```
// XmlConverter.h - XML CONVERSION FUNCTIONS
//
// MODULE INDEX
// NAME                     CONTENTS
// XmlRawMap_c::~XmlRawMap_c    Destruct a conversion map
// XmlRawMap_c::ReadInt         Read an integer from the input stream
// XmlRawMap_c::WriteInt    Write an integer to the output stream
// XmlRawMap_c::ReadString  Read a string from the input stream
// XmlRawMap_c::WriteString Write a string to the output stream
// WriteXml             Write the xml equivalent of a structure to a
//                      stream
// ReadTag              Read an xml tag
// ReadXml              Read a structure from its xml equivalent
// main                     Demonstration main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 05-09-11 JS  Original
//
//-----------------------------------------------------------------------

#include <cstring>      // C-style string manipulation functions
#include <string>       // C++ string declarations
#include <vector>       // C++ vector declarations
#include <istream>      // C++ input stream declarations
#include <ostream>      // C++ output stream declarations
#include <iostream>     // C++ input/output stream declarations
#include <sstream>      // C++ string stream declarations
using namespace std;            // Expand the standard namespace

//-----------------------------------------------------------------------

// XML FAULT EXCEPTION

class XmlFault_c {
    string      description;
                // Fault description
public:
    XmlFault_c () {}
                // Default constructor
    XmlFault_c (const char *desc) { description = desc; };
                // Value constructor
    const char *GetDescription () { return description.c_str(); }
                // Get description
};

//-----------------------------------------------------------------------

// FIELD PROCESSING CLASS

class XmlField_c {
public:
    string fieldName;
                // Field name string
    virtual void ReadField (istream *input, void *apStruct) = 0;
                // Read the field value
    virtual void WriteField (ostream *output, const void *apStruct) = 0;
                // Write the field value
    virtual ~XmlField_c() {}
```

```
                // Virtual destructor
};

//---------------------------------------------------------------------

// RAW CONVERSION MAP

class XmlRawMap_c {
protected:
    static void ReadInt (istream *input, int *intField);
                // Read an integer value
    static void WriteInt (ostream *output, const int *intField);
                // Write an integer value
    static void ReadString (istream *input, string *stringField);
                // Read a string value
    static void WriteString (ostream *output, const string *stringField);
                // Write a string value
public:
    vector<XmlField_c*> xmlFieldVec;
                // XML field vector
    ~XmlRawMap_c ();
                // Destructor
};

//---------------------------------------------------------------------

// CONVERTION MAP TEMPLATE

template<class Struct_t> class XmlMap_c : public XmlRawMap_c {

    // Integer Field Structure

    struct XmlInt_c : public XmlField_c {
        int Struct_t::  *intField;
    public:
        void ReadField (istream *input, void *apStruct)
        {
            ReadInt (input, &(static_cast<Struct_t*>(apStruct)
                ->*intField));
        }
        void WriteField (ostream *output, const void *apStruct)
        {
            WriteInt (output,
                &(static_cast<const Struct_t*>(apStruct)
                ->*intField));
        }
        XmlInt_c (const char *name, int Struct_t:: *intParm)
        {
            fieldName = name;
            intField = intParm;
        }
    };

    // String Field Structure

    struct XmlString_c : public XmlField_c {
        string Struct_t::   *stringField;
    public:
        void ReadField (istream *input, void *apStruct)
```

```
        {
            ReadString (input, &(static_cast<Struct_t*>(apStruct)
                ->*stringField));
        }
        void WriteField (ostream *output, const void *apStruct)
        {
            WriteString (output,
                &(static_cast<const Struct_t*>(apStruct)
                ->*stringField));
        }
        XmlString_c (const char *name, string Struct_t:: *stringParm)
        {
            fieldName = name;
            stringField = stringParm;
        }
    };

public:

    // Add an Integer Field to the Conversion Map

    void AddInt (const char *name, int Struct_t:: *intParm)
    {
        xmlFieldVec.push_back (new XmlInt_c(name, intParm));
    }

    // Add a String Field to the Conversion Map

    void AddString (const char *name, string Struct_t:: *stringParm)
    {
        xmlFieldVec.push_back (new XmlString_c(name, stringParm));
    }
};

//---------------------------------------------------------------------------

// DESTRUCT A CONVERSION MAP

XmlRawMap_c::~XmlRawMap_c ()
{
    size_t      i;      // General purpose index

    for (i = 0; i < xmlFieldVec.size(); i++)
        delete xmlFieldVec[i];
}

//---------------------------------------------------------------------------

// READ AN INTEGER FROM THE INPUT STREAM

void
XmlRawMap_c::ReadInt (
    istream     *input,     // Input stream
    int     *intField)  // Pointer to integer field
{
    istream::int_type c;        // Look-ahead character
    int     val;        // Integer value

    c = input->get();
```

```
    while (isspace (c)) c = input->get();
    if ( ! isdigit(c))
        throw XmlFault_c ("Invalid integer");
    val = 0;
    while (isdigit(c)) {
        val = val * 10 + c - '0';
        c = input->get();
    }
    while (isspace (c)) c = input->get();
    if (c != '<')
        throw XmlFault_c ("Invalid integer");
    input->unget();
    *intField = val;
}

//---------------------------------------------------------------------

// WRITE AN INTEGER TO THE OUTPUT STREAM

void
XmlRawMap_c::WriteInt (
    ostream     *output,    // Output stream
    const int   *intField)  // Pointer to integer field
{
    *output << *intField;
}

//---------------------------------------------------------------------

// READ A STRING FROM THE INPUT STREAM
void
XmlRawMap_c::ReadString (
    istream     *input,     // Input stream
    string      *stringField)   // Pointer to string field
{
    istream::int_type c;        // Look-ahead character

    stringField->erase ();
    c = input->get();
    while (c != istream::traits_type::eof() && c != '<') {
        *stringField += static_cast<char>(c);
        c = input->get();
    }
    if (c != '<')
        throw XmlFault_c ("Invalid string");
    input->unget();
}

//---------------------------------------------------------------------

// WRITE A STRING TO THE OUTPUT STREAM

void
XmlRawMap_c::WriteString (
    ostream     *output,    // Output stream
    const string    *stringField)   // Pointer to string field
{
    *output << *stringField;
}
```

```
//---------------------------------------------------------------------------

// WRITE THE XML EQUIVALENT OF A STRUCTURE TO A STREAM

void
WriteXml (
    ostream     *output,     // Output stream
    const char  *structName,     // Structure name
    const void  *structVal, // Structure value
    const XmlRawMap_c *rawMap)  // Pointer to raw map
{
    size_t      i;       // General purpose index

    *output << '<' << structName << ">\n";
    for (i = 0; i < rawMap->xmlFieldVec.size(); i++) {
        *output << "\t<" << rawMap->xmlFieldVec[i]->fieldName << '>';
        rawMap->xmlFieldVec[i]->WriteField (output, structVal);
        *output << "</" << rawMap->xmlFieldVec[i]->fieldName << ">\n";
    }
    *output << "</" << structName << ">\n";
}

//---------------------------------------------------------------------------

// READ AN XML TAG

void
ReadTag (
    istream     *input,      // Input stream
    string      *tagName,    // Received tag name
    bool        *closeFlag) // Received close tag flag
{
    istream::int_type c;         // Look-ahead character

    tagName->erase ();
    *closeFlag = 0;
    c = input->get();
    while (isspace(c)) c = input->get();
    if (c != '<') throw XmlFault_c ("No '<'");
    c = input->get();
    while (isspace(c)) c = input->get();
    if (c == '/') {
        *closeFlag = 1;
        c = input->get();
        while (isspace(c)) c = input->get();
    }
    while (c != istream::traits_type::eof() && c != '>') {
        *tagName += static_cast<char>(c);
        c = input->get();
    }
    if (c != '>')
        throw XmlFault_c ("Invalid tag");
    while (
        tagName->length() != 0 &&
        isspace(tagName->at(tagName->length()-1))
    ) tagName->resize (tagName->length() - 1);
}
```

```
//---------------------------------------------------------------------------

// READ A STRUCTURE FROM ITS XML EQUIVALENT
// Throws XmlFault_c on input data fault

void
ReadXml (
    istream     *input,      // Input stream
    const char  *structName,     // Structure name
    void        *structVal, // Structure value
    const XmlRawMap_c *rawMap)   // Pointer to raw map
{
    istream::int_type c;         // Look-ahead character
    string      tagName;     // Tag name
    string      closeName;   // Closing tag name
    bool        closeFlag;   // Closing tag flag
    vector<bool>    foundArr;    // Field found flags
    size_t      i;       // General purpose index

    // Reset the field found flags

    foundArr.resize(rawMap->xmlFieldVec.size());
    for (i = 0; i < rawMap->xmlFieldVec.size(); i++) foundArr[i] = 0;

    // Read and validate the initial opening tag

    ReadTag (input, &tagName, &closeFlag);
    if (tagName != structName || closeFlag)
        throw XmlFault_c ("Bad opening structure tag");

    // Read field tags

    for (;;) {
        ReadTag (input, &tagName, &closeFlag);
        if (tagName == structName && closeFlag) break;
        if (closeFlag)
            throw XmlFault_c ("Unmatched close tag");
        i = 0;
        while (
            i < rawMap->xmlFieldVec.size() &&
            rawMap->xmlFieldVec[i]->fieldName != tagName
        ) i ++ ;
        if (i >= rawMap->xmlFieldVec.size())
            throw XmlFault_c ("Unrecognised field tag");
        rawMap->xmlFieldVec[i]->ReadField (input, structVal);
        ReadTag (input, &closeName, &closeFlag);
        if (closeName != tagName || ! closeFlag)
            throw XmlFault_c ("Bad close tag");
    }
}

//---------------------------------------------------------------------------

// SAMPLE STRUCTURE DEFINITION

struct Item_t {
    int     itemNo;
    string      itemName;
    int     quantity;
```

```
};

//---------------------------------------------------------------------

// DEMONSTRATION MAIN LINE

int
main ()
{
    Item_t      item;       // Item to be read and written
    istringstream   iss;        // Input string stream

    XmlMap_c<Item_t> itemMap;
    itemMap.AddInt ("itemNo", &Item_t::itemNo);
    itemMap.AddString ("itemName", &Item_t::itemName);
    itemMap.AddInt ("quantity", &Item_t::quantity);

    iss.str (
        "<item>\n"
            "\t<itemNo>1234</itemNo>\n"
            "\t<itemName>Widget</itemName>\n"
            "\t<quantity>45</quantity>\n"
        "</item>\n"
    );
    try {
        ReadXml (&iss, "item", &item, &itemMap);
    } catch (XmlFault_c fault) {
        cerr << "Error: " << fault.GetDescription() << '\n';
        exit (1);
    }

    WriteXml (&cout, "item", &item, &itemMap);

    return 0;
}
```