Autodesk®
Scaleform®

ActionsScript 2 XML 用户手册

本文档描述了 Scaleform 4.2 中 ActionScript 2 XML 语言支持和配置

作者: Prasad Silva
版本: 3.0
上次编辑: July 28, 2010

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

Autodesk Scaleform 联系方式：

文档	XML 用户手册
地址	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
网站	www.scaleform.com
邮箱	info@scaleform.com
电话	(301) 446-3200
传真	(301) 446-3199

目录

介绍

Autodesk® Scaleform® SDK 是一个结构简洁、性能高效、功能丰富的 Flash®矢量图形引擎，它建立于净室工具，对控制台和 PC 游戏开发者特别有用。Scaleform 整合了广泛应用的可视化创作工具，如 Adobe® Flash® Studio，具备先进的硬件图形加速功能，满足了前沿游戏开发者的需求。

Flash 通过 AS2 XML API 函数支持 XML 扩展语言。XML 扩展语言和 XMLNode 类包含了加载、解析和 DOM 树管理功能。此类 XML 扩展语言的核心功能在 Scaleform 4.0 当中得到了严格遵循。Scaleform 执行单元内嵌了 DOM 树管理并开放了一个基于 SAX2 的接口，以兼容自定义 XML 解析器。在默认情况下，Scaleform Player 播放器使用了开源的 Expat 解析器。在支持 Scaleform 的所有软件平台上都支持 XML，包括：Windows®, Linux®, MacOS®, Xbox 360®, PSP® (PlayStation®Portable), PlayStation®2 (PS2), PlayStation®3 (PS3™), Nintendo® Wii。

本文档描述了 Scaleform XML 体系结构及其应用方法。并介绍了如何配置一个自定义解析器，最后还介绍了一些局限性。

1. XML 使用

下面为几个实例代码的片段，用简单的几条指令来帮助用户掌握 Scaleform XML 入门方法。

1.1 ActionScript XML 功能支持有效

为使 ActionScript XML 功能支持有效，在 Scaleform 初始化装载时就需要设置 XML 解析器状态参数。以下为 FxPlayer.cpp 文件的部分代码摘录，相同的模式，也可应用于自定义解析器。

```
…
#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++文档解析

解析器的配置参数可以被 C++程序调用并直接解析 XML 文档。如果使能 ActionScript XML 功能，以下代码可以用来从 XML 文件创建 DOM 树（需要有访问全部 Scaleform 源代码的权限）：

```
…
#include "GFx/XML/XML_DOM.h"
using namespace Scaleform;
…

// 创建一个可处理空字符的DOM编译器
XML::DOMBuilder domBuilder(Loader.GetXMLSupport());
// 或者创建一个忽略空字符的DOM编译器
XML::DOMBuilder domBuilder2(Loader.GetXMLSupport(), true);
…
// 处理xml文件并回到DOM树的根目录（新建一个内部对象管理器）
Ptr<XML::Document> pdoc = domBuilder.ParseFile("inputfile.xml",
                                               mLoader.GetFileOpener());

// 或者处理xml文件并回到DOM树的根目录（使用已提供的对象管理器）
Ptr<XML::ObjectManager> pobjMgr = *new XML::ObjectManager();
Ptr<XML::Document> pdoc2 = domBuilder.ParseFile("inputfile.xml",
                                                mLoader.GetFileOpener(), pobjMgr);
…
```

ActionScript XML 支持功能禁止，可以创建一个独立的解析器实例，如下代码所示：

```
…
#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);
```

```
…

// 创建一个可处理空字符的DOM编译器
XML::DOMBuilder domBuilder(pxmlSupport);
// 创建一个忽略空字符的DOM编译器
XML::DOMBuilder domBuilder2(pxmlSupport, true);
…
// 与上部分处理方式相同
…
```

1.3 C++中 DOM 树搜索

以下代码打印 DOM 树的内容（需要有访问全部 Scaleform 源代码的权限）：

```
…
…
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 配置调试报告

具有 Scaleform 全部源码的用户可以设置 GFxXMLConfig.h 头文件中的参数来改变 XML 对象默认存储空间和单元分配大小、XML 解析和 DOM 树结构报告标识。

```
// 调试报告输出和跟踪标记
#ifdef SF_BUILD_DEBUG
// 调试报告中列出了编译器 DOM 树型结构的信息
// #define SF_XML_DOCBUILDER_TRACE
//
// 将所有 DOM 树结构信息按照标准格式输出
// (警告：大文件时避免这种方法)
//
// #define SF_XML_DOCBUILDER_DOM_TREE_DUMP
//
#endif  // #ifdef SF_BUILD_DEBUG
```

2. 配置自定义解析器

Scaleform 提供了一个默认的解析器，使用外部函数库。若应用对象自身具有 XML 解析器，可以将其通过 GFx::XML::Parser 接口嵌入到 GFx XML 子系统，只需改变 XML_Support.h 文件中的定义参数即可。XML_Support.h 文件定义了 GFx::XML::ParserHandler 类作为 SAX2 接口机制。

```
namespace Scaleform { namespace GFx { namespace XML {
//
// XML解析器插件
//
// 为每个文件/字符串调用解析器创建实例。
// 为保证线程安全而添加。每个实例使用单个线程。
// XML状态需要用到解析器实例。
//
class Parser : public RefCountBaseNTS<Parser, Stat_Default_Mem>
{
  public:
    virtual ~Parser() {}

    //
    // Parse methods 解析方法
    //
    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
```

所有 XML 解析器必须用 DOM 编译对象注册一个 GFx::XML::ParserLocator 实例，在解析发生之前，将该实例传递给 ParseFile 和 ParseString 方法。对于所有适当的解析事件，根据预定参数调用 GFx::XM::LParserHandler 回收方法。如果发生错误，则调用对应的错误处理句柄方法函数来处理。

```
namespace Scaleform { namespace GFx { namespace XML {
// SAX2 固定句柄
// DOM编译器作为一个接口与SAX2解析器句柄类似。
// 解析器执行需要调用对应的回收处理程序。
// 特定事件方法。
//
class ParserHandler : public RefCountBase<ParserHandler, StatMV_XML_Mem>
{
  public:
    ParserHandler() {}
    virtual ~ParserHandler() {}

    // 文档开头和结尾
    virtual void        StartDocument() = 0;
    virtual void        EndDocument() = 0;

    // 标记单元开头和结尾
    virtual void        StartElement(const StringRef& prefix,
```

```
                                    const StringRef& localname,
                                    const ParserAttributes& atts) = 0;
    virtual void        EndElement(const StringRef& prefix,
                                   const StringRef& localname) = 0;
    // 命名空间定义。下个单元作为映射的参考
    virtual void        PrefixMapping(const StringRef& prefix,
                                      const StringRef& uri) = 0;
    // 文本数据，标记元素内部
    virtual void        Characters(const StringRef& text) = 0;

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

    // 未处理元素
    virtual void        SkippedEntity(const StringRef& name) = 0;


    // GFx::XMLParser执行器在回收发生之前必须设置文档。GFx::XML::ParserHandler句柄执行需要
    一// 本// 地对象用于错误报告和正确编码，以及xml版本和独立特性。
    virtual void    SetDocumentLocator(const ParserLocator* plocator) = 0;

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

    // 出错句柄回收
    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. 局限性

3.1 ActionScript 中无自定义 XML 节点特性

在 ActionScript XMLNode（和 XML）对象的所有引用被去除或者重复访问时，自定义属性将不能持续。当所有应用被取消时只有 ActionScript 对象被去除。后台的 DOM 树将保留。由于定制特性参数应用于 ActionScript 对象，而不是 DOM，所以其生命周期直接与 ActionScript 对象的生命周期密切相关。

在用 XML.load 导入文档时将创建一个顶层 XMLNode 对象，与后台 DOM 树相对应。若用户在 ActionScript 中采用 XMLNode 引用贯穿 DOM 树并将定制特性赋给这些临时引用，之后再去访问时这些引用信息将丢失。例如（假定拥有 XML 数据的 XML 或者 XMLNode 对象为根）：

```
// 获得 DOM 节点引用
XMLNode temp = root.firstChild;
// 分配节点定制特性
temp.someProperty = someValue;
// 引用丢失（所有引用）
XMLNode temp = temp.nextSibling;
// 引用回收
temp = temp.prevSibling;
// 查寻定制特性
trace(temp.someProperty)
```

Flash™中的一些参数值可以输出，但是在 Scaleform 中这些数值未有明确定义，因为在 Scaleform 中 XMLNode 对象不保存先前分配的特性。只有当引用在这个定制特性分配和状态跟踪中一直存在，这些特性才可以保留。

注意：这不影响附属于 XMLNode 节点的对象。设置这些对象的属性将持续有效，与 ActionScript XMLNode 引用状态无关。例如：

```
// 获得 DOM 节点引用
XMLNode temp = root.firstChild;
// 设置节点定制属性
temp.attributes.someProperty = someValue;
// 引用丢失（所有引用）
XMLNode temp = temp.nextSibling;
// 引用回收
temp = temp.prevSibling;
// 查寻定制特性
trace(temp.attributes.someProperty)
```

输出 Scaleform 中的某些参数值。

3.2 无 ActionScript XML.status 属性

这些属性将不能实现，这是由 ActionScript 2.0 XML 错误码和定制解析库之间的错误映射代码的不一致性导致的，例如，

ActionScript 错误码：

- 0 解析过程完全正确，无错误。
- -2 A CDATA 段异常终止
- -3 XML 声明异常终止
- -4 DOCTYPE 声明异常终止
- -5 注释异常终止
- -6 XML 元素异常
- -7 存储丢失
- -8 属性值异常终止
- -9 起始标签与终止标签不符
- -10 终止标签未找到起始标签

脱离 ActionScript 映射：

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

没有一一对应的错误码，无法维持状态信息。*GFx::XML::DOMBuilder* 在调试输出中打印详细的错误消息，这些数据来自 XML 解析器实例 GFx::XML::ParserLocator 对象。

3.3 无 ActionScript XML.docTypeDecl 属性

来自 Flash® 文档：“ActionScript XML 解析器不是合法解析器。DOCTYPE 声明由解析器读取并存储在 XML.docTypeDecl 属性当中，无 DTD 确认。”这些属性在 Flash 和 Scaleform 中都没有使用，因此将被忽略。