Qt LZO

Qt LZO Site : http://qtlzo.sourceforge.net

Qt LZO is a Lempel-Ziv-Oberhumer data compression wrapper for Qt.

Original LZO library site is http://www.oberhumer.com, author is Markus F.X.J. Oberhumer. Current version is "2.09". LZO was used on scientific researches such as NASA projects.

The author of QtLzo is Brian Lin, who is also an astrophysicist. QtLzo is used by the author on his own theoretical physics researches such as Oriphase theory, dark matters and Collective Intelligence Operating System(CIOS) for robotic applications.

For details, please pay a visit to each of the following languages:

English	正體中文	简体中文

Qt LZO is also a part of CIOS Compression Subsystem.

CIOS / Compression Subsystem			
QtGZip	QtBZip2	QtXz	QtLzo
QtLZip	QtUCL		
QtCompression			
QtTAR	QtZIP	QtRAR	Qt7z
QtArchivers			

Download

Method	Filename
7z	QtLzo-2015-11-02.7z
Tar.gz	QtLzo-2015-11-02.tar.gz
Tar.xz	QtLzo-2015-11-02.tar.xz
Mercurial	hg clone ssh://linfoxman@hg.code.sf.net/p/qtlzo/mercurial QtLzo
Git	git clone ssh://linfoxman@git.code.sf.net/p/qtlzo/git QtLzo

Contacts

Author : Brian Lin (a.k.a Vladimir Lin , Vladimir Forest , Владимир Лесной)

The author generated over 6 millions lines of codes manually in diverse fields and maintains hundreds of projects. Please be specific on which project you are referring. In case you want to report and ask questions about Qt LZO, please add a "Qt LZO" in a short and brief form when you report or ask questions.

E-mail	lin.foxman@gmail.com
E-mail	lin.vladimir@gmail.com
E-mail	wolfram_lin@yahoo.com
E-mail	wolfram_lin@sina.com
E-mail	wolfram_lin@163.com
Skype	wolfram_lin
WeChat	153-0271-7160
WhatsApp	153-0271-7160
ICQ	153-0271-7160
QQ	lin.vladimir@gmail.com

目錄

LZO 壓縮法說明

LZO 的完整名稱為Lempel-Ziv-Oberhumer，於1996 發表第一版的壓縮法，其後持續改進，共計有二十餘種變形壓縮方法。

LZO 是以壓縮、解壓的速度為致力的方向，為一種內容無損壓縮法。

由於LZO 的低資源需求，它被用於許多個太空計劃、科學研究計畫及機器人系統當中。

Qt LZO 當中支持的LZO 壓縮方式如下：

- LZO 1
- LZO 1 - X
- LZO 1 - 99

- LZO 1a
- LZO 1a - 99
- LZO 1b
- LZO 1b - 99
- LZO 1b - 999
- LZO 1c
- LZO 1c - 99
- LZO 1c - 999
- LZO 1f - 1
- LZO 1f - 999
- LZO 1x - 1
- LZO 1x - 1-11
- LZO 1x - 1-12
- LZO 1x - 1-15
- LZO 1x - 999
- LZO 1y - 1
- LZO 1y - 999
- LZO 1z - 999
- LZO 2a - 999

Qt LZO 的使用方式

Qt LZO 有兩種使用方式，編譯成Qt 模組或是直接將原始碼包含入您的開發計畫當中。

編譯成Qt 模組的方法，請見「QtLZO 編譯說明」。

直接將原始碼包含入您的開發計畫當中的方法，如下：

1. 下載「Qt LZO」（QtLzo-2015-11-02.7z），並且解開檔案。
2. 進入「QtLzo/Src/Embedded」目錄當中。
3. 將「QtLzo/Src/Embedded/LZO」目錄複製到您的開發計畫當中。
4. 在您的Qt PRO 計畫檔當中加入「LZO/LZO.pri」。
5. 將適當的LZO 編譯好的library 複製到編譯時期可以找到的目錄。

LZO 原始碼編譯說明

網站：http://www.oberhumer.com/
原始碼：http://www.oberhumer.com/opensource/lzo/download/lzo-2.09.tar.gz
版本：2.09

目錄

Windows Visual Studio 編譯方法

Windows Visual Studio 需要編譯八種模式 :

- Windows Visual Studio x64 (Static , Debug)
- Windows Visual Studio x64 (Static , Release)
- Windows Visual Studio x64 (DLL , Debug)
- Windows Visual Studio x64 (DLL , Release)
- Windows Visual Studio x86 (Static , Debug)
- Windows Visual Studio x86 (Static , Release)
- Windows Visual Studio x86 (DLL , Debug)
- Windows Visual Studio x86 (DLL , Release)

解壓縮

lzo-2.09.tar.gz 在「QtLzo/3rdparty/souces」當中可以找到 , 解壓縮lzo-2.09.tar.gz 以後 , 會留下一個lzo-2.09 目錄。

在平行的目錄下 :

```
mkdir lzo
cd lzo
```

Windows Visual Studio x64 (Static , Debug)

```
cmake -G "VIsual Studio 12 Win64" ..\lzo-2.09
cd ..
cmake-gui lzo
```

將ENABLE_SHARED 關閉掉 , 打開ENABLE_STATIC , 按下「Configure」 , 然後「Generate」。

如此可以產生Visual Studio 的sln 檔案 , 接下來選擇lzo.sln 打開Visual Studio , 選擇「Debug」 , 並且開始編譯。

Windows Visual Studio x64 (Static , Release)

```
cmake -G "VIsual Studio 12 Win64" ..\lzo-2.09
cd ..
cmake-gui lzo
```

將ENABLE_SHARED 關閉掉 , 打開ENABLE_STATIC , 按下「Configure」 , 然後

「Generate」。

如此可以產生Visual Studio 的sln 檔案，接下來選擇lzo.sln 打開Visual Studio，選擇「Release」，並且開始編譯。

Windows Visual Studio x64 (DLL , Debug)

```
cmake -G "VIsual Studio 12 Win64" ..\lzo-2.09
cd ..
cmake-gui lzo
```

將ENABLE_STATIC 關閉掉，打開ENABLE_SHARED，按下「Configure」，然後「Generate」，如此可以產生Visual Studio 的sln 檔案。

打開「lzo-2.09/include/lzo/lzoconf.h」，在大約220 行找到：

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1        /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2        /*empty*/
#endif
```

修改成為：

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1 __declspec(dllexport) /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2        /*empty*/
#endif
```

接下來選擇lzo.sln 打開Visual Studio，選擇「Debug」，並且開始編譯。

Windows Visual Studio x64 (DLL , Release)

```
cmake -G "VIsual Studio 12 Win64" ..\lzo-2.09
cd ..
cmake-gui lzo
```

將ENABLE_STATIC 關閉掉，打開ENABLE_SHARED，按下「Configure」，然後「Generate」，如此可以產生Visual Studio 的sln 檔案。

打開「lzo-2.09/include/lzo/lzoconf.h」，在大約220 行找到：

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1        /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2        /*empty*/
#endif
```

修改成為：

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1 __declspec(dllexport) /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2        /*empty*/
#endif
```

接下來選擇lzo.sln 打開Visual Studio，選擇「Release」，並且開始編譯。

Windows Visual Studio x86 (Static , Debug)

```
cmake ..\lzo-2.09
cd ..
cmake-gui lzo
```

將ENABLE_SHARED 關閉掉，打開ENABLE_STATIC，按下「Configure」，然後「Generate」。

如此可以產生Visual Studio 的sln 檔案，接下來選擇lzo.sln 打開Visual Studio，選擇「Debug」，並且開始編譯。

Windows Visual Studio x86 (Static , Release)

```
cmake ..\lzo-2.09
cd ..
cmake-gui lzo
```

將ENABLE_SHARED 關閉掉，打開ENABLE_STATIC，按下「Configure」，然後「Generate」。

如此可以產生Visual Studio 的sln 檔案，接下來選擇lzo.sln 打開Visual Studio，選擇「Release」，並且開始編譯。

Windows Visual Studio x86 (DLL , Debug)

```
cmake ..\lzo-2.09
```

```
cd ..
cmake-gui lzo
```

將ENABLE_STATIC 關閉掉，打開ENABLE_SHARED，按下「Configure」，然後「Generate」，如此可以產生Visual Studio 的sln 檔案。

打開「lzo-2.09/include/lzo/lzoconf.h」，在大約220 行找到：

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1        /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2        /*empty*/
#endif
```

修改成為：

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1 __declspec(dllexport) /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2        /*empty*/
#endif
```

接下來選擇lzo.sln 打開Visual Studio，選擇「Debug」，並且開始編譯。

Windows Visual Studio x86 (DLL , Release)

```
cmake ..\lzo-2.09
cd ..
cmake-gui lzo
```

將ENABLE_STATIC 關閉掉，打開ENABLE_SHARED，按下「Configure」，然後「Generate」，如此可以產生Visual Studio 的sln 檔案。

打開「lzo-2.09/include/lzo/lzoconf.h」，在大約220 行找到：

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1        /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2        /*empty*/
#endif
```

修改成為 :

```
/* DLL export information */
#if !defined(__LZO_EXPORT1)
#  define __LZO_EXPORT1 __declspec(dllexport) /*empty*/
#endif
#if !defined(__LZO_EXPORT2)
#  define __LZO_EXPORT2         /*empty*/
#endif
```

接下來選擇lzo.sln 打開Visual Studio，選擇「Release」，並且開始編譯。

Qt LZO 編譯說明

網站 : http://qtlzo.sourceforge.net
原始碼 : https://sourceforge.net/projects/qtlzo/files/
版本 : 2015-11-02

目錄

Qt LZO 目錄結構

- QtLzo
  - 3rdparty
    - scripts
      - Update-LZO.js
    - sources
      - lzo-2.09.tar.gz
    - Windows
      - x64
        - Debug
          - dll
            - lzo2.dll

- lzo2.def
- lzo2.ilk
- lzo2.lib
- lzo2.pdb
- lib
- lzo2.lib
- Release
- dll
- lzo2.dll
- lzo2.def
- lzo2.ilk
- lzo2.lib
- lzo2.pdb
- lib
- lzo2.lib
- x86
- Debug
- dll
- lzo2.dll
- lzo2.def
- lzo2.ilk
- lzo2.lib
- lzo2.pdb
- lib
- lzo2.lib
- Release
- dll
- lzo2.dll
- lzo2.def
- lzo2.ilk
- lzo2.lib
- lzo2.pdb
- lib
- lzo2.lib
- doc
- HTML
- index.html
- ODT
- QtLZO.odt
- QtLZO-TW.odt
- QtLZO-CN.odt

- PDF
  - QtLZO.pdf
  - QtLZO-TW.pdf
  - QtLZO-CN.pdf
- Qt
  - cn
    - examples.html
    - index.html
    - lzo.html
    - others.html
    - qtlzo.html
    - source.html
    - Replacements.txt
  - en
    - examples.html
    - index.html
    - lzo.html
    - others.html
    - qtlzo.html
    - source.html
  - tw
    - examples.html
    - index.html
    - lzo.html
    - others.html
    - qtlzo.html
    - source.html
  - classic.css
  - index.html
  - Qt.pri
  - QtLzo.qhp
- TeX
  - QtLZO.tex

- examples
  - lzotool
    - lzotool.cpp
    - lzotool.ico
    - lzotool.pro
    - lzotool.rc
    - LZO.js
  - examples.pro

- include
  - QtLzo
    - lzo
      - LZO Header files
    - headers.pri
    - QtLzo
    - qtlzo.hpp
- src
  - Embedded
    - LZO
      - lzo
        - LZO Header files
      - LZO.pri
      - qtlzo.cpp
      - qtlzo.hpp
      - ScriptableLzo.cpp
  - QtLzo
    - qtlzo.cpp
    - ScriptableLzo.cpp
    - QtLzo.pro
  - src.pro
- tests
  - auto
    - cmake
      - cmake.pro
      - CMakeLists
    - auto.pro
  - tests.pro

安排好LZO library 的檔案位置

如果您使用Unix 系統，您只需要將LZO 函式庫以正常方式編譯安裝到內定目錄即可。如果您使用Windows 系統，那麼您需要選取所需要的函式庫，擺到內定的編譯環境目錄下。

Unix

Unix 下編譯LZO 函式庫，只需要使用cmake 編譯安裝到內定位置即可。

```
cmake ..\lzo-2.09
make
make install
```

Windows

Windows 平台需要依照不同的架構來擺放需要的檔案。

x64 Static

- QtLzo\3rdparty\Windows\x64\Debug\lib\lzo2.lib
- QtLzo\3rdparty\Windows\x64\Release\lib\lzo2.lib

將「QtLzo\3rdparty\Windows\x64\Debug\lib\lzo2.lib」更名為「lzo2d.lib」。

將「lzo2.lib」及「lzo2d.lib」複製到「QtTargetDirectory\lib」當中即可使用靜態編譯的 Lzo 函式庫。

x64 Shared

- QtLzo\3rdparty\Windows\x64\Debug\dll\lzo2.dll
- QtLzo\3rdparty\Windows\x64\Debug\dll\lzo2.lib
- QtLzo\3rdparty\Windows\x64\Debug\dll\lzo2.ilk
- QtLzo\3rdparty\Windows\x64\Debug\dll\lzo2.pdb
- QtLzo\3rdparty\Windows\x64\Release\dll\lzo2.dll
- QtLzo\3rdparty\Windows\x64\Release\dll\lzo2.lib

將「QtLzo\3rdparty\Windows\x64\Debug\lib\lzo2.dll」更名為「lzo2d.dll」。
將「QtLzo\3rdparty\Windows\x64\Debug\lib\lzo2.lib」更名為「lzo2d.lib」。

將「lzo2.dll」及「lzo2d.dll」複製到「QtTargetDirectory\bin」。

將「lzo2.ilk」、「lzo2.pdb」、「lzo2.lib」及「lzo2d.lib」複製到「QtTargetDirectory\lib」當中即可使用靜態編譯的Lzo 函式庫。

x86 Static

- QtLzo\3rdparty\Windows\x86\Debug\lib\lzo2.lib
- QtLzo\3rdparty\Windows\x86\Release\lib\lzo2.lib

將「QtLzo\3rdparty\Windows\x86\Debug\lib\lzo2.lib」更名為「lzo2d.lib」。

將「lzo2.lib」及「lzo2d.lib」複製到「QtTargetDirectory\lib」當中即可使用靜態編譯的 Lzo 函式庫。

x86 Shared

- QtLzo\3rdparty\Windows\x86\Debug\dll\lzo2.dll
- QtLzo\3rdparty\Windows\x86\Debug\dll\lzo2.lib
- QtLzo\3rdparty\Windows\x86\Debug\dll\lzo2.ilk

- QtLzo\3rdparty\Windows\x86\Debug\dll\lzo2.pdb
- QtLzo\3rdparty\Windows\x86\Release\dll\lzo2.dll
- QtLzo\3rdparty\Windows\x86\Release\dll\lzo2.lib

將「QtLzo\3rdparty\Windows\x86\Debug\lib\lzo2.dll」更名為「lzo2d.dll」。
將「QtLzo\3rdparty\Windows\x86\Debug\lib\lzo2.lib」更名為「lzo2d.lib」。

將「lzo2.dll」及「lzo2d.dll」複製到「QtTargetDirectory\bin」。

將「lzo2.ilk」、「lzo2.pdb」、「lzo2.lib」及「lzo2d.lib」複製到「QtTargetDirectory\lib」當中即可使用靜態編譯的Lzo 函式庫。

編譯QtLzo

將LZO 函式庫安排好位置後，便可以準備開始編譯QtLzo。

您需要使用Qt 的原始碼來編譯QtLzo 模組，QtLzo 模組的擺設位置如下：

- QtTargetDirectory
- QtSourceDirectory
  - QtLzo
  - qtbase
  - gnuwin32
  - ...

編譯Qt 一般使用原始碼與目標目錄分開的方式，QtTargetDirectory 是Qt 編譯完成後的最終目錄，QtSourceDirectory 是Qt 的原始碼目錄。

當您編譯完成Qt 以後，不要進行任何清除的動作，將QtLzo 解壓縮以後，擺到QtSourceDirectory 目錄當中，並且更名為QtLzo。

Unix

```
cd QtLzo
qmake
make
make install
```

Windows

```
cd QtLzo
qmake
nmake
nmake install
```

或是

cd QtLzo
qmake
jom
nmake install

如此即可完成。

製作QtLzo 文件

切換到「QtLzo\doc\Qt」目錄當中：

cd QtLzo\doc\Qt
qhelpgenerator.exe QtLzo.qhp -o QtLzo.qch

新增文件到Qt Creator

打開Qt Assistant，「工具」→「選項」→「說明」→「文件」→「新增」，選取QtLzo.qch，按下完成即可新增。

新增文件到Qt Assistant

打開Qt Assistant，「編輯」→「喜好設定」→「文件」→「新增」，選取QtLzo.qch，按下完成即可新增。

Qt LZO 類別使用說明

目錄

QtLzo 類別

功能

處理數據的LZO 壓縮及解壓縮功能。

宣告

class QtLzo

```
{

  public:

    typedef enum       {

      LZO1x      =  1 ,

      LZO1       =  2 ,

      LZO1_99    =  3 ,

      LZO1a      =  4 ,

      LZO1a_99   =  5 ,

      LZO1b      =  6 ,

      LZO1b_99   =  7 ,

      LZO1b_999  =  8 ,

      LZO1c      =  9 ,

      LZO1c_99   = 10 ,

      LZO1c_999  = 11 ,

      LZO1f_1    = 12 ,

      LZO1f_999  = 13 ,

      LZO1x_1    = 14 ,

      LZO1x_1_11 = 15 ,

      LZO1x_1_12 = 16 ,

      LZO1x_1_15 = 17 ,

      LZO1x_999  = 18 ,

      LZO1y_1    = 19 ,

      LZO1y_999  = 20 ,

      LZO1z_999  = 21 ,

      LZO2a_999  = 22 }
```

```
    LzoMethods      ;

  explicit     QtLzo   (void) ;

  virtual     ~QtLzo   (void) ;

  virtual bool isLZO   (QByteArray & header) ;

  virtual void CleanUp (void) ;

  virtual bool IsCorrect          (int returnCode) ;

  virtual bool IsEnd              (int returnCode) ;

  virtual bool IsFault            (int returnCode) ;

  virtual int  BeginCompress      (int level = 9,int method = LZO1x) ;

  virtual int  BeginCompress      (QVariantList arguments = QVariantList() ) ;

  virtual int  doCompress         (const QByteArray & Source      ,
                                   QByteArray & Compressed) ;

  virtual int  doSection          (      QByteArray & Source      ,
                                   QByteArray & Compressed) ;

  virtual int  CompressDone       (QByteArray & Compressed) ;

  virtual int  BeginDecompress    (void) ;

  virtual int  doDecompress       (const QByteArray & Source        ,
                                   QByteArray & Decompressed) ;

  virtual int  undoSection        (      QByteArray & Source        ,
                                   QByteArray & Decompressed) ;

  virtual int  DecompressDone     (void) ;

  virtual bool IsTail             (QByteArray & header) ;

protected:

  void * LzoPacket ;

  virtual void CompressHeader     (QByteArray & Compressed) ;

  virtual void CompressTail       (unsigned int checksum,QByteArray &
Compressed) ;
```

```
    virtual int  DecompressHeader   (const QByteArray & Source) ;

  private:

};
```

說明

列舉型態

LzoMethods

名稱	值	作用
LZO1x	1	使用 LZO1 X 壓縮函式。
LZO1	2	使用 LZO1 通用壓縮函式。
LZO1_99	3	使用 LZO1-99 壓縮函式。
LZO1a	4	使用 LZO1a 通用壓縮函式。
LZO1a_99	5	使用 LZO1a-99 壓縮函式。
LZO1b	6	使用 LZO1b 通用壓縮函式。
LZO1b_99	7	使用 LZO1b-99 壓縮函式。
LZO1b_999	8	使用 LZO1b-999 壓縮函式。
LZO1c	9	使用 LZO1c 通用壓縮函式。
LZO1c_99	10	使用 LZO1c-99 壓縮函式。
LZO1c_999	11	使用 LZO1c-999 壓縮函式。
LZO1f_1	12	使用 LZO1f-1 壓縮函式。
LZO1f_999	13	使用 LZO1f-999 壓縮函式。
LZO1x_1	14	使用 LZO1x-1 壓縮函式。
LZO1x_1_11	15	使用 LZO1x-1-11 壓縮函式。
LZO1x_1_12	16	使用 LZO1x-1-12 壓縮函式。
LZO1x_1_15	17	使用 LZO1x-1-15 壓縮函式。
LZO1x_999	18	使用 LZO1x-999 壓縮函式。
LZO1y_1	19	使用 LZO1y-1 壓縮函式。

LZO1y_999	20	使用 LZO1y-999 壓縮函式。
LZO1z_999	21	使用 LZO1z-999 壓縮函式。
LZO2a_999	22	使用 LZO2a-999 壓縮函式。

```
bool isLZO(QByteArray & header)
```

判斷header 數據是否為LZO 的表頭。

LZO 的表頭為7 個位元組，如下：

0	1	2	3	4	5	6	7
0x00	0xE9	0x4C	0x5A	0x4F	0xFF	0x1A	

```
void CleanUp (void)
```

清除LzoPacket 以便重覆使用。如果您需要重覆使用QtLzo 類別內的函式，第二次使用時，最好呼叫一次CleanUp()。

```
bool IsCorrect(int returnCode)
```

判斷壓縮或解壓縮時期的返回碼是否為操作正確的返回碼。

```
bool IsEnd(int returnCode)
```

判斷操作時期的返回碼是否為壓縮或是解壓縮完成。

```
bool IsFault(int returnCode)
```

判斷操作時期的返回碼是否為發生嚴重錯誤的狀況。

```
int BeginCompress(int level,int method) ;
```

設定壓縮的參數。

1.level : 1~9，內定值為9。
2.method : lzoMethods，內定值為LZO1x。

```
int BeginCompress(QVariantList arguments)
```

設定壓縮的參數。

1.level : 1~9，內定值為9。
2.method : lzoMethods，內定值為LZO1x。
3.blocks : 每個區塊的大小，內定值為8192。

4.output length : 輸出長度，內定值為0。

```
int doCompress(const QByteArray & Source,QByteArray & Compressed)
```

將Source 壓縮到Compressed，成為LZO 壓縮數據格式。這個函式只需要使用一次，整個數據封包會被包裝成完整的LZO 檔案數據，主要使用於確定原始數據大小的狀況下使用。

返回值為操作碼。

```
int doSection(QByteArray & Source,QByteArray & Compressed)
```

將Source 壓縮到Compressed，成為LZO 壓縮數據格式。這個函式是將數據拆成小包的數據分解來壓縮，主要使用於不確定原始數據大小的狀況下使用，例如使用於網路即時傳輸用途時。

完成時需要呼叫CompressDone 函式。

這個函式會將Source 的數據由前面開始移除，移除的數據是已經完成壓縮的原始數據。

返回值為操作碼。

```
int CompressDone(QByteArray & Compressed)
```

使用doSection 的壓縮方式時，數據完成時，需要呼叫這個函式來寫入結尾。

```
int BeginDecompress(void)
```

準備開始解壓縮。

```
int doDecompress(const QByteArray & Source,QByteArray & Decompressed)
```

將壓縮的原始數據Source 解壓縮到Decompressed，Source 基本上是個完整的LZO 檔案。

返回值為操作碼。

```
int undoSection(QByteArray & Source,QByteArray & Decompressed)
```

將壓縮的原始數據Source 解壓縮到Decompressed，成為原始數據。這個函式是將數據拆成小包的數據分解來解壓縮，主要使用於不確定 原始數據大小的狀況下使用，例如使用於網路即時傳輸用途時。

完成時需要呼叫DecompressDone 函式。

這個函式會將Source 的數據由前面開始移除，移除的數據是已經完成解壓縮的LZO 數據。

返回值為操作碼。

```
int DecompressDone(void)
```

解壓縮完成時，需要呼叫這個函式。

```
bool IsTail(QByteArray & header)
```

判斷是否為LZO 檔案格式的結尾。

```
void CompressHeader(QByteArray & Compressed)
```

製作LZO 壓縮檔案格式表頭。

```
void CompressTail(unsigned int checksum,QByteArray & Compressed)
```

製作LZO 壓縮檔案格式結尾。checksum 是ADLER32。

```
int DecompressHeader(const QByteArray & Source)
```

檢查Source 是否為LZO 檔案表頭，並且返回數據開始的位置。

ScriptableLzo 類別

功能

Qt Script 的Javascript 當中，處理LZO 檔案的壓縮及解壓縮。

宣告

class ScriptableLzo : public QObject

```
                , public QScriptable
                , public QtLzo
{
  Q_OBJECT
  public:
    explicit     ScriptableLzo (QObject * parent) ;
```

```
    virtual     ~ScriptableLzo (void) ;

  protected:

  private:

  public slots:

    virtual bool ToLzo          (QString file,QString lzo,int level = 9,int
method = QtLzo::LZO1x) ;

    virtual bool ToFile         (QString lzo,QString file) ;

  protected slots:

  private slots:

  signals:

};
```

說明

```
bool ToLzo(QString file,QString lzo,int level,int method)
```

將檔案file 壓縮成lzo 檔，成功則返回true，失敗則返回false。

參數說明：

1.file : 原始檔案。
2.lzo : LZO 壓縮檔。
3.level : 1 ~ 9，內定值為9。
4.method : lzoMethods，內定值為LZO1x。

```
bool ToFile(QString lzo,QString file)
```

將lzo 檔案解壓縮成file，成功則返回true，失敗則返回false。

QtLzo 函數

Qt LZO 目前提供六個函數。

- bool ToLZO (const QByteArray & data,QByteArray & lzo,int level = 9,int method = QtLzo::LZO1x)
- bool FromLZO (const QByteArray & lzo ,QByteArray & data)
- bool SaveLZO (QString filename,QByteArray & data,int level = 9,int method = QtLzo::LZO1x)

- bool LoadLZO (QString filename,QByteArray & data)
- bool FileToLzo (QString file,QString lzo,int level = 9,int method = QtLzo::LZO1x)
- bool LzoToFile (QString lzo,QString file)

說明

bool ToLZO (const QByteArray & data,QByteArray & lzo,int level,int method)

將數據壓縮成LZO 壓縮格式，成功則返回true，失敗則返回false。

參數說明：

1.data：原始數據。
2.lzo：LZO 壓縮數據。
3.level：1 ~ 9，內定值為9。
4.method：lzoMethods，內定值為LZO1x。

bool FromLZO (const QByteArray & lzo ,QByteArray & data)

將LZO 數據解壓縮成原始數據，成功則返回true，失敗則返回false。

參數說明：

1.lzo：LZO 壓縮數據。
2.data：原始數據。

bool SaveLZO (QString filename,QByteArray & data,int level,int method)

將數據壓縮成LZO 壓縮格式，並儲存到檔案中，成功則返回true，失敗則返回false。

參數說明：

1.filename：LZO 檔案名稱。
2.data：原始數據。
3.level：1 ~ 9，內定值為9。
4.method：lzoMethods，內定值為LZO1x。

bool LoadLZO (QString filename,QByteArray & data)

將LZO 檔案解壓縮成原始數據，成功則返回true，失敗則返回false。

參數說明：

1.filename：LZO 檔案名稱。
2.data：原始數據。

bool FileToLzo (QString file,QString lzo,int level,int method)

將檔案file 壓縮成lzo 檔，成功則返回true，失敗則返回false。

參數說明：

1.file : 原始檔案。
2.lzo : LZO 壓縮檔。
3.level : 1 ~ 9，內定值為9。
4.method : lzoMethods，內定值為LZO1x。

bool LzoToFile (QString lzo,QString file)

將lzo 檔案解壓縮成file，成功則返回true，失敗則返回false。

QtLZO 使用範例

目錄

「QtLzo/examples/lzotool」目錄當中有使用範例程式。

僅使用記憶體的壓縮方式

僅使用記憶體的壓縮方式經常使用於網路傳輸的應用上。

完整的原始數據一次壓縮成LZO 格式

使用內定參數

使用函數的方式

```
QByteArray sourceData ;
QByteArray lzoData ;
```

```
// ...
// Load sourceData by your own code
// ...
if ( ToLZO ( sourceData , lzoData ) ) {
  // handling LZO compressed data
} else {
  // error when compress data into LZO format
}
```

使用QtLzo 物件的方式

```
QByteArray   sourceData                    ;
QByteArray   lzoData                       ;
QtLzo        L                             ;
int          r                             ;
QVariantList v                             ;
v << level                                 ;
v << method                                ;
v << blocks                                ;
v << outputLength                          ;
r = L . BeginCompress ( v )                ;
if ( L . IsCorrect ( r ) )                 {
  L . doCompress ( sourceData , lzoData ) ;
}                                          ;
```

未知長度的原始數據分解成多次壓縮成LZO 格式

```
QByteArray   sourceData                      ;
QByteArray   lzoData                         ;
QByteArray   Compressed                      ;
QtLzo        L                               ;
int          r                               ;
QVariantList v                               ;
v << level                                   ; // default 9
v << method                                  ; // default LZO1x
v << blocks                                  ; // default 8192
v << outputLength                            ; // default 0
r = L . BeginCompress ( v )                  ;
while ( MyOwnObtainData ( sourceData ) )     {
  r = L . doSection ( sourceData , lzoData ) ;
  if ( L . IsCorrect ( r ) )                 {
    if ( lzoData . size ( ) > 0 )            {
      Compressed . append ( lzoData )        ; // or handling the lzoData by your own funct
    }                                        ;
  }                                          ;
}                                            ;
L . CompressDone ( Compressed )              ;
```

僅使用記憶體的解壓方式

完整的壓縮數據一次解壓縮原始數據格式

使用函數的方式

```
QByteArray sourceData ;
QByteArray lzoData ;
// ...
// Load lzoData by your own code
// ...
if ( FromLZO ( lzoData , sourceData ) ) {
  // handling sourceData
} else {
  // error when decompress LZO data into sourceData
}
```

使用QtLzo 物件的方式

```
QByteArray sourceData                          ;
QByteArray lzoData                             ;
QtLzo      L                                   ;
int        r                                   ;
r = L . BeginDecompress ( )                    ;
if ( L . IsCorrect ( r ) )                     {
  L . doDecompress ( lzoData , sourceData ) ;
}                                              ;
```

未知長度的壓縮數據分解成多次解壓成原始數據格式

```
QByteArray sourceData                            ;
QByteArray lzoData                               ;
QByteArray Decompressed                          ;
QtLzo      L                                     ;
int        r                                     ;
r = L . BeginDecompress ( )                      ;
while ( MyOwnObtainLZO ( lzoData ) )             {
  r = L . undoSection ( lzoData , sourceData ) ;
  if ( L . IsCorrect ( r ) )                     {
    if ( sourceData . size ( ) > 0 )             {
      Decompressed . append ( sourceData )     ;
    }                                            ;
  }                                              ;
}                                                ;
```

將記憶體數據壓縮到檔案的方式

```
QByteArray sourceData ;
// ...
// Obtain sourceData by your own code
// ...
if ( SaveLZO ( "LzoFilename.lzo" , sourceData ) ) {
  // successful compress sourceData into LzoFilename.lzo file
} else {
  // error to compress into LZO file
}
```

將檔案解壓到記憶體數據的方式

```
QByteArray sourceData ;

if ( LoadLZO ( "LzoFilename.lzo" , sourceData ) ) {
  // successful decompress LzoFilename.lzo file into sourceData
} else {
  // error to decompress LZO file
}
```

將壓縮檔案解壓縮到另一個檔案的方式

```
if ( LzoToFile ( "LzoFilename.lzo" , "NormalFilename.dat" ) ) {
  // successful decompress LzoFilename.lzo file into NormalFilename.dat
} else {
  // error to decompress LZO file
}
```

將一般檔案壓縮到LZO 檔案

```
if ( FileToLzo ( "NormalFilename.dat" , "LzoFilename.lzo" ) ) {
  // successful compress NormalFilename.dat into LzoFilename.lzo
} else {
  // error to compress into LZO file
}
```

使用Javascript（Qt Script）的方式

使用Qt Script 呼叫ScriptableLzo

```
```

LZO.js

```
function CompressLZO()

{
  lzo = new ScriptableLzo() ;
  lzo . ToLzo ( "Testing.txt" , "Testing.lzo" , 9 , 1 ) ;
  delete lzo ;
}

function DecompressLZO()
{
  lzo = new ScriptableLzo() ;
  lzo . ToFile ( "Testing.lzo" , "Testing.txt" ) ;
  delete lzo ;
}
```

Qt LZO 其他需要注意的事項

目錄

修改所支援的Qt 版本

目前的QtLzo 支援Qt 5.5.0，如果您使用的Qt 版本有所更新，您需要修改以下檔案：

QtLzo/.qmake.conf

```
load(qt_build_config)

MODULE_VERSION = 5.5.0

# change version into current Qt version every time your upgrade your Qt

CIOS_VERSION = 1.6.0
```

將 MODULE_VERSION 修改成例如 5.5.1 即可。