BRISTOL

T • E • C • H • N • O • L • O • G • Y

eXactML 1.2

User's Guide

Bristol Technology Inc.
39 Old Ridgebury Road
Danbury, CT 06810-5113
USA
(203) 798-1007

Bristol Technology BV
Plotterweg 2A
3821 BB Amersfoort
The Netherlands
+31 (0)33 450 50 50

Printed July 25, 2000

This manual supports eXactML Release 1.2 and higher versions.
No part of this manual may be reproduced in any form or by any means without written permission of:

Bristol Technology Inc
39 Old Ridgebury Road
Danbury, CT 06810-5113 U.S.A.

Part No. EX1000628

Contents

Chapter 1
Getting Started with eXactML

What is eXactML?

eXactML makes it easy to add XML support to applications. It generates C++ interfaces for reading and writing valid XML content based on a specified DTD or schema. Use these interfaces in your C++ applications to add XML support quickly and easily.

How Does eXactML Work?

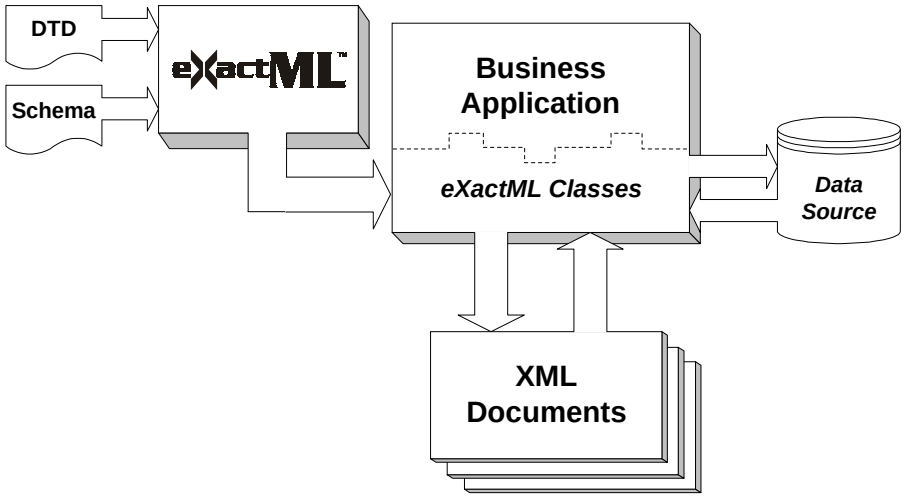
eXactML first parses the specified DTD or schema into an intermediate W3C XML Schema that orders elements according to dependencies and provides name attributes for all elements that do not have names in the original DTD or schema. You may save the intermediate schema generated by eXactML for future reference if you wish.

For complete instructions on generating eXactML interfaces, see Chapter 2, “Generating Interfaces.”

eXactML generates a header and source file that define interfaces for each element in the specified DTD or schema. Each interface provides a constructor and Get(), Set(), EmitXML(), and IsValid() methods. For more information about the interfaces, see the eXactML Class Reference in Chapter 4.

eXactML also generates a flex/bison parser that is customized to recognize XML documents for the given DTD or schema. The parser can be invoked to import a document and generate the appropriate element object structures. These objects may ben be manipulated programmatically using the generated interfaces.

Once you generate the eXactML interfaces and parser, follow the instructions in Chapter 3, “Using Generated Interfaces,” to add the classes to your application.



Installing eXactML

After downloading eXactML from the Bristol Technology web site, perform the following steps to install it on your PC or workstation:

1. Double-click the eXactML11.exe icon to open the eXactML Welcome dialog.
2. Click Finish to unpack the eXactML installation files and start the installation process.
3. Click Next> to view the eXactML license agreement.
4. Click Yes to agree to the license terms and continue with the installation.
5. Enter your name, company name, and serial number, then click Next> to continue.

   Your serial number was provided on the email you received from Bristol Technology. If you no longer have the email containing your serial number, request a serial number from support@bristol.com.
6. Choose the destination folder where you wish to install eXactML and click Next> to install the eXactML files.
7. When setup is complete, select “I would like to view the Release Notes” and click Finish.

Quick Start

Once you've installed eXactML, you can generate C++ interfaces to add XML to your application in just a few minutes:

1. Start Visual C++ and choose the File > New menu item.

   If you already have a project that you are adding XML support to, open it first to make the project generated by eXactML a subproject in your existing workspace.

2. On the Project tab, select eXactML AppWizard and specify the project name and location.

3. In the eXactML AppWizard, specify your DTD or schema location and class options, then click Finish.

That's it! eXactML generates C++ classes that you can use to add the ability to write valid XML to your applications. For more information about using the eXactML AppWizard or, if you don't use Visual C++, the eXactML command, see Chapter 2, "Generating Interfaces." For more information about using interfaces generated by eXactML in your application, see Chapter 3, "Using Generated Interfaces."

Chapter 2
Generating Interfaces with eXactML

There are two ways to use eXactML to generate interfaces based on your DTD or schema: the Visual C++ eXactML AppWizard and the eXactML command.

If you are using Microsoft Visual C++ for application development, the eXactML AppWizard is the easiest way to generate interfaces for your DTD or schema and use these interfaces in your application.

If you don't use Visual C++, however, you can use the eXactML command to generate interfaces.

Using the eXactML AppWizard

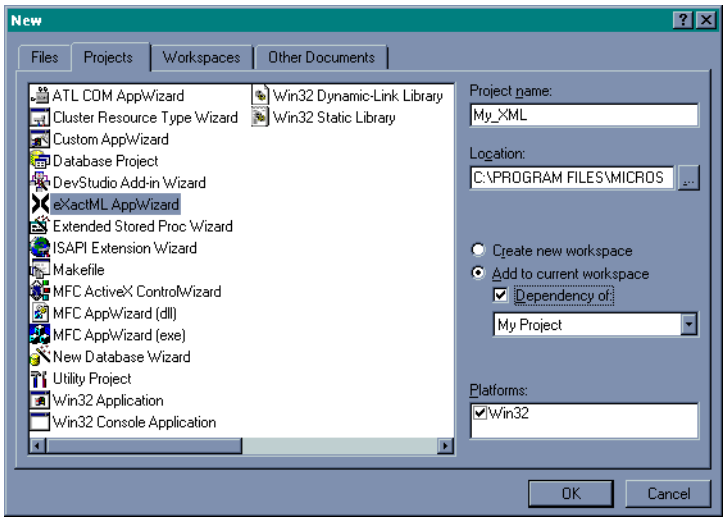
eXactML provides close integration with Visual C++ to make it easy to generate interfaces and use them in Visual C++ applications. eXactML runs as an AppWizard, provides the option of using MFC data types, and generates the XML interfaces in a Visual C++ project.

To use the eXactML AppWizard to generate interfaces in a Visual Studio project, perform the following steps:

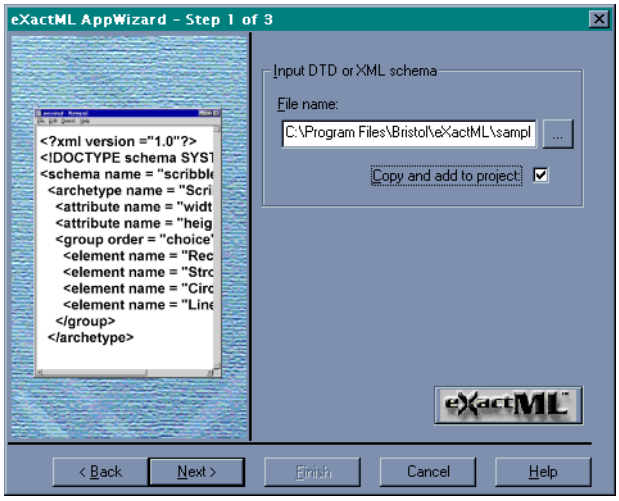
1. If you wish to use the interfaces generated by eXactML in an existing project, open the project in Visual C++ before running the eXactML AppWizard.
2. In Visual C++, choose the File > New menu item. The New dialog opens.

3. On the Projects page, select eXactML AppWizard and specify the following options:

Option	Description
Project name:	The name of the project to be created by the eXactML AppWizard.
Location:	The directory location of the project to be created by the eXactML AppWizard. Enter the full pathname or click **...** to browse to the location.
Workspace option	If your application project file is open, you may specify whether to create a new workspace or add the eXactML project to the current workspace. If no application project file is currently open, the eXactML AppWizard automatically creates a new workspace.



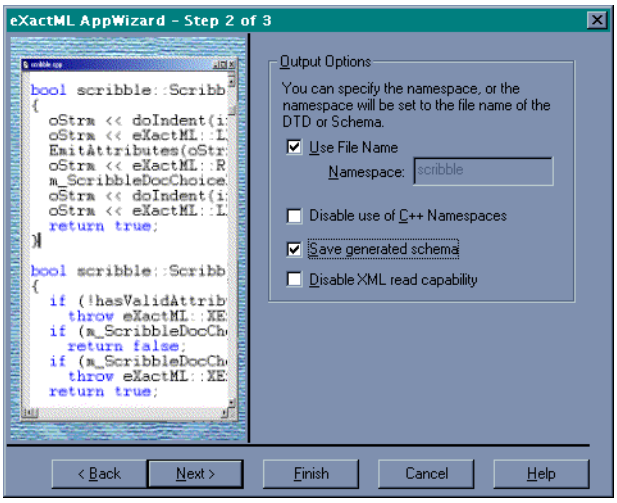
4. Click OK. The eXactML AppWizard Step 1 opens.



5. Specify the following options:

Option	Description
Input File Name	Enter the full pathname of the DTD or schema to use as the basis for your generated classes. Click … to select a file with the Open Schema dialog. eXactML supports the W3C XML Schema standard as well as DTD. Other schema standards such as XML-Data will be supported in upcoming releases. eXactML recognizes the following file extensions: .dtd DTD .xdr, .biz, .xml Microsoft XML Schema .xsd, other extensions W3C XML Schema
Copy and add	Check Copy and add to project to copy the specified DTD or schema file to your project. This option makes it easy to add your DTD or schema file to your source code control system along with the rest of your project.

6. Click Next> to display the eXactML AppWizard Step 2.

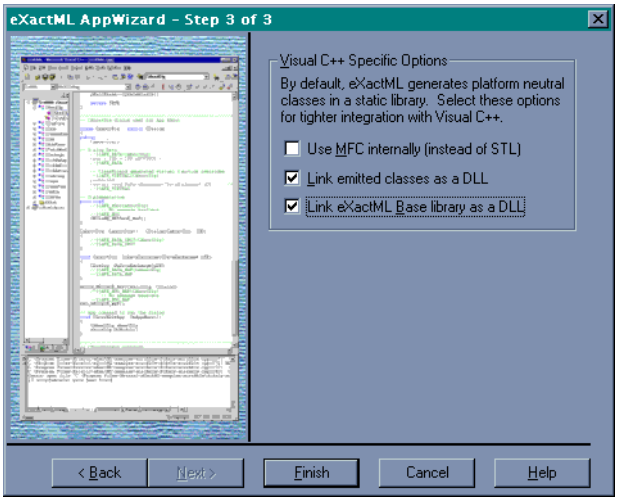


7. Specify the following options:

Option	Description
Namespace	Specify the namespace used by the generated classes. The default namespace is the name of the DTD or schema file (without the extension). To specify a different namespace, clear the Use File Name checkbox and type the namespace in the Namespace edit box.
Disable use of C++ Namespaces	If the compiler on your target platform does not support C++ namespaces (for example, SPARCWorks 4.2), select this option to cause all class definitions to be prefixed with *`namespace_`*.

Option	Description
Save generated schema	From your DTD or schema, eXactML generates an intermediate schema in W3C XML Schema format. In this schema, eXactML rearranges elements to account for dependencies between elements and assigns names to all attributes that are unnamed in the original DTD or schema. eXactML uses this schema to generate interfaces for your DTD or schema. To save the intermediate schema created in memory by eXactML, check Save Generated Schema. eXactML saves the generated schema in the current project with the name eXactML_*basename*.scm, where *basename* is the name of the DTD or schema entered in step 5. If you are generating classes from a DTD, this option is particularly useful because you can edit the schema to add type constraints not available in DTDs, then regenerate classes with eXactML using the edited schema, resulting in better classes.
Disable XML read capability	When checked, this option defines the macro - EXACTML_SUPPRESS_READ_SUPPORT in the generated project. When this macro is defined, the compiled eXactML classes will not contain XML read support, thus decreasing code size. Enable this option only if you do not want to use eXactML classes for reading XML.

8. Click Next> to display the eXactML AppWizard Step 3.



9. Specify the following options:

Option	Description
Use MFC	To generate classes that use MFC data types internally, check Use MFC internally. By default, eXactML generates classes that use STL data types. When this option is selected, eXactML classes may still use some STL components internally, but will mainly use MFC. **Note:** If any names in your DTD are the same as MFC macro names (for example, TEXT), the Visual C++ compiler will generate an error when you compile your classes. In this case, either change the name in your DTD or in the classes generated by eXactML before compiling.

Option	Description
Link emitted classes as a DLL	To build a DLL from the classes generated by eXactML, check Create a DLL. This setting causes eXactML to add DLL information to the classes it generates and configure project build settings to build a DLL. By default, eXactML configures project build settings to build a static library. The resulting static library or DLL import library must be added to your application project's link line.
Link eXactML Base library as a DLL	To link the eXactML Base library as a DLL, select Link eXactML Base library as a DLL. This option is only enabled if Link emitted classes as a DLL is selected. By default, eXactML configures project build settings to link the eXactML base library as a static library.

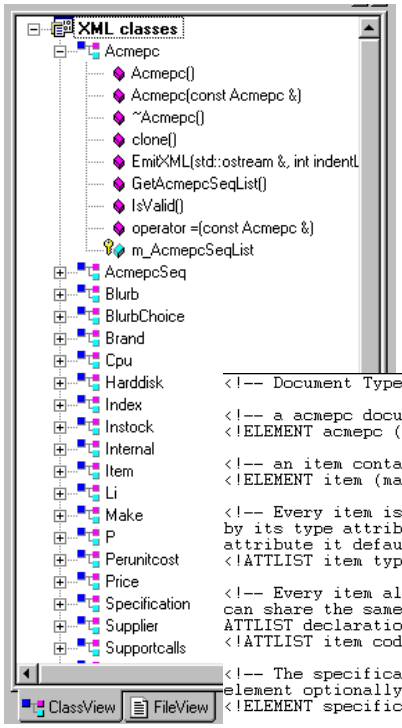
10. Click Finish. Visual C++ displays the New Project Information dialog, which lists the options you selected in the eXactML AppWizard.
11. Click OK.

eXactML creates source and header files that define interfaces for all elements in your DTD or schema. If you did not disable XML read capability, it also creates source files for the parser used to import XML documents into the interfaces.

If you specified, eXactML copies the original DTD or schema file and the generated schema file created by eXactML to your project. eXactML also configures your project settings to build the generated interfaces to your specification.

The classes generated by eXactML are particularly useful because they use names that match the element and attribute names in your DTD or schema rather than generic names such as "element" or "node."

Classes generated by eXactML



Source DTD

```
<!-- Document Type Definition for the acmepc catalog application -->

<!-- a acmepc document contains one or more items -->
<!ELEMENT acmepc (item)+>

<!-- an item contains these five sub-elements in this sequence -->
<!ELEMENT item (make,specification,price,blurb,internal)>

<!-- Every item is either a PC or a PRINTER which is indicated
by its type attribute. If a value is not supplied for this
attribute it defaults to PC -->
<!ATTLIST item type (PC|PRINTER) "PC">

<!-- Every item also has a code attribute. No two items
can share the same code. Note that we can use multiple
ATTLIST declarations for the same element if desired -->
<!ATTLIST item code ID #REQUIRED>

<!-- The specification element consists of a cpu
element optionally followed by a harddisk element -->
<!ELEMENT specification (cpu,harddisk?)>

<!-- The blurb element consits of one or more p or ul
elements -->
<!ELEMENT blurb (p|ul)+>
```

Using the eXactML Command

If you don't use Visual C++ for application development, you can still use eXactML to generate C++ classes by using the eXactML command. This command has the following syntax:

```
eXactML [-DLMNRX?] [-ddirectory] [-klicensekey]
[-llistfile] –nnamespace file
```

Flag	Description
-D	Configures the generated header file to export a DLL when compiled on Windows. Do not use this option if you are compiling on other platforms. By default, the generated classes are compiled into a static library.
-d	Specifies the directory where generated files are stored. The default is the current directory.
-k	Changes license information to use the specified license key and then exits. Use this option to change from an evaluation license to a development license.
-L	Returns the license status for your installation of eXactML.
-l	Specifies the name of the list file where eXactML writes a list of source files created as well as error information. The default is standard output.
-M	Uses MFC data types in generated classes. Use this option only if you plan to compile your classes on Windows only. Classes that use MFC cannot compile on other platforms. By default, eXactML uses STL data types in generated classes so the classes can be compiled on UNIX and Linux platforms. Note that if you use STL data types and compile on Windows, you must link to the multithreaded DLL version of the C++ runtime library. If you use the eXactML AppWizard, it configures this project setting for you automatically. eXactML may still use some STL components internally, but mainly uses MFC.

Flag	Description
-N	Causes eXactML to generate code that does not use namespaces. Use this option if your compiler does not support namespaces (for example, SPARCWorks 4.2). When this option is used, all classes are given the *namespace*_ prefix and all references to eXactML objects use `eXactML_` instead of `eXactML::`. Note that in situations where you need to compile on multiple platforms and wish to use namespaces when possible, you must run eXactML twice (once without this option and once with it) and maintain two sets of code. However, you may use code generated with the -N option on compilers that do have namespace support, enabling you to maintain a common source code base if desired.
-n	Specifies the namespace for the generated classes.
-R	Suppresses the generation of code to support reading of XML documents. Enable this option only if you do not want to use eXactML classes for reading XML.
-X	Saves the intermediate schema file generated by eXactML. From your DTD or schema, eXactML generates an intermediate schema in W3C XML Schema format. In this schema, eXactML rearranges elements to account for dependencies between elements and assigns names to all attributes that are unnamed in the original DTD or schema. eXactML uses this schema to generate interfaces for your DTD or schema. eXactML saves the generated schema with the name eXactML_*basename*.scm, where *basename* is the name of the DTD or schema file. If you are generating classes from a DTD, this option is particularly useful because you can edit the schema to add type constraints not available in DTDs, then regenerate classes with eXactML using the edited schema, resulting in better classes.
file	The pathname of the DTD or schema file generated classes should be based upon.

Chapter 3
Using Generated Interfaces

Types of Interfaces Generated

eXactML generates classes for each element in your DTD or schema. The names of the classes correspond to the element names in your DTD or schema. For each element, eXactML classes provide a default constructor, copy constructor, assignment operator, destructor, and the following member functions, all defined in the header file generated by eXactML.

- Get() member function
  eXactML generates a Get() member function for each element or attribute. This member function returns the attribute value. Call HasAttribute() first for attributes.
- Set() member function
  eXactML generates a Set() member function for each element attribute, unless the element is fixed. This member function sets the value of the attribute.
- EmitXML() member function
  This member function writes valid, well-formed XML content for the element to a stream. It returns trueon success.
- IsValid() member function
  This member function verifies that the element data is of the correct datatype. See the XML Schema Support section for a list of supported XML datatypes and the associated eXactML datatypes. Note that for datatypes implemented as strings in eXactML, eXactML only checks whether it is a string; it does not check values. For example, the XML datatype date is implemented as a string in eXactML. IsValid() checks that the value is a string, but does not check that it is a valid date. Regular expressions are not supported in string literals.
- freeElems() member function
  For list elements, this member function iterates over the list and deletes all pointers in it. Destructors for classes that use a list class call this member function.

- Has() member function
  For elements that are associated with the zero or one wildcard, this member function determines whether a specified element exists before calling the Get() member function.
- Remove() member function
  For elements that are associated with the zero or one wildcard, this member function removes a specified element.

For more detailed information about these member functions, see Chapter 4, “ExactML Class Reference.”

Read Interfaces

By default, eXactML generates an XML parser when it generates interfaces for your DTD or schema. This parser is customized to recognize XML documents for the specified DTD or schema. Use the XMLImporter class in your application to invoke the parser to import an XML document and generate the appropriate element object structures. These objects may then be manipulated programatically by the interfaces generated by eXactML.

If you do not plan to use the eXactML read capability in your application, disable read capability in the eXactML Wizard when you generate interfaces for your DTD or schema.

For more detailed information about XMLImporter, see Chapter 4, “eXactML Class Reference.”

Sample Applications

eXactML provides the following sample applications that demonstrate the use of classes generated by eXactML. These samples are located in *`eXactML_install_directory`*`/samples`.

Sample	Description
acmepc	eXactML classes are generated from a DTD and use STL. Demonstrates read and write functionality.
scribble/dtdafx	eXactML classes are generated from a DTD and use MFC. Demonstrates write functionality only.

Sample	Description
scribble/schemaafx	eXactML classes are generated from a W3C XML Schema and use MFC. Demonstrates read and write functionality.
scribble/schemastl	eXactML classes are generated from a W3C XML Schema and use STL. Demonstrates write functionality only.
marketwatch	eXactML classes are generated from a W3C XML Schema and use MFC. Demonstrates write functionality only.

Using Generated Interfaces in Applications

Once you've generated classes with eXactML, you can use them in your application just as you would any class library.

For an example of how you might use classes generated by eXactML in an application, let's look at the acmepc sample. This sample is a simple console application that reads catalog order information from a data file and writes it out as an XML document. The data file may be a text document (identified with the .dat filename extension) or an XML document (identified with the .xml filename extension).

Constructing XML Content

The acmepc application builds the XML document by setting the value for each element attribute.

In this example, the acmepc element consists of one or more item elements. Each item element consists of five subelements (make, specification, price, blurb, and internal) as well as a type attribute and a code attribute.

The way that the application constructs the output XML document depends on whether the input file is a text document or an XML document.

Parsing a Text File

If the input file ends with the .dat filename extension, the acmepc application first opens the data file and reads it into a data stream:

```
getline(dataStrm, inputbuf);
```

The following code uses the Set_code() class generated by eXactML to set the value for the code attribute:

```
dataStrm >> inputbuf;
debug(“Code=”);
item.Set_code(inputbuf);
```

Once each item element is complete, it is added to the sequence of items that make up the acmepc element:

```
AcmepcSeq *pAcmecpseq = new acmepcSeq;
pAcmepcseq->Set_item(item);
acmepc.Get_acmepcSeqList().push_back(pAcmepcseq);
```

Parsing an XML File

The application first calls the Initialize() function to register the create functions for the generated classes with the XMLImporter. This function must be called before using any generated classes or the XMLImporter.

```
acmepcxml::Initialize();
```

Next, the application imports the XML file, constructing the acmepc element object:

```
try {
importer.ImportFromFile(sInputFileName, fPreprocess);
}
catch (eXactML::XException & e)
{
    std::cerr << e.GetMsg() << std::endl;
    std::cerr << "in " << e.GetSourceFile() << " at line
number " << e.GetSourceLine() << std::endl;
    return 1;
}

cout << "Read in XML file with no errors." << std::endl;
```

The importer’s GetXObject() function returns a pointer to the root object of the XML document. Use dynamic_cast() to cast this pointer to the appropriate class:

```
acmepc *acmepc = dynamic_cast<acmepcxml::acmepc *>
(importer.GetXObject());

cout << "Successfully cast XML importer root to
acmepcxml::acmepc" << std::endl;
```

Validating XML Content

When the acmepc element is constructed, the application validates it against the DTD with the IsValid() member function before writing the XML document:

```
try {
    acmepc.IsValid();
}
catch (eXactML::XException e) {
    cout << “Exception in validating XML” << std::endl;
    cout << e.GetMsg() << std::endl;
    return 1;
}
```

Note that the following example data files are included with the sample. Some of them contain data that does not match the DTD:

- `acmepc.dat` contains valid data that will generate a valid XML document.
- `badtype.dat` contains an invalid value, Scanner, for the product type. The DTD only allows PC and Printer as product type values.

  ```
  <!ATTLIST item type (PC|PRINTER) “PC”>
  ```

- `acmepc.xml` contains a valid acmepc XML document.

Writing the XML Document

Finally, the application generates the XML document, either to a file or to standard output:

```
if (argc == 3)
       acmepc.EmitXMLToFile(argv[2]);
else
       acmepc.EmitXML(cout);
```

Deleting Imported XML Objects

After an XML document has been imported, the acmepc sample frees the objects created during the parsing of the XML document:

```
XMLImporterBase::DeleteImportedXObject()
```

If this function were not called, a memory leak would result.

Distributing Applications That Use eXactML Classes

When you distribute applications that use classes generated by eXactML, the code you distribute depends on your eXactML Wizard settings.

If you selected the Create DLL option, you must distribute this DLL along with your application. If you used the default setting, the generated classes are built as a static library. You must configure your application's project settings to link to this DLL or static library.

If your application uses eXactML's XML read capability, you must also distribute the `exactmlbase` library and configure your application's project settings to link to this library. The eXactML distribution includes several versions of the eXactML Base Library; the version you distribute depends on your eXactML Wizard settings. The following table describes the various eXactML Base libraries. In this table, the following codes are used:

d = Debug
s = Static
n = No Namespace

C++ Namespaces	Library Type	Release or Debug	eXactML Base Library Name
Enabled	DLL	Release	exactmlbase.lib/dll
Enabled	DLL	Debug	exactmlbase_d.lib/dll
Enabled	Static library	Release	exactmlbase_s.lib
Enabled	Static library	Debug	exactmlbase_sd.lib
Disabled	DLL	Release	exactmlbase_n.lib/dll
Disabled	DLL	Debug	exactmlbase_nd.lib/dll
Disabled	Static library	Release	exactmlbase_ns.lib
Disabled	Static library	Debug	exactmlbase_nsd.lib

To use read capability without distributing `exactmlbase`, build the source code provided in the Source folder of the eXactML installation directory into your application.

Important! The project settings for `exactmlbase.dll`, the DLL generated for your eXactML classes, and your main application must match. Otherwise, your application may fail during XML read while performing STL memory cleanup operations in `exactmlbase.dll`. Since `exactmlbase.dll` uses the multithreaded version of the Visual C++ C runtime library, the generated DLL and your main application must use this version in their project settings as well. The same applies to the debug configuration as well.

If you wish to statically link the C runtime library into your application, rebuild the exactmlbase debug and release DLLs using that version of the C runtime library.

Use the `exactmlbase.dsp` project file installed with eXactML to rebuild `exactmlbase.dll` with different project settings.

Important! If you are distributing your application on UNIX platforms, use the makefile provided in the Source folder to build the exactmlbase library on each platform.

Chapter 4
eXactML Class Reference

Namespaces

All eXactML classes are defined in the C++ namespace named "eXactML" unless the no namespace option is used. If this option is selected, no C++ namepsace is declared and all eXactML objects are prefixed with “eXactML_”. Note that many of these classes are implemented differently depending on whether you are using STL or MFC.

XObject

The XObject class acts as the base class for all other eXactML classes, including those generated from a DTD or schema. It implements the following public methods:

`XObject()`
The default constructor

`XObject(const XObject & other)`
The copy constructor

`~XObject()`
A virtual destructor

`XObject & operator=(const XObject & other)`
An assignment operator which copies another XObject's tag value into the current XObject

`int operator!=(const XObject & other)`
This inequality comparison operator returns false if the XObjects' tags are equal, otherwise it returns true.

`int operator==(const XObject & other)`
This equality comparison operator returns true if the XObjects' tags are equal, otherwise it returns false.

`bool EmitXMLToFile(const char * pFileName)`
Emits the current object's contents as XML to a given file. Returns false if the emit process fails, otherwise returns true.

`bool EmitXMLToFile(const std::string & sFileName)`
This function is identical to the previous one, but it takes a CString (in MFC mode) or std::string (in STL mode) argument instead of a const char *.

`void EmitXMLHeader(std::ostream & oStrm)`
Emits an XML header to a stream. It is called internally by `EmitXMLToFile()`, but may be called by an application that wants to emit directly to a stream but needs to emit the XML header first.

`bool EmitXML(std::ostream &, int indentLevel = 0)`
A pure virtual function. This function is implemented in every class derived from XObject to emit that object's contents as XML to the stream parameter.

`bool EmitAsAttribute(std::ostream &)`
Emits the contents of the object in attribute syntax to the stream parameter.

`void SetTag(const std::string & pTag)`
Sets the XML tag associated with the object.

`const std::string & GetTag() const`
Returns the XML tag associated with the object.

`bool IsValid()`
Checks an object to see if it conforms to the DTD or schema from which it was generated. It returns true if the object is valid, and false if it is not valid. `IsValid()` may throw an `eXactML::XException()` object when an invalid condition occurs.

`bool HasAttribute(std::string attrName)`
Returns true if the specified attribute name exists; otherwise, returns false. Use this method before the `Get` method.

`void RemoveAttribute(std::string attrName)`
Deletes the specified attribute.

`XObject * clone() const`
This pure virtual function is implemented by every derived class to create a deep copy of the current object.

XDatatype

XDatatype is a template class derived from XObject. It implements all the public XObject interfaces. It also uses the `eXactML::XFacet` class to implement XML schema datatypes with constraints. Generated classes derived from XDatatype will add the appropriate XFacets to the object during construction. When the `XDatatype::IsValid()` function is called, all of the appropriate XFacets are compared against the value in the datatype to determine if it meets the constraints.

XDatatype implements the following additional functions, where T is the template parameter type:

`T GetValue()`
Get the value stored in this datatype.

`void SetValue(const T value)`
Set the value stored in this datatype.

XList

This template class is derived from XObject and also from either CList (in MFC mode) or std::list (in STL mode). The template parameter for XList should always be a pointer to a class derived from XObject. XList inherits the interfaces of XObject and either CList or std::list. When pointers are passed into an XList, the XList is now responsible for deallocating that memory unless that pointer is removed from the list. To implement this behavior, the following XList member function is provided:

`void freeElems()`
Called during object destruction, this function iterates over the list and deletes all the element pointers. It then removes empties the list.

XMLImporterBase

This class is an abstract class used as the base for every XMLImporter class which is generated for a given DTD or schema. It implements the following public methods:

`void ImportFromFile(const std::string sFileName, bool fPreprocess = true)`

This function is used to read in an XML document whose filename is specified by `sFileName`. If the second parameter is false, the importer will not call the entity preprocessor before parsing the XML document. `ImportXMLFromFile()` will call the protected PreProcess() function to preprocess the file (if preprocessing is enabled) and then call the virtual Import() function to parse the file. If it is known that all XML documents for this DTD or schema do not make use of entity references, a significant performance gain may be achieved by setting fPreprocess to false. If the specified file cannot be found or opened, ImportFromFile() will throw an appropriate eXactML::XException. Exceptions may also be thrown by the PreProcess() or Import() functions which are called by ImportFromFile(). These exceptions will be passed up to the calling program.

`eXactML::XObject * GetXObject()`

After ImportFromFile() has been called, this function will return the root object of the XObject tree populated by the parsing of the XML document. Applications should use dynamic_cast to cast the XObject pointer to the appropriate expected root node. Note that the pointer to the actual allocated tree is passed here to avoid the overhead of making a copy. This tree will not be deleted automatically when the XMLImporter is destroyed. Instead, call the static XMLImporterBase::DeleteImportedXObject() function on the root node to delete the tree.

`std::string GetOriginalFileName()`

Returns the file name of the file passed into ImportFromFile(). Mainly to be used for problem determination when a file fails to parse.

`std::string GetPreProcessedFileName()`

Returns the filename of the file used during the PreProcess() step called by ImportFromFile(). This information may be needed for problem determination when a file fails to parse.

`static void DeleteImprotedXObject(eXactML::XObject * pXObject)`

This static function should be called to delete the tree of objects created during the ImportFromFile() XML document parse.

XObjectFactory

Under normal circumstances, an application will not directly reference the static functions of the XObjectFactory class. If an application derives classes from eXactML-generated classes, however, the XObjectFactory functions may be used to ensure that the derived classes will be used during the parsing of an XML document. XObjectFactory implements the following public methods:

```
static void RegisterCreateFunction(std::string
sObjectName, CREATE_FUNC_PTR pCreateFunc)
```

This function registers a create function with the XObjectFactory. A create function has the following prototype:

```
static XObject * create(void)
```

All eXactML-generated classes have a static create() member function. This function simply calls the new operator to allocate an instance of a class and return the allocated pointer. If an application uses classes derived from eXactML-generated classes, these derived classes should also implement a static create() member function. The RegisterCreateFunction() call maps fully-qualified class names to the corresponding create functions so that the XMLImporter() can create instances of the correct objects when parsing XML documents. Use this function to register your own create function for a derived class. If a create function is already registered for a given class name, the new function will override the existing function. Note that when registering a derived-class create function, the sObjectName parameter should be the fully-qualified base class name. This lets the XMLImporter create the appropriate subclass when it is importing an XML document. See the `scribble\schemaafx` sample for an example of how this is done.

Initialize Function

The eXactML tool generates an Initialize() function within the designated namespace whenever read functionality is enabled. This Initialize() function must be called before an XMLImporter is used to import an XML document. The generated Initialize function calls XObjectFactory::RegisterCreateFunction() to register all the create functions for the eXactML-generated classes. This lets the XMLImporter create the objects as it parses the XML document. If an application has derived classes from the eXactML-generated classes, it should call XObjectFactory::RegisterCreateFunction() for those classes after the Initialize() function has been called. See the `scribble\schemaafx` sample for an example of how this is done.

XML Schema Support

XML schema support in eXactML is based on the 24 September 1999 and 5 November 1999 drafts from the World Wide Web Consortium. Because this standard is still evolving, Bristol Technology will continue to track it closely and implement changes in upcoming releases.

Relevant URLs are:

- http://www.w3.org/TR/1999/WD-xmlschema-1-19991105/ - XML Schema Structures
- XML Schema Datatypes
  http://www.w3.org/TR/1999/WD-xmlschema-2-19991105/
- XML Schema Structures
  http://www.w3.org/TR/1999/WD-xmlschema-1-19990924/
- XML Schema Datatypes
  http://www.w3.org/TR/1999/WD-xmlschema-2-19990924/
- W3C XML Homepage
  http://www.w3.org/XML/

eXactML also supports the Microsoft XML Schema format. For more information on this format, see http://msdn.microsoft.com/xml/reference/schema/start.asp.

Structures

eXactML supports XML Schema structures except the following, for which the W3C working group has not yet reached consensus:

- Attribute Group Definition
- Named Model Group
- Archetype Refinement
- Entities and Notations
- Schema Composition and Namespaces

Datatypes

eXactML implements XML schema datatypes as follows. Note that eXactML classes validate that all generated XML is of the correct type, but do not necessarily validate that the data is correct. For example, if your XML content contains an element of datatype date, eXactML classes verify that the element is string but not that it is a valid date.

Schema Datatype	Datatype used by eXactML Classes (STL)	Datatype used by eXactML Classes (MFC)
string	std::string	CString
boolean	bool	bool
real	double	double
timeInstant	std::string	CString
timeDuration	std::string	CString
recurringInstant	std::string	CString
binary	std::string	CString
uri	std::string	CString
language	std::string	CString
NMTOKEN	std::string	CString
NMTOKENS	std::list<NMTOKEN>	CList<NMTOKEN>
Name	std::string	CString
NCName	std::string	CString
ID	std::string	CString
IDREF	std::string	CString
decimal	double	double
integer	long	long
non_negative_ integer	unsigned long	unsigned long
positive_integer	unsigned long	unsigned long
non_positive_ integer	long	long
negative_integer	long	long
date	std::string	CString

Schema Datatype	Datatype used by eXactML Classes (STL)	Datatype used by eXactML Classes (MFC)
time	std::string	CString

Groups

Groups in XML Schema are equivalent to parenthetical phrases in DTD !ELEMENT declarations. eXactML implements groups as separate classes rather than using a generic list of subelements for each class. This allows eXactML to provide easy checking to assure validity of subelements.

Unfortunately, XML Schema groups are often not named, and DTD parenthetical phrases certainly are unnamed. In these cases, eXactML generates a name based on the parent archetype name and the word "Choice" (in the case of a choice) or "Seq" (in the case of a sequence). For example, consider the following snippet from a DTD:

```
<!ELEMENT Item (Description, (UPC | SKU | Model))>
```

A schema representing this type might look like:

```
<archetype name="Item" order="seq">
  <element name="Description" type="Description"/>
  <group order="choice">
    <element name="UPC" type="UPC"/>
    <element name="SKU" type="SKU"/>
    <element name="Model" type="Model"/>
  </group>
</archetype>
```

In this case, eXactML will create a class for "Item" to represent the archetype and also one called "ItemChoice" to represent the choice of UPC, SKU or Model. Class Item will have a m_ItemChoice data member and corresponding accessor functions `SetItemChoice()` and `GetItemChoice()`. The ItemChoice class will contain a pointer to XObject, and have `SetChoice()` and `GetChoice()` functions. In this fashion, either a UPC, SKU or Model object can be placed into the ItemChoice object. When `ItemChoice::IsValid()` is called, ItemChoice will verify that the choice is a valid UPC, SKU or Model.

If you prefer to have better names for your choices or sequences, run the eXactML utility once and select the "Save generated schema" option. You can then modify the group names in the generated schema an rerun eXactML with the modified schema as input.

See the eXactML Release Notes for a description of current limitations in schema support.

Chapter 5
Error Messages

There are three categories of eXactML error messages: messages from the DTD-to-Schema parser, messages from the eXactML code generator, and messages from the eXactML AppWizard.

DTD-to-Schema Parser Error Messages

The following error messages may be generated when converting your DTD into a W3C XML Schema for use with eXactML. They are limited to DTD processing.

Error: Unable to open DTD input file
Make sure that you specified the correct pathname for your DTD. You must have read permission for this file.

Error: Unable to open DTD preprocessor output file
Make sure that you have write permission for the directory you specified for eXactML output.

Error: <DTD_filename>: line <#>: <description> at token "<token>" in the following line: <line number>
For details on the source of this error, compare this line against the appropriate production rule in the XML spec at http://www.w3.org/TR/REC-xml.
The specified token is not used in your DTD in accordance with the W3C XML specification. The <description> may be one of the following:

- Parse failed for DTD file: unable to continue.
- Reference to undefined symbol
- Defined symbol has no value

- Failed to open include file <filename>
- ftell failed
- Don’t know how to handle token ID
- Invalid character
- Problem with entity reference
- Entity reference undefined

Error: Stack Overflow - unable to continue
Send email with your DTD or schema file to support@bristol.com.

Error: Stack Underflow - unable to continue
Send email with your DTD or schema file to support@bristol.com.

Error: Symbol table full
Send email with your DTD or schema file to support@bristol.com.

Error: Lexemes array full
Send email with your DTD or schema file to support@bristol.com.

Error: Invalid symbol table index
Send email with your DTD or schema file to support@bristol.com.

Error: Too many errors - unable to continue
eXactML attempts to recover from errors and continue processing, but if it encounters an excessive number of errors in your DTD, it exits with this message.

Error: <DTD filename>: line <#>: External entity reference to URL at token “<token>” in the following line: <line text>.
This version of eXactML is unable to resolve external entity URL references such as `http://www.w3.org/TR/xhtml1/DTD/xhtml-lat.ent`. To work around this limitation, place the information from the specified URL into a file with the same name (for example, `./xhtml-lat.ent`). When eXactML encounters an external entity URL reference, it looks for a local file with the same name and, if one exists, uses it instead.

eXactML Class Generator Error Messages

The following errors may be generated by eXactML during the process of generating classes for your DTD or schema:

Error during XML4C2 Initialization
The IBM XML4C2 parser used to parse schemas is unable to start. This usually indicates an installation problem. Uninstall and then reinstall eXactML. If the message persists, see http://www.alphaworks.ibm.com/tech/xml4c for more information about the XML4C parser.

An Error occurred during parsing
The IBM XML4C2 parser used to parse schemas encountered an error. See http://www.alphaworks.ibm.com/tech/xml4c for more information about the XML4C parser.

eXactML License error, please contact Bristol for a license.
The serial number entered during eXactML installation is incorrect or has expired. Send email to support@bristol.com to request a new serial number.

Error: parse failed: unable to continue
The IBM XML4C2 parser used to parse schemas failed. This usually indicates an installation problem. Uninstall and then reinstall eXactML. If the message persists, see http://www.alphaworks.ibm.com/tech/xml4c for more information about the XML4C parser.

Error: Caught STL exception while <function>
Send email with your DTD or schema file to support@bristol.com.

Error: XMLException <exception> while <function>
Send email with your DTD or schema file to support@bristol.com.

Error: Caught a DOM exception with type <type> Message: <message>
Send email with your DTD or schema file to support@bristol.com.

Error: Caught an unknown exception while <function>.
Send email with your DTD or schema file to support@bristol.com.

Microsoft XML Schema Errors

The following errors may be generated during processing of Microsoft XML Schema files. They all indicate errors in the schema. For more information about the Microsoft XML Schema, see http://msdn.microsoft.com/xml/reference/schema/start.asp.

- Error: Found an XML-Data <datatype> declaration for ElementType ‘<item_name>’ without a dt:type attribute. Cannot continue.
- Error: Found an XML-Data ElementType without a name. Cannot continue.

- Error: Found an XML-Data ElemntType declaration with a null name. Cannot continue.
- Error: Found an XML-Data AtrributeType declaration without a name. Cannot continue.
- Error: Found an XML-Data AttributeType declaration with a null name. Cannot continue.
- Error: Found an XML-Data <datatype> declaration for AttributeType ‘<item_name>’ without a dt:type attribute. Cannot continue.
- Error: Found an XML-Data enumeration declaration for AttributeType ‘<item_name>’ without a dt:values attribute. Cannot continue.
- Error: Found an XML-Data <attribute> declaration with a type specification. Cannot continue.
- Error: Found an XML-Data <attribute> declaration of type ‘<item_name>’ without a corresponding AttributeType declaration. Cannot continue.

eXactML AppWizard Error Messages

The following error messages may be generated by the eXactML AppWizard:

Cannot create temporary file; out of disk space?

Make sure that you have available disk space.

Error: No input file

You must specify your DTD or schema on the first page of the eXactML AppWizard.

Error: License failure

The serial number entered during eXactML installation is incorrect or has expired. Send email to support@bristol.com to request a new serial number.

Error: Unknown exit code

Send email with your DTD or schema file to support@bristol.com.

Can’t open listfile to show errors!

eXactML was unable to open the listfile created during class generation that contains any errors.

Could not open exactml.exe listfile

eXactML was unable to open the listfile created during class generation that contains any errors.

Chapter 6
Frequently Asked Questions

Is eXactML a Windows product only, or is it available for other platforms?

The eXactML product itself runs only on Windows, but it generates classes that you can build and use on other platforms. For the current version, we've verified that you can build and use classes generated by eXactML on Solaris and Linux. For details, see the eXactML 1.0 Release Notes.

Do I have to use Visual C++ to use eXactML?

Tight integration with Visual C++ means that eXactML can automatically configure project settings required to build eXactML classes and build your eXactML generated classes along with your project. However, if you do not wish to use Visual C++, you can use the eXactML command line interface to generate your classes. The eXactML executable is located in *`eXactML_install_directory/`*`bin`.

Do I need to distribute any eXactML files with my application?

You must ship the code generated by eXactML (typically the DLL created by the AppWizard). If you use eXactML's XML read capability, you must also link your application to and distribute the `exactmlbase.dll` file. The eXactML installation includes the source for `exactmlbase.dll`, so you can build it with your application if you don't wish to distribute a separate DLL or if you need to build it for other platforms.

If you distribute the source code for your application, you may also distribute the source files for the eXactML classes. You may not distribute the source for `exactmlbase.dll`.

Does eXactML read in DTD or schema definitions?

eXactML reads in DTD, W3C XML Schema, and Microsoft XML Schema definitions. eXactML generates a corresponding W3C XML Schema definition, enabling you to quickly move from DTDs or Microsoft XML Schemas to W3C XML Schemas if you choose.

Which schema standard does eXactML support?

eXactML supports both the W3C XML Schema and the Microsoft XML Schema standards. For more information, see Chapter 4, "eXactML Class Reference."

What is the difference between eXactML and DOM?

The main difference is that the classes generated by eXactML unique to your DTD or W3C XML Schema, unlike the generic interfaces provided by DOM. For example, if your DTD has a Person element with attributes FirstName, LastName, and Age, the classes generated by eXactML use those element and attribute names, so that the class Person includes Get_FirstName, Get_LastName, and Get_Age member functions.

Can I use SGML extensions in my DTD?

No. eXactML supports DTD that conform to the XML 1.0 DTD only.

How do I upgrade my evaluation license to a developer license?

If you have an evaluation license of eXactML installed, upgrade to a development license as follows:

1. Purchase a developer license from http://store.bristol.com. A developer license key will be sent to you via email within one business day.
2. Run the following command at the DOS prompt, where *licensekey* is the developer license key you received from Bristol:

   ```
   exactml -klicensekey
   ```