development
------ Person
        name: Nathan Clemons
        email: nathan@windsofstorm.net
------
0 developers
=======
E:/ch/chlibxml2-2.0.0-win-binary/chlibxml2/demos# gjobread.c gjobs.xml
parseJob
parsePerson
1 Jobs registered
=======  Job
projectID: 3
application: GBackup
category: Development
------ Person
        name: Nathan Clemons
        email: nathan@windsofstorm.net
------
0 developers
=======
E:/ch/chlibxml2-2.0.0-win-binary/chlibxml2/demos# _
```

(a)

Ch libxml2 XML C parser and toolkit of Gnome - Microsoft Internet Explorer
Main Menu
Home
System requirment
Download
Demos
Documents
News
Contact us

```
parseJob
parsePerson
1 Jobs registered
=======  Job
projectID: 3
application: GBackup
category: Development
------ Person
        name: Nathan Clemons
        email: nathan@windsofstorm.net
------
0 developers
=======
```

(b)

Figure 3. (a) Execution results of the C program gjobread.c and binary executable gjobread.exe; (b) output from executing the C program gjobread.ch through the Internet.

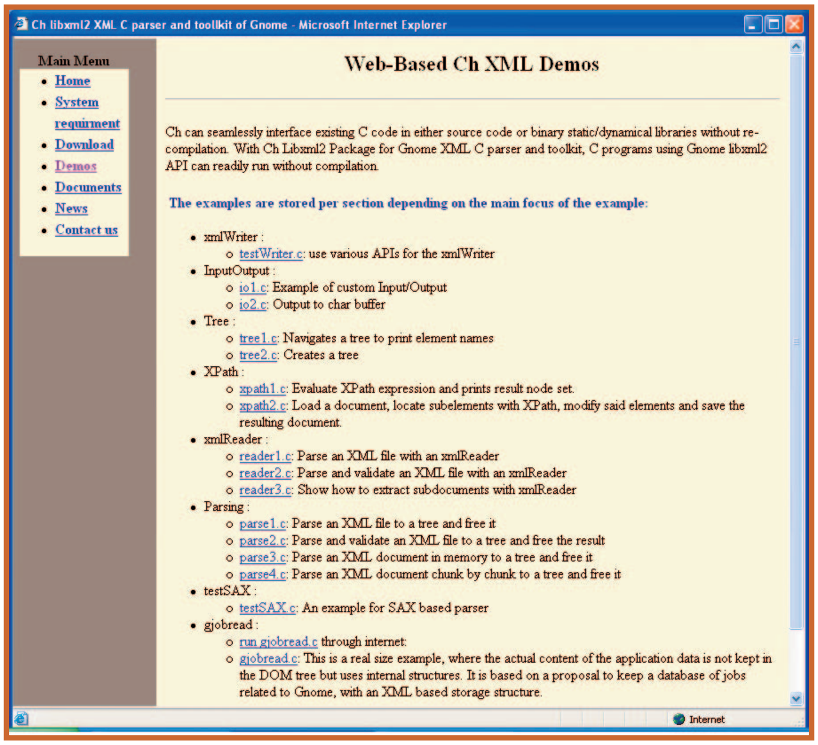
Application examples

The Gnome libxml2 XML C parser and toolkit package provides several C demo programs for processing XML documents. Figure 3a shows the interactive execution of the C program gjobread.c in a Ch command shell. The program handles the Gnome XML format for job descriptions. It parses the XML data file gjobs.xml and prints out the content information about jobs. Typing a file name in a command shell causes gjobread.c to execute and displays the output, as shown in figure 3a. The result is the same as the output of the compiled binary executable program gjobread.exe. We can also execute gjobread.c from a Ch-compatible integrated development environment. Several IDEs, such as ChSciTE, are readily available for editing and running Ch programs.

XML documents are widely used for Web-based application and integration. Like Perl, Python, or PHP, interpretive C scripts can be used in Ch to create dynamic Web pages. The Ch CGI Toolkit (www.softintegration.com/products/toolkit/cgi) contains four classes—Request, Response, Server, and Cookie—with APIs similar to ActiveServer Pages and JavaServer Pages. For a Web-based application, Ch CGI scripts typically have the ".ch" file extension. For example, with a proper Web server setup, we can launch the program gjobread.ch by clicking on a Web browser's hyperlink. The program can print out and display its results inside the Web browser.

Figure 4 shows a demo page for a Web-based application using the Ch XML package. The "run gjobread.c" link executes a CGI demo program. The CGI program gjobread.ch is modified from the original C program gjobread.c, adding the following CGI code to display the output as plain text in a Web browser:

```
class CResponse Response;
 Response.setContentType("text/
    plain");
 Response.begin();
 ...
 Response.end();
```



Ch libxml2 XML C parser and toolkit of Gnome - Microsoft Internet Explorer
Main Menu
Home
System requirment
Download
Demos
Documents
News
Contact us
Internet

Web-Based Ch XML Demos

Ch can seamlessly interface existing C code in either source code or binary static/dynamical libraries without re-compilation. With Ch Libxml2 Package for Gnome XML C parser and toolkit, C programs using Gnome libxml2 API can readily run without compilation.

The examples are stored per section depending on the main focus of the example:

- xmlWriter :
  - testWriter.c: use various APIs for the xmlWriter
- InputOutput :
  - io1.c: Example of custom Input/Output
  - io2.c: Output to char buffer
- Tree :
  - tree1.c: Navigates a tree to print element names
  - tree2.c: Creates a tree
- XPath :
  - xpath1.c: Evaluate XPath expression and prints result node set.
  - xpath2.c: Load a document, locate subelements with XPath, modify said elements and save the resulting document.
- xmlReader :
  - reader1.c: Parse an XML file with an xmlReader
  - reader2.c: Parse and validate an XML file with an xmlReader
  - reader3.c: Show how to extract subdocuments with xmlReader
- Parsing :
  - parse1.c: Parse an XML file to a tree and free it
  - parse2.c: Parse and validate an XML file to a tree and free the result
  - parse3.c: Parse an XML document in memory to a tree and free it
  - parse4.c: Parse an XML document chunk by chunk to a tree and free it
- testSAX :
  - testSAX.c: An example for SAX based parser
- gjobread :
  - run gjobread.c through internet:
  - gjobread.c: This is a real size example, where the actual content of the application data is not kept in the DOM tree but uses internal structures. It is based on a proposal to keep a database of jobs related to Gnome, with an XML based storage structure.

Figure 4. The Ch XML demo Web page for Gnome libxml2.

```
#include <embedch.h>                         // header file for Embedded Ch
 ChInterp_t interp;
 char *argvv[] = {"gjobread.c", NULL};      // argument for a C/C++ script
 Ch_Initialize(&interp, NULL);              // initialize an embedded Ch
 Ch_RunScript(interp, argvv);               // run an Embedded C/C++ script indicated by argvv
 ....
 Ch_End(interp);                            // release memory and terminate the interpreter
```

Figure 5. Embedding Ch in the C/C++ code as a scripting engine.

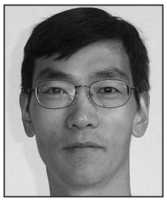
The output is the same whether we execute the program in a Ch command shell or in a CGI-based Web page (see figure 3b).

If we embed Ch as a scripting engine in an application program, the program can process XML documents using C scripts that the application program dynamically controls. Using a Ch XML package, we can easily process portable XML data in mobile code or mobile agents in various network-based applications.[8] As a simple example, in the C/C++ code fragment shown in figure 5, Ch is embedded as a scripting engine.

The program initializes an embedded Ch interpreter `interp` using function `Ch_Initialize()`. The application then runs C script gjobread.c through function `Ch_RunScript()`. Finally, function `Ch_End()` releases the interpreter. The mobile code using XML functions can also be generated dynamically and sent through the network. It can process XML documents using a Ch XML package and access a database using open database connectivity. The script can also invoke functions or classes in the binary host application program.

The example applications show the potential for using Ch XML in applications that need dynamically executable codes to process dynamic XML data. Because Ch is an embeddable C/C++ interpreter, applications—especially those developed in C/C++—can embed Ch as a scripting engine to process portable XML documents using portable C/C++ scripts through the Internet. Scripting in C/C++ for processing XML data is ideal for rapid prototyping,[9] Web-based applications,[7] mobile computing,[8] and teaching and learning XML.

About the Authors



Zhaoqing Wang is an associate professor and the director of the Instructional Division for Computing Technology in the Zhejiang Sci-Tech University, Zhejiang, China. His research interests include network computing, engineering software design, real-time and embedded control systems, XML data processing, Web technology, and Ch and its applications. He received his PhD in electrical engineering from the Shanghai Jiaotong University, Shanghai, and was a post-doctoral researcher at the University of California, Davis. Contact him at Instructional Division for Computing Technology, Zhejiang Sci-Tech University, Hangzhou, Zhejiang, China, 310018; thezqwang@yahoo.com.

Harry H. Cheng is a professor at the University of California, Davis, and is the director of the university's Integration Engineering Laboratory. His research interests include information technology, mobile multiagent systems, information-driven systems, design and manufacturing, mechatronics, and intelligent transportation systems. He received his PhD in mechanical engineering from the University of Illinois at Chicago. He's a member of the American Society of Mechanical Engineers, the IEEE, the IEEE Robotics and Automation Society, and the IEEE Computer Society. He holds one US patent and has published over 110 papers in refereed journals and conference proceedings. Contact him at the Integration Engineering Laboratory, Dept. Mechanical and Aeronautical Engineering, Univ. of California, Davis, CA 95616; hhcheng@ucdavis.edu.

References

1. R. Vidgen and S. Goodwin, "XML: What Is It Good For?" *Computing & Control Eng. J.*, vol. 11 , no. 3, 2000, pp. 119–124.
2. J.P. Morgenthal, "Portable Data / Portable Code: XML & Java Technologies," white paper, Sun Microsystems, 2003, http://java.sun.com/xml/ncfocus.html.
3. H.H. Cheng, "Scientific Computing in the Ch Programming Language," *Scientific Programming*, vol. 2, no. 3, 1993, pp. 49–75.
4. H.H. Cheng, "Ch: A C/C++ Interpreter for Script Computing," *C/C++ User's J.*, vol. 24, no. 1, 2006, pp. 6–12.
5. Z. Wang and H.H. Cheng, *Integrating Portable XML Data with Portable C/C++ Code*, tech. report, Oracle, 2004; www.oracle.com/technology/pub/articles/wang_ch.html.
6. J. K. Ousterhout, "Scripting: Higher-level Programming for the 21st Century," *Computer*, vol. 31, no. 3, 1998, pp. 23–30.
7. Q. Yu, B. Chen, and H.H. Cheng, "Web-Based Control System Design and Analysis," *IEEE Control Systems Magazine*, vol. 24, no. 3, 2004, pp. 45–57.
8. B. Chen and H.H. Cheng, "A Run-Time Support Environment for Mobile Agents," *Proc. ASME/IEEE Int'l Conf. Mechatronic and Embedded Systems and Applications*, CD-ROM, paper #DETC2005-85389, 2005.
9. B. Chen and H.H. Cheng, "Interpretive OpenGL for Computer Graphics," *Computers and Graphics*, vol. 29, no. 2, 2005, pp. 331–339.