

# **PlayStation®Edge LZMA 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>
EdgeLzmaInflateQueueElement .....	6
EdgeLzmaInflateQHandle .....	8
EdgeLzmaInflateTaskProcessing .....	9
<b>Shared PPU/SPU Functions</b> .....	<b>10</b>
edgeLzmaAddInflateQueueElement .....	11
edgeLzmaTryAddInflateQueueElement .....	13
edgeLzmaAddInflateQueueElementPartialCopyOut.....	15
edgeLzmaTryAddInflateQueueElementPartialCopyOut .....	18
<b>PPU Functions</b> .....	<b>21</b>
edgeLzmaGetInflateQueueSize .....	22
edgeLzmaCreateInflateQueue .....	23
edgeLzmaShutdownInflateQueue .....	24
edgeLzmaGetInflateTaskContextSaveSize.....	25
edgeLzmaCreateInflateTask .....	26
<b>SPU Functions</b> .....	<b>27</b>
edgeLzmaInflateRawData .....	28

# Preface

---

# About This Document

---

## Purpose

This document provides an API reference for the Edge LZMA component of the Edge library. Use this component to efficiently decompress or move data through the SPUs.

## 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 LZO 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>fixed-width font + bold</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

# EdgeLzmaInflateQueueElement

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

## Definition

```
#include <edge/lzma/edgelzma_inflate_queue_element.h>
typedef struct EdgeLzmaInflateQueueElement
{
    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_properties0;
    uint16_t m_pad8;
} EdgeLzmaInflateQueueElement __attribute__((aligned(16)));
```

## Members

<i>m_eaCompressed</i>	Effective Address of compressed data.
<i>m_eaUncompressed</i>	Effective Address of uncompressed data.
<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 is done. If the SPU has an error with the compressed data, it will set the high bit of the counter to indicate that an error occurred. May be NULL.
<i>m_eaEventFlag</i>	Least significant bit of this EA is a flag that indicates if segment was stored compressed (1) or uncompressed (0) Effective Address of 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 <i>N</i> bytes of the uncompressed output.
<i>m_outputUncompSkipEndSize</i>	Do not output the last <i>N</i> bytes of the uncompressed output.
<i>m_properties0</i>	The first byte of the properties data.
<i>m_pad8</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**

---

[edgeLzmaAddInflateQueueElement](#), [edgeLzmaAddInflateQueueElementPartialCopyOut](#),  
[edgeLzmaCreateInflateQueue](#)

---

# EdgeLzmaInflateQHandle

---

A handle for the Inflate Queue.

## Definition (on PPU)

---

```
#include <edge/lzma/edgeLzma_ppu.h>
typedef void* EdgeLzmaInflateQHandle;
```

## Definition (on SPU)

---

```
#include <edge/lzma/edgeLzma_spu.h>
typedef uint32_t EdgeLzmaInflateQHandle;
```

## Description

---

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

## See Also

---

[EdgeLzmaInflateQueueElement](#), [edgeLzmaCreateInflateQueue](#)



---

# EdgeLzmaInflateTaskProcessing

---

Specifies whether a segment is copied or decompressed.

## Definition

---

```
#include <edge/lzma/edgeLzma_inflate_queue_element.h>

typedef enum EdgeLzmaInflateTaskProcessing
{
    kEdgeLzmaInflateTask_Memcpy = 0,
    kEdgeLzmaInflateTask_Inflate = 1,
} EdgeLzmaInflateTaskProcessing;
```

## Members

---

<i>kEdgeLzmaInflateTask_Memcpy</i>	Copy memory from one location to another.
<i>kEdgeLzmaInflateTask_Inflate</i>	Decompress memory from one location to another.

## Description

---

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

## See Also

---

[edgeLzmaAddInflateQueueElement](#), [edgeLzmaAddInflateQueueElementPartialCopyOut](#)

# Shared PPU/SPU Functions

# edgeLzmaAddInflateQueueElement

Adds a new piece of work to the Inflate Queue.

## Definition (on PPU)

```
#include <edge/lzma/edgelzma_ppu.h>
void edgeLzmaAddInflateQueueElement
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    const void* eaInputCompressedData,
    uint32_t compressedSize,
    void* eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzmaInflateTaskProcessing processing
)
```

## Definition (on SPU)

```
#include <edge/lzma/edgelzma_spu.h>
void edgeLzmaAddInflateQueueElement
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    uint32_t eaInputCompressedData,
    uint32_t compressedSize,
    uint32_t eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzmaInflateTaskProcessing processing,
    uint32_t dmaTagId
)
```

## 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>pProperties</i>	A pointer to the properties data. This is an Effective Address for the PPU function, or an LS (Local Store) address for the SPU function.
<i>propertiesSize</i>	The size of the properties data.
<i>eaInputCompressedData</i>	The Effective Address of the input data that will 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 indicate that an error occurred. Can be NULL.
<i>eaEventFlag</i>	The event flag to set. Can be NULL.
<i>eventFlagBits</i>	The value used to set the event flag.
<i>processing</i>	Choose what kind of processing the task will do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTagId</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### **Return Values**

---

None.

### **Description**

---

This function 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 *maxNumQueueEntries* has been reached), 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 that are being waited on. Then when each item is completed, the counter will decrement by one; 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, then the event flag will be set after this segment is done.

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

This function expects both a pointer to the properties data and the Effective Address of the raw compressed data.

### **See Also**

---

[EdgeLzmaInflateQueueElement](#), [edgeLzmaAddInflateQueueElementPartialCopyOut](#), [edgeLzmaCreateInflateTask](#)

# edgeLzmaTryAddInflateQueueElement

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

## Definition (on PPU)

```
#include <edge/lzma/edgelzma_ppu.h>
bool edgeLzmaTryAddInflateQueueElement
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    const void* eaInputCompressedData,
    uint32_t compressedSize,
    void* eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t* eaWorkToDoCounter,
    CellSpursEventFlag* eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzmaInflateTaskProcessing processing
)
```

## Definition (on SPU)

```
#include <edge/lzma/edgelzma_spu.h>
bool edgeLzmaTryAddInflateQueueElement
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    uint32_t eaInputCompressedData,
    uint32_t compressedSize,
    uint32_t eaOutputUncompressed,
    uint32_t expectedUncompressedSize,
    uint32_t eaWorkToDoCounter,
    uint32_t eaEventFlag,
    uint16_t eventFlagBits,
    EdgeLzmaInflateTaskProcessing processing,
    uint32_t dmaTagId
)
```

## 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>pProperties</i>	A pointer to the properties data. This is an Effective Address for the PPU function, or an LS (Local Store) address for the SPU function.
<i>propertiesSize</i>	The size of the properties data.
<i>eaInputCompressedData</i>	The Effective Address of the input data that will 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 encounters an error while processing the compressed data, it will set the high bit of the counter to indicate that an error occurred. Can be NULL.
<i>eaEventFlag</i>	The event flag to set. Can be NULL.
<i>eventFlagBits</i>	The value used to set the event flag.
<i>processing</i>	Choose what kind of processing the task will do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTagId</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

---

This function 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 that are being waited on. Then when each item is completed, the counter will decrement by one; 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, the Inflate Task can be used for this by passing the appropriate value for *processing*.

This function expects both a pointer to the properties data and the Effective Address of the raw compressed data.

### See Also

---

[EdgeLzmaInflateQueueElement](#), [edgeLzmaAddInflateQueueElementPartialCopyOut](#), [edgeLzmaCreateInflateQueue](#)

# edgeLzmaAddInflateQueueElementPartialCopyOut

Adds a new piece of work to the Inflate Queue.

## Definition (on PPU)

```
#include <edge/lzma/edgelzma_ppu.h>
void edgeLzmaAddInflateQueueElementPartialCopyOut
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    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,
    EdgeLzmaInflateTaskProcessing processing
)
```

## Definition (on SPU)

```
#include <edge/lzma/edgelzma_spu.h>
void edgeLzmaAddInflateQueueElementPartialCopyOut
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    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,
    EdgeLzmaInflateTaskProcessing processing,
    uint32_t dmaTagId
)
```

## 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>pProperties</i>	A pointer to the properties data. This is an Effective Address for the PPU function, or an LS (Local Store) address for the SPU function.

---

<i>propertiesSize</i>	The size of the properties data.
<i>eaInputCompressedData</i>	The Effective Address of the input data which 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>eaOutputUncompPartialBuff</i>	The Effective Address of the output buffer for the uncompressed data.
<i>outputUncompSkipBeginSize</i>	Do not output the first <i>N</i> 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 <i>N</i> 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 to indicate that an error occurred.
<i>eaEventFlag</i>	Can be NULL. The event flag to set.
<i>eventFlagBits</i>	Can be NULL. The value used to set the event flag.
<i>processing</i>	Choose what kind of processing the task will do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTagId</i>	(SPU only) The DMA tag to be used for DMAs performed inside this function.

### Return Values

---

None

### Description

---

This function 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 *maxNumQueueEntries* has been reached), 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 that are being waited on. Then when each item is completed, the counter will decrement by one; 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, the Inflate Task can be used for this by passing the appropriate value for *processing*.

This function expects both a pointer to the properties data and the Effective Address of the raw compressed data.

This function is very similar to [edgeLzmaAddInflateQueueElement\(\)](#). 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**

---

[EdgeLzmaInflateQueueElement](#), [edgeLzmaAddInflateQueueElement](#), [edgeLzmaCreateInflateTask](#)

# edgeLzmaTryAddInflateQueueElementPartialCopyOut

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

## Definition (on PPU)

```
#include <edge/lzma/edgelzma_ppu.h>
bool edgeLzmaTryAddInflateQueueElementPartialCopyOut
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    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,
    EdgeLzmaInflateTaskProcessing processing
)
```

## Definition (on SPU)

```
#include <edge/lzma/edgelzma_spu.h>
bool edgeLzmaTryAddInflateQueueElementPartialCopyOut
(
    EdgeLzmaInflateQHandle handle,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    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,
    EdgeLzmaInflateTaskProcessing processing,
    uint32_t dmaTagId
)
```

## 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

*handle*

The handle of the Inflate Queue that the entry will be pushed onto.

---

<i>pProperties</i>	A pointer to the properties data. This is an Effective Address for the PPU function, or an LS (Local Store) address for the SPU function.
<i>propertiesSize</i>	The size of the properties data.
<i>eaInputCompressedData</i>	The Effective Address of the input data that will 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>eaOutputUncompPartialBuff</i>	The Effective Address of the output buffer for the uncompressed data.
<i>outputUncompSkipBeginSize</i>	Do not output the first <i>N</i> 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 <i>N</i> 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 to indicate that an error occurred.
<i>eaEventFlag</i>	Can be NULL. The event flag to set.
<i>eventFlagBits</i>	Can be NULL. The value used to set the event flag.
<i>processing</i>	Choose what kind of processing the task will do on the data. Can perform decompression or merely move the raw data from <i>eaInputCompressedData</i> to <i>eaOutputUncompressedData</i> .
<i>dmaTagId</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

---

This function 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 that are being waited on. Then when each item is completed, the counter will decrement by one; 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 then 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, the Inflate Task can be used for this by passing the appropriate value for *processing*.

This function expects both a pointer to the properties data and the Effective Address of raw compressed data.

This function is very similar to [edgeLzmaAddInflateQueueElement](#). 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**

---

[EdgeLzmaInflateQueueElement](#), [edgeLzmaAddInflateQueueElement](#), [edgeLzmaCreateInflateTask](#)

# PPU Functions

---

# edgeLzmaGetInflateQueueSize

---

Returns the required buffer size needed for an Inflate Queue.

## Definition

---

```
#include <edge/lzma/edgeLzma_ppu.h>
uint32_t edgeLzmaGetInflateQueueSize
(
    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

---

This function returns the required buffer size needed for an Inflate Queue, which can hold at most the specified number of elements at one time.

## Notes

---

The allocated buffer must be 128-byte aligned.

## See Also

---

[EdgeLzmaInflateQueueElement](#), [edgeLzmaCreateInflateQueue](#), [edgeLzmaCreateInflateTask](#)

# edgeLzmaCreateInflateQueue

Creates an Inflate Queue.

## Definition

```
#include <edge/lzma/edgeLzma_ppu.h>
EdgeLzmaInflateQHandle edgeLzmaCreateInflateQueue
(
    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="#">edgeLzmaGetInflateQueueSize()</a> .

## Return Values

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

## Description

This function creates 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.

## See Also

[EdgeLzmaInflateQueueElement](#), [edgeLzmaAddInflateQueueElement](#), [edgeLzmaAddInflateQueueElementPartialCopyOut](#), [edgeLzmaGetInflateQueueSize](#), [edgeLzmaShutdownInflateQueue](#), [edgeLzmaCreateInflateTask](#)

---

# edgeLzmaShutdownInflateQueue

---

Shuts down the Inflate Queue.

## Definition

---

```
#include <edge/lzma/edgeLzma_ppu.h>
void edgeLzmaShutdownInflateQueue
(
    EdgeLzmaInflateQHandle 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

---

This function shuts down the Inflate Queue.

## See Also

---

[edgeLzmaCreateInflateQueue](#)



---

## edgeLzmaGetInflateTaskContextSaveSize

---

Returns the buffer size needed for storing the context data of an Inflate Task.

### Definition

---

```
#include <edge/lzma/edgeLzma_ppu.h>
uint32_t edgeLzmaGetInflateTaskContextSaveSize( 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 an Inflate Task.

### Description

---

This function returns the buffer size needed for storing the context data of an Inflate Task.

### Notes

---

The allocated buffer must be 16-byte aligned.

### See Also

---

[edgeLzmaCreateInflateTask](#)

# edgeLzmaCreateInflateTask

Create a SPURS task for performing decompression on an SPU.

## Definition

```
#include <edge/lzma/edgeLzma_ppu.h>
CellSpursTaskId edgeLzmaCreateInflateTask
(
    CellSpursTaskset* pTaskSet,
    void* pTaskContext,
    EdgeLzmaInflateQHandle 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 task set 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. The required size of this buffer can be queried by calling <a href="#">edgeLzmaGetInflateTaskContextSaveSize()</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

This function creates a SPURS task that will pull work off the Inflate Queue when work exists.

If there is no work to do, the task will sleep, storing its context to 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 6 SPUs (and your SPURS instance has 6 SPUs in it) you should create 6 Inflate Tasks, all in the same task set, and all working on the same Inflate Queue.

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

## See Also

[edgeLzmaAddInflateQueueElement](#), [edgeLzmaAddInflateQueueElementPartialCopyOut](#), [edgeLzmaGetInflateQueueSize](#), [edgeLzmaCreateInflateQueue](#), [edgeLzmaShutdownInflateQueue](#), [edgeLzmaGetInflateTaskContextSaveSize](#)

# SPU Functions

# edgeLzmaInflateRawData

---

Decompresses a buffer of compressed data.

## Definition

---

```
#include <edge/lzma/edgeLzma_spu.h>
extern int edgeLzmaInflateRawData
(
    unsigned char* pUncompr,
    uint32_t expectedUncompSize,
    const unsigned char* pProperties,
    uint32_t propertiesSize,
    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>pProperties</i>	Pointer to the properties data in Local Store.
<i>propertiesSize</i>	The size in bytes of the properties data.
<i>pComprData</i>	Pointer to where the compressed data will be read from in Local Store.
<i>comprDataSize</i>	This should be a pointer to the raw data without any header. Size of the compressed data buffer to be read.

## Return Values

---

Zero on success. Non-zero on failure.

## Description

---

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

## Notes

---

The input and output buffers are both in Local Store. The function that calls [edgeLzmaInflateRawData\(\)](#) is expected to provide the input data and deal with the output data. This function does not do any DMAs internally.

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

This function expects both a pointer to the properties data and a pointer to the raw compressed data in Local Store.

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.