Xerces-C documentation

Table of Contents

1
Xerces C++ Parser

Xerces-C++ Version 1.1.0

Xerces-C is a validating XML parser written in a portable subset of C++. Xerces-C makes it easy to give your application the ability to read and write XML data. A shared library is provided for parsing, generating, manipulating, and validating XML documents. Xerces-C is faithful to the XML 1.0 recommendation and associated standards (DOM 1.0, SAX 1.0, Namespaces). It also provides early implementations of DOM Level 2 version 1.0 and soon it will support XMLSchema, both the Structures and the Datatypes. The parser provides high performance, modularity, and scalability. Source code, samples and API documentation are provided with the parser. For portability, care has been taken to make minimal use of templates, no RTTI, no C++ namespaces, limited use of exceptions and minimal use of #ifdefs.

Applications of the Xerces Parser

Xerces has rich generating and validating capabilities. The parser is used for:

- Building XML-savvy Web servers
- Building next generation of vertical applications that use XML as their data format
- On-the-fly validation for creating XML editors
- Ensuring the integrity of e-business data expressed in XML
- Building truly internationalized XML applications

Features

- Conforms to XML Spec 1.0
- Tracking of latest DOM (Level 1.0), SAX and Namespace specifications.
- Experimental DOM Level 2.0 implementation
- Source code, samples, and documentation is provided.
- Programmatic generation and validation of XML
- Pluggable catalogs, validators and encodings
- High performance
- Customizable error handling

Platforms with Binaries

- Win32 using MSVC 6.0 SP3
- Linux (RedHat 6.1) using egcs-2.91.66 and glibc-2.1.2-11
- Solaris 2.6 using Sun Workshop 4.2 **XML4C only**
- AIX 4.2.1 using xlC 3.6.4 **XML4C only**
- HP-UX 10.2 using CC A.10.36 **XML4C only**
- HP-UX 11 using aCC A.03.13 with pthreads **XML4C only**

Other ports...

- OS/390

- AS/400
- SGI IRIX
- FreeBSD
- Unixware
- and more!

2
Installation

Window NT/98

Install the binary Xerces-C release by using `unzip` on the file-win32.zip archive in the Windows environment. You can use WinZip, or any other UnZip utility.

```
    unzip xerces-c-1_1_0-win32.zip
```

This creates a 'xerces-c-1_1_0-win32' sub-directory containing the Xerces-C distribution.

You need to add the 'xerces-c-1_1_0-win32\bin' directory to your path:

To do this under Windows NT, go to the start menu, click the settings menu and select control panel. When the control panel opens, double click on System and select the 'Environment' tab. Locate the PATH variable under system variables and add <full_path_to_xerces-c-1_1_0>\bin to the PATH variable. To do this under Windows 95/98 add this line to your AUTOEXEC.BAT file:

```
    SET PATH=<full_path_to_xerces-c-1_1_0>\bin;%PATH%
```

or run the `SET PATH` command in your shell window.

UNIX

Binary installation of this release is to extract the files from the compressed .tar archive (using 'tar').

```
    cd $HOME
    gunzip xerces-c-1_1_0-linux.tar.gz
    tar -xvf xerces-c-1_1_0-linux.tar
```

This will create an 'xerces-c-1_1_0-linux' sub-directory (in the home directory) which contains the Xerces-C distribution. You will need to add the xerces-c-1_1_0-linux/bin directory to your PATH environment variable:

For Bourne Shell, K Shell or Bash, type:

```
    export PATH="$PATH:$HOME/xerces-c-1_1_0-linux/bin"
```

For C Shell, type:

```
    setenv PATH "$PATH:$HOME/xerces-c-1_1_0-linux/bin"
```

If you wish to make this setting permanent, you need to change your profile by changing your setup files which can be either .profile or .kshrc.

In addition, you will also need to set the environment variables XERCESCROOT, ICUROOT and the library search path. (LIBPATH on AIX, LD_LIBRARY_PATH on Solaris and Linux, SHLIB_PATH on HP-UX).

> ***Note:*** *XERCESCROOT and ICUROOT are needed only if you intend to recompile the samples or build your own applications. The library path is necessary to link the shared libraries at runtime.*

For Bourne Shell, K Shell or Bash, type:

```
export XERCESCROOT=<wherever you installed Xerces-C>
export ICUROOT=<wherever you installed ICU>
export LIBPATH=$XERCESCROOT/lib:$LIBPATH (on AIX)
export LD_LIBRARY_PATH=$XERCESCROOT/lib:$LD_LIBRARY_PATH (on Solaris, Linux)
export SHLIB_PATH=$XERCESCROOT/lib:$SHLIB_PATH (on HP-UX)
```

For C Shell, type:

```
setenv XERCESCROOT "<wherever you installed Xerces-C>"
setenv ICUROOT "<wherever you installed ICU>"
setenv LIBPATH "$XERCESCROOT/lib:$LIBPATH" (on AIX)
setenv LD_LIBRARY_PATH "$XERCESCROOT/lib:$LD_LIBRARY_PATH" (on Solaris, Linux)
setenv SHLIB_PATH "$XERCESCROOT/lib:$SHLIB_PATH" (on HP-UX)
```

Note: *If you need to build the samples after installation, make sure you read and follow the build instructions given in the FAQ.*

3
Building Xerces-C

This page answers the following questions:

- Building Xerces-C on Windows.
- Building Xerces-C on UNIX.
- Building Xerces-C on Windows using Visual Age.
- Building Xerces-C on OS/2 using Visual Age.
- Building Xerces-C on AS/400.
- Building Xerces-C on Macintosh.
- Building ICU.
- Building COM module on Windows 98/NT/2000.
- How to build the User Documentation?.
- I wish to port Xerces to my favourite platform. Do you have any suggestions?
- What should I define XMLCh to be?
- How can I generate Xerces-C binaries which includes the sample NetAccessor implementation using Libwww?
- Where can I look for more help?

Building on Windows NT/98

Building Xerces-C library

To build Xerces-C from it source (using MSVC), you will need to open the workspace containing the project. If you are building your application, you may want to add the Xerces-C project inside your applications's workspace.

The workspace containing the Xerces-C project file and all other samples is:

```
    xerces-c-src-1_1_0\Projects\Win32\VC6\xerces-all\xerces-all.dsw
```

Once you are inside MSVC, you need to build the project marked **XercesLib**.

If you want to include the Xerces-C project separately, you need to pick up:

```
    xerces-c-src-1_1_0\Projects\Win32\VC6\xerces-all\XercesLib\XercesLib.dsp
```

You must make sure that you are linking your application with the xerces-c_1.lib library and also make sure that the associated DLL is somewhere in your path.

> ***Note:*** *If you are working on the AlphaWorks version which uses ICU, you must have the ICU data DLL named* `icudata.dll` *available from your path setting. For finding out where you can get ICU from and build it, look at the last section of this page.*

Building samples

Inside the same workspace (xerces-all.dsw), you'll find several other projects. These are for the samples. Select all the samples and right click on the selection. Then choose "Build (selection only)" to build all the samples in one shot.

Building on UNIX platforms

Xerces-C uses GNU tools like Autoconf and GNU Make to build the system. You must first make sure you have these tools installed on your system before proceeding. If you don not have required tools, ask your system administrator to get them for you. These tools are free under the GNU Public Licence and may be obtained from the Free Software Foundation.

Do not jump into the build directly before reading this.

Spending some time reading the following instructions will save you a lot of wasted time and support-related e-mail communication. The Xerces-C build instructions are a little different from normal product builds. Specifically, there are some wrapper-scripts that have been written to make life easier for you. You are free not to use these scripts and use Autoconf and GNU Make directly, but we want to make sure you know what you are by-passing and what risks you are taking. So read the following instructions carefully before attempting to build it yourself.

Besides having all necessary build tools, you also need to know what compilers we have tested Xerces-C on. The following table lists the relevant platforms and compilers.

Operating System	Compiler
Redhat Linux 6.1	g++, gcc (egcs)
AIX 4.3.3 and higher	xlC_r, xlc_r
Solaris 2.6	CC, cc
HP-UX 10.2	CC, cc
HP-UX 11	aCC, cc

If you are not using any of these compilers, you are taking a calculated risk by exploring new grounds. Your effort in making Xerces-C work on this new compiler is greatly appreciated and any problems you face can be addressed on the Xerces-C mailing list.

Differences between the UNIX platforms: The description below is generic, but as every programmer is aware, there are minor differences within the various UNIX flavors the world has been bestowed with. The one difference that you need to watch out in the discussion below, pertains to the system environment variable for finding libraries. On **Linux and Solaris**, the environment variable name is called `LD_LIBRARY_PATH`, on **AIX** it is `LIBPATH`, while on **HP-UX** it is `SHLIB_PATH`. The following discussion assumes you are working on Linux, but it is with subtle understanding that you know how to interpret it for the other UNIX flavors.

> ***Note:*** *If you wish to build Xerces-C with ICU, look at the last section of this page. It tells you where you can find ICU and how you can build Xerces-C to include the ICU international library.*

Setting build environment variables

Before doing the build, you must first set your environment variables to pick-up the compiler and also specify where you extracted Xerces-C on your machine. While the first one is probably set for you by the system administrator, just make sure you can invoke the compiler. You may do so by typing the compiler invocation command without any parameters (e.g. xlc_r, or g++, or cc) and check if you get a proper response back.

Next set your Xerces-C root path as follows:

```
    export XERCESCROOT=<full path to xerces-c-src-1_1_0>
```

This should be the full path of the directory where you extracted Xerces-C.

Building Xerces-C library

As mentioned earlier, you must be ready with the GNU tools like autoconf and gmake before you attempt the build.

The autoconf tool is required on only one platform and produces a set of portable scripts (configure) that you can run on all other platforms without actually having the autoconf tool installed everywhere. In all probability the autoconf-generated script (called `configure`) is already in your `src` directory. If not, type:

```
cd $XERCESCROOT/src
autoconf
```

This generates a shell-script called `configure`. It is tempting to run this script directly as is normally the case, but wait a minute. If you are using the default compilers like gcc and g++ you do not have a problem. But if you are not on the standard GNU compilers, you need to export a few more environment variables before you can invoke configure.

Rather than make you to figure out what strange environment variables you need to use, we have provided you with a wrapper script that does the job for you. All you need to tell the script is what your compiler is, and what options you are going to use inside your build, and the script does everything for you. Here is what the script takes as input:

```
runConfigure
runConfigure: Helper script to run "configure" for one of the
              supported platforms.
Usage: runConfigure "options"
       where options may be any of the following:
       -p <platform> (accepts 'aix', 'linux', 'solaris',
                           'hp-10', 'hp-11', 'irix', 'unixware')
       -c <C compiler name> (e.g. xlc_r, gcc, cc)
       -x <C++ compiler name> (e.g. xlC_r, g++, CC, aCC)
       -d (specifies that you want to build debug version)
       -m <message loader> can be 'inmem', 'icu', 'iconv'
       -n <net accessor> can be 'fileonly', 'libwww'
       -t <transcoder> can be 'icu' or 'native'
       -r <thread option> can be 'pthread' or 'dce' (only used on HP-11)
       -l <extra linker options>
       -z <extra compiler options>
       -h (to get help on the above commands)
```

Note: *Xerces-C can be built as either a standalone library or as a library dependent on International Components for Unicode (ICU). For simplicity, the following discussion only explains standalone builds.*

One of the common ways to build Xerces-C is as follows:

```
runConfigure -plinux -cgcc -xg++ -minmem -nfileonly -tnative
```

The response will be something like this:

```
Generating makefiles with the following options ...
Platform: linux
C Compiler: gcc
C++ Compiler: g++
Extra compile options:
Extra link options:
Message Loader: inmem
Net Accessor: fileonly
Transcoder: native
Thread option:
Debug is OFF

creating cache ./config.cache
checking for gcc... gcc
```

```
 checking whether the C compiler (gcc -O -DXML_USE_NATIVE_TRANSCODER
-DXML_USE_INMEM_MESSAGELOADER   ) works... yes
 checking whether the C compiler (gcc -O -DXML_USE_NATIVE_TRANSCODER
-DXML_USE_INMEM_MESSAGELOADER   ) is a cross-compiler... no
 checking whether we are using GNU C... yes
 checking whether gcc accepts -g... yes
 checking for c++... g++
 checking whether the C++ compiler (g++ -O -DXML_USE_NATIVE_TRANSCODER
-DXML_USE_INMEM_MESSAGELOADER   ) works... yes
 checking whether the C++ compiler (g++ -O -DXML_USE_NATIVE_TRANSCODER
-DXML_USE_INMEM_MESSAGELOADER   ) is a cross-compiler... no
 checking whether we are using GNU C++... yes
 checking whether g++ accepts -g... yes
 checking for a BSD compatible install... /usr/bin/install -c
 checking for autoconf... autoconf
 checking for floor in -lm... yes
 checking how to run the C preprocessor... gcc -E
 checking for ANSI C header files... yes
 checking for XMLByte... no
 checking host system type... i686-pc-linux-gnu
 updating cache ./config.cache
 creating ./config.status
 creating Makefile
 creating util/Makefile
 creating util/Transcoders/ICU/Makefile
 creating util/Transcoders/Iconv/Makefile
 creating util/Transcoders/Iconv390/Makefile
 creating util/Transcoders/Iconv400/Makefile
 creating util/Platforms/Makefile
 creating util/Compilers/Makefile
 creating util/MsgLoaders/InMemory/Makefile
 creating util/MsgLoaders/ICU/Makefile
 creating util/MsgLoaders/MsgCatalog/Makefile
 creating util/MsgLoaders/MsgFile/Makefile
 creating validators/DTD/Makefile
 creating framework/Makefile
 creating dom/Makefile
 creating parsers/Makefile
 creating internal/Makefile
 creating sax/Makefile
 creating ../obj/Makefile
 creating conf.h
 cat: ./conf.h.in: No such file or directory
 conf.h is unchanged

 Having build problems? Read instructions at
http://xml.apache.org/xerces-c/build.html
 Still cannot resolve it? Find out if someone else had the same problem before.
 Go to http://xml-archive.webweaving.org/xml-archive-xerces/

 In future, you may also directly type the following commands to create the
Makefiles.

 export TRANSCODER=NATIVE
 export MESSAGELOADER=INMEM
 export USELIBWWW=0
 export CC=gcc
 export CXX=g++
```

```
export CXXFLAGS=-O -DXML_USE_NATIVE_TRANSCODER -DXML_USE_INMEM_MESSAGELOADER
export CFLAGS=-O -DXML_USE_NATIVE_TRANSCODER -DXML_USE_INMEM_MESSAGELOADER
export LIBS= -lpthread
configure

If the result of the above commands look OK to you, go to the directory
XERCESCROOT and type "gmake" to make the XERCES-C system.
```

> ***Note:*** *The error message concerning `conf.h` is NOT an indication of a problem. This code has been inserted to make it work on AS/400, but it gives this message which appears to be an error. The problem will be fixed in future.*

So now you see what the wrapper script has actually been doing! It has invoked `configure` to create the Makefiles in the individual sub-directories, but in addition to that, it has set a few environment variables to correctly configure your compiler and compiler flags too.

Now that the Makefiles are all created, you are ready to do the actual build.

```
gmake
```

Is that it? Yes, that's all you need to build Xerces-C.

Building samples

Similarly, you can build the samples by giving the same commands in the `samples` directory.

```
cd $XERCESCROOT/samples
runConfigure -plinux -cgcc -xg++
gmake
```

The samples get built in the `bin` directory. Before you run the samples, you must make sure that your library path is set to pick up libraries from `$XERCESCROOT/lib`. If not, type the following to set your library path properly.

```
export LD_LIBRARY_PATH=$XERCESCROOT/lib:$LD_LIBRARY_PATH
```

You are now set to run the sample applications.

Building Xerces-C on Windows using Visual Age C++

A few unsupported projects are also packaged with Xerces-C. Due to origins of Xerces-C inside IBM labs, we do have projects for IBM's Visual Age C++ compiler on Windows. The following describes the steps you need to build Xerces-C using Visual Age C++.

Building Xerces-C library

Requirements:

- VisualAge C++ Version 4.0 with Fixpak 1:
  Download the Fixpak from the IBM VisualAge C++ Corrective Services web page.

To include the ICU library:

- ICU Build:
  You should have the ICU Library in the same directory as the Xerces-C library. For example if Xerces-C is at the top level of the d drive, put the ICU library at the top level of d e.g. d:/xml4c, d:/icu.

Instructions:

1. Change the directory to d:\xml4c\Projects\Win32
2. If a d:\xml4c\Project\Win32\VACPP40 directory does not exist, create it.
3. Copy the IBM VisualAge project file, `XML4C2X.icc`, to the VACPP40 directory.
4. From the VisualAge main menu enter the project file name and path.
5. When the build finishes the status bar displays this message: Last Compile completed Successfully

with warnings on date.

Note: *These instructions assume that you install in drive d:\. Replace d with the appropriate drive letter.*

Building on OS/2 using Visual Age C++

OS/2 is a favourite IBM PC platforms. The only option in this platform is to use Visual Age C++ compiler. Here are the steps you need to build Xerces-C using Visual Age C++ on OS/2.

Building Xerces-C library

Requirements:

- VisualAge C++ Version 4.0 with Fixpak 1:
  Download the Fixpak from the IBM VisualAge C++ Corrective Services web page.

To include the ICU library:

- ICU Build:
  You should have the ICU Library in the same directory as the Xerces-C library. For example if Xerces-C is at the top level of the d drive, put the ICU library at the top level of d e.g. d:/xml4c, d:/icu.

Instructions

1. Change directory to d:\xml4c\Projects\OS2
2. If a d:\xml4c\Project\OS2\VACPP40 directory does not exist, create it.
3. Copy the IBM VisualAge project file, XML4C2X.icc, to the VACPP40 directory.
4. From the VisualAge main menu enter the project file name and path.
5. When the build finishes the status bar displays this message: Last Compile completed Successfully with warnings on date.

   Note: *These instructions assume that you install in drive d:\. Replace d with the appropriate drive letter.*

Building on AS/400

The following addresses the requirements and build of Xerces-C natively on the AS/400.

Building Xerces-C library

Requirements:

- `QSHELL` interpreter installed (install base option 30, operating system)
- QShell Utilities, PRPQ 5799-XEH
- ILE C++ for AS/400, PRPQ 5799-GDW
- GNU facilities (the gnu facilities are currently available by request only. Send e-mail to rchgo400@us.ibm.com)

Recommendations:

- There are a couple of options when building the XML4C parser on AS/400. For messaging support, you can use the in memory message option or the message file support. For code page translation, you can use the AS/400 native `Iconv400` support or ICU. If you choose ICU, follow the instructions to build the ICU service program with the ICU download. Those instructions are not included here.
- Currently we recommend that you take the options of `MsgFile` and `Iconv400` (see below)

Setup Instructions:

- Make sure that you have the requirements installed on your AS/400. We highly recommend that you read the writeup that accompanies the gnu facilities download. There are install instructions as well as information about how modules, programs and service programs can be created in Unix-like fashion using gnu utilities. Note that symbolic links are use in the file system to point to actual AS/400 `*module`, `*pgm` and `*srvpgm` objects in libraries.
- Download the tar file (unix version) to the AS/400 (using a mapped drive), and decompress and `untar` the source. We have had difficulty with the tar command on AS/400. This is under

investigation. If you have trouble, we recommend the following work around:

```
qsh:
gunzip -d <tar file.gz>
pax -r -f <uncompressed tar file>
```

- Create AS400 target library. This library will be the target for the resulting modules and Xerces-C service program. You will specify this library on the `OUTPUTDIR` environment variable in step 4
- Set up the following environment variables in your build process (use `ADDENVVAR` or `WRKENVVAR CL` commands):

```
XERCESCROOT - <the full path to your Xerces-C sources>
PLATFORM  - 'OS400'
MAKE   - '/usr/bin/gmake'
OUTPUTDIR  - <identifies target as400 library for *module, *pgm and *srvpgm
objects>
ICUROOT - (optional if using ICU)  <the path of your ICU includes>
```

- Add `QCXXN`, to your build process library list. This results in the resolution of `CRTCPPMOD` used by the `icc` compiler.
- The runConfigure instruction below uses `'egrep'`. This is not on the AS/400 but you can create it by doing the following: `edtf '/usr/bin/egrep'` with the following source:

```
#!/usr/bin/sh
/usr/bin/grep -e "$@"
```

You may want to put the environment variables and library list setup instructions in a `CL` program so you will not forget these steps during your build.

Configure

To configure the make files for an AS/400 build do the following:

```
qsh
cd <full path to Xerces-C>/src
runConfigure -p os400 -x icc -c icc -m MsgFile -t Iconv400
```

Troubleshooting:

```
error: configure: error: installation or configuration problem:
C compiler cannot create executables.
```

If during `runConfigure` you see the above error message, it can mean one of two things. Either `QCXXN` is not on your library list **OR** the `runConfigure` cannot create the temporary modules `CONFTest1`, etc) it uses to test out the compiler options. The second reason happens because the test modules already exist from a previous run of `runConfigure`. To correct the problem, do the following:

```
DLTMOD <your OUTPUTDIR library>/CONFT* and
DLTPGM your <OUTPUTDIR library>/CONFT*
```

Build

```
qsh
gmake -e
```

The above gmake will result in a service program being created in your specified library and a symbolic link to that service program placed in <path to Xerces-C/lib>. You can either bind your XML application programs directly to the parser's service program via the `BNDSRVPGM` option on the `CRTPGM` or `CRTSRVPGM` command or you can specify a binding directory on your `icc` command. To specify an

archive file to bind to, use the `-L, -l` binding options on icc. An archive file on AS/400 is a binding directory. To create an archive file, use `qar` command. (see the gnu facilities write up).

After building the Xerces-C service program, create a binding directory by doing the following (note, this binding directory is used when building the samples):

```
 qsh
 cd <full path to Xerces-C>/lib>
 qar -cuv libxercesc1_1.a *.o
 command = CRTBNDDIR BNDDIR(yourlib/libxercesc)
TEXT('/yourlib/Xerces-C/lib/libxercesc1_1.a')
 command = ADDBNDDIRE BNDDIR(yourlib/libxercesc) OBJ((yourlib/LIBXERCESC *SRVPGM)
)
```

Troubleshooting:

If you are on a V4R3 system, you will get a bind problem `'descriptor QlgCvtTextDescToDesc not found'` using Iconv400. On V4R3 the system doesn't automatically pick up the `QSYS/QLGUSR` service program for you when resolving this function. This is not the case on V4R4. To fix this, you can either manually create the service program after creating all the resulting modules in your <OUTPUTDIR> library or you can create a symbolic link to a binding directory that points to the `QLGUSR` service program and then specify an additional `-L, -l` on the `EXTRA_LINK_OPTIONS` in `Makefile.incl`. See the `ln` and `qar` function in the gnu utilities.

To build for transcoder ICU:

1. Make sure you have an `ICUROOT` path set up so that you can find the ICU header files (usually `/usr/local`)
2. Make sure you have created a binding directory (symbolic link) in the file system so that you can bind the Xerces-C service program to the ICU service program and specify that on the `EXTRA_LINK_OPTIONS` in `src/Makefile.incl` (usually the default is a link in `/usr/local/lib`).

Creating AS400 XML parser message file:

As specified earlier, the `-m` MsgFile support on the `runConfigure` enable the parser messages to be pulled from an AS/400 message file. To view the source for creating the message file and the XML parser messages, see the following stream file:

```
EDTF <full path to Xerces-C>/src/util/MsgLoaders/MsgFile/CrtXMLMsgs
```

In the prolog of `CrtXMLMsgs` there are instructions to create the message file:

1. Use the `CPYFRMSTMF` to copy the CL source to an AS/400 source physical file. Note that the target source file needs to have record length of about 200 bytes to avoid any truncation.
2. Create the CL program to create the message file and add the various message descriptions
3. Call the CL program, providing the name of the message file (use `QXMLMSG` as default) and a library (this can be any library, including any product library in which you wish to embed the xml parser)

Note that the Xerces-C source code for resolving parser messages is using by default message file `QXMLMSG, *LIBL`. If you want to change either the message file name or explicitly qualify the library to match your product needs, you must edit the following `.cpp` files prior to your build.

```
<full path to Xerces-C>/src/util/MsgLoaders/MsgFile/MsgLoader.cpp
<full path to Xerces-C>/src/util/Platforms/OS400/OS400PlatformUtils.cpp
```

Troubleshooting:

If you are using the parser and are failing to get any message text for error codes, it may be because of the `*LIBL` resolution of the message file.

Building Samples on AS/400

```
qsh
cd <full path to Xerces-C>/samples
runConfigure -p os400 -x icc -c icc
gmake -e
```

Troubleshooting:

If you take a `'sed'` error, while trying to make the samples. This is an AS400 anomaly having to do with certain new line character and the `sed` function. A temporary work around is to use `EDTF` on the configure stream file (`../samples/configure`) and delete the following line near the bottom: `s%@DEFS@%$DEFS%g`.

Building on Macintosh using CodeWarrior

Building Xerces-C library

The directions in this file cover installing and building Xerces-C and ICU under the MacOS using CodeWarrior.

1. **Create a folder:**
   for the Xerces-C and ICU distributions, the "src drop" folder
2. **Download and uncompress:**
   the ICU and Xerces-C source distribution
   the ICU and Xerces-C binary distributions, for the documentation included
3. **Move the new folders:**
   move the newly created Xerces-C and icu124 folders to the "src drop" folder.
4. **Drag and drop:**
   the Xerces-C folder into the "rename file" application located in the same folder as this readme.
   This is a MacPerl script that renames files that have names too long to fit in a HFS/HFS+ filesystem. It also searches through all of the source code and changes the #include statements to refer to the new file names.
5. **Move the MacOS folder:**
   from the in the Projects folder to "src drop:Xerces-C:Projects".
6. **Open and build Xerces-C:**
   open the CodeWarrior project file "src drop:Xerces-C:Projects:MacOS:Xerces-C:Xerces-C" and build the Xerces-C library.
7. **Open and build ICU:**
   open the CodeWarrior project file "src drop:Xerces-C:Projects:MacOS:icu:icu" and build the ICU library.
8. **Binary distribution:**
   If you wish, you can create projects for and build the rest of the tools and test suites. They are not needed if you just want to use Xerces-C. I suggest that you use the binary data files distributed with the binary distribution of ICU instead of creating your own from the text data files in the ICE source distribution.

There are some things to be aware of when creating your own projects using Xerces-C.

1. You will need to link against both the ICU and Xerces-C libraries.
2. The options "Always search user paths" and "Interpret DOS and Unix Paths" are very useful. Some of the code won't compile without them set.
3. Most of the tools and test code will require slight modification to compile and run correctly (typecasts, command line parameters, etc), but it is possible to get them working correctly.
4. You will most likely have to set up the Access Paths. The access paths in the Xerces-C projects should serve as a good example.

> ***Note:*** *These instructions were originally contributed by J. Bellardo. Xerces-C has undergone many changes since these instructions were written. So, these instructions are not upto date. But it will give you a jump start if you are struggling to get it to work for the first time. We will be glad to get your changes. Please respond to xerces-c-dev@xml.apache.org with your comments and corrections.*

How to Build ICU

As mentioned earlier, Xerces-C may be built in stand-alone mode using native encoding support and also using ICU where you get support for 100's of encodings. ICU stands for International Components for Unicode and is an open source distribution from IBM. You can get ICU libraries from IBM's developerWorks site or go to the ICU download page directly.

Buiding ICU for Xerces-C

You can find generic instructions to build ICU in the ICU documentation. What we describe below are the minimal steps needed to build ICU for Xerces-C. Not all ICU components need to be built to make it work with Xerces-C.

> ***Note: Important:*** *Please remember that **ICU and Xerces-C must be built with the same compiler**, preferably with the same version. You cannot for example, build ICU with a threaded version of the xlC compiler and build Xerces-C with a non-threaded one.*

Building ICU on Windows

To build ICU from its source, invoke the project `\icu\source\allinone\allinone.dsw` and build the sub-project labeled `all`.

You must make sure that you are linking your application with the xerces-c_1.lib library and also make sure that the associated Xerces-C DLL is somewhere in your path. Note that at runtime, your application will need the ICU data DLL called `icudata.dll` which must also be available from your path setting.

Building ICU on UNIX platforms

To build ICU on all UNIX platforms you at least need the `autoconf` tool and GNU's `gmake` utility.

First make sure that you have defined the following environment variables:

```
    export ICUROOT = <icu_installdir>
```

Next, go to the directory, the following commands will create a shell script called `configure`:

```
    cd $ICUROOT
    cd source
    autoconf
```

Commands for specific UNIX platforms are different and are described separately below.

You will get a more detailed description of the use of configure in the ICU documentation. The differences lie in the arguments passed to the configure script, which is a platform-independent generated shell-script (through `autoconf`) and is used to generate platform-specific `Makefiles` from generic `Makefile.in` files.

For AIX:

Type the following:

```
    env CC="xlc_r -L/usr/lpp/xlC/lib" CXX="xlC_r -L/usr/lpp/xlC/lib"
        C_FLAGS="-w -O" CXX_FLAGS="-w -O"
    configure --prefix=$ICUROOT
    gmake
    gmake install
```

The first line is different for different platforms as outlined below:

For Solaris and Linux:

```
env CC="cc" CXX="CC" C_FLAGS="-w -O" CXX_FLAGS="-w -O"
    ./configure --prefix=$ICUROOT
```

For HP-UX with the aCC compiler:

```
env CC="cc" CXX="aCC" C_FLAGS="+DAportable -w -O"
    CXX_FLAGS="+DAportable -w -O" ./configure --prefix=$ICUROOT
```

For HP-UX with the CC compiler:

```
env CC="cc" CXX="CC" C_FLAGS="+DAportable -w -O"
    CXX_FLAGS="+eh +DAportable -w -O" ./configure --prefix=$ICUROOT
```

How to build XML for COM on Windows

To build the COM module for use with XML on Windows platforms, you must first set up your machine appropriately with necessary tools and software modules and then try to compile it. The end result is an additional library that you can use along with the standard Xerces-C for writing VB templates or for use with IE 5.0 using JavaScript.

Setting up your machine for COM

To build the COM project you will need to install the MS PlatformSDK. Some of the header files we use don't come with Visual C++ 6.0. You may download it from Microsoft's Website at http://www.microsoft.com/msdownload/platformsdk/setuplauncher.htm or directly FTP it from ftp://ftp.microsoft.com/developr/PlatformSDK/April2000/Msi/WinNT/x86/InstMsi.exe.

The installation is huge, but you don't need most of it. So you may do a **custom install** by just selecting "Build Environment" and choosing the required components. First select the top level Platform SDK. Then click the down arrow and make all of the components unavailable. Next open the "Build Environment" branch and select only the following items:

- Win32 Build Environment
- COM Headers and Libraries
- Internet Explorer Headers and Libraries

Important: When the installation is complete you need to update VC6's include path to include `..\platformsdk\include\atl30`. You do this by choosing "Tools -> Options -> Directories". This path should be placed second after the normal PlatformSDK include. You change the order of the paths by clicking the up and down arrows.

> ***Note:*** *The order in which the directories appear on your path is important. Your first include path should be `..\platformsdk\include`. The second one should be `..\platformsdk\include\atl30`.*

Building COM module for Xerces-C

Once you have set up your machine, build Xerces-C COM module by choosing the project named 'xml4com' inside the workspace. Then select your build mode to be **xml4com - Win32 Release MinDependency**. Finally build the project. This will produce a DLL named `xerces-com.dll` which needs to be present in your path (on local machine) before you can use it.

Testing the COM module

There are some sample test programs in the `test/COMTest` directory which show examples of navigating and searching an XML tree using DOM. You need to browse the HTML files in this directory using IE 5.0. Make sure that your build has worked properly, specially the registration of the ActiveX controls that happens in the final step.

How to build the User Documentation?

The user documentation (this very page that you are reading on the browser right now), was generated using an XML application called StyleBook. This application makes use of Xerces-J and Xalan to create the HTML file from the XML source files. The XML source files for the documentation are part of the Xerces-C module. These files reside in the `doc` directory.

Pre-requisites for building the user documentation are:

- JDK 1.2.2 (or later).
- Xerces-J (1.0.0 or later).
- Xalan (0.19.3 or later)
- Stylebook 1.0-b2
- The Apache Style files (dtd's and .xsl files)

Setup PATH to include the JDK 1.2.2 bin directory. Also setup CLASSPATH environment variable as follows:

- Under Windows (assumes all jars are in '\jars' directory:)
  ```
  CLASSPATH=\jars\stylebook-1.0-b2.jar;\jars\xalan.jar;\jars\xerces.jar
  ```
- Under Unix's (assumes all jars are in '~/jars' directory):
  ```
  export
  CLASSPATH="~/jars/stylebook-1.0-b2.jar:~/jars/xalan.jar:~/jars/xerces.jar"
  ```

Next, cd to the Xerces-C source drop root directory, and enter

- Under Windows:
  ```
  createDocs
  ```
- Under Unix's:
  ```
  sh createDocs.bat
  ```

This should generate the .html files in the 'doc/html' directory.

If you are wondering where to get the three `jar` files referred above, here is where you would find it.

- JDK 1.2.2 is available from http://java.sun.com/products/jdk/1.2/
- Xerces-J is available from http://xml.apache.org/dist/. Extract the xerces.jar file from the binary drop and store it in the 'jars' directory as mentioned above.
- Xalan is also available from http://xml.apache.org/dist/. Extract the xalan.jar file from the 'jar' distribution that you just downloaded and store it in the same 'jars' directory as mentioned above.
- Getting to Stylebook is little more involved. You will have to download one of the 'xml-stylebook' tar balls from http://xml.apache.org/from-cvs/xml-stylebook/ and then extract the file:
  ```
  xml-stylebook/bin/stylebook-1.0-b2.jar
  ```
  Under Unix's you may enter:
  ```
  gzip -d -c xml-stylebook_20000207231311.tar.gz | tar xf -
  xml-stylebook/bin/stylebook-1.0-b2.jar
  ```
  to extract this file (in this gzip command, subsitute the tar file name with the one you downloaded). Copy it to the 'jars' directory as mentioned above.

  Under Windows you may use 'WinZip' to extract the jar file from the tar ball.
- Lastly, the Apache Style (dtd's and .xsl) files reside in the same 'stylebook' tar ball, as described above. The script `createdocs.bat` assumes that these styles are installed relative to where it is located in the `../../xml-stylebook/styles/apachexml` directory. If the directory structure on your build machine differs, you can edit this script file to reflect the difference. To extractt the Apache style files enter:
  ```
  cd <parent of Xerces-C source directory>
  gzip -d -c xml-stylebook_20000207231311.tar.gz | tar xf -
  xml-stylebook/styles
  ```

I wish to port Xerces to my favourite platform. Do you have any suggestions?

All platform dependent code in Xerces has been isolated to a couple of files, which should ease the porting effort. Here are the basic steps that should be followed to port Xerces.

1. The directory `src/util/Platforms` contains the platform sensitive files while `src/util/Compilers` contains all development environment sensitive files. Each operating system has a file of its own and each development environment has another one of its own too.

   As an example, the Win32 platform as a `Win32Defs.hpp` file and the Visual C++ environment has a `VCPPDefs.hpp` file. These files set up certain define tokens, typedefs, constants, etc... that will drive the rest of the code to do the right thing for that platform and development environment. AIX/CSet have their own `AIXDefs.hpp` and `CSetDefs.hpp` files, and so on. You should create new versions of these files for your platform and environment and follow the comments in them to set up your own. Probably the comments in the Win32 and Visual C++ will be the best to follow, since that is where the main development is done.
2. Next, edit the file `XML4CDefs.hpp`, which is where all of the fundamental stuff comes into the system. You will see conditional sections in there where the above per-platform and per-environment headers are brought in. Add the new ones for your platform under the appropriate conditionals.
3. Now edit `AutoSense.hpp`. Here we set canonical Xerces internal `#define` tokens which indicate the platform and compiler. These definitions are based on known platform and compiler defines.

   `AutoSense.hpp` is included in `XML4CDefs.hpp` and the canonical platform and compiler settings thus defined will make the particular platform and compiler headers to be the included at compilation.

   It might be a little tricky to decipher this file so be careful. If you are using say another compiler on Win32, probably it will use similar tokens so that the platform will get picked up already using what is already there.
4. Once this is done, you will then need to implement a version of the platform utilities for your platform. Each operating system has a file which implements some methods of the XMLPlatformUtils class, specific to that operating system. These are not terribly complex, so it should not be a lot of work. The Win32 verions is called `Win32PlatformUtils.cpp`, the AIX version is `AIXPlatformUtils.cpp` and so on. Create one for your platform, with the correct name, and empty out all of the implementation so that just the empty shells of the methods are there (with dummy returns where needed to make the compiler happy.) Once you've done that, you can start to get it to build without any real implementation.
5. Once you have the system building, then start implementing your own platform utilties methods. Follow the comments in the Win32 version as to what they do, the comments will be improved in subsequent versions, but they should be fairly obvious now. Once you have these implementations done, you should be able to start debugging the system using the demo programs.

That is the work required in a nutshell!

What should I define XMLCh to be?

The answer is 'it depends'. We will mention some of the quirks that affect this decision. Hopefully, after reading whats below, you will be able to best decide what the right definition should be. We could not however, resist making a suggestion. Some observations first:

- Xerces-C uses XMLCh as the fundamental type to hold one Unicode character as, all processing inside Xerces-C happens in Unicode.
- Most modern C++ compilers today provide 'wchar_t' as a fundamental type representing a 'wide character'. Most of them define it in using a typedef. This typedef definition is not consistent on all the platforms that we have come across.
- The size of wchar_t varies among the various compilers. Its either 16-bit or 32-bit. Fortunately, this only affects how much memory you need, to process the XML data, while everything is still in memory.
- Again on most platforms wchar_t represents a unicode character. HPUX, is one exception to this, that we know, where wchar_t **does not** represent a unicode character, rather its a native wide character.

- Lastly, most OS's/compilers provide a system library to manipulate wide character strings taking wchar_t and wchar_t* arguments. Most applications which support wide-characters make these system calls.

Our suggestion is:

If your compiler defines wchar_t to represent a unicode character, then define XMLCh to be wchar_t. Such a definition will allow you to pass the data returned by the parser (all api's return XMLCh, which is wchar_t) directly to the wide-character system api's for i/o or manipulation. This is most efficient and convenient.

However, if your compiler defines wchar_t to be just a wide-character which is not Unicode, then define XMLCh to be unsigned short. For the Xerces-C parser, XMLCh is always Unicode. By defining it to be unsigned short and not wchar_t, the compiler will not let you accidently pass what is returned, via the parser API's, directly to the wide-character library calls. To use the wide-character library of functions, you will have to in your application, call some transcoding function which will convert it from Unicode to the native wide-character form. Again, if your application desires for whatever reason, you may define XMLCh to be 'unsigned long'. By doing so, you have just doubled the memory required to process the XML file.

Hopefully, you will agree that the answer 'it depends' was the right one.

How can I generate Xerces-C binaries which includes the sample NetAccessor implementation using Libwww?

This sample implementation has only been minimally tested only under Windows NT using Libwww 5.2.8. We have not stress tested our implementation can cannot guarantee that there are no memory leaks. The error reporting is also not adequate. Further, it only handles HTTP style URL's. As you can see, this implementation is only for illustrative purposes. Much more work is required to have a robust cross-platform implementation. We would welcome any volunteers who would contribute code to make this happen on various platforms.

The software that you need are:

- You need the Xerces-C source archive for Windows.
- LibWWW 5.2.8. Win32 binaries are available at: http://www.idm.ru/libwww.htm. Source archives and other details on LibWWW are available at http://www.w3.org/Library/.

All required changes in Xerces-C are restricted to the Project file settings for the XercesLib. To simplify, we will make certain assumptions about how LibWWW binaries (.lib) and header files are installed on your machine.

1. First generate all the LibWWW binaries by using the project file supplied. Create a top level (say) `\libWWW` directory on the same disk drive where you installed the Xerces-C sources. Copy all the `.lib` files to `\libWWW\lib` directory. Next, copy all the `.dll` files to `\libWWW\bin` directory and all the header (`*.h`) files to `\libWWW\include` directory.
2. Next make the following changes to the Xerces-C lib project settings. Invoke the project settings dialog box.
   1. In the 'C/C++ : Preprocessor : Preprocessor definitions' add `XML_USE_NETACCESSOR_LIBWWW`
   2. In the 'C/C++ : Preprocessor : Additional include directories' add `\libWWW\include`.
3. Next, rather than listing all the 20 some LibWWW .lib files in the link settings, add them as external files to the XercesLib project. Right-Click on 'XercesLib files' and choose the 'Add Files to Project' menu item. Next choose all the *.lib files in \libWWW\lib directory and press 'ok'.
4. Next, create a new sub-folder in XercesLib:util folder, by right-clicking on 'util' and choosing 'New Folder'. Call it 'libWWW'.
5. Add netaccessor files into this 'libWWW' folder again, by right-clicking on 'libWWW' folder and choosing 'Add Files to Folder'. Choose the four files in `\src\util\NetAccessors\libWWW` directory. These files are: `BinURLInputStream.[ch]pp and LibWWWNetAccessor.[ch]pp`.
6. Rebuild the Xerces-C library.

Make sure you have `\libWWW\bin` in your `PATH` environment variable, before you run the samples and refer to a XML file containing HTTP URL's to remote resources.

Where can I look for more help?

If you have read this page, followed the instructions, and still cannot resolve your problem(s), there is more help. You can find out if others have solved this same problem before you, by checking the Xerces mailing list archives.

If all else fails, you may ask for help by subscribing to the Xerces-C mailing list.

4
API Documentation

API Docs for SAX and DOM

Xerces-C is packaged with the API documentation for SAX and DOM, the two most common programming interfaces for XML. The most common framework classes have also been documented.

Xerces-C DOM is an implementation of the Document Object Model (Core) Level 1 as defined in the W3C Recommendation of 1 October, 1998. For a complete understanding of how the Xerces-C APIs work, we recommend you to read the DOM Level 1 specification.

Xerces-C SAX is an implementation of the SAX 1.0 specification. You are encouraged to read this document for a better understanding of the SAX API in Xerces-C.

Click here for the Xerces-C API documentation.

Note: *The API documentation is automatically generated using doxygen and GraphViz.*

5
Xerces-C Samples

Building the Samples

Xerces-C comes packaged with ten sample applications that demonstrate salient features of the parser using simple applications written on top of the SAX and DOM APIs provided by the parser.

Once you have set up your PATH variable, you can run the samples by opening a command window (or your shell prompt for UNIX environments). Sample XML data files are provided in the samples/data directory.

The installation process for the samples is same on all UNIX platforms. Note that **runConfigure** is just a helper script and you are free to use **./configure** with the correct parameters to make it work on any platform-compiler combination of your choice. The script needs the following parameters:

```
Usage: runConfigure "options"
       where options may be any of the following:
       -p <platform> (accepts 'aix', 'linux', 'solaris', 'hp-10', 'hp-11')
       -c <C compiler name> (e.g. gcc, xlc_r, cc or aCC)
       -x <C++ compiler name> (e.g. g++, xlC_r, CC or aCC)
       -d (specifies that you want to build debug version)
       -h (get help on the above commands)
```

Note: NOTE:*The code samples in this section assume that you are are working on the Linux binary drop. If you are using some other UNIX flavor, please replace '-linux' with the appropriate platform name in the code samples.*

Running the Samples

The sample applications are dependent on the Xerces-C shared library (and could also depend on the ICU library if you built Xerces-C with ICU). Therefore, on Windows platforms you must make sure that your `PATH` environment variable is set properly to pick up these shared libraries at runtime.

On UNIX platforms you must ensure that LIBPATH environment variable is set properly to pick up the shared libraries at runtime. (UNIX gurus will understand here that LIBPATH actually translates to **LD_LIBRARY_PATH** on Solaris and Linux, **SHLIB_PATH** on HP-UX and stays as **LIBPATH** on AIX).

To set you LIBPATH (on AIX for example), you would type:

```
export LIBPATH=xerces-c-1_1_0/lib:$LIBPATH
```

Xerces-C Samples

- SAXCount

  SAXCount counts the elements, attributes, spaces and characters in an XML file.
- SAXPrint

  SAXPrint parses an XML file and prints it out.

- DOMCount
  DOMCount counts the elements in a XML file.
- DOMPrint
  DOMPrint parses an XML file and prints it out.
- MemParse
  MemParse parses XML in a memory buffer, outputing the number of elements and attributes.
- Redirect
  Redirect redirects the input stream for external entities.
- PParse
  PParse demonstrates progressive parsing.
- StdInParse
  StdInParse demonstrates streaming XML data from standard input.
- EnumVal
  EnumVal shows how to enumerate the markup decls in a DTD Validator.
- CreateDOMDocument
  CreateDOMDocument creates a DOM tree in memory from scratch.

Xerces-C Sample 1

SAXCount

SAXCount is the simplest application that counts the elements and characters of a given XML file using the (event based) SAX API.

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked SAXCount.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd SAXCount
gmake
```

This will create the object files in the current directory and the executable named SAXCount in 'xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running SAXCount

The SAXCount sample parses an XML file and prints out a count of the number of elements in the file. To run SAXCount, enter the following

```
SAXCount <XML File>
```

To use the validating parser, use

```
SAXCount -v <XML file>
```

Here is a sample output from SAXCount

```
cd xerces-c-1_1_0-linux/samples/data
SAXCount -v personal.xml
personal.xml: 60 ms (37 elems, 12 attrs, 134 spaces, 134 chars)
```

Running SAXCount with the validating parser gives a different result because ignorable white-space is counted separately from regular characters.

```
SAXCount personal.xml
personal.xml: 10 ms (37 elems, 12 attrs, 0 spaces, 268 chars)
```

Note that the sum of spaces and chracters in both versions is the same.

> ***Note:*** *The time reported by the program may be different depending on your machine processor.*

Xerces-C Sample 2

SAXPrint

SAXPrint uses the SAX APIs to parse an XML file and print it back. Do note that the output of this sample is not exactly the same as the input (in terms of whitespaces, first line), but the output has the same information content as the input.

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked SAXPrint.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd SAXPrint
gmake
```

This will create the object files in the current directory and the executable named SAXPrint in 'xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running SAXPrint

The SAXPrint sample parses an XML file and prints out the contents again in XML (some loss occurs). To run SAXPrint, enter the following

```
SAXPrint <XML file>
```

To use the validating parser, use

```
SAXPrint -v <XML file>
```

Here is a sample output from SAXPrint

```
cd xerces-c-1_1_0-linux/samples/data
SAXPrint -v personal.xml

<personnel>

  <person id="Big.Boss">
    <name><family>Boss</family> <given>Big</given></name>
    <email>chief@foo.com</email>
    <link subordinates="one.worker two.worker three.worker
                        four.worker five.worker"></link>
  </person>

  <person id="one.worker">
    <name><family>Worker</family> <given>One</given></name>
    <email>one@foo.com</email>
    <link manager="Big.Boss"></link>
  </person>

  <person id="two.worker">
    <name><family>Worker</family> <given>Two</given></name>
    <email>two@foo.com</email>
    <link manager="Big.Boss"></link>
```

```
    </person>

    <person id="three.worker">
      <name><family>Worker</family> <given>Three</given></name>
      <email>three@foo.com</email>
      <link manager="Big.Boss"></link>
    </person>

    <person id="four.worker">
      <name><family>Worker</family> <given>Four</given></name>
      <email>four@foo.com</email>
      <link manager="Big.Boss"></link>
    </person>

    <person id="five.worker">
      <name><family>Worker</family> <given>Five</given></name>
      <email>five@foo.com</email>
      <link manager="Big.Boss"></link>
    </person>

  </personnel>
```

Note: *SAXPrint does not reproduce the original XML file. Also SAXPrint and DOMPrint produce different results because of the way the two APIs store data and capture events.*

Xerces-C Sample 3

DOMCount

DOMCount uses the provided DOM API to parse an XML file, constructs the DOM tree and walks through the tree counting the elements (using just one API call).

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked DOMCount.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd DOMCount
gmake
```

This will create the object files in the current directory and the executable named DOMCount in ' xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running DOMCount

The DOMCount sample parses an XML file and prints out a count of the number of elements in the file. To run DOMCount, enter the following

```
DOMCount <XML file>
```

To use the validating parser, use

```
DOMCount -v <XML file>
```

Here is a sample output from DOMCount

```
cd xerces-c-1_1_0-linux/samples/data
DOMCount -v personal.xml
personal.xml: 20 ms (37 elems)
```

The output of both versions should be same.

> ***Note:*** *The time reported by the system may be different, depending on your processor type.*

Xerces-C Sample 4

DOMPrint

DOMPrint parses an XML file, constructs the DOM tree, and walks through the tree printing each element. It thus dumps the XML back (output same as SAXPrint).

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked DOMPrint.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd DOMPrint
gmake
```

This will create the object files in the current directory and the executable named DOMPrint in 'xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running DOMPrint

The DOMPrint sample parses an XML file, using either a validating or non-validating DOM parser configuration, builds a DOM tree, and then walks the tree and outputs the contents of the nodes in a 'canonical' format. To run DOMPrint, enter the following:

```
DOMPrint [-v] <XML file>
```

The -v option is used when you wish to use a validating parser. Here is a sample output for DOMPrint when the validating parser is used:

```
cd xerces-c-1_1_0-linux/samples/data
DOMPrint -v personal.xml
```

Here is a sample output from DOMPrint

```
cd xerces-c-1_1_0-linux/samples/data
DOMPrint -v personal.xml

<?xml version='1.0' encoding='utf-8?>
<!-- Revision: 63 1.7 samples/data/personal.xml -->
<personnel>

<person id="Big.Boss">
  <name><family>Boss</family> <given>Big</given></name>
  <email>chief@foo.com</email>
  <link subordinates="one.worker two.worker three.worker
                      four.worker five.worker"></link>
</person>

<person id="one.worker">
  <name><family>Worker</family> <given>One</given></name>
  <email>one@foo.com</email>
  <link manager="Big.Boss"></link>
</person>
```

```
    <person id="two.worker">
      <name><family>Worker</family> <given>Two</given></name>
      <email>two@foo.com</email>
      <link manager="Big.Boss"></link>
    </person>

    <person id="three.worker">
      <name><family>Worker</family> <given>Three</given></name>
      <email>three@foo.com</email>
      <link manager="Big.Boss"></link>
    </person>

    <person id="four.worker">
      <name><family>Worker</family> <given>Four</given></name>
      <email>four@foo.com</email>
      <link manager="Big.Boss"></link>
    </person>

    <person id="five.worker">
      <name><family>Worker</family> <given>Five</given></name>
      <email>five@foo.com</email>
      <link manager="Big.Boss"></link>
    </person>

    </personnel>
```

Note that DOMPrint does not reproduce the original XML file. Also DOMPrint and SAXPrint produce different results because of the way the two APIs store data and capture events.

Xerces-C Sample 5

MemParse

MemParse uses the Validating SAX Parser to parse a memory buffer containing XML statements, and reports the number of elements and attributes found.

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked MemParse.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd MemParse
gmake
```

This will create the object files in the current directory and the executable named MemParse in ' xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running MemParse

This program uses the SAX Parser to parse a memory buffer containing XML statements, and reports the number of elements and attributes found.

```
MemParse [-v]
```

The -v option is used to invoke the Validating SAX Parser instead. When invoked with a validating parser:

```
cd xerces-c-1_1_0-linux/samples/data
MemParse -v
```

The output is the following:

```
Finished parsing the memory buffer containing the following XML statements:

<?xml version='1.0' encoding='ascii'?>
<!DOCTYPE company [
<!ELEMENT company     (product,category,developedAt)>
<!ELEMENT product     (#PCDATA)>
<!ELEMENT category    (#PCDATA)>
<!ATTLIST category idea CDATA #IMPLIED>
<!ELEMENT developedAt (#PCDATA)>
]>

<company>
  <product>Xerces-C</product>
  <category idea='great'>XML Parsing Tools</category>
  <developedAt>
    IBM Center for Java Technology, Silicon Valley, Cupertino, CA
  </developedAt>
</company>

Parsing took 0 ms (4 elements, 1 attributes, 16 spaces, 95 characters).
```

Xerces-C Sample 6

Redirect

Redirect uses the SAX EntityResolver handler to redirect the input stream for external entities. It installs an entity resolver, traps the call to the external DTD file and redirects it to another specific file which contains the actual DTD.

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked Redirect.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd Redirect
gmake
```

This will create the object files in the current directory and the executable named Redirect in 'xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running Redirect

This program illustrates how a XML application can use the SAX EntityResolver handler to redirect the input stream for external entities. It installs an entity resolver, traps the call to the external DTD file and redirects it to another specific file which contains the actual DTD.

The program then counts and reports the number of elements and attributes in the given XML file.

```
Redirect [-v] <XML file>
```

The -v option is used to invoke the Validating SAX Parser instead.

When invoked as follows:

```
cd xerces-c-1_1_0-linux/samples/data
Redirect -v personal.xml
```

The output is the following:

```
cd xerces-c-1_1_0-linux/samples/data
Redirect -v personal.xml
personal.xml: 30 ms (37 elems, 12 attrs, 134 spaces, 134 chars)
```

External files required to run this sample are 'personal.xml', 'personal.dtd' and 'redirect.dtd', which are all present in the 'samples/data' directory. Make sure that you run redirect in the samples/data directory.

The 'resolveEntity' callback in this sample looks for an external entity with system id as 'personal.dtd'. When it is asked to resolve this particular external entity, it creates and returns a new InputSource for the file 'redirect.dtd'.

A real-world XML application can similarly do application specific processing when encountering external entities. For example, an application might want to redirect all references to entities outside of its domain to local cached copies.

Xerces-C Sample 7

PParse

PParse demonstrates progressive parsing.

In this example, the programmer doesn't have to depend upon throwing an exception to terminate the parsing operation. Calling parseFirst() will cause the DTD to be parsed (both internal and external subsets) and any pre-content, i.e. everything up to but not including the root element. Subsequent calls to parseNext() will cause one more piece of markup to be parsed, and spit out from the core scanning code to the parser. You can quit the parse any time by just not calling parseNext() anymore and breaking out of the loop. When you call parseNext() and the end of the root element is the next piece of markup, the parser will continue on to the end of the file and return false, to let you know that the parse is done.

Building on Windows

Load the xerces-c-1_1_0win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked PParse.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd PParse
gmake
```

This will create the object files in the current directory and the executable named PParse in ' xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running PParse

The program looks for the first 16 elements of the XML file, and reports if successful.

```
PParse [-v] <XML file>
```

The output is the following:

```
Got the required 16 elements.
```

Xerces-C Sample 8

StdInParse

StdInParse demonstrates streaming XML data from standard input.

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked StdInParse.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd StdInParse
gmake
```

This will create the object files in the current directory and the executable named StdInParse in ' xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running StdInParse

The StdInParse sample parses an XML file and prints out a count of the number of elements in the file. To run StdInParse, enter the following:

```
StdInParse < <XML file>
```

Here is a sample output from StdInParse

```
cd xerces-c-1_1_0-linux/samples/data
StdInParse < personal.xml
personal.xml: 60 ms (37 elems, 12 attrs, 0 spaces, 268 chars)
```

Note: *The time reported by the program may be different depending on your machine processor.*

Xerces-C Sample 9

EnumVal

EnumVal shows how to enumerate the markup decls in a DTD Validator.

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked EnumVal.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd EnumVal
gmake
```

This will create the object files in the current directory and the executable named EnumVal in ' xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running EnumVal

This program parses a file, then shows how to enumerate the contents of the validator pools. To run EnumVal, enter the following

```
EnumVal <XML file>
```

Here is a sample output from EnumVal

```
cd xerces-c-1_1_0-linux/samples/data
EnumVal personal.xml

ELEMENTS:
----------------------------
  Name: personnel
  Content Model: (person)+

  Name: person
  Content Model: (name,email*,url*,link?)
  Attributes:
    Name:id, Type: ID

  Name: name
  Content Model: (#PCDATA|family|given)*

  Name: email
  Content Model: (#PCDATA)*

  Name: url
  Content Model: EMPTY
  Attributes:
    Name:href, Type: CDATA

  Name: link
  Content Model: EMPTY
  Attributes:
    Name:subordinates, Type: IDREF(S)
```

```
        Name:manager, Type: IDREF(S)

      Name: family
      Content Model: (#PCDATA)*

      Name: given
      Content Model: (#PCDATA)*
```

Xerces-C Sample 10

CreateDOMDocument

CreateDOMDocument, illustrates how you can create a DOM tree in memory from scratch. It then reports the elements in the tree that was just created.

Building on Windows

Load the xerces-c-1_1_0-win32\samples\Projects\Win32\VC6\samples.dsw Microsoft Visual C++ workspace inside your MSVC IDE. Then build the project marked DOMCount.

Building on UNIX

```
cd xerces-c-1_1_0-linux/samples
./runConfigure -p<platform> -c<C_compiler> -x<C++_compiler>
cd CreateDOMDocument
gmake
```

This will create the object files in the current directory and the executable named CreateDOMDocument in ' xerces-c-1_1_0-linux/bin' directory.

To delete all the generated object files and executables, type

```
gmake clean
```

Running CreateDOMDocument

The CreateDOMDocument sample illustrates how you can create a DOM tree in memory from scratch. To run CreateDOMDocument, enter the following

```
CreateDOMDocument
```

Here is a sample output from CreateDOMDocument

```
cd xerces-c-1_1_0-linux/samples/data
CreateDOMDocument
The tree just created contains: 4 elements.
```

6
Programming Guide

This page has sections on the following topics:

- SAX Programming Guide
  - Constructing a parser
  - Using the SAX API
- DOM Programming Guide
  - Comparision of Java and C++ DOM's
    - Accessing the API from application code
    - Class Names
    - Objects and Memory Management
  - DOMString
    - Equality Testing
  - Downcasting
  - Subclassing

SAX Programming Guide

Constructing a parser

In order to use Xerces-C to parse XML files, you will need to create an instance of the SAXParser class. The example below shows the code you need in order to create an instance of SAXParser. The DocumentHandler and ErrorHandler instances required by the SAX API are provided using the HandlerBase class supplied with Xerces-C.

```
int main (int argc, char* args[]) {

    try {
        XMLPlatformUtils::Initialize();
    }
    catch (const XMLException& toCatch) {
        cout << "Error during initialization! :\n"
             << toCatch.getMessage() << "\n";
        return 1;
    }

    char* xmlFile = "x1.xml";
    SAXParser* parser = new SAXParser();
    parser->setDoValidation(true);    // optional.
        parser->setDoNamespaces(true);    // optional

    DocumentHandler* docHandler = new HandlerBase();
    ErrorHandler* errHandler = (ErrorHandler*) docHandler;
    parser->setDocumentHandler(docHandler);
```

```
        parser->setErrorHandler(errHandler);

        try {
            parser->parse(xmlFile);
        }
        catch (const XMLException& toCatch) {
            cout << "\nFile not found: '" << xmlFile << "'\n"
                 << "Exception message is: \n"
                 << toCatch.getMessage() << "\n" ;
            return -1;
        }
    }
```

Using the SAX API

The SAX API for XML parsers was originally developed for Java. Please be aware that there is no standard SAX API for C++, and that use of the Xerces-C SAX API does not guarantee client code compatibility with other C++ XML parsers.

The SAX API presents a callback based API to the parser. An application that uses SAX provides an instance of a handler class to the parser. When the parser detects XML constructs, it calls the methods of the handler class, passing them information about the construct that was detected. The most commonly used handler classes are DocumentHandler which is called when XML constructs are recognized, and ErrorHandler which is called when an error occurs. The header files for the various SAX handler classes are in '<xerces-c-1_1_0>/include/sax'

As a convenience, Xerces-C provides the class HandlerBase, which is a single class which is publicly derived from all the Handler classes. HandlerBase's default implementation of the handler callback methods is to do nothing. A convenient way to get started with Xerces-C is to derive your own handler class from HandlerBase and override just those methods in HandlerBase which you are interested in customizing. This simple example shows how to create a handler which will print element names, and print fatal error messages. The source code for the sample applications show additional examples of how to write handler classes.

This is the header file MySAXHandler.hpp:

```
    #include <sax/HandlerBase.hpp>

    class MySAXHandler : public HandlerBase {
    public:
        void startElement(const XMLCh* const, AttributeList&);
        void fatalError(const SAXParseException&);
    };
```

This is the implementation file MySAXHandler.cpp:

```
    #include "MySAXHandler.hpp"
    #include <iostream.h>

    MySAXHandler::MySAXHandler()
    {
    }

    MySAXHandler::startElement(const XMLCh* const name,
                               AttributeList& attributes)
    {
        // transcode() is an user application defined function which
        // converts unicode strings to usual 'char *'. Look at
```

```
        // the sample program SAXCount for an example implementation.
        cout << "I saw element: " << transcode(name) << endl;
    }

    MySAXHandler::fatalError(const SAXParseException& exception)
    {
        cout << "Fatal Error: " << transcode(exception.getMessage())
             << " at line: " << exception.getLineNumber()
             << endl;
    }
```

The XMLCh and AttributeList types are supplied by Xerces-C and are documented in the include files. Examples of their usage appear in the source code to the sample applications.

DOM Programming Guide

Java and C++ DOM comparisons

The C++ DOM API is very similar in design and use, to the Java DOM API bindings. As a consequence, conversion of existing Java code that makes use of the DOM to C++ is a straight forward process.

This section outlines the differences between Java and C++ bindings.

Accessing the API from application code

```
    // C++
    #include <dom/DOM.hpp>
```

```
    // Java
    import org.w3c.dom.*
```

The header file <dom/DOM.hpp> includes all the individual headers for the DOM API classes.

Class Names

The C++ class names are prefixed with "DOM_". The intent is to prevent conflicts between DOM class names and other names that may already be in use by an application or other libraries that a DOM based application must link with.

The use of C++ namespaces would also have solved this conflict problem, but for the fact that many compilers do not yet support them.

```
    DOM_Document   myDocument;   // C++
    DOM_Node       aNode;
    DOM_Text       someText;
```

```
    Document       myDocument;   // Java
    Node           aNode;
    Text           someText;
```

If you wish to use the Java class names in C++, then you need to typedef them in C++. This is not advisable for the general case - conflicts really do occur - but can be very useful when converting a body of existing Java code to C++.

```
    typedef DOM_Document  Document;
    typedef DOM_Node      Node;
```

```
    Document   myDocument;        // Now C++ usage is
                                  // indistinguishable from Java
    Node       aNode;
```

Objects and Memory Management

The C++ DOM implementation uses automatic memory management, implemented using reference counting. As a result, the C++ code for most DOM operations is very similar to the equivalent Java code, right down to the use of factory methods in the DOM document class for nearly all object creation, and the lack of any explicit object deletion.

Consider the following code snippets

```
    // This is C++
    DOM_Node       aNode;
    aNode = someDocument.createElement("ElementName");
    DOM_Node docRootNode = someDoc.getDocumentElement();
    docRootNode.AppendChild(aNode);
```

```
    // This is Java
    Node       aNode;
    aNode = someDocument.createElement("ElementName");
    Node docRootNode = someDoc.getDocumentElement();
    docRootNode.AppendChild(aNode);
```

The Java and the C++ are identical on the surface, except for the class names, and this similarity remains true for most DOM code.

However, Java and C++ handle objects in somewhat different ways, making it important to understand a little bit of what is going on beneath the surface.

In Java, the variable `aNode` is an object reference , essentially a pointer. It is initially == null, and references an object only after the assignment statement in the second line of the code.

In C++ the variable `aNode` is, from the C++ language's perspective, an actual live object. It is constructed when the first line of the code executes, and DOM_Node::operator = () executes at the second line. The C++ class DOM_Node essentially a form of a smart-pointer; it implements much of the behavior of a Java Object Reference variable, and delegates the DOM behaviors to an implementation class that lives behind the scenes.

Key points to remember when using the C++ DOM classes:

- Create them as local variables, or as member variables of some other class. Never "new" a DOM object into the heap or make an ordinary C pointer variable to one, as this will greatly confuse the automatic memory management.
- The "real" DOM objects - nodes, attributes, CData sections, whatever, do live on the heap, are created with the create... methods on class DOM_Document. DOM_Node and the other DOM classes serve as reference variables to the underlying heap objects.
- The visible DOM classes may be freely copied (assigned), passed as parameters to functions, or returned by value from functions.
- Memory management of the underlying DOM heap objects is automatic, implemented by means of reference counting. So long as some part of a document can be reached, directly or indirectly, via reference variables that are still alive in the application program, the corresponding document data will stay alive in the heap. When all possible paths of access have been closed off (all of the application's DOM objects have gone out of scope) the heap data itself will be automatically deleted.
- There are restrictions on the ability to subclass the DOM classes.

DOMString

Class DOMString provides the mechanism for passing string data to and from the DOM API. DOMString

is not intended to be a completely general string class, but rather to meet the specific needs of the DOM API.

The design derives from two primary sources: from the DOM's CharacterData interface and from class java.lang.string

Main features are:

- It stores Unicode text.
- Automatic memory management, using reference counting.
- DOMStrings are mutable - characters can be inserted, deleted or appended.

When a string is passed into a method of the DOM, when setting the value of a Node, for example, the string is cloned so that any subsequent alteration or reuse of the string by the application will not alter the document contents. Similarly, when strings from the document are returned to an application via the DOM API, the string is cloned so that the document can not be inadvertently altered by subsequent edits to the string.

> ***Note:*** *The ICU classes are a more general solution to UNICODE character handling for C++ applications. ICU is an Open Source Unicode library, available at the IBM DeveloperWorks website.*

Equality Testing

The DOMString equality operators (and all of the rest of the DOM class conventions) are modeled after the Java equivalents. The equals() method compares the content of the string, while the == operator checks whether the string reference variables (the application program variables) refer to the same underlying string in memory. This is also true of DOM_Node, DOM_Element, etc., in that operator == tells whether the variables in the application are referring to the same actual node or not. It's all very Java-like

- bool operator == () is true if the DOMString variables refer to the same underlying storage.
- bool equals() is true if the strings contain the same characters.

Here is an example of how the equality operators work:

```
DOMString a = "Hello";
DOMString b = a;
DOMString c = a.clone();
if (b == a)           //  This is true
if (a == c)           //  This is false
if (a.equals(c))       //  This is true
b = b + " World";
if (b == a)           // Still true, and the string's
                      //    value is "Hello World"
if (a.equals(c))      // false.  a is "Hello World";
                      //    c is still "Hello".
```

Downcasting

Application code sometimes must cast an object reference from DOM_Node to one of the classes deriving from DOM_Node, DOM_Element, for example. The syntax for doing this in C++ is different from that in Java.

```
// This is C++
DOM_Node       aNode = someFunctionReturningNode();
DOM_Element    el = (Element &) aNode;
```

```
// This is Java
Node       aNode = someFunctionReturningNode();
```

```
    Element    el = (Element) aNode;
```

The C++ cast is not type-safe; the Java cast is checked for compatible types at runtime. If necessary, a type-check can be made in C++ using the node type information:

```
    // This is C++

    DOM_Node       aNode = someFunctionReturningNode();
    DOM_Element    el;    // by default, el will == null.

    if (anode.getNodeType() == DOM_Node::ELEMENT_NODE)
       el = (Element &) aNode;
    else
       // aNode does not refer to an element.
       // Do something to recover here.
```

Subclassing

The C++ DOM classes, DOM_Node, DOM_Attr, DOM_Document, etc., are not designed to be subclassed by an application program.

As an alternative, the DOM_Node class provides a User Data field for use by applications as a hook for extending nodes by referencing additional data or objects. See the API description for DOM_Node for details.

7
Migrating from XML4C 2.x

This document is a discussion of the technical differences between XML4C 2.x code base and the new Xerces-C 1.1.0 code base.

Topics discussed are:

- General Improvements
  - Compliance
  - Bug Fixes
  - Speed
- Summary of changes required to migrate from XML4C 2.x to Xerces-C 1.1.0
- The Samples
- Parser Classes
- DOM Level 2 support
- Progressive Parsing
- Namespace support
- Moved Classes to src/framework
- Loadable Message Text
- Pluggable Validators
- Pluggable Transcoders
- Util directory Reorganization
  - util - The platform independent utility stuff

General Improvements

The new version is improved in many ways. Some general improvements are: significantly better conformance to the XML spec, cleaner internal architecture, many bug fixes, and faster speed.

Compliance

Except for a couple of the very obscure (mostly related to the 'standalone' mode), this version should be quite compliant. We have more than a thousand tests, some collected from various public sources and some IBM generated, which are used to do regression testing. The C++ parser is now passing all but a handful of them.

Bug Fixes

This version has many bug fixes with regard to XML4C version 2.x. Some of these were reported by users and some were brought up by way of the conformance testing.

Speed

Much work was done to speed up this version. Some of the new features, such as namespaces, and conformance checks ended up eating up some of these gains, but overall the new version is significantly faster than previous versions, even while doing more.

Summary of changes required to migrate from XML4C 2.x to Xerces-C 1.1.0

As mentioned, there are some major architectural changes between the 2.3.x and Xerces-C 1.1.0 releases of the parser, and as a result the code has undergone significant restructuring. The list below mentions the public api's which existed in 2.3.x and no longer exist in Xerces-C 1.1.0. It also mentions the Xerces-C 1.1.0 api which will give you the same functionality. Note: This list is not exhaustive. The API docs (and ultimately the header files) supplement this information.

- `parsers/[Non]Validating[DOM/SAX]parser.hpp`
  These files/classes have all been consolidated in the new version to just two files/classes: `[DOM/SAX]Parser.hpp`. Validation is now a property which may be set before invoking the `parse`. Now, the `setDoValidation()` method controls the validation processing.
- The `framework/XMLDocumentTypeHandler.hpp` been replaced with `validators/DTD/DocTypeHandler.hpp`.
- The following methods now have different set of parameters because the underlying base class methods have changed in the 3.x release. These methods belong to one of `XMLDocumentHandler`, `XMLErrorReporter` or `DocTypeHandler` interfaces.
  - `[Non]Validating[DOM/SAX]Parser::docComment`
  - `[Non]Validating[DOM/SAX]Parser::doctypePI`
  - `[Non]ValidatingSAXParser::elementDecl`
  - `[Non]ValidatingSAXParser::endAttList`
  - `[Non]ValidatingSAXParser::entityDecl`
  - `[Non]ValidatingSAXParser::notationDecl`
  - `[Non]ValidatingSAXParser::startAttList`
  - `[Non]ValidatingSAXParser::TextDecl`
  - `[Non]ValidatingSAXParser::docComment`
  - `[Non]ValidatingSAXParser::docPI`
  - `[Non]Validating[DOM/SAX]Parser::endElement`
  - `[Non]Validating[DOM/SAX]Parser::startElement`
  - `[Non]Validating[DOM/SAX]Parser::XMLDecl`
  - `[Non]Validating[DOM/SAX]Parser::error`
- The following methods/data members changed visibility from `protected` in 2.3.x to `private` (with public setters and getters, as appropriate).
  - `[Non]ValidatingDOMParser::fDocument`
  - `[Non]ValidatingDOMParser::fCurrentParent`
  - `[Non]ValidatingDOMParser::fCurrentNode`
  - `[Non]ValidatingDOMParser::fNodeStack`
- The following files have moved, possibly requiring changes in the `#include` statements.
  - `MemBufInputSource.hpp`
  - `StdInInputSource.hpp`
  - `URLInputSource.hpp`
- All the DTD validator code was moved from `internal` to separate `validators/DTD` directory.
- The error code definitions which were earlier in `internal/ErrorCodes.hpp` are now splitup into the following files:
  - `framework/XMLErrorCodes.hpp` - Core XML errors
  - `framework/XMLValidityCodes.hpp` - DTD validity errors
  - `util/XMLExceptMsgs.hpp` - C++ specific exception codes.

The Samples

The sample programs no longer use any of the unsupported util/xxx classes. They only existed to allow us to write portable samples. But, since we feel that the wide character APIs are supported on a lot of platforms these days, it was decided to go ahead and just write the samples in terms of these. If your system does not support these APIs, you will not be able to build and run the samples. On some platforms, these APIs might perhaps be optional packages or require runtime updates or some such action.

More samples have been added as well. These highlight some of the new functionality introduced in the new code base. And the existing ones have been cleaned up as well.

The new samples are:

1. PParse - Demonstrates 'progressive parse', (see below)
2. StdInParse - Demonstrates use of the standard in input source
3. EnumVal - Shows how to enumerate the markup decls in a DTD Validator

Parser Classes

In the XML4C 2.x code base, there were the following parser classes (in the src/parsers/ source directory): NonValidatingSAXParser, ValidatingSAXParser, NonValidatingDOMParser, ValidatingDOMParser. The non-validating ones were the base classes and the validating ones just derived from them and turned on the validation. This was deemed a little bit overblown, considering the tiny amount of code required to turn on validation and the fact that it makes people use a pointer to the parser in most cases (if they needed to support either validating or non-validating versions.)

The new code base just has SAXParer and DOMParser classes. These are capable of handling both validating and non-validating modes, according to the state of a flag that you can set on them. For instance, here is a code snippet that shows this in action.

```
    void ParseThis(const  XMLCh* const fileToParse,
                   const bool validate)
    {
      //
      // Create a SAXParser. It can now just be
      // created by value on the stack if we want
      // to parse something within this scope.
      //
      SAXParser myParser;

      // Tell it whether to validate or not
      myParser.setDoValidation(validate);

      // Parse and catch exceptions...
      try
      {
        myParser.parse(fileToParse);
      }
        ...
    };
```

We feel that this is a simpler architecture, and that it makes things easier for you. In the above example, for instance, the parser will be cleaned up for you automatically upon exit since you don't have to allocate it anymore.

DOM Level 2 support

Experimental early support for some parts of the DOM level 2 specification have been added. These address some of the shortcomings in our DOM implementation, such as a simple, standard mechanism for tree traversal.

Progressive Parsing

The new parser classes support, in addition to the parse() method, two new parsing methods, parseFirst() and parseNext(). These are designed to support 'progressive parsing', so that you don't have to depend upon throwing an exception to terminate the parsing operation. Calling parseFirst() will cause the DTD (or in the future, Schema) to be parsed (both internal and external subsets) and any pre-content, i.e. everything up to but not including the root element. Subsequent calls to parseNext() will cause one more pieces of markup to be parsed, and spit out from the core scanning code to the parser (and hence either on

to you if using SAX or into the DOM tree if using DOM.) You can quit the parse any time by just not calling parseNext() anymore and breaking out of the loop. When you call parseNext() and the end of the root element is the next piece of markup, the parser will continue on to the end of the file and return false, to let you know that the parse is done. So a typical progressive parse loop will look like this:

```
    // Create a progressive scan token
    XMLPScanToken token;

    if (!parser.parseFirst(xmlFile, token))
    {
      cerr << "scanFirst() failed\n" << endl;
      return 1;
    }

    //
    // We started ok, so lets call scanNext()
    // until we find what we want or hit the end.
    //
    bool gotMore = true;
    while (gotMore && !handler.getDone())
      gotMore = parser.parseNext(token);
```

In this case, our event handler object (named 'handler' surprisingly enough) is watching form some criteria and will return a status from its getDone() method. Since the handler sees the SAX events coming out of the SAXParser, it can tell when it finds what it wants. So we loop until we get no more data or our handler indicates that it saw what it wanted to see.

When doing non-progressive parses, the parser can easily know when the parse is complete and insure that any used resources are cleaned up. Even in the case of a fatal parsing error, it can clean up all per-parse resources. However, when progressive parsing is done, the client code doing the parse loop might choose to stop the parse before the end of the primary file is reached. In such cases, the parser will not know that the parse has ended, so any resources will not be reclaimed until the parser is destroyed or another parse is started.

This might not seem like such a bad thing; however, in this case, the files and sockets which were opened in order to parse the referenced XML entities will remain open. This could cause serious problems. Therefore, you should destroy the parser instance in such cases, or restart another parse immediately. In a future release, a reset method will be provided to do this more cleanly.

Also note that you must create a scan token and pass it back in on each call. This insures that things don't get done out of sequence. When you call parseFirst() or parse(), any previous scan tokens are invalidated and will cause an error if used again. This prevents incorrect mixed use of the two different parsing schemes or incorrect calls to parseNext().

Namespace support

The C++ parser now supports namespaces. With current XML interfaces (SAX/DOM) this doesn't mean very much because these APIs are incapable of passing on the namespace information. However, if you are using our internal APIs to write your own parsers, you can make use of this new information. Since the internal event APIs must be able to now support both namespace and non-namespace information, they have more parameters. These allow namespace information to be passed along.

Most of the samples now have a new command line parameter to turn on namespace support. You turn on namespaces like this:

```
    SAXParser myParser;

    // Tell it whether to do namespacse
    myParser.setDoNamespaces(true);
```

Moved Classes to src/framework

Some of the classes previously in the src/internal/ directory have been moved to their more correct location in the src/framework/ directory. These are classes used by the outside world and should have been framework classes to begin with. Also, to avoid name classes in the absense of C++ namespace support, some of these clashes have been renamed to make them more XML specific and less likely to clash. More classes might end up being moved to framework as well.

So you might have to change a few include statements to find these classes in their new locations. And you might have to rename some of the names of the classes, if you used any of the ones whose names were changed.

Loadable Message Text

The system now supoprts loadable message text, instead of having it hard coded into the program. The current drop still just supports English, but it can now support other languages. Anyone interested in contributing any translations should contact us. This would be an extremely useful service.

In order to support the local message loading services, we have created a pretty flexible framework for supporting loadable text. Firstly, there is now an XML file, in the src/NLS/ directory, which contains all of the error messages. There is a simple program, in the Tools/NLSXlat/ directory, which can spit out that text in various formats. It currently supports a simple 'in memory' format (i.e. an array of strings), the Win32 resource format, and the message catalog format. The 'in memory' format is intended for very simple installations or for use when porting to a new platform (since you can use it until you can get your own local message loading support done.)

In the src/util/ directory, there is now an XMLMsgLoader class. This is an abstraction from which any number of message loading services can be derived. Your platform driver file can create whichever type of message loader it wants to use on that platform. We currently have versions for the in memory format, the Win32 resource format, and the message catalog format. An ICU one is present but not implemented yet. Some of the platforms can support multiple message loaders, in which case a #define token is used to control which one is used. You can set this in your build projects to control the message loader type used.

Both the Java and C++ parsers emit the same messages for an XML error since they are being taken from the same message file.

Pluggable Validators

In a preliminary move to support Schemas, and to make them first class citizens just like DTDs, the system has been reworked internally to make validators completely pluggable. So now the DTD validator code is under the src/validators/DTD/ directory, with a future Schema validator probably going into the src/validators. The core scanner architecture now works completely in terms of the framework/XMLValidator abstract interface and knows almost nothing about DTDs or Schemas. For now, if you don't pass in a validator to the parsers, they will just create a DTDValidator. This means that, theoretically, you could write your own validator. But we would not encourage this for a while, until the semantics of the XMLValidator interface are completely worked out and proven to handle DTD and Schema cleanly.

Pluggable Transcoders

Another abstract framework added in the src/util/ directory is to support pluggable transcoding services. The XMLTransService class is an abtract API that can be derived from, to support any desired transcoding service. XMLTranscoder is the abstract API for a particular instance of a transcoder for a particular encoding. The platform driver file decides what specific type of transcoder to use, which allows each platform to use its native transcoding services, or the ICU service if desired.

Implementations are provided for Win32 native services, ICU services, and the iconv services available on many Unix platforms. The Win32 version only provides native code page services, so it can only handle XML code in the intrinsic encodings ASCII, UTF-8, UTF-16 (Big/Small Endian), UCS4

(Big/Small Endian), EBCDIC code pages IBM037 and IBM1140 encodings, ISO-8859-1 (aka Latin1) and Windows-1252. The ICU version provides all of the encodings that ICU supports. The iconv version will support the encodings supported by the local system. You can use transcoders we provide or create your own if you feel ours are insufficient in some way, or if your platform requires an implementation that we do not provide.

Util directory Reorganization

The src/util directory was becoming somewhat of a dumping ground of platform and compiler stuff. So we reworked that directory to better spread things out. The new scheme is:

util - The platform independent utility stuff

- MsgLoaders - Holds the msg loader implementations
  1. ICU
  2. InMemory
  3. MsgCatalog
  4. Win32
- Compilers - All the compiler specific files
- Transcoders - Holds the transcoder implementations
  1. Iconv
  2. ICU
  3. Win32
- Platforms
  1. AIX
  2. HP-UX
  3. Linux
  4. Solaris
  5. ....
  6. Win32

This organization makes things much easier to understand. And it makes it easier to find which files you need and which are optional. Note that only per-platform files have any hard coded references to specific message loaders or transcoders. So if you don't include the ICU implementations of these services, you don't need to link in ICU or use any ICU headers. The rest of the system works only in terms of the abstraction APIs.

8
Frequently Asked Questions

Distributing Xerces-C

What are the differences between Xerces-C and XML4C?

Xerces-C has intrinsic support for ASCII, UTF-8, UTF-16 (Big/Small Endian), UCS4 (Big/Small Endian), EBCDIC code pages IBM037 and IBM1140 encodings, ISO-8859-1 (aka Latin1) and Windows-1252. This means that it can parse input XML files in these above mentioned encodings.

However, if you wish to parse XML files in any other encodings, say in Shift-JIS, Big5 etc., then you cannot use Xerces-C. XML4C addresses this need. It combines Xerces-C and International Components for Unicode (ICU) and provides support for over 100 different encodings.

ICU is also an open source project but is licensed under the IBM Public License. XML4C is published by IBM and can be downloaded from their Alphaworks site. The license to use XML4C is simply to comply with the Apache license (because of Xerces-C) and IBM Public License (because of ICU).

XML4C binaries are published for Solaris using SunWorkshop compiler, HPUX 10.20 and 11.0 using CC and aCC, Redhat Linux using gcc, Windows NT using MSVC, AIX using xlC.

Which DLL's do I need to distribute with my application?

As mentioned above, there are two configurations in which Xerces-C binaries are shipped. One is from the Apache site, while the other is from IBM published at IBM's Alphaworks Site.

If you are using the binaries from the Apache download site site, then you only need to distribute **one** file:
xerces-c_1.dll for Windows NT/95/98, or
libxerces-c1_1.a for AIX, or
libxerces-c1_1.so for Solaris/Linux, or
libxerces-c1_1.sl for HP-UX.

However, if you are using the XML4C binaries then in **addition** to the Xerces-C library file mentioned above, you also need to ship:

1. **ICU shared library file**:
   icuuc.dll for Windows NT/95/98, or
   libicu-uc.a for AIX, or
   libicu-uc.so for Solaris/Linux, or
   libicu-uc.sl for HP-UX.
2. **ICU converter data shared library file:**
   icudata.dll for Windows NT/95/98, or
   libicudata.a for AIX, or
   libicudata.so for Solaris/Linux, or
   libicudata.sl for HP-UX.

How do I package the sources to create a binary drop?

You have to first compile the sources inside your IDE to create the required DLLs and EXEs. Then you need to copy over the binaries to another directory for the binary drop. A perl script has been provided to give you a jump start. You need to install perl on your machine for the script to work. If you have changed your source tree, you have to modify the script to suit your current directory structure. To invoke the script, go to the \<Xerces>\scripts directory, and type:

```
perl packageBinaries.pl
```

You will get a message that somewhat looks like this (changes always happpen, we are evolving you see!):

```
Usage is: packageBinaries <options>
options are:  -s <source_directory>
              -o <target_directory>
              -c <C compiler name> (e.g. gcc or xlc_r)
              -x <C++ compiler name> (e.g. g++ or xlC_r)
              -m <message loader> can be 'inmem', 'icu' or 'iconv'
              -n <net accessor> can be 'fileonly' or 'libwww'
              -t <transcoder> can be 'icu' or 'native'
              -r <thread option> can be 'pthread' or 'dce' (only used on HP-11)
              -h to get help on these commands
Example: perl packageBinaries.pl -s$HOME/xerces-c_1_0_0
                                 -o$HOME/xerces-c_1_0_0
                                 -cgcc -xg++ -minmem
                                 -nfileonly -tnative
```

Make sure that your compiler can be invoked from the command line and follow the instructions to produce a binary drop.

I do not see binaries for my platform. When will they be available?">

The reason why you see binaries only for some specific platforms is that we have had the maximum requests for them. Moreover, we have limited resources and hence cannot publish binaries for every platform. If you wish to contribute your time and effort in building binaries for a specific platform/environment then please send a mail to xerces-dev@xml.apache.org. We can definitely use any extra help in this open source project

When will a port to my platform be available?

We would like to see Xerces ported to as many platforms as there are. Again, due to limited resources we cannot do all the ports. We will help you make this port happen. Here are some Porting Guidelines.

We strongly encourage you to submit the changes that are required to make it work on another platform. We will incorporate these changes in the source code base and make them available in the future releases.

All porting changes may be sent to: xerces-dev@xml.apache.org .

How can I port Xerces to my favourite platform?

Some porting information is mentioned on the build page.

What application did you used to create the documentation?

We have used an internal XML based application to create the documentation. The documentation files are all written in XML and the application, internally codenamed StyleBook, makes use of XSL to transform it into an HTML document that you are seeing right now. It is currently available on the Apache open source website as Cocoon.

The API documentation is created using DOC++.

Can I get the source code for the C++ Builder TreeViewer application?

In view of the numerous requests that we have received for the TreeViewer sample application (written using C++ Builder), we have decided to make it available as an independent download from the IBM AlphaWorks portal. The archive to get from the 'Download' page is 'DOMSampleTreeViewer.zip'. Please note, this is provided on a "as-is, no support" basis. This sample works with XML4C 2.3.1 and we have

not verified if it works with XML4C 3.x releases.

TreeViewer parses the XML file, using Xerces, and displays the data as a tree.

We welcome your additional feedback at: < xerces-c-dev@xml.apache.org >

Can I use Xerces in my product?

Yes! Read the license agreement first and if you still have further questions, then please address them to xerces-c-dev@xml.apache.org > .

Parsing with Xerces-C

Why does my application crash on AIX when I run it under a multi-threaded environment?

AIX maintains two kinds of libraries on the system, thread-safe and non-thread safe. Multi-threaded libraries on AIX follow a different naming convention, Usually the multi-threaded library names are followed with "_r". For example, libc.a is single threaded whereas libc_r.a is multi-threaded.

To make your multi-threaded application run on AIX, you MUST ensure that you do not have a 'system library path' in your LIBPATH environment variable when you run the application. The appropriate libraries (threaded or non-threaded) are automatically picked up at runtime. An application usually crashes when you build your application for multi-threaded operation but don't point to the thread-safe version of the system libraries. For example, LIBPATH can be simply set as:

```
    LIBPATH=$HOME/<Xerces>/lib
```

Where <Xerces> points to the directory where Xerces application resides.

If for any reason, unrelated to Xerces, you need to keep a 'system library path' in your LIBPATH environment variable, you must make sure that you have placed the thread-safe path before you specify the normal system path. For example, you must place /lib/threads before /lib in your LIBPATH variable. That is to say your LIBPATH may look like this:

```
    export LIBPATH=$HOME/<Xerces>/lib:/usr/lib/threads:/usr/lib
```

Where /usr/lib is where your system libraries are.

What compilers are being used on the supported platforms?

Xerces has been built on the following platforms with these compilers

Operating System	Compiler
Windows NT SP5/98	MSVC 6.0
Redhat Linux 6.0	gcc
AIX 4.1.4 and higher	xlC 3.1
Solaris 2.6	CC version 4.2
HP-UX B10.2	aCC and CC
HP-UX B11	aCC and CC

I cannot run my sample applications. What is wrong?

In order to run an application built using Xerces you must set up your path and library search path properly. In the standalone version from Apache, you must have the Xerces-C runtime library available from your path settings. On Windows this library is called `xerces-c_1.dll` which must be available from your `PATH` settings. On UNIX platforms the library is called `libxerces-c1_1.so` (or `.a` or `.sl`) which must be available from your `LD_LIBRARY_PATH` (or `SHLIB_PATH` or `LIBPATH`) environment variable.

Thus, if you installed your binaries under `$HOME/fastxmlparser`, you need to point your library path to that directory.

```
    export LIBPATH=$LIBPATH:$HOME/fastxmlparser/lib # (AIX)
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/fastxmlparser/lib # (Solaris,
   Linux)
```

```
export SHLIB_PATH=$SHLIB_PATH:$HOME/fastxmlparser/lib # (HP-UX)
```

If you are using the enhanced version of this parser from IBM, you will need to put in two additional DLLs. In the Windows build these are `icuuc.dll` and `icudata.dll` which must be available from your PATH settings. On UNIX, these libraries are called `libicu-uc.so` and `libicudata.so` (or `.sl` for HP-UX or `.a` for AIX) which must be available from your library search path.

I just built my own application using the Xerces parser. Why does it crash?

In order to work with the Xerces parser, you have to first initialize the XML subsystem. The most common mistake is to forget this initialization. Before you make any calls to Xerces APIs, you must call

```
XMLPlatformUtils::Initialize():
try {
   XMLPlatformUtils::Initialize();
}
catch (const XMLException& toCatch) {
   // Do your failure processing here
}
```

This initializes the Xerces system and sets its internal variables. Note that you must the include `util/PlatformUtils.hpp` file for this to work.

Is Xerces thread-safe?

This is not a question that has a simple yes/no answer. Here are the rules for using Xerces in a multi-threaded environment:

Within an address space, an instance of the parser may be used without restriction from a single thread, or an instance of the parser can be accessed from multiple threads, provided the application guarantees that only one thread has entered a method of the parser at any one time.

When two or more parser instances exist in a process, the instances can be used concurrently, and without external synchronization. That is, in an application containing two parsers and two threads, one pareser can be running within the first thread concurrently with the second parser running within the second thread.

The same rules apply to Xerces DOM documents - multiple document instances may be concurrently accessed from different threads, but any given document instance can only be accessed by one thread at a time.

DOMStrings allow multiple concurrent readers. All DOMString const methods are thread safe, and can be concurrently entered by multiple threads. Non-const DOMString methods, such as appendData(), are not thread safe and the application must guarantee that no other methods (including const methods) are executed concurrently with them.

Why does my multi-threaded application crash on Solaris?

The problem appears because the throw call on Solaris 2.6 is not multi-thread safe. Sun Microsystems provides a patch to solve this problem. To get the latest patch for solving this problem, go to SunSolve.sun.com and get the appropriate patch for your operating system. For Intel machines running Solaris, you need to get Patch ID 104678. For SPARC machines you need to get Patch ID #105591.

How do I find out what version of Xerces I am using?

The version string for Xerces happens to be in one of the source files. Look inside the file `src/util/XML4CDefs.hpp` and find out what the static variable `gXML4CFullVersionStr` is defined to be. (It is usually of type 3.0.0 or something similar). This is the version of XML you are using.

If you don't have the source code, you have to find the version information from the shared library name. On Windows NT/95/98 right click on the DLL name xerces-c_1.dll in the bin directory and look up properties. The version information may be found on the Version tab.

On AIX, just look for the library name libxerces-c1_1.a (or libxerces-c1_1.so on Solaris/Linux and libxerces-c1_1.sl on HP-UX). The version number is coded in the name of the library.

How do I uninstall Xerces?

Xerces only installs itself in a single directory and does not set any registry entries. Thus, to un-install, you only need to remove the directory where you installed it, and all Xerces related files will be removed.

How are entity reference nodes handled in DOM?

If you are using the native DOM classes, the function `setExpandEntityReferences` controls how entities appear in the DOM tree. When setExpandEntityReferences is set to false (the default), an occurance of an entity reference in the XML document will be represented by a subtree with an EntityReference node at the root whose children represent the entity expansion. Entity expansion will be a DOM tree representing the structure of the entity expansion, not a text node containing the entity expansion as text.

If setExpandEntityReferences is true, an entity reference in the XML document is represented by only the nodes that represent the entity expansion. The DOM tree will not contain any entityReference nodes.

What kinds of URLs are currently supported in Xerces?

The `XMLURL` class provides for limited URL support. It understands the `file://, http://`, and `ftp://` URL types, and is capable or parsing them into their constituent components, and normalizing them. It also supports the commonly required action of conglomerating a base and relative URL into a single URL. In other words, it performs the limited set of functions required by an XML parser.

Another thing that URLs commonly do are to create an input stream that provides access to the entity referenced. The parser, as shipped, only supports this functionality on URLs in the form `file:///` and `file://localhost/`, i.e. only when the URL refers to a local file.

You may enable support for HTTP and FTP URLs by implementing and installing a NetAccessor object. When a NetAccessor object is installed, the URL class will use it to create input streams for the remote entities refered to by such URLs.

How can I add support for URL's with HTTP/FTP protocols?

To address the need to make remote connections to resources specified using other protocols like HTTP, FTP etc..., Xerces-C now provides the `NetAccessor` interface. The header file is `src/util/XMLNetAccessor.hpp`. This interface allows you to plug in your own implementation of URL networking code into the Xerces-C parser.

One such implementation **(tested minimally under WinNT only)** is already provided in Xerces-C source code drop, using W3C's Libwww library. Libwww is available for free and has been ported to various platforms. Click here to read how you can rebuild Xerces-C binaries with this implementation.

Some more notes about the NetAccessor implementation using Libwww:

- This implementation only supports HTTP and does not return adequate error messages when connections cannot be made to the remote resources. It however illustrates how you can add support for HTTP and FTP URL's.
- The Xerces-C team will **NOT** be able to address any questions related to how things work in Libwww. You can get some help with Libwww by subscribing to the <www-lib@w3.org> public mailing list.
- However, we will welcome any feedback on the design of the NetAccessor interface. Please send all such feedback to xerces-dev@xml.apache.org.
- You do not have to recompile Xerces-C to plugin your NetAccessor implementation. You can simply point the static pointer variable `XMLPlatformUtils::fgNetAccessor` to an instance of your NetAccessor implementation. Please refer to the files `src/util/PlatformUtils.cpp` and `src/util/Platforms/Win32/Win32PlatformUtils.cpp` to see how we have done this simple illustrative implementation.

Can I use Xerces to parse HTML?

Yes, if it follows the XML spec rules. Most HTML, however, does not follow the XML rules, and will therefore generate XML well-formedness errors.

I keep getting an error: "invalid UTF-8 character". What's wrong?

There are many Unicode characters that are not allowed in your XML document, according to the XML

spec. Typical disallowed characters are control characters, even if you escape them using the Character Reference form: See the XML spec, sections 2.2 and 4.1 for details. If the parser is generating this error, it is very likely that there's a character in there that you can't see. You can generally use a UNIX command like "od -hc" to find it.

Another reason for this error is that your file is in some non UTF/ASCII encoding but you gave no encoding="" string in your file to tell the parser what its real encoding is.

What encodings are supported by Xerces-C / XML4C?

Xerces-C has intrinsic support for ASCII, UTF-8, UTF-16 (Big/Small Endian), UCS4 (Big/Small Endian), EBCDIC code pages IBM037 and IBM1140 encodings, ISO-8859-1 (aka Latin1) and Windows-1252. This means that it can parse input XML files in these above mentioned encodings.

XML4C - the version of Xerces-C available from IBM - extends this set to include the encodings listed in the table below.

Common Name	Use this name in XML
8 bit Unicode	UTF-8
ISO Latin 1	ISO-8859-1
ISO Latin 2	ISO-8859-2
ISO Latin 3	ISO-8859-3
ISO Latin 4	ISO-8859-4
ISO Latin Cyrillic	ISO-8859-5
ISO Latin Arabic	ISO-8859-6
ISO Latin Greek	ISO-8859-7
ISO Latin Hebrew	ISO-8859-8
ISO Latin 5	ISO-8859-9
EBCDIC US	ebcdic-cp-us
EBCDIC with Euro symbol	ibm1140
Chinese, PRC	gb2312
Chinese, Big5	Big5
Cyrillic	koi8-r
Japanese, Shift JIS	Shift_JIS
Korean, Extended UNIX code	euc-kr

Some implementations or ports of Xerces-C provide support for additional encodings. The exact set will depend on the supplier of the parser and on the character set transcoding services in use.

What character encoding should I use when creating XML documents?

The best choice in most cases is either utf-8 or utf-16. Advantages of these encodings include

- The best portability. These encodings are more widely supported by XML processors than any others, meaning that your documents will have the best possible chance of being read correctly, no matter where they end up.
- Full international character support. Both utf-8 and utf-16 cover the full Unicode character set, which includes all of the characters from all major national, international and industry character sets.
- Efficient. utf-8 has the smaller storage requirements for documents that are primarily composed of of characters from the Latin alphabet. utf-16 is more efficient for encoding Asian languages. But both encodings cover all languages without loss.

The only drawback of utf-8 or utf-16 is that they are not the native text file format for most systems, meaning that common text file editors and viewers can not be directly used.

A second choice of encoding would be any of the others listed in the table above. This works best when the xml encoding is the same as the default system encoding on the machine where the XML document is being prepared, because the document will then display correctly as a plain text file. For UNIX systems in

countries speaking Western European languages, the encoding will usually be iso-8859-1.

The versions of Xerces, both C and Java, distributed by IBM as XML4C and XML4J, include all of the encodings listed in the above table, on all platforms.

A word of caution for Windows users: The default character set on Windows systems is windows-1252, not iso-8859-1. While Xerces-c does recognize this Windows encoding, it is a poor choice for portable XML data because it is not widely recoginized by other XML processing tools. If you are using a Windows based editing tool to generate XML, check which character set it generates, and make sure that the resulting XML specifies the correct name in the encoding="..." declaration.

Is EBCDIC supported?

Yes, Xerces-C supports EBCDIC. When creating EBCDIC encoded XML data, the preferred encoding is ibm1140. Also supported is ibm037 (and its alternate name, ebcdic-cp-us); this encoding is almost the same as ibm1140, but it lacks the Euro symbol

These two encodings, ibm1140 and ibm037, are available on both Xerces-C and IBM XML4C, on all platforms.

On IBM System 390, XML4C also supports two alternative forms, ibm037-s390 and ibm1140-s390. These are similar to the base ibm037 and ibm1140 encodings, but with alternate mappings of the EBCDIC new-line character, which allows them to appear as normal text files on System 390s. These encodings are not supported on other platforms, and should not be used for portable data.

XML4C on System 390 and AS/400 also provides additional EBCDIC encodings, including those for the character sets of different countries. The exact set supported will be platform dependent, and these encodings are not recommended for portable XML data.

Other Xerces-C Questions

I can't use C++. Do you have a Java version?

Yes. The Xerces family of products also has a Java version. More information is available at: http://xml.apache.org/xerces-j/index.html

I found a defect - whom do I report it to?

Send the defect reports to < xerces-c-dev@xml.apache.org > with the version number, the exact OS release number, the compiler version number, and a copy of the XML document that generates the error. Do mention the archive (source or binary) you are using. If you have rebuilt the samples/library then please mention that too. The more information we get, the faster we will be able to reproduce and fix the defect.

Is there any kind of support available for Xerces-C/XML4C?

Limited support is available with **no guarantees** via the Xerces-C mailing list xerces-c-dev@xml.apache.org > and XML4C discussion forum on Alphaworks. We strongly recommend all Xerces-C users to subscribe to the Xerces-C mailing list and monitor the discussions on the XML4C discussion forums.

IBM, in the past, has set up some separate special support contracts for customers who absolutely insisted on having support. If you have similar requirements, then you may contact IBM via email at xml4c@us.ibm.com.

I have a question not covered here -- who may I contact?

Hopefully, you have read this entire FAQ and still could not find the answer to your question.

Next, we would like you to search the archives (addresses given below) and on going discussion threads (URL's mentioned in the answer to the previous question above) to see if someone else has already asked the same question. Its been a while since the first public release of this product was made, so most common questions have been asked and answered. Many people actively subscribe/monitor these lists and is the fastest way to get an answer.

The Apache's Xerces-C mailing list is archived at http://xml-archive.webweaving.org.

The archives for IBM's XML4C discussion forum are available at here.

If you are using XML4C then you may also send your question to xml4c@us.ibm.com. Do note that the packaging and bundling of XML4C sources and binaries differs from whats available at the Apache's site, so please address your questions accordingly. Again, the response from this email address may not be as prompt as what you may receive from the mailing lists/discussion forums.

Its critical, that you very clearly mention the following information along with your question. The product name, version, source/binary archive that you downloaded, whether you rebuilt the samples/library or are using the stock ones that we release, platform, OS version number, compiler, compiler version number.

9 Releases

Xerces-C Version 1.1.0: February 28, 2000

- Simplified platform support (removed need to support local standard output streams or to find the location of the parser DLL/SharedLib.)
- Added support for the NetAccessor plug in abstraction, which allows the parser to support HTTP/FTP based data sources
- Added EBCDIC-US and ISO-8859-1 as intrinsic encodings
- Added more DOM Level II features
- Support for ICU 1.4, which makes Xerces-C Unicode 3.0 compliant when using ICU
- New samples and tests (DOM test, programmatic DOM sample, thread test)
- Added support for multiply nested entities using relative paths or URLs
- Significant internal architecture improvements in the handling of encodings and transcoding services.

Xerces-C Version 1.0.1: December 15, 1999

- Port to Solaris.
- Improved error recovery and clarified error messages.
- Added DOMTest program.

Xerces C++ Parser Version 1.0.0: December 7, 1999

- Released Xerces-C after incorporating ICU as a value-added plug-in.
- Has bug fixes, better conformance, better speed and cleaner internal internal architecture
- Three additional samples added: PParse, StdInParse and EnumVal
- Experimental DOM Level 2 support
- Support for namespaces
- Loadable message text enabling future translations to be easily plugged-in
- Pluggable validators
- Pluggable transcoders
- Reorganized the util directory to better manage different platforms and compilers

Xerces-C BETA November 5, 1999

- Created initial code base derived from IBM's XML4C Version 2.0
- Modified documentation to reflect new name (Xerces-C)

10
Caveats and Limitations

Miscellaneous

- SAXPrint does not output the <?XML ... ?> prologue line (this means that it cannot process its own output). This is because the SAX API doesn't provide a callback handler for the prologue.
- ??????????

11 Feedback Procedures

Questions or Comments

Please browse through this bundled documentation completely. Most of the common questions have been answered in the FAQ's. Specifically, do read the answer to "I have a question not covered here - who may I contact?". Browsing this documentation, may be the quickest way to get an answer. Ofcourse, if all else fails, as mentioned in the link above, you can post a question to the Xerces-C-dev mailing list.

If you are reporting a defect (greatly appreciated!), please provide the following information:

- Version number of Xerces-C (1.1.0?)
- Which OS platform/version you are using (NT4+SP4? Win98? Redhat Linux 6.0? Solaris2.6? AIX 4.3? HP-UX10? HP-UX11?)
- Which compiler/version you are using (MSVC6? gcc? cc? aCC?)
- Sample XML file that causes the bug
- Sample Schema file (if required to recreate the bug)
- Sample DTD file (if required to recreate the bug)

Acknowledgements

Ever since this source code base was initially created, many people have helped to port the code to different platforms and provide constructive feedback to fix bugs and enhance features.

Listed below are some names (in alphabetical order) of people to whom we would like to give special thanks.

- Anupam Bagchi
- Matthew Baker
- Sumit Chawla
- John Bellardo
- Arundhati Bhowmick
- Paul Ferguson
- Pierpaolo Fumagalli
- Susan Hardenbrook
- Andy Heninger
- Rahul Jain
- Richard Ko
- Paul Kramer
- Andy Levine
- David Nickerson
- Michael Ottati
- Mike Pogue
- John Ponzo
- Dean Roddey
- Steven Rosenthal

- Gereon Steffens
- Rick J. Stevens
- Linda M. Swan
- Tom Watson
- Roger Webster

12
PDF Documentation

PDF Documentation

You can get the entire Xerces-C documentation in PDF format for printing and offline reference.

> ***Note:*** *A word of caution! The tools to create the PDF documentation are still experimental. So the resulting PDF document is not perfect. We would be glad to receive your comments on Xerces-C mailing list. We know, for example, that links do not work properly, images are not supported yet, and the shading within the document (around source code) does not print properly.*