

# **PlayStation®Edge LZO Library Reference**

# Table of Contents

<b>Preface.....</b>	<b>3</b>
About This Document.....	4
<b>Shared PPU/SPU Runtime Data Types.....</b>	<b>5</b>
EdgeLzo1xDeflateQueueElement.....	6
EdgeLzo1xInflateQueueElement .....	8
EdgeLzo1xDeflateQHandle.....	10
EdgeLzo1xInflateQHandle .....	11
EdgeLzo1xDeflateTaskProcessing.....	12
EdgeLzo1xInflateTaskProcessing .....	13
EdgeLzo1xDeflateTaskProcessingStored .....	14
<b>Shared PPU/SPU Functions.....</b>	<b>15</b>
edgeLzo1xAddDeflateQueueElement.....	16
edgeLzo1xTryAddDeflateQueueElement.....	18
edgeLzo1xAddInflateQueueElement .....	20
edgeLzo1xTryAddInflateQueueElement .....	22
edgeLzo1xAddInflateQueueElementPartialCopyOut.....	24
edgeLzo1xTryAddInflateQueueElementPartialCopyOut.....	26
<b>PPU Functions .....</b>	<b>28</b>
edgeLzo1xGetDeflateQueueSize.....	29
edgeLzo1xGetInflateQueueSize .....	30
edgeLzo1xCreateDeflateQueue.....	31
edgeLzo1xCreateInflateQueue .....	32
edgeLzo1xShutdownDeflateQueue .....	33
edgeLzo1xShutdownInflateQueue .....	34
edgeLzo1xGetDeflateTaskContextSaveSize .....	35
edgeLzo1xGetInflateTaskContextSaveSize .....	36
edgeLzo1xCreateDeflateTask .....	37
edgeLzo1xCreateInflateTask.....	38
<b>SPU Functions .....</b>	<b>39</b>
edgeLzo1xDeflateRawData .....	40
edgeLzo1xInflateRawData .....	41

# Preface

---

# About This Document

---

## Purpose

This document provides an API reference for the Edge LZO component of the Edge library. Use this component to perform lossless data decompression and compression on SPU.

## Audience and Prerequisites

This document was written for PlayStation®3 developers who want to write high-performance applications for the PlayStation®3. It is assumed that such developers have familiarity with the following:

- C and C++
- PlayStation®3 hardware
- SCE standard library functions

## Related Documentation

In combination with this reference, the following documents provide complete usage and reference information about the Edge library:

- *PlayStation®Edge Library Overview*
- *PlayStation®Edge Geometry Library: Quick Start*
- *PlayStation®Edge Geometry Library Reference*
- *PlayStation®Edge Geometry Library for Offline Tool: Reference*
- *PlayStation®Edge Animation Library Reference*
- *PlayStation®Edge Animation Library for Offline Tool: Reference*
- *PlayStation®Edge Zlib Library Reference*
- *PlayStation®Edge LZMA Library Reference*
- *PlayStation®Edge DXT Library Reference*
- *PlayStation®Edge Post Library Reference*

## Typographic Conventions

This document uses the following typographic conventions:

Convention	Meaning
<code>fixed-width font</code>	Indicates programming code and literals, such as processing instructions, register names, data types, events, and file names. Also indicates function, structure, and macro names.
<b><code>fixed-width font + bold</code></b>	In structure/function definitions only, indicates names of structures and functions.
<i>fixed-width font + italics</i>	Indicates arguments, parameters, and variables.
<a href="#">blue + underlined text</a>	Indicates a hyperlink (blue displays in color printers or online only).

# Shared PPU/SPU Runtime Data Types

# EdgeLzo1xDeflateQueueElement

An entry in the queue that tells the Deflate Task to compress a single segment of data.

## Definition

```
#include <edge/lzo/edgelzo1x_deflate_queue_element.h>
typedef struct EdgeLzo1xDeflateQueueElement
{
    uint32_t m_eaInputUncompressedData;
    uint32_t m_eaOutputCompressedData;
    uint32_t m_uncompressedSize;
    uint32_t m_maxCompressedOutputSize;
    uint32_t m_eaOutputCompressedSize;
    uint32_t m_eaWorkToDoCounter;
    uint32_t m_eaEventFlag;
    uint16_t m_eventFlagBits;
    uint16_t m_pad16;
} EdgeLzo1xDeflateQueueElement __attribute__((aligned(16)));
```

## Members

<i>m_eaInputUncompressedData</i>	Effective address of input uncompressed data. Warning: low 16 bits of address affect contents of returned compressed data. This is inherent in the LZO1x compression.
<i>m_eaOutputCompressedData</i>	Effective address where compressed data will be placed.
<i>m_uncompressedSize</i>	Size of uncompressed data.
<i>m_maxCompressedOutputSize</i>	Maximum size of output buffer for compressed data.
<i>m_eaOutputCompressedSize</i>	Effective address where SPU will return the size of the outputted data with the high bit set if the data was compressed.
<i>m_eaWorkToDoCounter</i>	Effective address of counter to atomically decrement when compression is done. May be NULL.
<i>m_eaEventFlag</i>	The least significant bit gives a flag of type <a href="#">EdgeLzo1xDeflateTaskProcessing</a> .
<i>m_eventFlagBits</i>	Effective address of the event flag. May be NULL.
<i>m_pad16</i>	The event flag will be set to this value after decompression is completed. Ignored.

## Description

This structure contains information about one segment of data. Each entry will cause a Deflate Task to compress this memory.

## Notes

The address of the compressed and uncompressed data can have any alignment.

The input uncompressed data must be no more than 64 KB.

The output compressed data will be no more than  $m\_uncompressedSize + (m\_uncompressedSize/16) + 64 + 3$ .

Warning: The low 16 bits of the address of the input uncompressed data is used by Lzo1x in deciding how to encode the data, and changing these low 16 bits will thus result in differently encoded (but completely correct) compressed data. Therefore, compressing the same uncompressed file may

produce different compressed data at different times, but both compressed data sets will decompress to the same uncompressed data.

#### **See Also**

---

[EdgeLzo1xDeflateQueueElement](#), [edgeLzo1xCreateDeflateQueue](#)

# EdgeLzo1xInflateQueueElement

An entry in the queue that tells the Inflate Task to decompress or move a single segment of data.

## Definition

```
#include <edge/lzo/edgelzo1x_inflate_queue_element.h>
typedef struct EdgeLzo1xInflateQueueElement
{
    uint32_t m_eaCompressed;
    uint32_t m_eaUncompressed;
    uint32_t m_compressedSize;
    uint32_t m_outputUncompPartialBuffSize;
    uint32_t m_eaWorkToDoCounter;
    uint32_t m_eaEventFlag;
    uint16_t m_eventFlagBits;
    uint16_t m_outputUncompSkipBeginSize;
    uint16_t m_outputUncompSkipEndSize;
    uint16_t m_pad16;
} EdgeLzo1xInflateQueueElement __attribute__((aligned(16)));
```

## Members

<i>m_eaCompressed</i>	Effective address of compressed data.
<i>m_eaUncompressed</i>	Effective address where uncompressed data will be placed.
<i>m_compressedSize</i>	Size of compressed data.
<i>m_outputUncompPartialBuffSize</i>	Size of the uncompressed data to be DMAed out.
<i>m_eaWorkToDoCounter</i>	Effective address of counter to atomically decrement when decompression done. May be NULL.
<i>m_eaEventFlag</i>	Least significant bit is a flag that indicates if the segment was stored compressed (1) or uncompressed (0). Effective address of the event flag. May be NULL.
<i>m_eventFlagBits</i>	The event flag will be set to this value after decompression is completed.
<i>m_outputUncompSkipBeginSize</i>	Do not output the first N bytes of the uncompressed output.
<i>m_outputUncompSkipEndSize</i>	Do not output the last N bytes of the uncompressed output.
<i>m_pad16</i>	Ignored.

## Description

This structure contains information about one segment of data. Each entry will cause an Inflate Task to decompress or move this memory.

## Notes

The address of the compressed and uncompressed data can have any alignment.  
 The compressed data must be no more than 64 KB.  
 The uncompressed data must be no more than 64 KB.  
 The sum of *m\_outputUncompSkipBeginSize*, *m\_outputUncompPartialBuffSize*, and *m\_outputUncompSkipEndSize* gives the expected size that the compressed data will decompress to.  
 If this value is incorrect, the SPU code will assert.



**See Also**

---

[edgeLzo1xAddInflateQueueElement](#), [edgeLzo1xAddInflateQueueElementPartialCopyOut](#),  
[edgeLzo1xCreateInflateQueue](#)

# EdgeLzo1xDeflateQHandle

---

---

A handle for the Deflate Queue.

## Definition on PPU

---

```
#include <edge/lzo/edgelzo_ppu.h>
typedef void* EdgeLzo1xDeflateQHandle;
```

## Definition on SPU

---

```
#include <edge/lzo/edgelzo_spu.h>
typedef uint32_t EdgeLzo1xDeflateQHandle;
```

## Description

---

This is a handle for the Deflate Queue in main memory.

## See Also

---

[EdgeLzo1xDeflateQueueElement](#), [edgeLzo1xCreateDeflateQueue](#)

# EdgeLzo1xInflateQHandle

---

---

A handle for the Inflate Queue.

## Definition on PPU

---

```
#include <edge/lzo/edgelzo_ppu.h>
typedef void* EdgeLzo1xInflateQHandle;
```

## Definition on SPU

---

```
#include <edge/lzo/edgelzo_spu.h>
typedef uint32_t EdgeLzo1xInflateQHandle;
```

## Description

---

This is a handle for the Inflate Queue in main memory.

## See Also

---

[EdgeLzo1xInflateQueueElement](#), [edgeLzo1xCreateInflateQueue](#)

---

# EdgeLzo1xDeflateTaskProcessing

---

Specifies whether a segment is to be stored compressed, or if the smaller of the compressed and uncompressed data is to be stored.

## Definition

---

```
#include <edge/lzo/edgelzo1x_deflate_queue_element.h>

typedef enum EdgeLzo1xDeflateTaskProcessing
{
    kEdgeLzo1xDeflateTask_DeflateStoreCompressed = 0,
    kEdgeLzo1xDeflateTask_DeflateStoreSmallest = 1,
} EdgeLzo1xDeflateTaskProcessing;
```

## Members

---

<i>kEdgeLzo1xDeflateTask_DeflateStoreCompressed</i>	Always store compressed data.
<i>kEdgeLzo1xDeflateTask_DeflateStoreSmallest</i>	Store either compressed or uncompressed data, whichever is smaller.

## Description

---

In the case where compression does not reduce the size of the data, this is used to declare whether to store the compressed data anyway (even though it may be bigger), or whether to store whichever is the smaller.

## See Also

---

[edgeLzo1xAddDeflateQueueElement](#)

---

# EdgeLzo1xInflateTaskProcessing

---

Specifies whether a segment is copied or decompressed.

## Definition

---

```
#include <edge/lzo/edgelzo1x_inflate_queue_element.h>

typedef enum EdgeLzo1xInflateTaskProcessing
{
    kEdgeLzo1xInflateTask_Memcpy = 0,
    kEdgeLzo1xInflateTask_Inflate = 1,
} EdgeLzo1xInflateTaskProcessing;
```

## Members

---

<i>kEdgeLzo1xInflateTask_Memcpy</i>	Copy memory from one location to another.
<i>kEdgeLzo1xInflateTask_Inflate</i>	Decompress memory from one location to another.

## Description

---

This is used to declare whether the segment is to be copied or decompressed by the Inflate Task.

## See Also

---

[edgeLzo1xAddInflateQueueElement](#), [edgeLzo1xAddInflateQueueElementPartialCopyOut](#)

---

# EdgeLzo1xDeflateTaskProcessingStored

---

The Deflate Task specifies whether the output data was stored compressed or uncompressed.

## Definition

---

```
#include <edge/lzo/edgelzo1x_deflate_queue_element.h>

typedef enum EdgeLzo1xDeflateTaskProcessingStored
{
    kEdgeLzo1xDeflateTask_UncompressedWasStored = 0x00000000,
    kEdgeLzo1xDeflateTask_CompressedWasStored = 0x80000000,
} EdgeLzo1xDeflateTaskProcessingStored;
```

## Members

---

<i>kEdgeLzo1xDeflateTask_UncompressedWasStored</i>	Uncompressed data was stored.
<i>kEdgeLzo1xDeflateTask_CompressedWasStored</i>	Compressed data was stored.

## Description

---

When the SPU writes out the output size, the high bit is used to declare whether the data was stored compressed or uncompressed.

## See Also

---

[edgeLzo1xAddDeflateQueueElement](#)

# Shared PPU/SPU Functions

# edgeLzo1xAddDeflateQueueElement

Adds a new piece of work to the Deflate Queue.

## Definition on PPU

```
#include <edge/lzo/edgelzo_ppu.h>
void edgeLzo1xAddDeflateQueueElement
(
    EdgeLzo1xDeflateQHandle handle,
    const void* eaInputUncompressedData,
    uint32_t uncompressedSize,
    void* eaOutputCompressedData,
    uint32_t maxCompressedOutputSize,
    uint32_t* eaOutputSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xDeflateTaskProcessing processing
)
```

## Definition on SPU

```
#include <edge/lzo/edgelzo_spu.h>
void edgeLzo1xAddDeflateQueueElement
(
    EdgeLzo1xDeflateQHandle handle,
    uint32_t eaInputUncompressedData,
    uint32_t uncompressedSize,
    uint32_t eaOutputCompressedData,
    uint32_t maxCompressedOutputSize,
    uint32_t eaOutputSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xDeflateTaskProcessing processing,
    uint32_t dmaTag
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>handle</i>	The handle of the Deflate Queue that the entry will be pushed onto.
<i>eaInputUncompressedData</i>	The Effective Address of the input data that is to be compressed. Warning: low 16 bits of address affect contents of returned compressed data. This is inherent in the Lzo1x compression.
<i>uncompressedSize</i>	The size of the uncompressed input data.
<i>eaOutputCompressedData</i>	The Effective Address of the output buffer for the compressed data.
<i>maxCompressedOutputSize</i>	The maximum space available for the compressed data buffer.



---

<i>eaOutputSize</i>	The Effective Address of the <code>uint32_t</code> into which the output compressed size will be written. The top bit of the written counter indicates if the data was stored compressed or if the uncompressed original data was chosen for storing.
<i>eaWorkToDoCounter</i>	A counter in main memory that will be atomically decremented after this item has been decompressed, and if the SPU has an error with the compressed data it will set the high bit to alert the fact that an error occurred.
<i>eaEventFlag</i>	Can be NULL. The event flag to set.
<i>eventFlagBits</i>	Can be NULL. The event flag will be set to this value after compression is completed.
<i>processing</i>	Choose whether to always store the compressed data, or whether to store whichever is the smaller of the compressed and uncompressed data.
<i>dmaTag</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### Return Values

---

None.

### Description

---

Adds a new piece of work to the Deflate Queue. The work added to this queue will be taken by one of the Deflate Tasks at the next opportunity.

Warning: The low 16 bits of the address of the input uncompressed data is used by Lzo1x in deciding how to encode the data, and changing these low 16 bits will thus result in differently encoded (but completely correct) compressed data. Therefore, compressing the same uncompressed file may produce different compressed data at different times, but both compressed data sets will decompress to the same uncompressed data.

### Notes

---

If the queue is full (that is, if it has reached *maxNumQueueEntries*), and another item is pushed onto the queue, then this function will block until there is space.

The work-to-do-counter can be used to track completion of the processing on a collection of segments by initializing it to the number of segments being waited on. Then when each item completes, it will decrement by one, and so when the value reaches zero all segments will have been done. At this point the specified event flag will be set. Note that if the SPU has an error with compressed data, it will set the high bit of the counter.

If *eaWorkToDoCounter* is NULL but *eaEventFlag* is valid, the event flag will be set after this segment is done.

### See Also

---

[EdgeLzo1xDeflateQueueElement](#), [edgeLzo1xCreateDeflateTask](#), [edgeLzo1xTryAddDeflateQueueElement](#)

# edgeLzo1xTryAddDeflateQueueElement

Tries to add a new piece of work to the Deflate Queue.

## Definition on PPU

```
#include <edge/lzo/edgelzo_ppu.h>
bool edgeLzo1xTryAddDeflateQueueElement
(
    EdgeLzo1xDeflateQHandle handle,
    const void* eaInputUncompressedData,
    uint32_t uncompressedSize,
    void* eaOutputCompressedData,
    uint32_t maxCompressedOutputSize,
    uint32_t* eaOutputCompressedSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xDeflateTaskProcessing processing
)
```

## Definition on SPU

```
#include <edge/lzo/edgelzo_spu.h>
bool edgeLzo1xTryAddDeflateQueueElement
(
    EdgeLzo1xDeflateQHandle handle,
    uint32_t eaInputUncompressedData,
    uint32_t uncompressedSize,
    uint32_t eaOutputCompressedData,
    uint32_t maxCompressedOutputSize,
    uint32_t eaOutputCompressedSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xDeflateTaskProcessing processing,
    uint32_t dmaTag
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>handle</i>	The handle of the Deflate Queue that the entry will be pushed onto.
<i>eaInputUncompressedData</i>	The Effective Address of the input data that is to be compressed. Warning: low 16 bits of address affect contents of returned compressed data. This is inherent in the Lzo1x compression.
<i>uncompressedSize</i>	The size of the uncompressed input data.
<i>eaOutputCompressedData</i>	The Effective Address of the output buffer for the compressed data.
<i>maxCompressedOutputSize</i>	The maximum space available for the compressed data.

---

<i>eaOutputCompressedSize</i>	The Effective Address of the <code>uint32_t</code> into which the output compressed size will be written.
<i>eaWorkToDoCounter</i>	The top bit of the written counter indicates if the data was stored compressed or if the uncompressed data was chosen for storing. A counter in main memory that will be atomically decremented after this item has been decompressed, and if the SPU has an error with the compressed data it will set the high bit to alert the PPU. Can be NULL.
<i>eaEventFlag</i>	The event flag to set. Can be NULL.
<i>eventFlagBits</i>	The event flag will be set to this value after compression is completed.
<i>processing</i>	Choose whether to always store the compressed data, or whether to store whichever is the smaller of the compressed and uncompressed data.
<i>dmaTag</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### Return Values

---

`true` if element is added to the deflate queue; `false` if it could not be added at this time.

### Description

---

Tries to add a new piece of work to the Deflate Queue. If added, the work added to this queue will be taken by one of the Deflate Tasks at the next opportunity.

Warning: The low 16 bits of the address of the input uncompressed data is used by Lzo1x in deciding how to encode the data, and changing these low 16 bits will thus result in differently encoded (but completely correct) compressed data. Therefore, compressing the same uncompressed file may produce different compressed data at different times, but both compressed data sets will decompress to the same uncompressed data.

### Notes

---

If the queue is full (that is, if it has reached `maxNumQueueEntries`), and another item is pushed onto the queue, then this function will return `false` immediately.

The work-to-do-counter can be used to track completion of the processing on a collection of segments by initializing it to the number of segments being waited on. Then when each item completes, it will decrement by one, and so when the value reaches zero all segments will have been done. At this point the specified event flag will be set. Note that if the SPU has an error with compressed data, it will set the high bit of the counter.

If *eaWorkToDoCounter* is NULL but *eaEventFlag* is valid, the event flag will be set after this segment is done.

### See Also

---

[EdgeLzo1xDeflateQueueElement](#), [edgeLzo1xCreateDeflateTask](#), [edgeLzo1xAddDeflateQueueElement](#)

# edgeLzo1xAddInflateQueueElement

Adds a new piece of work to the Inflate Queue.

## Definition on PPU

```
#include <edge/lzo/edgelzo_ppu.h>
void edgeLzo1xAddInflateQueueElement
(
    EdgeLzo1xInflateQHandle handle,
    const void* eaInputCompressedData,
    uint32_t compressedSize,
    void* eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing
)
```

## Definition on SPU

```
#include <edge/lzo/edgelzo_spu.h>
void edgeLzo1xAddInflateQueueElement
(
    EdgeLzo1xInflateQHandle handle,
    uint32_t eaInputCompressedData,
    uint32_t compressedSize,
    uint32_t eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing,
    uint32_t dmaTag
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>handle</i>	The handle of the Inflate Queue that the entry will be pushed onto.
<i>eaInputCompressedData</i>	The Effective Address of the input data that is to be decompressed. This should be a pointer to the raw data without any header.
<i>compressedSize</i>	The size of the compressed input data.
<i>eaOutputUncompressed</i>	The Effective Address of the output buffer for the uncompressed data.
<i>expectedUncompressedSize</i>	The expected size of the uncompressed data. The SPU will assert if this value is incorrect.
<i>eaWorkToDoCounter</i>	A counter in main memory that will be atomically decremented after this item has been decompressed. Can be NULL.

---

<i>eaEventFlag</i>	The event flag to set. Can be NULL.
<i>eventFlagBits</i>	The event flag will be set to this value after decompression is completed.
<i>processing</i>	Choose what kind of processing the task has to do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTag</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### Return Values

---

None.

### Description

---

Adds a new piece of work to the Inflate Queue. The work added to this queue will be taken by one of the Inflate Tasks at the next opportunity.

### Notes

---

If the queue is full (that is, if it has reached *maxNumQueueEntries*), and another item is pushed onto the queue, then this function will block until there is space.

The work-to-do-counter can be used to track completion of the processing on a collection of segments by initializing it to the number of segments being waited on. Then when each item completes it will decrement by one, and so when the value reaches zero all segments will have been done. At this point the specified event flag will be set. Note that if the SPU has an error with compressed data, it will set the high bit of the counter.

If *eaWorkToDoCounter* is NULL but *eaEventFlag* is valid, the event flag will be set after this segment is done.

If the data was stored uncompressed and merely needs to be moved to its destination, the Inflate Task can be used for this by passing the appropriate value for performing only a memory copy.

This function expects a pointer to the raw compressed data without any header.

### See Also

---

[EdgeLzo1xInflateQueueElement](#), [edgeLzo1xAddInflateQueueElementPartialCopyOut](#), [edgeLzo1xCreateInflateTask](#)

# edgeLzo1xTryAddInflateQueueElement

Tries to add a new piece of work to the Inflate Queue.

## Definition on PPU

```
#include <edge/lzo/edgelzo_ppu.h>
bool edgeLzo1xTryAddInflateQueueElement
(
    EdgeLzo1xInflateQHandle handle,
    const void* eaInputCompressedData,
    uint32_t compressedSize,
    void* eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing
)
```

## Definition on SPU

```
#include <edge/lzo/edgelzo_spu.h>
bool edgeLzo1xTryAddInflateQueueElement
(
    EdgeLzo1xInflateQHandle handle,
    uint32_t eaInputCompressedData,
    uint32_t compressedSize,
    uint32_t eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing,
    uint32_t dmaTag
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>handle</i>	The handle of the Inflate Queue that the entry will be pushed onto.
<i>eaInputCompressedData</i>	The Effective Address of the input data that is to be decompressed. This should be a pointer to the raw data without any header.
<i>compressedSize</i>	The size of the compressed input data.
<i>eaOutputUncompressed</i>	The Effective Address of the output buffer for the uncompressed data.
<i>expectedUncompressedSize</i>	The expected size of the uncompressed data. The SPU will assert if this value is incorrect.

---

<i>eaWorkToDoCounter</i>	A counter in main memory that will be atomically decremented after this item has been decompressed. If the SPU has an error with the compressed data it will set the high bit of the counter to alert the fact that an error occurred. Can be NULL.
<i>eaEventFlag</i>	The event flag to set. Can be NULL.
<i>eventFlagBits</i>	The value to set to the event flag.
<i>processing</i>	Choose what kind of processing the task has to do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTag</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### Return Values

---

`true` if the item was successfully added to the queue; `false` if it failed to be added.

### Description

---

Tries to add a new piece of work to the Inflate Queue. The work added to this queue will be taken by one of the Inflate Tasks at the next opportunity.

### Notes

---

If the item is added, this function will return `true`. If the queue is full (that is, if *maxNumQueueEntries* has been reached), and another item is pushed onto the queue, this function will return `false`.

The work-to-do-counter can be used to track completion of the processing on a collection of segments by initializing it to the number of segments being waited on. Then when each item completes it will decrement by one, and so when the value reaches zero all segments will have been done. At this point the specified event flag will be set. Note that if the SPU has an error with the compressed data, it will set the high bit of the counter.

If *eaWorkToDoCounter* is NULL but *eaEventFlag* is valid, the event flag will be set after this segment is done.

If the data was stored uncompressed and merely needs moving to its destination, the Inflate Task can be used for this by passing the appropriate value for *processing*.

This function expects a pointer to the raw compressed data without any header.

### See Also

---

[EdgeLzo1xInflateQueueElement](#), [edgeLzo1xAddInflateQueueElementPartialCopyOut](#), [edgeLzo1xCreateInflateTask](#)

# edgeLzo1xAddInflateQueueElementPartialCopyOut

Adds a new piece of work to the Inflate Queue.

## Definition on PPU

```
#include <edge/lzo/edgelzo_ppu.h>
void edgeLzo1xAddInflateQueueElementPartialCopyOut
(
    EdgeLzo1xInflateQHandle handle,
    const void* eaInputCompressedData,
    uint32_t compressedSize,
    void* eaOutputUncompPartialBuff,
    uint16_t outputUncompSkipBeginSize,
    uint32_t outputUncompPartialBuffSize,
    uint16_t outputUncompSkipEndSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing
)
```

## Definition on SPU

```
#include <edge/lzo/edgelzo_spu.h>
void edgeLzo1xAddInflateQueueElementPartialCopyOut
(
    EdgeLzo1xInflateQHandle handle,
    uint32_t eaInputCompressedData,
    uint32_t compressedSize,
    uint32_t eaOutputUncompPartialBuff,
    uint16_t outputUncompSkipBeginSize,
    uint32_t outputUncompPartialBuffSize,
    uint16_t outputUncompSkipEndSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing,
    uint32_t dmaTag
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>handle</i>	The handle of the Inflate Queue that the entry will be pushed onto.
<i>eaInputCompressedData</i>	The Effective Address of the input data that is to be decompressed.
<i>compressedSize</i>	This should be a pointer to the raw data without any header.
<i>eaOutputUncompPartialBuff</i>	The size of the compressed input data.
	The Effective Address of the output buffer for the uncompressed data.



---

<i>outputUncompSkipBeginSize</i>	Do not output the first N bytes of the uncompressed output.
<i>outputUncompPartialBuffSize</i>	The size of the uncompressed data that will be DMAed out.
<i>outputUncompSkipEndSize</i>	Do not output the last N bytes of the uncompressed output.
<i>eaWorkToDoCounter</i>	A counter in main memory that will be atomically decremented after this item has been decompressed. Can be NULL.
<i>eaEventFlag</i>	The event flag to set. Can be NULL.
<i>eventFlagBits</i>	The value to set to the event flag
<i>processing</i>	Choose what kind of processing the task has to do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTag</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### Return Values

---

None.

### Description

---

Adds a new piece of work to the Inflate Queue. The work added to this queue will be taken by one of the Inflate Tasks at the next opportunity.

### Notes

---

If the queue is full (that is, if it has reached *maxNumQueueEntries*), and another item is pushed onto the queue, this function will block until there is space.

The work-to-do-counter can be used to track completion of the processing on a collection of segments by initializing it to the number of segments being waited on. Then when each item completes it will decrement by one, and so when the value reaches zero all segments will have been done. At this point the specified event flag will be set. Note that if the SPU has an error with the compressed data, it will set the high bit of the counter.

If *eaWorkToDoCounter* is NULL but *eaEventFlag* is valid, the event flag will be set after this segment is done.

If the data was stored uncompressed and merely needs to be moved to its destination, the Inflate Task can be used for this by passing the appropriate value for *processing*.

This function expects a pointer to the raw compressed data without any header.

This function is very similar to [edgeLzo1xAddInflateQueueElement\(\)](#). The additional parameters *skipOutputBeginSize* and *skipOutputEndSize* exist so that decompression of a segment can be done, but only a portion of the output may need to be written.

The sum of the *outputUncompSkipBeginSize*, *outputUncompPartialBuffSize*, and *outputUncompSkipEndSize* gives the expected uncompressed size of the compressed data. The SPU will assert if this value is incorrect.

### See Also

---

[EdgeLzo1xInflateQueueElement](#), [edgeLzo1xAddInflateQueueElement](#), [edgeLzo1xCreateInflateTask](#)

# edgeLzo1xTryAddInflateQueueElementPartialCopyOut

Tries to add a new piece of work to the Inflate Queue.

## Definition on PPU

```
#include <edge/lzo/edgelzo_ppu.h>
bool edgeLzo1xTryAddInflateQueueElementPartialCopyOut
(
    EdgeLzo1xInflateQHandle handle,
    const void* eaInputCompressedData,
    uint32_t compressedSize,
    void* eaOutputUncompPartialBuff,
    uint16_t outputUncompSkipBeginSize,
    uint32_t outputUncompPartialBuffSize,
    uint16_t outputUncompSkipEndSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing
)
```

## Definition on SPU

```
#include <edge/lzo/edgelzo_spu.h>
bool edgeLzo1xTryAddInflateQueueElementPartialCopyOut
(
    EdgeLzo1xInflateQHandle handle,
    uint32_t eaInputCompressedData,
    uint32_t compressedSize,
    uint32_t eaOutputUncompPartialBuff,
    uint16_t outputUncompSkipBeginSize,
    uint32_t outputUncompPartialBuffSize,
    uint16_t outputUncompSkipEndSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzo1xInflateTaskProcessing processing,
    uint32_t dmaTag
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>handle</i>	The handle of the Inflate Queue that the entry will be pushed onto.
<i>eaInputCompressedData</i>	The Effective Address of the input data that is to be decompressed.
<i>compressedSize</i>	This should be a pointer to the raw data without any header. The size of the compressed input data.

---

<i>eaOutputUncompPartialBuff</i>	The Effective Address of the output buffer for the uncompressed data.
<i>outputUncompSkipBeginSize</i>	Do not output the first N bytes of the uncompressed output.
<i>outputUncompPartialBuffSize</i>	The size of the uncompressed data that will be DMAed out.
<i>outputUncompSkipEndSize</i>	Do not output the last N bytes of the uncompressed output.
<i>eaWorkToDoCounter</i>	A counter in main memory that will be atomically decremented after this item has been decompressed. If the SPU has an error with the compressed data it will set the high bit of the counter as an error flag. Can be NULL.
<i>eaEventFlag</i>	The event flag to set. Can be NULL.
<i>eventFlagBits</i>	The value to set to the event flag.
<i>processing</i>	Choose what kind of processing the task has to do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTag</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### Return Values

---

`true` if the item was successfully added to the queue; `false` if it failed to be added.

### Description

---

Tries to add a new piece of work to the Inflate Queue. The work added to this queue will be taken by one of the Inflate Tasks at the next opportunity.

### Notes

---

If the item is added, this function will return `true`. If the queue is full (that is, if *maxNumQueueEntries* has been reached), and another item is pushed onto the queue, this function will return `false`.

The work-to-do-counter can be used to track completion of the processing on a collection of segments by initializing it to the number of segments being waited on. Then when each item completes it will decrement by one, and so when the value reaches zero all segments will have been done. At this point the specified event flag will be set. Note that if the SPU has an error with the compressed data, it will set the high bit of the counter.

If *eaWorkToDoCounter* is NULL but *eaEventFlag* is valid, the event flag will be set after this segment is done.

If the data was stored uncompressed and merely needs to be moved to its destination, the Inflate Task can be used for this by passing the appropriate value for *processing*.

This function expects a pointer to the raw compressed data without any header.

This function is very similar to [edgeLzo1xAddInflateQueueElement\(\)](#). The additional parameters *skipOutputBeginSize* and *skipOutputEndSize* exist so that decompression of a segment can be done, but only a portion of the output may need to be written.

The sum of the *outputUncompSkipBeginSize*, *outputUncompPartialBuffSize*, and *outputUncompSkipEndSize* gives the expected uncompressed size of the compressed data. The SPU will assert if this value is incorrect.

### See Also

---

[EdgeLzo1xInflateQueueElement](#), [edgeLzo1xAddInflateQueueElement](#), [edgeLzo1xCreateInflateTask](#)

# PPU Functions

---

# edgeLzo1xGetDeflateQueueSize

---

Returns the required buffer size needed for a Deflate Queue.

## Definition

---

```
#include <edge/lzo/edgelzo_ppu.h>
uint32_t edgeLzo1xGetDeflateQueueSize
(
    uint32_t maxNumQueueEntries
)
```

## Calling Conditions

---

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

---

*maxNumQueueEntries* The maximum number of entries the queue will hold at one time (maximum: 32767).

## Return Values

---

Returns the required buffer size needed.

## Description

---

Returns the required buffer size needed by a Deflate Queue to hold the specified number of elements.

## Notes

---

The allocated buffer must be 128-byte aligned.

## See Also

---

[EdgeLzo1xDeflateQueueElement](#), [edgeLzo1xCreateDeflateQueue](#), [edgeLzo1xCreateDeflateTask](#)

---

# edgeLzo1xGetInflateQueueSize

---

Returns the required buffer size needed for an Inflate Queue.

## Definition

---

```
#include <edge/lzo/edgelzo_ppu.h>
uint32_t edgeLzo1xGetInflateQueueSize
(
    uint32_t maxNumQueueEntries
)
```

## Calling Conditions

---

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

---

*maxNumQueueEntries* The maximum number of entries the queue will hold at one time (maximum: 32767).

## Return Values

---

Returns the required buffer size needed.

## Description

---

Returns the required buffer size needed by an Inflate Queue to hold the specified number of elements.

## Notes

---

The allocated buffer must be 128-byte aligned.

## See Also

---

[EdgeLzo1xInflateQueueElement](#), [edgeLzo1xCreateInflateQueue](#), [edgeLzo1xCreateInflateTask](#)

# edgeLzo1xCreateDeflateQueue

Creates a Deflate Queue.

## Definition

```
#include <edge/lzo/edgelzo_ppu.h>
EdgeLzo1xDeflateQHandle edgeLzo1xCreateDeflateQueue
(
    CellSpurs* pSpurs,
    uint32_t maxNumQueueEntries,
    void* pBuffer,
    uint32_t bufferSize
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>pSpurs</i>	The SPURS instance that this Deflate Queue will be associated with.
<i>maxNumQueueEntries</i>	The maximum number of entries this queue will hold at one time (maximum: 32767).
<i>pBuffer</i>	The buffer in main memory to be used for this Deflate Queue. Must be aligned to 128 bytes.
<i>bufferSize</i>	The size of the provided buffer in main memory. The required size of this buffer for a given number of queue elements can be queried by calling <a href="#">edgeLzo1xGetDeflateQueueSize()</a> .

## Return Values

The [EdgeLzo1xDeflateQHandle](#) of the created Deflate Queue.

## Description

Creates a Deflate Queue. This will hold the list of work (segments to compress) that is queued up for the Deflate Tasks to work on.

## Notes

The Deflate Queue is a FIFO and will stall as necessary if work is pushed onto a full queue, so the queue size can safely be lower than the actual maximum number of elements needed.

## See Also

[EdgeLzo1xDeflateQueueElement](#), [edgeLzo1xAddDeflateQueueElement](#), [edgeLzo1xGetDeflateQueueSize](#), [edgeLzo1xShutdownDeflateQueue](#), [edgeLzo1xCreateDeflateTask](#)

# edgeLzo1xCreateInflateQueue

Creates an Inflate Queue.

## Definition

```
#include <edge/lzo/edgeLzo_ppu.h>
EdgeLzo1xInflateQHandle edgeLzo1xCreateInflateQueue
(
    CellSpurs* pSpurs,
    uint32_t maxNumQueueEntries,
    void* pBuffer,
    uint32_t bufferSize
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>pSpurs</i>	The SPURS instance that this Inflate Queue will be associated with.
<i>maxNumQueueEntries</i>	The maximum number of entries this queue will hold at one time (maximum: 32767).
<i>pBuffer</i>	The buffer in main memory to be used for this Inflate Queue. Must be aligned to 128 bytes.
<i>bufferSize</i>	The size of the provided buffer in main memory. The required size of this buffer for a given number of queue elements can be queried by calling <a href="#">edgeLzo1xGetInflateQueueSize()</a> .

## Return Values

The [EdgeLzo1xInflateQHandle](#) of the created Inflate Queue.

## Description

Create an Inflate Queue. This will hold the list of work (segments to decompress or move) that is queued up for the Inflate Tasks to work on.

## Notes

The Inflate Queue is a FIFO and will stall as necessary if work is pushed onto a full queue, so the queue size can safely be lower than the actual maximum number of elements needed.

## See Also

[EdgeLzo1xInflateQueueElement](#), [edgeLzo1xAddInflateQueueElement](#), [edgeLzo1xAddInflateQueueElementPartialCopyOut](#), [edgeLzo1xGetInflateQueueSize](#), [edgeLzo1xShutdownInflateQueue](#), [edgeLzo1xCreateInflateTask](#)



---

# edgeLzo1xShutdownDeflateQueue

---

Shuts down the Deflate Queue.

## Definition

---

```
#include <edge/lzo/edgelzo_ppu.h>
void edgeLzo1xShutdownDeflateQueue
(
    EdgeLzo1xDeflateQHandle handle
)
```

## Calling Conditions

---

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

---

<i>handle</i>	The handle of the Deflate Queue to shutdown
---------------	---

## Return Values

---

None

## Description

---

Shuts down the Deflate Queue.

## See Also

---

[edgeLzo1xCreateDeflateQueue](#)

---

# edgeLzo1xShutdownInflateQueue

---

Shuts down the Inflate Queue.

## Definition

---

```
#include <edge/lzo/edgeLzo_ppu.h>
void edgeLzo1xShutdownInflateQueue
(
    EdgeLzo1xInflateQHandle handle
)
```

## Calling Conditions

---

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

---

<i>handle</i>	The handle of the Inflate Queue to shutdown
---------------	---

## Return Values

---

None

## Description

---

Shuts down the Inflate Queue.

## See Also

---

[edgeLzo1xCreateInflateQueue](#)

---

# edgeLzo1xGetDeflateTaskContextSaveSize

---

Returns the required buffer size needed by one Deflate Task to store its context.

## Definition

---

```
#include <edge/lzo/edge_lzo_ppu.h>
uint32_t edgeLzo1xGetDeflateTaskContextSaveSize( void )
```

## Calling Conditions

---

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

---

None.

## Return Values

---

The size of the buffer needed to store the context data of one Deflate Task.

## Description

---

This function returns the required buffer size that the library needs to store the context of a single Deflate Task.

## Notes

---

The allocated buffer must be 16-byte aligned.

## See Also

---

[edgeLzo1xCreateDeflateTask](#)

---

## edgeLzo1xGetInflateTaskContextSaveSize

---

Returns the required buffer size needed by one Inflate Task to store its context.

### Definition

---

```
#include <edge/lzo/edge_lzo_ppu.h>
uint32_t edgeLzo1xGetInflateTaskContextSaveSize( void )
```

### Calling Conditions

---

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

### Arguments

---

None.

### Return Values

---

The size of the buffer needed for storing the context data of one Inflate Task.

### Description

---

This function returns the required buffer size that the library needs to store the context of a single Inflate Task.

### Notes

---

The allocated buffer must be 16-byte aligned.

### See Also

---

[edgeLzo1xCreateInflateTask](#)

# edgeLzo1xCreateDeflateTask

Creates a SPURS Task for performing compression on one SPU.

## Definition

```
#include <edge/lzo/edgelzo_ppu.h>
CellSpursTaskId edgeLzo1xCreateDeflateTask
(
    CellSpursTaskset* pTaskSet,
    void* pTaskContext,
    EdgeLzo1xDeflateQHandle handle
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>pTaskSet</i>	Pointer to the SPURS Taskset that this Deflate Task should be attached to.
<i>pTaskContext</i>	Pointer to the main memory buffer that the task uses for storing its context to. The required size of this buffer can be queried by calling <a href="#">edgeLzo1xGetDeflateTaskContextSaveSize()</a> .
<i>handle</i>	The handle for the Deflate Queue that this task will be pulling from.

## Return Values

The `CellSpursTaskId` of the created task.

## Description

When work exists, this Task will pull work off the Deflate Queue.

If there is no work to do, the task will sleep, in which case it will store its context in the location specified by *pTaskContext*.

## Notes

Create one Deflate Task for each SPU you want to run on. So, if you want compression to be able to run in parallel on six SPUs (and your SPURS Instance has six SPUs in it), then you should create six Deflate Tasks, all in the same taskset, and all working on the same Deflate Queue.

The necessary size for the buffer pointed to by *pTaskContext* may be queried by calling [edgeLzo1xGetDeflateTaskContextSaveSize\(\)](#).

## See Also

[edgeLzo1xAddDeflateQueueElement](#), [edgeLzo1xGetDeflateQueueSize](#),  
[edgeLzo1xCreateDeflateQueue](#), [edgeLzo1xShutdownDeflateQueue](#),  
[edgeLzo1xGetDeflateTaskContextSaveSize](#)

# edgeLzo1xCreateInflateTask

Creates a SPURS Task for performing decompression on one SPU.

## Definition

```
#include <edge/lzo/edgelzo_ppu.h>
CellSpursTaskId edgeLzo1xCreateInflateTask
(
    CellSpursTaskset* pTaskSet,
    void* pTaskContext,
    EdgeLzo1xInflateQHandle handle
)
```

## Calling Conditions

Can be called from an interrupt handler.

Can be called from a thread (does not depend on interrupt-disabled or -enabled state).

Multithread safe.

## Arguments

<i>pTaskSet</i>	Pointer to the SPURS Taskset that this Inflate Task should be attached to.
<i>pTaskContext</i>	Pointer to the main memory buffer that the task uses for storing its context to. The required size of this buffer can be queried by calling <a href="#">edgeLzo1xGetInflateTaskContextSaveSize()</a> .
<i>handle</i>	The handle for the Inflate Queue that this task will be pulling from.

## Return Values

The `CellSpursTaskId` of the created task.

## Description

When work exists, this Task will pull work off the Inflate Queue.

If there is no work to do, the task will sleep, in which case it will store its context in the location specified by *pTaskContext*.

## Notes

Create one Inflate Task for each SPU you want to run on. So, if you want decompression to be able to run in parallel on six SPUs (and your SPURS Instance has six SPUs in it), then you should create six Inflate Tasks, all in the same taskset, and all working on the same Inflate Queue.

The necessary size for the buffer pointed to by *pTaskContext* may be queried by calling [edgeLzo1xGetInflateTaskContextSaveSize\(\)](#).

## See Also

[edgeLzo1xAddInflateQueueElement](#), [edgeLzo1xAddInflateQueueElementPartialCopyOut](#), [edgeLzo1xGetInflateQueueSize](#), [edgeLzo1xCreateInflateQueue](#), [edgeLzo1xShutdownInflateQueue](#), [edgeLzo1xGetInflateTaskContextSaveSize](#)

# SPU Functions

# edgeLzo1xDeflateRawData

Compresses a buffer of uncompressed data.

## Definition

```
#include <edge/lzo/edgelzo_spu.h>
extern int edgeLzo1xDeflateRawData
(
    const unsigned char* pUncompr,
    uint32_t uncompSize,
    unsigned char* pComprData,
    uint32_t maxCompressedDataSize,
    uint32_t* pOutputCompressedSize
);
```

## Arguments

<i>pUncompr</i>	Pointer to where the uncompressed data will be read from in Local Store. Warning: low 16 bits of address affect contents of returned compressed data. This is inherent in the Lzo1x compression.
<i>uncompSize</i>	The size of the input compressed data that will be compressed.
<i>pComprData</i>	Pointer to where the compressed data will be written to in Local Store.
<i>maxCompressedDataSize</i>	Maximum size of the compressed data buffer.
<i>pOutputCompressedSize</i>	Returns the output size of the compressed data.

## Return Values

Zero on success; non-zero on failure.

## Description

This function compresses a buffer of uncompressed data within Local Store (LS).

## Notes

The input and output buffers are both in LS. Providing the input data and dealing with the output data is expected to be handled by the function that calls [edgeLzo1xDeflateRawData](#). This function does not do any DMAs internally.

Warning: The low 16 bits of the address of the input uncompressed data is used by Lzo1x in deciding how to encode the data, and changing these low 16 bits will thus result in differently encoded (but completely correct) compressed data. Therefore, compressing the same uncompressed file may produce different compressed data at different times, but both compressed data sets will decompress to the same uncompressed data.



---

# edgeLzo1xInflateRawData

---

Decompresses a buffer of compressed data.

## Definition

---

```
#include <edge/lzo/edgelzo_spu.h>
extern int edgeLzo1xInflateRawData
(
    unsigned char* pUncompr,
    uint32_t expectedUncompSize,
    const unsigned char* pComprData,
    uint32_t comprDataSize
);
```

## Arguments

---

<i>pUncompr</i>	Pointer to where the uncompressed data will be written to in Local Store.
<i>expectedUncompSize</i>	The expected output size of the uncompressed data. The output buffer pointed to by <i>pUncompr</i> must be at least this large.
<i>pComprData</i>	Pointer to where the compressed data will be read from in Local Store. This should be a pointer to the raw data without any header.
<i>comprDataSize</i>	Size of the compressed data buffer to be read.

## Return Values

---

Zero on success; nonzero on failure.

## Description

---

This function decompresses a buffer of compressed data within Local Store (LS).

## Notes

---

The input and output buffers are both in LS. Providing the input data and dealing with the output data is expected to be handled by the function that calls [edgeLzo1xInflateRawData](#). This function does not do any DMAs internally.

This function will assert if the *expectedUncompSize* does not equal the actual uncompressed size.

This function expects a pointer to the raw compressed data without any header.

If any data error is encountered, this function will not assert (unless a conditional define is changed) and instead returns an error value to its caller.