Autodesk®
Scaleform®

ActionScript 2 XML 用戶手冊

本文檔描述了 Scaleform 4.4 中 ActionScript 2 XML 語言支援和配置

作者: Prasad Silva
版本: 3.0
上次編輯: July 28, 2010

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

Autodesk Scaleform 聯繫方式：

地址	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
網站	www.scaleform.com
郵箱	info@scaleform.com
電話	(301) 446-3200
傳真	(301) 446-3199

目錄

介紹 1

1. XML 使用 2

1.1 ActionScript XML 功能支援有效 2

1.2 C++文檔解析 2

1.3 C++中 DOM 樹搜索 3

1.4 配置調試報告 4

2. 配置自定義解析器 6

3. 局限性 8

3.1 ActionScript 中無自定義 XML 節點特性 8

3.2 無 ActionScript XML.status 屬性 9

3.3 無 ActionScript XML.docTypeDecl 屬性 9

介紹

Autodesk® Scaleform® SDK 是一個結構簡潔、性能高效、功能豐富的 Flash 向量圖形引擎，它建立於淨室工具，對控制臺和 PC 遊戲開發者特別有用。Scaleform 整合了廣泛應用的視覺化創作工具，如 Adobe® Flash® Studio，具備先進的硬體圖形加速功能，滿足了前沿遊戲開發者的需求。

Flash 通過 AS2 XML API 函數支援 ActionScript 2 XML 擴展語言。XML 擴展語言和 XMLNode 類包含了載入、解析和 DOM 樹管理功能。此類 XML 擴展語言的核心功能在 Scaleform 4.4 當中得到了嚴格遵循。Scaleform 執行單元內嵌了 DOM 樹管理並開放了一個基於 SAX2 的介面，以相容自定義 XML 解析器。在默認情況下，Scaleform Player 播放器使用了開源的 Expat 解析器。在支援 Scaleform 的所有軟體平臺上都支援 XML，包括：Windows®, Linux®, MacOS®, Xbox 360®, PSP® (PlayStation®Portable), PlayStation®2 (PS2), PlayStation®3 (PS3™), Nintendo® Wii。

本文檔描述了 Scaleform XML 體系結構及其應用方法。並介紹了如何配置一個自定義解析器，最後還介紹了一些局限性。

1. XML 使用

下麵為幾個實例代碼的片段，用簡單的幾條指令來幫助用戶掌握 Scaleform XML 入門方法。

1.1 ActionScript XML 功能支援有效

為使 ActionScript XML 功能支援有效，在 Scaleform 初始化裝載時就需要設置 XML 解析器狀態參數。以下為 FxPlayer.cpp 檔的部分代碼摘錄，相同的模式，也可應用於自定義解析器。

```
…
#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++文檔解析

解析器的配置參數可以被 C++程式調用並直接解析 XML 文檔。如果使能 ActionScript XML 功能，以下代碼可以用來從 XML 檔創建 DOM 樹（需要有訪問全部 Scaleform 源代碼的許可權）：

```
…
#include "GFx/XML/XML_DOM.h"
using namespace Scaleform;
…
// 創建一個可處理空字元的DOM編譯器
XML::DOMBuilder domBuilder(Loader.GetXMLSupport());
// 或者創建一個忽略空字元的DOM編譯器
XML::DOMBuilder domBuilder2(Loader.GetXMLSupport(), true);
…
// 處理xml檔並回到DOM樹的根目錄（新建一個內部物件管理器）
Ptr<XML::Document> pdoc = domBuilder.ParseFile("inputfile.xml",
                                               mLoader.GetFileOpener());

// 或者處理xml檔並回到DOM樹的根目錄（使用已提供的物件管理器）
Ptr<XML::ObjectManager> pobjMgr = *new XML::ObjectManager();
Ptr<XML::Document> pdoc2 = domBuilder.ParseFile("inputfile.xml",
                                                mLoader.GetFileOpener(), pobjMgr);
…
```

ActionScript XML 支援功能禁止，可以創建一個獨立的解析器實例，如下代碼所示：

```
…
#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);
…

// 創建一個可處理空字元的DOM編譯器
XML::DOMBuilder domBuilder(pxmlSupport);
// 創建一個忽略空字元的DOM編譯器
XML::DOMBuilder domBuilder2(pxmlSupport, true);
…
// 與上部分處理方式相同
…
```

1.3 C++中 DOM 樹搜索

以下代碼列印 DOM 樹的內容（需要有訪問全部 Scaleform 源代碼的許可權）：

```
…
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 配置調試報告

具有 Scaleform 全部源碼的用戶可以設置 XML_DOM.cpp 頭檔中的參數來改變 XML 物件默認存儲空間和單元分配大小、XML 解析和 DOM 樹結構報告標識。

```
//
// 調試報告輸出和跟蹤標記
#ifdef SF_BUILD_DEBUG
////
```

```
// 調試報告中列出了編譯器 DOM 樹型結構的資訊
// #define SF_XML_DOCBUILDER_TRACE
//
// 將所有 DOM 樹結構資訊按照標準格式輸出
// (警告：大檔時避免這種方法)
//
// #define SF_XML_DOCBUILDER_DOM_TREE_DUMP
//
#endif  // #ifdef SF_BUILD_DEBUG
```

2. 配置自定義解析器

Scaleform 提供了一個默認的解析器，使用外部函數庫。若應用物件自身具有 XML 解析器，可以將其通過 GFx::XML::Parser 介面嵌入到 Scaleform XML 子系統，只需改變 XML_Support.h 檔中的定義參數即可。XML_Support.h 檔定義了 GFx::XML::ParserHandler 類作為 SAX2 介面機制。

```
namespace Scaleform { namespace GFx { namespace XML {
// XML解析器插件
//
// 為每個檔/字串調用解析器創建實例。
// 為保證線程安全而添加。每個實例使用單個線程。
// XML狀態需要用到解析器實例。
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 解析器必須用 DOM 編譯物件註冊一個 GFx::XML::ParserLocator 實例，在解析發生之前，將該實例傳遞給 ParseFile 和 ParseString 方法。對於所有適當的解析事件，根據預定參數調用 GFx::XML::ParserHandler 回收方法。如果發生錯誤，則調用對應的錯誤處理控制碼方法函數來處理。

```
namespace Scaleform { namespace GFx { namespace XML {
// SAX2 固定控制碼
// DOM編譯器作為一個介面與SAX2解析器控制碼類似。
// 解析器執行需要調用對應的回收處理程式。
// 特定事件方法。
//
class ParserHandler : public RefCountBase<ParserHandler, StatMV_XML_Mem>
{
  public:
    ParserHandler() {}
    virtual ~ParserHandler() {}
```

```
    // 文檔開頭和結尾
    virtual void        StartDocument() = 0;
    virtual void        EndDocument() = 0;

    // 標記單元開頭和結尾
    virtual void        StartElement(const StringRef& prefix,
                                     const StringRef& localname,
                                     const ParserAttributes& atts) = 0;
    virtual void        EndElement(const StringRef& prefix,
                                   const StringRef& localname) = 0;
    // 命名空間定義。下個單元作為映射的參考
    virtual void        PrefixMapping(const StringRef& prefix,
                                      const StringRef& uri) = 0;
    // 文本資料，標記元素內部
    virtual void        Characters(const StringRef& text) = 0;

    // 空字元
    virtual void        IgnorableWhitespace(const XMLStringRef& ws) = 0;

    // 未處理元素
    virtual void        SkippedEntity(const XMLStringRef& name) = 0;


    // GFx::XML::Parser執行器在回收發生之前必須設置文檔。GFx::XML::ParserHandler控制碼執行
    需要一// //本地物件用於錯誤報告和正確編碼，以及xml版本和獨立特性。
    virtual void    SetDocumentLocator(const ParserLocator* plocator) = 0;

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

    // 出錯控制碼回收
    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 中無自定義 XML 節點特性

在 ActionScript XMLNode（和 XML）物件的所有引用被去除或者重複訪問時，自定義屬性將不能持續。當所有應用被取消時只有 ActionScript 物件被去除。後臺的 DOM 樹將保留。由於定制特性參數應用於 ActionScript 物件，而不是 DOM，所以其生命週期直接與 ActionScript 物件的生命週期密切相關。

在用 XML.load 導入文檔時將創建一個頂層 XMLNode 物件，與後臺 DOM 樹相對應。若用戶在 ActionScript 中採用 XMLNode 引用貫穿 DOM 樹並將定制特性賦給這些臨時引用，之後再去訪問時這些引用資訊將丟失。例如（假定擁有 XML 資料的 XML 或者 XMLNode 物件為根）：

```
// 獲得 DOM 節點引用
XMLNode temp = root.firstChild;
// 分配節點定制特性
temp.someProperty = someValue;
// 引用丟失（所有引用）
XMLNode temp = temp.nextSibling;
// 引用回收
temp = temp.prevSibling;
// 查尋定制特性
trace(temp.someProperty)
```

Flash™中的一些參數值可以輸出，但是在 Scaleform 中這些數值未有明確定義，因為在 Scaleform 中 XMLNode 物件不保存先前分配的特性。只有當引用在這個定制特性分配和狀態跟蹤中一直存在，這些特性才可以保留。

注意：這不影響附屬於 XMLNode 節點的對象。設置這些物件的屬性將持續有效，與 ActionScript XMLNode 引用狀態無關。例如：

```
// 獲得 DOM 節點引用
XMLNode temp = root.firstChild;
// 設置節點定制屬性
temp.attributes.someProperty = someValue;
// 引用丟失（所有引用）
XMLNode temp = temp.nextSibling;
// 引用回收
temp = temp.prevSibling;
// 查尋定制特性
trace(temp.attributes.someProperty)
```

輸出 Scaleform 中的某些參數值。

3.2 無 ActionScript XML.status 屬性

這些屬性將不能實現，這是由 ActionScript 2.0 XML 錯誤碼和定制解析庫之間的錯誤映射代碼的不一致性導致的，例如，

ActionScript 錯誤碼：

- 0 解析過程完全正確，無錯誤。
- -2 A CDATA 段異常終止
- -3 XML 聲明異常終止
- -4 DOCTYPE 聲明異常終止
- -5 注釋異常終止
- -6 XML 元素異常
- -7 存儲丟失
- -8 屬性值異常終止
- -9 起始標籤與終止標籤不符
- -10 終止標籤未找到起始標籤

脫離 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
//
```

沒有一一對應的錯誤碼，無法維持狀態資訊。*GFx::XML::DOMBuilder* 在調試輸出中列印詳細的錯誤消息，這些資料來自 XML 解析器實例 GFx::XML::ParserLocator 物件。

3.3 無 ActionScript XML.docTypeDecl 屬性

來自 Flash® 文檔：“ActionScript XML 解析器不是合法解析器。DOCTYPE 聲明由解析器讀取並存儲在 XML.docTypeDecl 屬性當中，無 DTD 確認。”這些屬性在 Flash 和 Scaleform 中都沒有使用，因此將被忽略。