

XML User Guide

This document describes configuring XML support available in Scaleform GfX.

Authors: Prasad Silva
Version: 2.08
Last Edited: July 2, 2009

Copyright Notice

Autodesk® Scaleform® 3

© 2011 Autodesk, Inc. All rights reserved. Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

The following are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries: 3DEC (design/logo), 3December, 3December.com, 3ds Max, Algor, Alias, Alias (swirl design/logo), AliasStudio, Alias|Wavefront (design/logo), ATC, AUGI, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk Intent, Autodesk Inventor, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSnap, AutoSketch, AutoTrack, Backburner, Backdraft, Beast, Built with ObjectARX (logo), Burn, Buzzsaw, CAiCE, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, DesignServer, DesignStudio, Design Web Format, Discreet, DWF, DWG, DWG (logo), DWG Extreme, DWG TrueConvert, DWG TrueView, DXF, Ecotect, Exposure, Extending the Design Team, Face Robot, FBX, Fempro, Fire, Flame, Flare, Flint, FMDesktop, Freewheel, GDX Driver, Green Building Studio, Heads-up Design, Heidi, HumanIK, IDEA Server, i-drop, Illuminate Labs AB (design/logo), ImageModeler, iMOUT, Incinerator, Inferno, Inventor, Inventor LT, Kynapse, Kynogon, LandXplorer, LiquidLight, LiquidLight (design/logo), Lustre, MatchMover, Maya, Mechanical Desktop, Moldflow, Moldflow Plastics Advisers, MPI, Moldflow Plastics Insight, Moldflow Plastics Xpert, Moondust, MotionBuilder, Movimento, MPA, MPA (design/logo), MPX, MPX (design/logo), Mudbox, Multi-Master Editing, Navisworks, ObjectARX, ObjectDBX, Opticore, Pipeplus, PolarSnap, PortfolioWall, Powered with Autodesk Technology, Productstream, ProMaterials, RasterDWG, RealDWG, Real-time Roto, Recognize, Render Queue, Retimer, Reveal, Revit, RiverCAD, Robot, Scaleform, Scaleform AMP, Scaleform CLIK, Scaleform GFx, Scaleform IME, Scaleform Video, Showcase, Show Me, ShowMotion, SketchBook, Smoke, Softimage, Softimage|XSI (design/logo), Sparks, SteeringWheels, Stitcher, Stone, StormNET, StudioTools, ToolClip, Topobase, Toxik, TrustedDWG, U-Vis, ViewCube, Visual, Visual LISP, Volo, Vtour, WaterNetworks, Wire, Wiretap, WiretapCentral, XSI.

All other brand names, product names or trademarks belong to their respective holders.

Disclaimer

THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. "AS IS". AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

How to Contact Autodesk Scaleform:

Document	XML User Guide
Address	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.....	1
1. Using XML.....	2
1.1 Enabling ActionScript XML Support.....	2
1.2 Document Parsing from C++.....	2
1.3 Traversing the DOM Tree in C++.....	3
1.4 Configuring Allocation Sizes and Debug Reporting.....	5
2. Implementing a Custom Parser.....	6
3. Known Limitations.....	8
3.1 No Custom XML node Properties in ActionScript.....	8
3.2 No ActionScript XML.status Property.....	9
3.3 No ActionScript XML.docTypeDecl Property.....	10

Introduction

The Scaleform GfX 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 GfX 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.

XML support is provided by Flash through its ActionScript 2.0 API. The XML and XMLNode classes wrap the loading, parsing, and DOM tree management functionalities. This core XML support is faithfully reproduced in GfX 3.x/2.2. The GfX implementation provides built-in DOM tree management and also exposes a SAX2-based interface to plug in custom XML parsers. By default, the FxPlayer application uses a parser implementation that wraps the open source Expat parser. XML is supported on all platforms that support GfX: Windows®, Linux®, MacOS®, Xbox 360®, PSP® (PlayStation®Portable), PlayStation®2 (PS2), PlayStation®3 (PS3™), Nintendo® Wii™.

This document describes the GfX 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 GfX XML implementation.

1.1 Enabling ActionScript XML Support

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

```
...
#include "GFxXMLExpat.h"
...
GPtr<GFxXMLParser> pexpatXmlParser = *new GFxXMLParserExpat;
GPtr<GFxXMLSupport> pxmlSupport = *new GFxXMLSupport(pexpatXmlParser);
Loader.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 GfX source):

```
...
#include "GFxXML.h"
...
// Create a DOM builder that processes whitespace
GFxXMLDOMBuilder domBuilder(Loader.GetXMLSupport());
// OR create a DOM builder that ignores whitespace
GFxXMLDOMBuilder domBuilder2(Loader.GetXMLSupport(), true);
...
// Process the xml file and return the root of the DOM tree (creates an
// object manager internally)
GPtr<GFxXMLDocument> pdoc = domBuilder.ParseFile("inputfile.xml",
Loader.GetFileOpener());
// OR process the xml file and return the root of the DOM tree (use provided object
manager)
GPtr<GFxXMLObjectManager> pobjMgr = *new GFxXMLObjectManager();
```

```

GPtr<GFxXMLDocument> pdoc2 = domBuilder.ParseFile("inputfile.xml",
Loader.GetFileOpener(), pObjMgr);
...

```

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

```

...
#include "GFxXML.h"
#include "GFxXMLExpat.h"
...
GPtr<GFxXMLParser> pexpatXmlParser = *new GFxXMLParserExpat;
GPtr<GFxXMLSupport> pxmlSupport = *new GFxXMLSupport(pexpatXmlParser);
...
// Create a DOM builder that processes whitespace
GFxXMLDOMBuilder domBuilder(pxmlSupport);
// Create a DOM builder that ignores whitespace
GFxXMLDOMBuilder 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 GFx source):

```

...
void PrintDOMTree(GFxXMLNode* proot)
{
    switch (proot->Type)
    {
        case GFxElementNodeType:
        {
            GFxXMLElementNode* pnode =
                static_cast<GFxXMLElementNode*>(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 (GFxXMLAttribute* attr = pnode->FirstAttribute;
         attr != NULL; attr = attr->Next)
    {
        printf(" {%s,%s}", attr->Name.ToCStr(),
               attr->Value.ToCStr());
    }
    printf("\n");
    for (GFxXMLNode* child = pnode->FirstChild; child != NULL;
         child = child->NextSibling)
        PrintDOMTree(child);
    break;
}
case GFxTextNodeType:
{
    printf("TEXT - '%s'\n", proot->Value.ToCStr());
    break;
}
default:
{
    printf("UNKN\n");
}
}
}
...
GPtr<GFxXMLDocument> pdoc = domBuilder.ParseFile("inputfile.xml",
Loader.GetFileOpener());
PrintDOMTree(root)

```


1.4 Configuring Allocation Sizes and Debug Reporting

Users who have access to the GfX source files are able to set/unset flags in GfXMLConfig.h to specify the default allocation sizes of the XML object memory pools as well as the allocation, XML parsing, and DOM construction reporting flags.

```
//
// Memory Management
//
// These defines configure the default allocation sizes of each
// memory pool page. Configuring these values for specific applications
// could potentially save runtime memory.
//
#define GFC_XML_DOCUMENTS_PER_PAGE      4
#define GFC_XML_ELEMENTNODES_PER_PAGE  32
#define GFC_XML_TEXTNODES_PER_PAGE     32
#define GFC_XML_ATTRIBUTES_PER_PAGE    32
#define GFC_XML_PREFIXES_PER_PAGE      8
#define GFC_XML_SHADOWREFS_PER_PAGE    32
#define GFC_XML_ROOTNODES_PER_PAGE     32

//
// Debug Output & Tracing Flags
//
#ifdef GFC_BUILD_DEBUG
//
// Trace the document builder's DOM tree construction in debug output
//
// #define GFC_XML_DOCBUILDER_TRACE
//
// Dump all constructed DOM trees to standard out
// (Warning: Avoid this for large files)
//
// #define GFC_XML_DOCBUILDER_DOM_TREE_DUMP
//
// Dump total memory allocation for XML objects to debug output
// for all XML object managers.
//
// #define GFC_XML_MEM_ALLOC_TOTALS_DUMP
//
#endif // #ifdef GFC_BUILD_DEBUG
```

2. Implementing a Custom Parser

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

```
//  
// 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 GFxXMLParser : public GRefCountBase<GFxXMLParser>  
{  
public:  
    virtual ~GFxXMLParser() {}  
  
    // Parse methods  
  
    virtual bool    ParseFile(const char* pfilename, GFxFileOpenerBase* pfo,  
                             GFxXMLParserHandler* pphandler) = 0;  
    virtual bool    ParseString(const char* pdata, UPInt len,  
                               GFxXMLParserHandler* pphandler) = 0;  
};
```

All XML parser implementations must register a GFxXMLParserLocator instance with the DOM builder object passed to the ParseFile and ParseString methods, before any parsing occurs. For all appropriate parsing events, the complementary GFxXMLParserHandler 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.

```
// 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 GFxXMLParserHandler : public GRefCountBase<GFxXMLParserHandler>  
{  
public:  
    GFxXMLParserHandler() { SetRefCountMode(GFC_REFCOUNT_THREADSAFE); }
```

```

virtual ~GFXMLParserHandler() {}

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

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

// Namespace declaration. Next element will be the parent of the mappings
virtual void      PrefixMapping(const GFXMLStringRef& prefix,
                                const GFXMLStringRef& uri) = 0;

// Text data, in between tag elements
virtual void      Characters(const GFXMLStringRef& text) = 0;

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

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

// GFXMLParser implementors are REQUIRED to set a document locator
// before any callbacks occur. GFXMLParserHandler implementations
// require a locator object for error reporting and correctly processing
// the encoding, xmlversion and standalone properties.
virtual void      SetDocumentLocator(const GFXMLParserLocator* plocator) = 0;

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

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

};

```

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 GfX 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 someValue in GFx.

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 *GFxXMLDOMBuilder* prints verbose error messages to debug output using data from the *GFxXMLParserLocator* 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 GfX, therefore it was omitted.