Easy XML Parser (v.0.8)

Содержание

Введение

Easy XMLParser (v.0.8)

В свое время пришлось плотно поработать с PalmO, и тема xml парсера была достаточно актуальной. Обычно данные в PalmOs хранятся в бинарном виде. У данного подхода только один плюс – это быстрая скорость обработки данных. Все остальное – это минусы. Приходится тщательно соблюдать формат данных в базе, сложность в тестировании, сложность в изменении и расширении формата данных – все это может запросто отбить охоту в хранении двоичных данных. Все таки, для палма это часто имеет смысл, и разработчики смирились с этим.

Так уж получилось в мое практике, что xml парсер стал жизненно необходимым для нашего проекта на палме. В результате большого труда, как поиск готового продукта, серфинг, анализ открытых исходных кодов, внесение своих идей, исправление ошибок, был получен xml парсер, который прекрасно работал под управлением системы PalmOs 4.0 и выше. Парсер был максимально оптимизирован по скорости работы и по потреблению памяти на этой системе и прекрасно справлялся с xml документом любой сложности и любой сложенности.

Некоторое время позже возник интерес по переносу этого парсера на другую операционную систему, для начала в Windows. При этом, в основе переноса была идея свободного переноса парсера и на другие системы в будущем, например Linux, Symbian. Учитывая то, что парсер потребляет мало операционной памяти и достаточно быстр, я подумал, что он может быть востребован в приложениях, которые требуют быстрой загрузки, быстрой обработки, и не привязаны к системно-зависимым технологиям, таким как COM к примеру.

В результате я получил версию парсера 0.8. почему версия 0.8?

1. эта версия работает только с документами ASCII
2. эта версия предназначена только для *чтения*, но не для создания

В случае исправления этих пунктов я смогу присвоить парсеру версию 1.0.
Итак, почему вам стоит использовать этот парсер?

1. Он потребляет мало памяти. Это достигнуто за счет того, что для каждого нода создается структура, элементами которой являются не отдельные строки или выделенные участки памяти, а указатели на имеющийся документ (xml строку).
2. Он достаточно быстрый, и устойчивый.
3. Код парсера является переносимым, так как в нем используются стандартные C функции для работы с памятью. Его нужно только перекомпиллировать под нужной системой.
4. Парсер абстрагируется от приложения при помощи интерфейсов. Парсер находится внутри библиотеки, которая удобно расширяет свою функциональность при помощи интерфейсов. Это гарантирует, что код приложения будет работать с библиотекой даже в том случае, если появятся новые функции. Они будут просто вынесены в отдельный интерфейс.
5. Парсер абсолютно бесплатный. Кроме того, я буду отслеживать и фиксить обнаруженные ошибки, а также учитывать пожелания пользователей.

Описание продукта

- *Easy XML Parser* версии 0.8 предназначен *для чтения*.

- *Easy XML Parser* версии 0.8 работает только xml строками.

- *Easy XML Parser* версии 0.8 не работает по стандарту DOM или SAX. Он по умолчанию не возвращает управление программе до тех пор, пока полностью не отпарсит текущий нод. Если приложению нужно получать результаты работы парсера во время самого парсинга, то для этого библиотека предусматривает интерфейс, обеспечивающий обратную связь между парсером и приложением.

SDK

Easy XML Parser предоставляет следующий набор функций для обработки XML документа. Эти функции является частью интерфейса **IparserDispatcher** и описаны в файле **IParserDispatcher.h**

Функции для чтения XML документа:

*Инициализировать структуру **XMLObject.***

```
void Initialize ( XMLObject* _pXMLO );
```

Параметры

_pXMLO
[in,out] указатель на структуру **XMLObject** для ее инициализации

*Освободить память, зарезервированную структурой **XMLObject.***

```
void ReleaseXml ( XMLObject* _pXMLO );
```

Параметры

_pXMLO
[in] указатель на структуру **XMLObject** для ее корректного освобождения. Вызывайте эту функцию в случае, когда очередной XML node был обработан.

Обработать XML node

```
Bool Parse ( XMLObject* _pXMLO,
             char* _pstrXmlDocument,
             int _wEnd,
             int* _pwElementSize,
             bool* _pwShortCircuit,
             int* _pwErr,
             int* _pwErrOffset );
```

Параметры

_pXMLO
[in, out] указатель на структуру **XMLObject**, в которой возвращается текущий XML node.

_pstrXmlDocument
[in] строка, в которой хранится документ.

_wEnd
[in] размер node в байтах

_pwElementSize
[out] в этом параметре возвращается реальный размер node.

_pwShortCircuit
[out] равен true только для заголовочного node

_pwErr
[out] в случае сбоя в этом параметре возвращается код ошибки

_pwErrOffset
[out] в случае сбоя в этом параметре возвращается смещение внутри node относительно его начала, где была обнаружена ошибка

Возвращаемые значения

True, если Xml Parser отработал без ошибок. В случае возникновения ошибки ее код можно посмотреть в параметре *err*

Вернуть XML элемент (полностью весь node)

```
char* GetElement ( char* _pstrXmlDocument,
                   XMLObject* _pXMLO,
                   int* _pwElementsize );
```

Параметры

_pstrXmlDocument
[in] указатель на весь XML документ

_pXMLO
[in] указатель на структуру XMLObject

_pwElementsize
[out] в этом параметре возвращается размер искомого элемента.

Возвращаемые значения

Возвращает позицию в XML документе, с которой начинается node

Вернуть имя XML node

```
char* GetTagName ( char* _pstrXmlDocument,
                   XMLObject* _pXMLO,
                   int* _pwLen );
```

Параметры

_pstrXmlDocument
[in] указатель на XML документ

_pXMLO
[in] указатель на структуру XMLObject

_pwLen
[out] в этом параметре возвращается длина имени node

Возвращаемые значения

Возвращается позиция внутри XML документа, с которой начинается имя node

Получить контекст XML нода

```
char* GetContent ( char* _pstrXmlDocument,
                   XMLObject* _pXMLO,
                   int* _pwLen,
                   int* _pwNumEncodings );
```

Параметры

_pstrXmlDocument
[in] указатель на XML документ

_pXMLO
[in] указатель на структуру **XMLObject**

_pwLen
[out] в этом параметре возвращается длина контекста

_pwNumEncodings
[out] содержит количество символов ‘&’ в контексте

Возвращаемые значения

Возвращает указатель на требуемый контекст

Вернуть дочерний XML node

```
XMLObject* GetChild ( char* _pstrXmlDocument,
                      char* _pstrChild,
                      XMLObject* _pXMLO );
```

Параметры

_pstrXmlDocument
[in] указатель на XML документ

_pstrChild
[in] имя дочернего node

_pXMLO
[in] указатель на структуру **XMLObject**

Возвращаемые значения

Возвращает указатель на структуру **XMLObject** с дочерним node

Вернуть дочерний node (способ 2)

```
XMLObject* GetChild ( XMLObject* _pXMLO,
                      int_n );
```

Параметры

_pXMLO
[in] указатель на структуру **XMLObject**

_n
[in] индекс дочернего node

Возвращаемые значения
Возвращает указатель на структуру **XMLObject** с дочерним node

Вернуть значение атрибута

```
char* GetAttribute ( char* _pstrXmlDocument,
                     char* _pstrAttribute,
                     XMLObject* _pXMLO,
                     int* _pwLen );
```

Параметры

_pstrXmlDocument
[in] указатель на XML документ

_pstrAttribute
[in] имя атрибута

_pXMLO
[in] указатель на структуру **XMLObject**

_pwLen
[out] в этом параметре возвращается значение атрибута

Возвращаемые значения

Возвращается указатель внутри строки с node на значение атрибута.

Вернуть количество дочерних nodes

```
int GetNumChildren ( XMLObject* _pXMLO );
```

Параметры

_pXMLO
[in] указатель на структуру **XMLObject**

Возвращаемые значения

Количество дочерних nodes

Возвращает имя атрибута по его индексу

```
char* GetAttributeByIndex ( char* _pstrXmlDocument,
                            XMLObject* _pXMLO,
                            int _wIndex,
                            int* _pwLen );
```

Параметры

_pstrXmlDocument
[in] указатель на XML документ

_pXMLO
[in] указатель на структуру **XMLObject**

_wIndex
[in] номер индекса искомого атрибута

_pwLen
[out] в этом параметре возвращается длина имени атрибута

Возвращаемые значения

Возвращается указатель внутри XML документа на позицию, с которой начинается имя атрибута.

Примеры кода

Пример использования Easy XML Parser вы можете найти в папке **xmlreader**. Это консольная программа, написанная для Visual Studio 6.0. Ниже следуют краткие пояснения к коду.

Парсер находится внутри библиотеки **easyxmlparser.dll**, которая выставляет наружу интерфейсы для его использования.

Версия 0.8 поддерживает два интерфейса:
1. **IParserDispatcher** – для непосрадетсвенной работы с парсером.
2. **ICcallbackDispatcher –** для обеспечения обратной связи между парсером и приложением.

Второй интерфейс нужен для нетерпеливых. На самом деле, парсер не возвращает управления до тех пор, покуда полностью не обработает полученный xml нод. Если приложение хочет получать информацию о нодах по мере их обработки, оно должно поспользоваться интерфейсом **ICallbackDispatcher**. Так как парсер активно использует рекурсию в своей работе, обработка нодов ведется в обратной последовательности. То есть в callback вернется сначала обработанный нод с более глубоким уровнем вложения, и только после него – нод верхнего уровня.

Итак, рассмотрим пример работы с **easy xmlparser** и сделаем необходимые пояснения.

Файл **xmlreader.cpp**

Здесь используется библиотека **easyxmlparser.dll**, которая экспортирует только одну функцию **GetCore.** Задача этой функции, вернуть указатель на интерфейс **ICore**, который описан в файле **ICore.h**.

```
typedef ICore* ( *GETCORE )();
```

Задача интерфейса **ICore** – обеспечить доступ к внутренним интерфейсам библиотеки по их идентификатору. Идентификаторы описаны в файле **iid.h** (перечисление **E_QUERY**).

Доступ к самому парсеру осуществляется при помощи интерфейса **IParserDispatcher,** который получается следующим образом.

```
IParserDispatcher* pParserDispatcher = NULL;
_pCore->QueryDispatcher( IID_QUERY::EASCII_DISPATCHER,
                         ( IPluginCore** )&pParserDispatcher );
```

Для того, чтобы получить интерфейс, который обеспечивает обратную связь парсера с приложением, нужно получить интерфейс **ICallbackDispatcher**, который описан в файле **ICallbackDispatcher.h**.

```
ICallbackDispatcher* pCallbackDispatcher = NULL;
_pCore->QueryDispatcher( IID_QUERY::ECALLBACK_DISPATCHER,
                                ( IPluginCore** )&pCallbackDispatcher );
```

Изучите в примере функцию **Parse**, которая получает оба интерфейса и использует их.

```
szNode = &szNode[ wSize ];
```

Обратите внимание на вышеприведенный код. Он смещает указатель внутри строки на начало следующего нода, который будет скормлен парсеру. Это означает, что реально парсер обрабатывает текущий нод. Строка должна указывать на его начало. Таким образом, сам xml документ может содержать много корневых нодов, парсер будет обрабатывать только текущий. После обработки очередного парсера нужно освободить память.

```
pParserDispatcher->ReleaseXml( &XMLO );
```

В примере чтение данных из парсера происходит в классе, который обеспечивает обратную связь с парсером. Как уже отмечалось выше, такой подход не дает правильной последовательности нодов. Для того, чтобы прочесть xml строку в той последовательности, в которой она создана, нужно дождаться полной обработки xml строки, и после этого читать данные из парсера при помощи инитерфейса **IParserDispatcher**.

Полезные советы

Не допускайте наличие утечек памяти в вашем приложении.

Xml Parser при обработке документа потребляет мало памяти. Это достигнуто путем манипуляции указателями в пределах строки с *XML* документом. При обработке node приложением *Xml Parser* возвращает не отдельный зарезервированный участок памяти, а указатель в пределах строки с *XML* документом. Поэтому не надо освобождать память, которую возвращает *Xml Parser*. Это обеспечивает высокую скорость работы. Если приложению нужно получить отдельный *element*, *node*, *context* или *attribute*, то оно само должно заботиться о выделении и освобождении памяти.