Autodesk®
Scaleform®

ActionScript 2 XML 사용자

이 문서는 Scaleform 4.2 에서 가용한 ActionScript 2 XML 지원 설정을 다룬다.

저자: Prasad Silva
버전: 3.0
최종 편집: 2010 년 7 월 28 일

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

연락처:

문서	XML 사용자 가이드
주소	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
홈페이지	www.scaleform.com
이메일	info@scaleform.com
직통전화	(301) 446-3200
팩스	(301) 446-3199

목차

서문 **1**

1. XML 사용 **2**

1.1 ActionScript XML 지원 가능 2

1.2 C++로부터의 문서 파싱 2

1.3 C++에서 DOM 트리의 전환 3

1.4 디버그 보고 설정 5

2. 커스텀 파서의 구현 **6**

3. 알려진 제한사항 **8**

3.1 ActionScript 내 커스텀 XML 노드 속성이 없음 8

3.2 ActionScript XML.status 속성이 없음 9

3.3 ActionScript XML.doc TypeDecl 속성이 없음 10

서문

Autodesk® Scaleform® SDK 은 경량의 높은 성능을 갖는 미디어가 풍부한 플래시 벡터 그래픽 엔진으로 특히 콘솔 및 PC 게임 개발자들을 위한 클린룸 구현을 바탕으로 수립되었다. Scaleform 는 어도비 플래시 플레이어 등과 같이 이미 검증된 비주얼 제작 툴의 쉬운 확장성 및 개발을 첨단 게임 개발자들이 요구하는 최신의 하드웨어 그래픽과 결합한다.

XML 지원은 AS2 XML API 를 통해 플래시를 통해 제공된다. XML 및 XMLNode 클래스는 로딩, 파싱 및 DOM 트리 관리 기능을 묶는다. 이러한 코어 XML 지원은 Scaleform 4.1 에서 충실히 재현되었다. Scaleform 구현은 빌트인 DOM 트리 관리를 제공하며 또한 SAX2 기반의 인터페이스를 제공하여 커스텀 XML 파서로의 플러그 인이 가능하다. 기본적으로 Scaleform Player 어플리케이션은 오픈 소스 Expat 파서를 포함하는 파서 구현을 사용한다. XML 은 Windows®, Linux®, MacOS®, Xbox 360®, PSP® (PlayStation®Portable), PlayStation®2 (PS2), PlayStation®3 (PS3™), Nintendo® Wii™와 같이 Scaleform 를 지원하는 모든 플랫폼에서 지원된다.

이 문서는 Scaleform XML 아키텍처와 어플리케이션에서의 사용에 대해 다룬다. 커스텀 파서의 설정과 알려진 제한사항에 대해선 마지막에 다룬다.

1. XML 사용

다음은 Scaleform XML 구현을 시작하려는 사용자를 돕기 위한 단순한 지침 및 예제 코드이다.

1.1 ActionScript XML 지원 가능

ActionScript XML 처리 지원을 가능하게 하기 위해 XML 파서 상태는 Scaleform 초기화 동안 Scaleform 로더에 대해 설정될 필요가 있다. 다음은 기본 FxPlayer.cpp 파일의 일부이다. 동일한 패턴은 커스텀 파서 구현에 사용할 수 있다.

```
…
#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 C++로부터의 문서 파싱

파서 상태는 C++에서 직접 XML 문서를 파싱하기 위해 사용 가능하다. 만약 ActionScript XML 지원이 가능하면 다음을 사용하여 XML 파일에서 DOM 트리를 생성할 수 있다 (이를 위해 Scaleform 소스에 완전히 접근할 필요가 있다.).

```
…
#include "GFx/XML/XML_DOM.h"
using namespace Scaleform;
…
// 빈 공간을 처리하는 DOM 빌더 생성
XML::DOMBuilder domBuilder(Loader.GetXMLSupport());
// 또는 빈 공간을 무시하는 DOM 빌더 생성
XML::DOMBuilder domBuilder2(Loader.GetXMLSupport(), true);
…
// xml 파일을 처리하고 DOM 트리의 루트를 반환 (객체 관리자를 내부적으로 생성)
Ptr<XML::Document> pdoc = domBuilder.ParseFile("inputfile.xml",
                                               mLoader.GetFileOpener());
// 또는 xml 파일을 처리하고 DOM 트리의 루트를 반환 (제공된 객체 관리자 사용)
```

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

ActionScript XML 지원이 불가능한 경우 파서의 스탠드 얼론 인스턴스를 생성해서 다음과 같이 사용할 수 있다.

```
…
#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);
…
// 빈 공간을 처리하는 DOM 빌더 생성
XML::DOMBuilder domBuilder(pxmlSupport);
// 빈 공간을 무시하는 DOM 빌더 생성
XML::DOMBuilder domBuilder2(pxmlSupport, true);
…
// 이전과 동일한 처리
…
```

1.3 C++에서 DOM 트리의 전환

다음의 코드는 DOM 트리의 컨텐츠를 출력한다 (이를 위해 Scaleform 소스에 완전히 접근할 필요가 있다).

```
…
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 디버그 보고 설정

Scaleform 소스 파일에 접근 권한이 있는 사용자는 XML 파싱 및 DOM 구성 보고 플래그를 지정하기 위해 XML_DOM.cpp 내의 플래그를 설정/해제할 수 있다

```
// 디버그 출력 및 플래그 추적

#ifdef SF_BUILD_DEBUG
//
// 디버그 출력에 있는 문서 빌더의 DOM 트리 구성을 추적
//#define SF_XML_DOCBUILDER_TRACE
//
// 구성된 모든 DOM 트리를 표준 출력으로 덤프한다
// (경고: 큰 파일에 대해선 사용하지 않는다)
//#define SF_XML_DOCBUILDER_DOM_TREE_DUMP
//
#endif  // #ifdef SF_BUILD_DEBUG
```

2. 커스텀 파서의 구현

Scaleform 는 expat 라이브러리를 사용해서 기본 파서 구현을 제공한다. 만약 타깃 어플리케이션이 이미 XML 파서를 제공한다면 파서를 GFx::XML::Parser 인터페이스 내에서 파서를 묶음으로써 Scaleform XML 서브 시스템으로 결합이 가능하며 이는 XML_Support.h 에 나와 있다. XML_Parser.h 파일은 GFx::XML::ParserHandler 클래스를 SAX2 인터페이스 메커니즘으로 정의한다.

```
namespace Scaleform { namespace GFx { namespace XML {
//
// 플러그가 가능한 XML 파서
//
// 각각의 파서 파일/문자열 호출에 대한 파서 인스턴스 생성
// 스레드 안전을 위해 추가된다. 각 파서 인스턴스는
// 단일 스레드 안에서 사용되어야 한다. 파서의 인스턴스는
// XML의 상태에 따를 것으로 예상된다.
//
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
```

모든 XML 파서 구현은 ParseFile 과 ParseString 메소드로 전달된 DOM 빌더 객체에 GFx::XML::ParserLocator 인스턴스를 파싱 발생 전에 등록해야 한다. 모든 적절한 파싱 이벤트에 대해 보충의 역할을 하는 GFx::XML::ParserHandler 콜 백 메소드를 필요한 매개변수와 호출해야 한다. 에러 발생 시 적절한 에러 핸들러 메소드를 그 정도에 따라 호출해야 한다.

```
namespace Scaleform { namespace GFx { namespace XML {
 //
// SAX2 결합 핸들러
//
// DOM 빌더는 SAX2 파서 핸들러와 유사한 인터페이스이다.
// 파서를 구현함으로써 특정 이벤트에 대한 적절한
// 콜 백을 호출할 것으로 예상된다.
```

```
//

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

    // 문서의 시작과 끝
    virtual void        StartDocument() = 0;
    virtual void        EndDocument() = 0;

    // 태그 요소의 시작과 끝
    virtual void        StartElement(const StringRef& prefix,
                                     const StringRef& localname,
                                     const ParserAttributes& atts) = 0;
    virtual void        EndElement(const StringRef& prefix,
                                   const StringRef& localname) = 0;

    // 이름 공간 선언. 다음 요소는 매핑의 부모가 된다.
    virtual void        PrefixMapping(const StringRef& prefix,
                                      const StringRef& uri) = 0;
    // 태그 요소 사이의 텍스트 데이터
    virtual void        Characters(const StringRef& text) = 0;

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

    // 처리되지 않은 요소
    virtual void        SkippedEntity(const StringRef& name) = 0;

    // GFx::XML::Parser 구현자는 문서 위치자를 콜 백 발생 전에
    // 설정해야 한다. GFx::XML::ParserHandler 구현에는
     // 에러 보고 및 올바른 인코딩 처리를 위한 위치가 객체,
    // xmlversion 및 스탠드 얼론 속성이 필요하다.

    virtual void    SetDocumentLocator(const ParserLocator* plocator) = 0;

    // 설명
    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. 알려진 제한사항

3.1 ActionScript 내 커스텀 XML 노드 속성이 없음

ActionScript XMLNode (및 XML) 객체에 할당된 커스텀 속성은 만약 해당 객체로의 모든 참조를 드롭하고 다시 획득한 경우라면 지속되지 않는다. 모든 참조를 드롭했을 때 ActionScript 객체만 제거된다. 이 배경에 있는 DOM 트리는 여전히 존재한다. 그러나 커스텀 속성이 ActionScript 객체에 적용되고 DOM 에는 적용되지 않기 때문에 속성의 수명은 직접적으로 ActionScript 객체의 수명과 관련이 있다.

문서를 XML.load 로 로딩할 때 발생할 수 있는 상황이며 이 때 배경에 완전한 DOM 트리에 음영을 주는 상위 수준의 XMLNode 객체를 생성한다. 만약 사용자가 임시 XMLNode 참조를 사용해서 DOM 트리를 전환하고 커스텀 속성을 이러한 임시 속성에 할당하는 경우 차후 접속할 때에는 존재하지 않는다. 예를 들어 (루트는 유효한 XML 데이터를 갖는 XML 또는 XMLNode 라고 가정)

```
// DOM 노드에서 참조 획득
XMLNode temp = root.firstChild;
// 노드에 커스텀 속성 설정
temp.someProperty = someValue;
// 참조 드롭 (모든 참조)
XMLNode temp = temp.nextSibling;
// 참조 가져오기
temp = temp.prevSibling;
// 커스텀 속성 찾기
trace(temp.someProperty)
```

출력은 플래시 내 someValue 가 되나 Scaleform 에선 정의되지 않는데 그 이유는 XMLNode 객체가 이전에 할당된 속성을 더 이상 보유하지 않기 때문이다. 이 곳으로의 참조가 커스텀 속성 할당 및 추적 명령을 통해 지속되는 경우 속성이 존재한다.

주: 이는 XMLNode 에 부착된 속성 객체에 영향을 주지 않는다. 이 객체로 속성을 설정하는 것은 ActionScript XMLNode 의 참조에 상관 없이 지속된다. 예를 들어

```
// DOM 노드에서 참조 획득
XMLNode temp = root.firstChild;
// 커스텀 속성을 노드에 설정
temp.attributes.someProperty = someValue;
// 참조 드롭 (모든 참조)
```

```
XMLNode temp = temp.nextSibling;
// 참조 가져오기
temp = temp.prevSibling;
// 커스텀 속성 찾기
trace(temp.attributes.someProperty)
```

출력값은 Scaleform 내에서 someValue 가 된다.

3.2 ActionScript XML.status 속성이 없음

이 속성은 ActionScript 2.0 XML 에러 코드 등으로의 커스텀 파서 라이브러리로부터 매핑 에러 코드가 발생하는 등의 다양한 불일치 때문에 구현되지 않는다.

ActionScript 에러 코드

- 0 에러 없음, 파싱이 성공적으로 이뤄짐
- -2 CDATA 부분이 제대로 종료되지 않음
- -3 XML 선언이 제대로 종료되지 않음
- -4 DOCTYPE 선언이 제대로 종료되지 않음
- -5 설명이 선언이 제대로 종료되지 않음
- -6 XML 요소가 잘못됨
- -7 메모리초과
- -8 속성값이 제대로 종료되지 않음
- -9 시작 태그와 종료 태그가 매치하지 않음
- -10 종료 태그가 시작 태그와 매치되지 않은 상태로 직면함

ActionScript 매핑에서의 Expat

```
//
// 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
//
```

일관적인 상태 속성을 구현하는 것은 에러 코드의 1 대 1 매핑이 없이는 불가능하다.
GFx::XML::DOMBuilder 는 XML 파서 인스턴스를 통해 등록된 GFx::XML::ParserLocator 객체로부터 데이터를 사용하는 출력을 디버깅하기 위한 에러 메시지를 출력한다.

3.3 ActionScript XML.doc TypeDecl 속성이 없음

플래시 문서로부터 "ActionScript XML 파서는 유효한 파서가 아니다. DOCTYPE 선언을 파서가 읽고 XML.docTypeDecl 속성에 저장하나 DTD 유효화가 이뤄지지 않는다." 이러한 속성은 플래시 및 Scaleform 모두에서 무효하므로 생략한다.