

XML en C++ con CodeSynthesis XSD

SÍNTESIS

AtlantCar, Fotolia

Procesamos XML en programas C++ sin tener que lidiar con DOM. **POR BEN MARTIN**

Los desarrolladores de software suelen utilizar XML para el paso de datos de entrada o de salida en sus programas. Hay varias APIs que proporcionan medios con los que procesar datos XML. Algunas opciones populares, como SAX (*Simple API for XML*) o DOM (*Document Object Model*), ofrecen sistemas estándar de procesado de datos XML desde los programas. Otra opción interesante a la hora de trabajar con datos XML en C++ es CodeSynthesis XSD, un compilador de fuentes abiertas de asociación de datos (Figura 1). Dado un esquema XML, CodeSynthesis genera clases de C++ con las que acceder a los datos guardados en el documento XML...É

É utilizado tipos y funciones que se corresponden automáticamente con el dominio de la aplicación” [1].

CodeSynthesis XSD simplifica la tarea de parsear el XML y elimina la necesidad de utilizar toscas instancias DOM.

CodeSynthesis XSD

CodeSynthesis XSD hace del manejo de XML algo menos tedioso: Se acabó el tener que parsear y el tener que interactuar con DOM [2]; en lugar de eso, sólo trabajamos normalmente con objetos C++. CodeSynthesis XSD es capaz además de serializar estos objetos de nuevo a XML sin que haya que lidiar con la lectura y escritura de XML, y permitiendo al des-

arrollador centrarse en el funcionamiento de la aplicación en vez de en los detalles del código.

CodeSynthesis XSD tiene dos modos de operación: Podemos usarlo con un árbol en memoria (al estilo de DOM) o en modo parseado (al estilo de SAX). Para la elaboración de este artículo usaremos la modalidad de árbol. Aunque los ejemplos que se pueden encontrar en el sitio web de CodeSynthesis XSD utilizan todos *xsd* como comando principal de CodeSynthesis XSD, en Fedora 11, vimos que debíamos usar *xsdccx* para llegar hasta el comando de CodeSynthesis XSD. Como ejemplo funcional usaremos el archivo *customers.xml* mostrado en el Listado 1.

Listado 1: customers.xml

| | | |
|---------------------------------|-------------------------------|--------------------------------|
| 01 <?xml version="1.0"?> | 07 </customer> | 12 <gender>male</gender> |
| 02 <customers> | 08 <customer id="2"> | 13 |
| 03 <customer id="1"> | 09 <first-name>Charles | <dob>1903-11-20T06:30:13</dob> |
| 04 | </first-name> | 14 </customer> |
| <first-name>Bart</first-name> | 10 <middle-name>Montgomery | 15 </customer> |
| 05 <sur-name>Simpson</sur-name> | </middle-name> | |
| 06 <gender>male</gender> | 11 <sur-name>Burns</sur-name> | |

Listado 2: Esquema de customers.xsd

```

01 <?xml version="1.0"?>
02 <xs:schema
   xmlns:xs="http://www.w3.org/20
01/XMLSchema">
03 <xs:simpleType
   name="gender_t">
04 <xs:restriction
   base="xs:string">
05 <xs:enumeration
   value="male"/>
06 <xs:enumeration
   value="female"/>
07 </xs:restriction>
08 </xs:simpleType>
09 <xs:complexType name="cus-
   tomer_t">
10 <xs:sequence>
11 <xs:element
   name="first-name"
   type="xs:string"/>
12 <xs:element
   name="middle-name"
   type="xs:string" minOccurs="0"/>
13 <xs:element name="sir-name"
   type="xs:string"/>
14 <xs:element name="gender"
   type="gender_t"/>
15 <xs:element name="dob"
   type="xs:dateTime" minOccurs="0"/>
16 </xs:sequence>
17 <xs:attribute name="id"
   type="xs:unsignedInt"
   use="required"/>
18 </xs:complexType>
19 <xs:complexType name="cus-
   tomers_t">
20 <xs:sequence>
21 <xs:element name="customer"
   type="customer_t"
   minOccurs="0" maxOccurs="unbounded"/>
22 </xs:sequence>
23 </xs:complexType>
24 <xs:element name="customers"
   type="customers_t"/>
25 </xs:schema>

```

Por el camino, haremos algunos cambios menores conforme vayamos explorando nuevas funcionalidades. Cualquier parecido de los datos contenidos en el archivo *customers.xml* con la realidad o la ficción es pura coincidencia.

Como su propio nombre indica, el interés de CodeSynthesis XSD se centra princi-

palmente en los archivos *.xsd* que proporcionan el esquema XML. Para crear una asociación C++ que parsee un archivo XML con CodeSynthesis XSD, necesitamos un archivo de esquema XML. El archivo *customers.xml* cumple con el esquema del archivo *customers.xsd* mostrado en el Listado 2. Leyendo el esquema

de arriba a abajo, se puede distinguir el elemento *customers*, de tipo *customers_t*. El elemento de tipo *customers_t* contiene un listado de elementos de tipo *customer_t*.

Cada *customer_t* tiene varios nombres, un género y una fecha de nacimiento (*dob*). Nótese que, dado que el nombre, el

Listado 3: Cliente C++ para Parsear customers.xml

```

01 #include <iostream>
02 #include "customers.hxx"
03
04 using namespace std;
05 int main (int argc, char* argv[])
06 {
07     try {
08         xml_schema::properties props;
09         props.no_namespace_schema_
10         location ("customers.xsd");
11         auto_ptr<customers_t>
12         all_customers( customers(
13         argv[1], 0, props ));
14         customers_t::customer_sequence
15         & l = all_customers->customer();
16         for(
17             customers_t::customer_sequence
18             ::iterator ci = l.begin(); ci !=
19             l.end(); ++ci ) {
20                 cout << "first name:" <<
21                 ci->first_name() << " sirname:"
22                 << ci->sir_name();
23                 if( ci->middle_name().pre-
24                 sent() ) {
25                     cout << " segundo:" <<
26                     ci->middle_name();
27                 }
28                 if( ci->dob().present() ) {
29                     ::xml_schema::date_time dt =
30                     ci->dob().get();
31                     cout << " dob:" << dt.year()
32                     << "/" << dt.month() << "/" <<
33                     dt.day();
34                 }
35             }
36     } catch (const xml_schema::exception& e) {
37         cerr << e.what() << endl;
38         return 1;
39     } catch (const std::exception& e) {
40         cerr << e.what() << endl;
41         return 1;
42     }
43 }

```

Listado 4: Nuevo customers.xsd

```

01 $ cat customers.xsd
02 ...
03 <xs:element name="gender"
   type="gender_t"/>
04 <xs:element name="dob"
   type="xs:dateTime" minOccurs="0"/>
05 </xs:sequence>
06 <xs:attribute name="id"
   type="xs:ID" use="required"/>
07 </xs:complexType>
08
09 $ cat customers.xml
10 <?xml version="1.0"?>
11 customers
   xmlns:xsi="http://www.w3.org/2
001/XMLSchema-instance"
   xsi:noNamespaceSchemaLoca-
   tion="customers.xsd" >
12 ...
13 <customer id="c2">
14   <first-name>Charles</first-nam
   e>
15   <middle-name>Montgomery</mid-
   dle-name>
16 ...

```



Figura 1: CodeSynthesis XSD es mantenido y distribuido por la empresa sudafricana CodeSynthesis Tools CC.

Como se puede apreciar, incluso sin estar familiarizados con los archivos de esquema de XSD, no hay nada particularmente difícil en la creación de un esquema para un documento XML. Basta con describir los elementos neces-

arios, sus hijos y atributos. Todo lo que se describa en el esquema tendrá un tipo asociado; por ejemplo, *dob* es un *date-time*, de modo que se puede guardar el momento exacto del nacimiento procedente de un certificado en caso de disponer del mismo.

Ahora que ya tenemos el archivo de esquema XML, el código del cliente C++ +

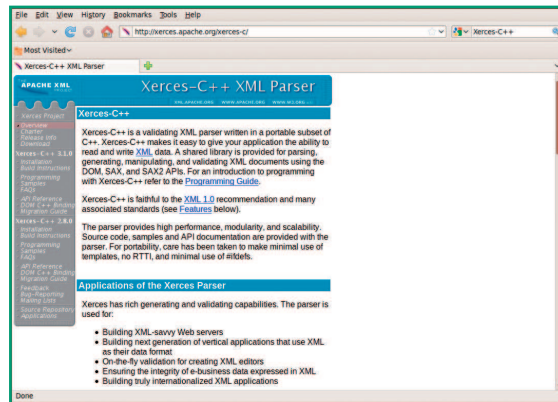


Figura 2: CodeSynthesis XSD, Xalan-C++ y XQilla están todos basados o pueden usar la librería XML Xerces-C++, que está disponible a través del sitio web de Apache.

mostrado en el Listado 3 es más simple si cabe. Del código hemos quitado intencionalmente cualquier dificultad: El archivo XML no referencia al archivo de esquema XSD al que sigue. Muchos sistemas XML han sido creados sin tener en cuenta los archivos de esquema, y por tanto, no refieren ninguno. Para manejar tales archivos XML sin CodeSynthesis XSD, hemos

El género está restringido a unas pocas posibilidades, por lo que debe contar con un *simpleType* propio en el archivo de esquema. La fecha de nacimiento y el segundo apellido son opcionales, de modo que incluyen un *minOccurs = "0"* en su esquema.

Listado 5: Buscando a Monty

```

01 #include <iostream>                                && ci->first_name() == "Charles"    y) );
02 #include "customers.hxx"                            ) {                                34   customer_t& montyByID =
03 using namespace std;                               18   monty = &( *ci);                (dynamic_cast<customer_t&>
04 using namespace xercesc;                           19   break;                           (t));
05                                                     20   }                                35
06 int main (int argc, char* argv[])                  21   }                                36   cout << "Buscando por ID..."
07 {                                                  22   // ¿Lo tenemos?                 << endl;
08 // Parseamos el XML                               23   if( !monty ) {                  37   cout << "Nombre:" << monty-
09 XMLPlatformUtils::Initialize                       24   cerr << "¡No podemos encontrar  ByID.first_name() << " apeli-
10 ();                                                al pobre Monty! saliendo..." << lido:" << montyByID.sir_name()
11                                                     endl;                                << endl;
12 auto_ptr<customers_t> all_cus-                       25   exit(1);                        38
13 tomers( customers( argv[1],                          26   }                                39   // Cambiamos a través de una
14 xml_schema::flags::keep_dom |                       27   cout << "nombre:" <<           referencia, visualizamos a
15 xml_schema::flags::dont_initialize, props );        monty->first_name() << " apeli-   través de otra
16 for(                                                 lido:" << monty->sir_name() <<   40   montyByID.first_name() =
17 customers_t::customer_sequence                       endl;                                "fred";
18 & l = all_customers->customer();                    28                                  41   cout << "puntero a monty, nom-
19 for(                                                 29   // Lo buscamos "por id"         bre:" << monty->first_name() <<
20 customers_t::customer_sequence                       30   const xercesc::DOMNode* root =   endl;
21 ::iterator ci = l.begin(); ci !=                     all_customers->_node();           42   }
22 l.end(); ++ci ) {                                   31   if( DOMDocument* dom =          43   }
23 if( ci->sir_name() == "Burns"                       root->getOwnerDocument() ) {     44   catch (const
24                                                     32   if( DOMElement* e =             xml_schema::exception& e ) {
25                                                     dom->getElementById( XML-        45   cerr << e << endl;
26                                                     String::transcode("c2") ) ) {   46   return 1;
27                                                     33   xml_schema::type& t ( *reinter- 47   }
28                                                     pret_cast<xml_schema::type*     48   }
29                                                     > ( e->getUserData
30                                                     (xml_schema::dom::tree_node_ke

```

de decirle al sistema qué archivo XSD sigue el archivo XML.

Las dos primeras líneas configuran la variable *props* con esta información, mientras que la tercera parsea el archivo *customers.xml* haciendo uso del esquema *customers.xsd*. El objeto C++ *all_customers* nos permite obtener cualquier cliente a partir de una interfaz al estilo de STL. Es posible iterar desde *begin()* hasta *end()* y obtener los nombres mediante el uso de funciones miembro en vez de hacerlo interactuando, con la probabilidad de errores que ello implica, con el DOM directamente. Los campos opcionales, como el segundo apellido o la fecha de nacimiento, los gestiona el método *present()* para ver si están presentes en el elemento actual.

El código del Listado 3 está basado en clases C++ de *customers.hxx*, generadas por CodeSynthesis XSD utilizando *customers.xsd*. El Makefile mostrado a continuación compilará un ejecutable a partir de este código C++ y del archivo *customers.xsd*. CodeSynthesis XSD presenta una dependencia de tipo sólo-en-tiempo-de-compilación, y no hace falta ninguna librería comparada adicional.

```
main: main.cxx customers.cxx
g++ main.cxx customers.cxx \
-lxerces-c -o main

customers.cxx: customers.xsd
Makefile
xsdcxx cxx-tree customers.xsd
```

El binario *main* generado es ejecutable, arrojando los siguientes resultados:

```
$. /main customers.xml
nombre:Bart apellido:Simpson
nombre:Charles apellido:Burns
segundo:Montgomery
dob:1903/11/20
```

Del Nodo Xerces-C++ DOM al Objeto C++

Puede que el desarrollador ya tenga un archivo XML con atributos ID en sus elementos, o que tenga pensado encontrar dichos elementos utilizando el lenguaje XPath (*XML Path Language*) [3]. Una vez encontrado un DOMNode con XPath, se puede convertir a un objeto de C++ con CodeSynthesis XSD. Si además nos encar-

XQuery: De XQilla a Objetos C++

Hasta ahora este artículo se ha centrado en cómo hacer uso de CodeSynthesis XSD para simplificar el proceso de consumir y generar XML desde C++. Sin embargo, no hay por qué dar el control absoluto a CodeSynthesis XSD; en lugar de eso, podemos buscar elementos de un DOM por nuestros propios medios y pedir a CodeSynthesis XSD que nos dé el objeto C++ correspondiente al elemento XML que previamente hemos encontrado.

El proyecto XQilla project (Figura 3) [8] [9], usado para consultar datos XML, proporciona soporte tanto para el lenguaje de rutas XML XPath 2.0 [3] como para el lenguaje XQuery [10]. XQilla supone un enorme potencial a la hora de generar XML y de definir elementos XML. Mediante el uso conjunto de CodeSynthesis XSD y XQilla, aunamos la disponibilidad de información XML y el acceso a ésta a través de objetos C++. Para aquellos que no estén familiarizados con XQuery, se trata de un

lenguaje de consulta FLOWR (*For, Let, Order by, Where, Return*) [11] parecido al lenguaje de consulta SQL. Con XQuery podemos integrar consultas en XML y también generar nuevo XML a partir de los resultados de las consultas. El siguiente ejemplo es un XQuery que devuelve un único cliente desde el archivo *customers.xml*:

```
$ cat customers.xq
<customers>
{
  for $c in //customer
  where $c/middle-name = "Montgomery"
  return $c
}
</customers>
```

Nótese que el elemento XML del cliente aparece en el propio XQuery. La consulta a ejecutar se encapsula entre { }, y el resultado de la declaración *return* reemplazará completamente lo que quiera que haya entre los corchetes, así como los propios corchetes.

La declaración *for* opera sobre los nodos seleccionados con XPath; en este caso inspeccionando cada nodo de cliente. La cláusula *where* se podría haber ubicado dentro de la expresión XPath, formando *//customer[@middle-name="Montgomery"]*. La declaración *return* simplemente devuelve el elemento XML de cliente al completo.

El comando mostrado en el Listado 8 ejecuta el anterior archivo de XQuery *customers.xq*. Cabe destacar que el programa *xqilla* no decora los resultados, por lo que se los enviamos a *xmllint* para que adopten un aspecto algo más asejable para el ojo humano. El argumento *-i* de *xqilla* le indica que debe asociar el archivo *customers.xml* como contexto predeterminado. En el anterior XQuery se puede apreciar que no se hace mención a archivo alguno; la consulta simplemente da por sentado que podrá obtener el/los cliente(s) de alguna parte en "el documento". Esta separación es muy potente, en tanto que se puede usar el mismo XQuery para procesar muchos archivos. En este caso, queremos *customers.xml*, por lo que lo asociamos como "el documento" que usa la consulta. El resultado de la ejecución de la consulta sobre *customers.xml* es un archivo XML que cumple con el esquema del archivo *customers.xsd* pero que sólo contiene un único cliente: Mr. Burns.

El truco de usar CodeSynthesis XSD con XQilla consiste en generar el DOM con XQilla y pasárselo a CodeSynthesis XSD para crear el modelo del objeto C++ a partir de un DOM que seguimos poseyendo. Una vez en disposición del DOM y de los objetos C++, podemos buscar elementos DOM particulares con XQilla para convertirlos a objetos C++. Consultar el sitio web de Linux Magazine para ver un listado que muestra cómo convertir los resultados de XQuery a objetos C++ [12].



Figura 3: La librería XQilla está disponible para descarga bajo la licencia Apache v2.

gamos de que el atributo *id* sea un atributo ID de XML [4] correcto, entonces podremos incluso encontrar directamente al cliente mediante `DOMDocument::getElementById()` [5].

El archivo de esquema `customers.xsd` debe cambiar para indicar que el atributo *id* es de tipo `xs:ID` y no un simple entero. El archivo `customers.xml` tiene que cambiar, de modo que cada ID comience por una letra. Cabe destacar que el archivo `customers.xml` incluye ahora un enlace a su archivo de esquema XML.

El código C++ se muestra en el Listado 5. El archivo XML se carga de un modo algo diferente. Para empezar, se ini-

cializa el tiempo de ejecución y se usa `keep_dom` para indicar a CodeSynthesis XSD que debería conservar el DOM después de parsear el XML. Durante la primera vuelta, al iterar sobre los nombres de los clientes, se toma un puntero a Mr. Burns. Tras mostrar Mr. Burns al usuario, se obtiene el DOM mediante el uso del método `_node()` del objeto C++ que creó CodeSynthesis XSD para el XML.

A través de este `DOMNode` se obtiene el `DOMDocument`, y se usa `getElementById()` para sacar a Mr. Burns por su ID de XML. Por supuesto, una vez disponemos del `DOMElement*` correspondiente a Mr. Burns, podemos usar la API de DOM para

obtener más información. Sin embargo, como ya estamos usando CodeSynthesis XSD, quizá convenga más obtener el objeto CodeSynthesis de C++ generado mediante el XSD correspondiente a Monty desde el `DOMElement`, que es lo que hacen `reinterpret_cast` y `dynamic_cast`.

A fin de demostrar que ambas referencias a Mr. Burns llevan al mismo objeto, cambiamos su nombre usando la referencia `reinterpret_cast`, y la sacamos por pantalla mediante la referencia original obtenida a partir del objeto C++.

En vez de simplemente coger los elementos por su ID, puede que queramos encontrarlos usando XPath. El ejemplo

Listado 6: Xalan-C++ y XPath a Objetos C++

```

01 #include <iostream>           ize();
02 #include "customers.hxx"     23 xml_schema::properties props;
03 #include <Xalan/Include/PlatformDefinitions.hpp> 24
04 #include <Xalan/PlatformSupport/XSLException.hpp> 25 auto_ptr<customers_t>
05 #include                    26 all_customers( customers(
    <Xalan/DOMSupport/XalanDocumentPrefixResolver.hpp> 27     argv[1],
06 #include <Xalan/XPath/XObject.hpp> 28     xml_schema::flags::keep_dom |
07 #include <Xalan/XPath/XPathEvaluator.hpp> 29     xml_schema::flags::dont_initialize, props ) );
08 #include <Xalan/XalanSourceTree/XalanSourceTreeDOMSupport.hpp> 30
09 #include <Xalan/XercesParserLiaison/XercesParserLiaison.hpp> 31 // Creamos un documento Xalan-
10 #include <Xalan/XercesParserLiaison/XercesDOMSupport.hpp> 32     Document basado en doc.
11 #include <Xalan/XalanTransformer/XercesDOMWrapperParsedSource.hpp> 33 DOMDocument* xercesDoc =
12 #include <Xalan/XercesParserLiaison/XercesDocumentWrapper.hpp> 34     all_customers->_node
13                                     35     (->getOwnerDocument());
14 using namespace std;         36 XercesDOMSupport theDOMSupport;
15 using namespace xercesc;     37 XercesDocumentWrapper
16 using namespace XALAN_CPP_NAMESPACE; 38     theWrapper( XalanMemMgrs::get-
17                                     39     DefaultXercesMemMgr(), xerces-
18 int main (int argc, char* argv[]) 40     Doc );
19 {                               41 XalanNode* xalanContextNode =
20     // Parseamos el XML         42     theWrapper.getDocumentEle-
21     XMLPlatformUtils::Initialize 43     ment();
22     XPathEvaluator::initial-    44     XalanDocumentPrefixResolver
23     ize();                     45     thePrefixResolver(
24     xml_schema::properties props; 46     theWrapper.getDocumentEle-
25     auto_ptr<customers_t>       47     ment()->getOwnerDocument() );
26     all_customers( customers(    48     XPathEvaluator theEvaluator;
27     argv[1],                   49
28     xml_schema::flags::keep_dom | 50     cerr << "Evaluando XPath" <<
29     xml_schema::flags::dont_initialize, props ) ); 51     endl;
30
31 // Creamos un documento Xalan- 52     // Evaluate XPath
32     Document basado en doc.     53     XalanNode* const resultX-
33     DOMDocument* xercesDoc =    54     alanNode =
34     all_customers->_node        55     theEvaluator.selectSingleN-
35     (->getOwnerDocument());     56     ode(
36     XercesDOMSupport theDOMSupport; 57     theDOMSupport,
37     XercesDocumentWrapper       58     xalanContextNode,
38     theWrapper( XalanMemMgrs::get- 59     XalanDOMString( "/cus-
39     DefaultXercesMemMgr(), xerces- 60     tomers/customer[1]").c_str(),
40     Doc );                       61     thePrefixResolver
41     XalanNode* xalanContextNode = 62     );
42     theWrapper.getDocumentEle-    63     );
43     ment();                       64     //
44     XalanDocumentPrefixResolver   65     // Volvemos a objetos CodeSyn-
45     thePrefixResolver(           66     thesis XSD C++
46     theWrapper.getDocumentEle-    67     //
47     ment()->getOwnerDocument() ); 68     if( DOMNode* xercesNode =
48     XPathEvaluator theEvaluator; 69     (DOMNode*)theWrapper.mapNode(
49                                     70     resultXalanNode ) ) {
50     cerr << "De la búsqueda por ID 71     cerr << "nodo resultado
51     ..." << endl;             72     xerces-c:" << xercesNode <<
52     cout << "nombre:" << monty- 73     endl;
53     ByID.first_name() << "apellido:" << montyByID.sir_name() << endl;
54     << endl;
55     }
56     }
57     catch (const
58     xml_schema::exception& e) {
59     cerr << e << endl;
60     }
61     return 1;

```

Listado 7: Objetos C++ de Vuelta a XML de Nuevo

```

01 #include <iostream>                                ::iterator ci=l.begin(); ci != 24
02 #include "customers.hxx"                            l.end(); ++ci ) { 25 // customers (std::cout,
03                                                     13 if (ci->first_name() == *all_customers );
04 using namespace std;                               "Bart" ) { 26
05                                                     14 int year=1980; xml_schema::namespace_infomap
06 int main (int argc, char* argv[])                 15 unsigned short month=2; map;
07 {                                                  16 unsigned short day=22; 27 map[""].schema =
08 try {                                              17 unsigned short hours=1; "customers.xsd";
09 auto_ptr<customers_t>                               18 unsigned short minutes=2; 28 customers (std::cout,
10 all_customers(                                     19 double seconds=3; *all_customers, map);
09 customers(argv[1]) );                             20 29 }
11                                                     21 ci->dob() = 30 catch (const
customers_t::customer_sequence                       xml_schema::date_time( year, xml_schema::exception& e) {
&l=all_customers->customer();                       month, day, hours, minutes, sec- 31 cerr << e << endl;
12 for(                                               22 } 32 return 1;
customers_t::customer_sequence                       23 } 33 }
                                                     34 }

```

mostrado en el Listado 6 usa Xalan-C++ [6] para evaluar la expresión XPath. Para usar Xalan-C++ para evaluar el XPath, hemos de envolver los objetos Xerces-C++ objects [7] en objetos Xalan-C++.

El código de base del bloque previo a `selectSingleNode()` se usa para conseguir los wrappers Xalan-C++. Una vez evaluado el XPath, se usa el objeto `XercesDocumentWrapper` para convertir el Xalan-Node de nuevo en un `DOMNode` de Xerces-C++. Una vez disponemos del `DOMNode` de nuevo, la conversión de vuelta a objetos CodeSynthesis XSD C++ es la misma que para el ejemplo del `DOMDocument::getElementById()`.

Objetos C++ de Vuelta a XML

Claro está que en algún punto queremos convertir de nuevo los objetos de C++ a XML. El único cambio necesario para soportarlo es pasar `-generate-serialization` como argumento al ejecutar `xsdccx`. El código C++ mostrado en el Listado 7 opera sobre los mismos archivos `custo-`

`mers.xsd` y `customers.xml` usados en los ejemplos anteriores.

Debido a que el archivo `customers.xml` enlaza ahora a su archivo `xsd`, su carga se convierte en una operación de una sola línea. Al encontrar a Bart, creamos un `date_time` y se lo asignamos a su fecha de nacimiento. La línea comentada volcará todos los clientes a la salida estándar en forma de archivo XML válido.

Esta versión, ligeramente más larga, incluirá en su salida un enlace al archivo `customers.xsd`. Debido a que el programa espera que el archivo XML de entrada contenga un enlace a su archivo de esquema, lo lógico es que el XML de salida contenga también un enlace al esquema que le corresponde. De este modo, el XML resultante es también una entrada válida para el programa.

Conclusión

En este artículo se resumen algunas de las ventajas del uso de CodeSynthesis XSD para integrar XML con C++, frente al uso de interfaces de programación como

DOM. Por supuesto, los detalles de cómo usemos CodeSynthesis XSD, DOM, XQilla u otras herramientas orientadas a XML, dependerán de nuestros propios métodos de desarrollo y de las peculiaridades de nuestros proyectos. ■

RECURSOS

- [1] CodeSynthesis XSD: <http://codesynthesis.com/projects/xsd/>
- [2] Document object model: http://en.wikipedia.org/wiki/Document_Object_Model
- [3] XPath: <http://www.w3.org/TR/xpath/>
- [4] Atributo ID de XML: <http://www.w3.org/TR/xmlschema-2/#ID>
- [5] DOMDocument::getElementById: <http://xerces.apache.org/xerces-c/apiDocs-2/classDOMDocument.html#fb3e89ba1247d689c4570f40003ea5db>
- [6] Xalan-C++: <http://xml.apache.org/xalan-c/>
- [7] Xerces-C++: <http://xerces.apache.org/xerces-c/>
- [8] XQilla: <http://xqilla.sourceforge.net/>
- [9] Funciones de extensión de XQilla: <http://xqilla.sourceforge.net/ExtensionFunctions>
- [10] XQuery: <http://www.w3.org/TR/xquery/>
- [11] Lenguajes de consulta FLOWR: For, Let, Order by, Where, Return: <http://en.wikipedia.org/wiki/FLWOR>
- [12] Listados y recursos para este artículo: <http://www.linux-magazine.es/Magazine/Downloads/62/XSD>

Listado 8: Ejecutando customers.xq

```

01 $ xqilla -i customers.xml cus- e>
tomers.xq | xmllint --format - 06
02 <?xml version="1.0"?> <middle-name>Montgomery</mid-
03 <customers> dle-name>
04 <customer 07 <sir-name>Burns</sir-name>
xmlns:xsi="http://www.w3.org/2 08 <gender>male</gender>
001/XMLSchema-instance" 09
id="c2"> <dob>1903-11-20T06:30:13</dob>
05 10 </customer>
<first-name>Charles</first-nam 11 </customers>

```