Autodesk®
Scaleform®

ActionScript 2 XML User Guide

This document describes configuring ActionScript 2 XML support available in Scaleform 4.3.

Authors: Prasad Silva

Version: 3.0
Last Edited: July 6, 2012

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.3

How to Contact Autodesk Scaleform:

Document	XML User Guide
Address	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	www.scaleform.com
Email	info@scaleform.com
Direct	(301) 446-3200
Fax	(301) 446-3199

Table of Contents

Introduction

The Autodesk Scaleform SDK is a light-weight, high-performance rich media Flash® vector graphics engine, built on a clean-room implementation specifically for Console and PC game developers. Scaleform combines the scalability and development ease of proven visual authoring tools, such as the Adobe® Flash® Studio, with the latest hardware graphics acceleration that cutting-edge game developers demand.

ActionScript 2 XML support is provided by Flash through its AS2 XML API. The XML and XMLNode classes wrap the loading, parsing, and DOM tree management functionalities. This core XML support is faithfully reproduced in Scaleform 4.3. The Scaleform implementation provides built-in DOM tree management and also exposes a SAX2-based interface to plug in custom XML parsers. By default, the GFxPlayer application uses a parser implementation that wraps the open source Expat parser. XML is supported on all platforms that support Scaleform: Windows®, Linux®, MacOS®, Xbox 360®, PSP® (PlayStation®Portable), PlayStation®2 (PS2), PlayStation®3 (PS3™), Nintendo® Wii™.

This document describes the Scaleform XML architecture and how to use it in an application. A how-to on configuring a custom parser and known limitations are described at the end.

1. Using XML

The following are example code snippets and simple instructions to help users get started with the Scaleform XML implementation.

1.1 Enabling ActionScript XML Support

To enable ActionScript XML processing support, the XML parser state needs to be set for the Scaleform loader during Scaleform initialization. The following is an excerpt from the default FxPlayer.cpp file. The same pattern can be used with a custom parser implementation:

```
…
#include "XML/XML_Expat.h"
using namespace Scaleform;
…
Ptr<XML::Parser> pexpatXmlParser = *new XML::ParserExpat;
Ptr<XML::SupportBase> pxmlSupport = *new XML::Support(pexpatXmlParser);
mLoader.SetXMLSupport(pxmlSupport);…
```

1.2 Document Parsing from C++

The parser state can be used to parse XML documents directly from C++. If ActionScript XML support is enabled, the following can be used to create a DOM tree from an XML file (This requires access to full Scaleform source):

```
…
#include "GFx/XML/XML_DOM.h"
using namespace Scaleform;
…
// Create a DOM builder that processes whitespace
XML::DOMBuilder domBuilder(Loader.GetXMLSupport());
// OR create a DOM builder that ignores whitespace
XML::DOMBuilder domBuilder2(Loader.GetXMLSupport(), true);
…
// Process the xml file and return the root of the DOM tree (creates an
// object manager internally)
Ptr<XML::Document> pdoc = domBuilder.ParseFile("inputfile.xml",
                                               mLoader.GetFileOpener());
// OR process the xml file and return the root of the DOM tree (use provided object
manager)
```

```
Ptr<XML::ObjectManager> pobjMgr = *new XML::ObjectManager();
Ptr<XML::Document> pdoc2 = domBuilder.ParseFile("inputfile.xml",
                                                mLoader.GetFileOpener(), pobjMgr);
…
```

If ActionScript XML support is not enabled, a stand-alone instance of the parser can be created and used as follows:

```
…
#include "GFx/XML/XML_DOM.h"
#include "GFx/XML/XML_Expat.h"
using namespace Scaleform;
…
Ptr<XML::Parser> pexpatXmlParser = *new XML::ParserExpat;
Ptr<XML::Support> pxmlSupport = *new XML::Support(pexpatXmlParser);
…
// Create a DOM builder that processes whitespace
XML::DOMBuilder domBuilder(pxmlSupport);
// Create a DOM builder that ignores whitespace
XML::DOMBuilder domBuilder2(pxmlSupport, true);
…
// Same as processing from previous section
…
```

1.3 Traversing the DOM Tree in C++

The following code prints out the contents of a DOM tree (This requires access to full Scaleform source):

```
…
using namespace Scaleform;
…
void PrintDOMTree(XML::Node* proot)
{
    switch (proot->Type)
    {
        case XML::ElementNodeType:
        {
            XML::ElementNode* pnode =
                static_cast< XML::ElementNode*>(proot);
            if (pnode->Prefix.GetSize() > 0)
            {
                printf("ELEM - '%s:%s' ns:'%s' prefix:'%s'"
                       " localname: '%s'",
```

```
                    pnode->Prefix.ToCStr(),
                    pnode->Value.ToCStr(),
                    pnode->Namespace.ToCStr(),
                    pnode->Prefix.ToCStr(),
                    pnode->Value.ToCStr());
            }
            else
            {
                printf("ELEM - '%s' ns:'%s' prefix:''"
                    " localname: '%s'",
                    pnode->Value.ToCStr(),
                    pnode->Namespace.ToCStr(),
                    pnode->Value.ToCStr());
            }
            for (XML::Attribute* attr = pnode->FirstAttribute;
                  attr != NULL; attr = attr->Next)
            {
                printf(" {%s,%s}", attr->Name.ToCStr(),
                        attr->Value.ToCStr());
            }
            printf("\n");
            for (XML::Node* child = pnode->FirstChild; child != NULL;
                  child = child->NextSibling)
                PrintDOMTree(child);
            break;
        }
        case TextNodeType:
        {
            printf("TEXT - '%s'\n", proot->Value.ToCStr());
            break;
        }
        default:
        {
            printf("UNKN\n");
        }
    }
}
…
Ptr<XML::Document> pdoc = domBuilder.ParseFile("inputfile.xml",
                                                Loader.GetFileOpener());
PrintDOMTree(root);
```

1.4 Configuring Debug Reporting

Users who have access to the Scaleform source files are able to set/unset flags in XML_DOM.cpp to specify the XML parsing, and DOM construction reporting flags.

```
//
// Debug Output & Tracing Flags
//
#ifdef SF_BUILD_DEBUG
//
// Trace the document builder's DOM tree construction in debug output
//
// #define SF_XML_DOCBUILDER_TRACE
//
// Dump all constructed DOM trees to standard out
// (Warning: Avoid this for large files)
//
// #define SF_XML_DOCBUILDER_DOM_TREE_DUMP
//
#endif  // #ifdef SF_BUILD_DEBUG
```

2. Implementing a Custom Parser

Scaleform provides a default parser implementation using the expat library. If the target application already provides an XML parser, it can be plugged into the Scaleform XML subsystem by wrapping the parser in the GFx::XML::Parser interface, which is defined in XML_Support.h. The XML_Parser.h file defines the GFx::XML::ParserHandler class as a SAX2 interface mechanism.

```
namespace Scaleform { namespace GFx { namespace XML {
//
// The pluggable XML parser
//
// Creates instances of parsers for each parse file/string call.
// This was added for thread safety. Each parser instance is guaranteed
// to be used within a single thread. An instance of a parser is expected
// by the XML state.
//
class Parser : public RefCountBaseNTS<Parser, Stat_Default_Mem>
{
  public:
    virtual ~Parser() {}

    // Parse methods

    virtual bool    ParseFile(const char* pfilename, FileOpenerBase* pfo,
                              ParserHandler* pphandler) = 0;
    virtual bool    ParseString(const char* pdata, UPInt len,
                                ParserHandler* pphandler) = 0;
};
}}} //SF::GFx::XML
```

All XML parser implementations must register a GFx::XML::ParserLocator instance with the DOM builder object passed to the ParseFile and ParseString methods, before any parsing occurs. For all appropriate parsing events, the complementary GFx::XML::ParserHandler callback method should be invoked with the required parameters. If an error occurs, the appropriate error handler method should be invoked based on its severity.

```
namespace Scaleform { namespace GFx { namespace XML {
//
// SAX2 Consolidated Handler
//
// The DOM builder is an interface similar to a SAX2 parser handler.
// The parser implementation is expected to call the appropriate callback
// method for certain events.
//
```

```
class ParserHandler : public RefCountBase<ParserHandler, StatMV_XML_Mem>
{
  public:
    ParserHandler() {}
    virtual ~ParserHandler() {}

    // Beginning and end of documents
    virtual void        StartDocument() = 0;
    virtual void        EndDocument() = 0;

    // Start and end of a tag element
    virtual void        StartElement(const StringRef& prefix,
                                     const StringRef& localname,
                                     const ParserAttributes& atts) = 0;
    virtual void        EndElement(const StringRef& prefix,
                                   const StringRef& localname) = 0;

    // Namespace declaration. Next element will be the parent of the mappings
    virtual void        PrefixMapping(const StringRef& prefix,
                                      const StringRef& uri) = 0;
    // Text data, in between tag elements
    virtual void        Characters(const StringRef& text) = 0;


    // Whitespace
    virtual void        IgnorableWhitespace(const StringRef& ws) = 0;


    // Unprocessed elements
    virtual void        SkippedEntity(const StringRef& name) = 0;


    // GFx::XML::Parser implementors are REQURIED to set a document locator
    // before any callbacks occur. GFx::XML::ParserHandler implementations
    // require a locator object for error reporting and correctly processing
    // the encoding, xmlversion and standalone properties.
    virtual void    SetDocumentLocator(const ParserLocator* plocator) = 0;

    // Comments
    virtual void        Comment(const StringRef& text) = 0;

    // ErrorHandler Callbacks
    virtual void        Error(const ParserException& exception) = 0;
    virtual void        FatalError(const ParserException& exception) = 0;
    virtual void        Warning(const ParserException& exception) = 0;
};

}}} //SF::GFx::XML
```

3. Known Limitations

3.1 No Custom XML node Properties in ActionScript

Custom properties assigned to an ActionScript XMLNode (and XML) object will not persist if all references to that object is dropped and accessed again. When all references are dropped, only the ActionScript object is removed. The DOM tree behind the scenes will persist. But since custom properties are applied to the ActionScript object and not the DOM, the properties' lifetime is directly related to the lifetime of the ActionScript object.

This case can occur when a document is loaded through XML.load, which creates a top level XMLNode object that shadows a complete DOM tree behind the scenes. If the user traverses the DOM tree in ActionScript using temporary XMLNode references and assigns custom properties to those temporary references, they will not exist when accessed later on. For example (assuming root is an XML or XMLNode object that has valid XML data):

```
// Get reference to a DOM node
XMLNode temp = root.firstChild;
// Set a custom property to the node
temp.someProperty = someValue;
// Drop the reference (all references)
XMLNode temp = temp.nextSibling;
// Get back the reference
temp = temp.prevSibling;
// Look for the custom property
trace(temp.someProperty)
```

The output will be someValue in Flash™, but will be undefined in Scaleform because the XMLNode object does not hold the previously assigned property anymore. The property will exist if a reference to it persisted through custom property assignment and the trace statement.

Note: This does not affect the attributes object that is attached to an XMLNode. Setting properties to this object will persist regardless of the state of ActionScript XMLNode references. For example:

```
// Get reference to a DOM node
XMLNode temp = root.firstChild;
// Set a custom attribute to the node
temp.attributes.someProperty = someValue;
// Drop the reference (all references)
XMLNode temp = temp.nextSibling;
// Get back the reference
```

```
temp = temp.prevSibling;
// Look for the custom attribute
trace(temp.attributes.someProperty)
```

The output will be some Value in Scaleform.

3.2 No ActionScript XML.status Property

This property was not implemented because of the various inconsistencies that arise when mapping error codes from a custom parser library to the ActionScript 2.0 XML error codes, e.g.,

ActionScript error codes:

- 0 No error; parse was completed successfully.
- -2 A CDATA section was not properly terminated.
- -3 The `XML` declaration was not properly terminated.
- -4 The `DOCTYPE` declaration was not properly terminated.
- -5 A comment was not properly terminated.
- -6 An XML element was malformed.
- -7 Out of memory.
- -8 An attribute value was not properly terminated.
- -9 A start-tag was not matched with an end-tag.
- -10 An end-tag was encountered without a matching start-tag.

Expat to ActionScript mapping:

```
//
// XML_ERROR_NONE           = 0
// XML_ERROR_INVALID_TOKEN  = 0 (ie: XML.parse("http://www.google.com"))
// XML_ERROR_UNCLOSED_CDATA_SECTION = -2
// XML_ERROR_UNCLOSED_TOKEN = -3
// XML_ERROR_INVALID_TOKEN  = -4
// XML_ERROR_INVALID_TOKEN  = -5
// XML_ERROR_UNCLOSED_TOKEN = -8
// XML_ERROR_NO_ELEMENTS    = -9
// XML_ERROR_TAG_MISMATCH   = -10
//
```

Implementing a consistent status property is not possible without a one-to-one mapping of the error codes. The *GFx::XML::DOMBuilder* prints verbose error messages to debug output using data from the GFx::XML::ParserLocator object registered with it by the XML parser instance.

3.3 No ActionScript XML.docTypeDecl Property

From Flash® documentation: "The ActionScript XML parser is not a validating parser. The `DOCTYPE` declaration is read by the parser and stored in the `XML.docTypeDecl` property, but no DTD validation is performed." This property has no use in both Flash and Scaleform, therefore it was omitted.