Queue

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers

Table of Contents

RTAC LIBRARY

Queue

Introduction

See the ACSELERATOR RTAC Library Extensions Instruction Manual (LibraryExtensionsIM) for an explanation of the concepts used by the object-oriented extensions to the IEC 61131-3 standard.

Queues

A queue is a fundamental data structure in computer science, and is implemented within this library as an object.

The term "queue" is often used interchangeably with the term "First-In-First-Out" (FIFO), which is a more descriptive name. The queue is often used as a buffer, allowing information to be queued up to be processed in the same order that it was received in, but at a different rate. This library defines the front of a queue as the oldest element pushed into the queue and the back of the queue is the newest element pushed into the queue.

This is easily visualized and remembered by using the common image of customers standing in a line, also known as a queue. The customer that has been in line the longest is at the front of the queue, and people awaiting service are added to the back of the queue.

All queues in this library assume that all elements within the queue are the same size.

Double-Ended Queues (Deques)

This library provides a a double-ended queue implementation (a deque). A deque can do anything that a standard queue can do, but can have items added or removed from either the front or the back.

With a deque, the library assembly can move priority information to the front of a queue, or balance several parallel queues by removing items from the back of one queue and reassigning them to the back of other queues. These are operations that cannot be performed on a pure queue.

The deque implementation used in this library also ensures that all the data within the queue are kept in contiguous memory, which is not guaranteed in all queue implementations.

Special Considerations

Classes in this library have memory allocated inside them. As such, they should only be created in environments of permanent scope (e.g., Programs, Global Variable Lists, or VAR_STAT sections).

Copying classes from this library causes unwanted behavior. This means the following:

1. The assignment operator ":=" must not be used on any class from this library; consider assigning pointers to the objects instead.

```
// This is bad and in most cases will provide a compiler error such
    as:
// "C0328: Assignment not allowed for type class_QueueObject"
myQueueObject := otherQueueObject;
```

```
// This is fine
someVariable := myQueueObject.value;
// As is this
pt_myQueueObject := ADR(myQueueObject);
```

2. Classes from this library must never be VAR_INPUT or VAR_OUTPUT members in function blocks, functions, or methods. Place them in the VAR_IN_OUT section or use pointers instead.

Supported Firmware Versions

You can use this library on any device configured using ACSELERATOR RTAC® SEL-5033 Software with firmware version R143 or higher.

Versions 3.5.0.0 and older can be used on RTAC firmware version R132 and higher.

Global Parameters

The library applies the following values as maximums; they can be modified when the library is included in a project.

Name	IEC 61131 Type	Value	Description
g_p_DefaultQueueSize	UDINT	32	The default number of elements that a queue can hold.

Interfaces

This library provides the following interface.

I_Queue (Interface)

This interface is implemented by any class that provides a queue data type.

Properties

Name	IEC 61131 Type	Access	Description
ElementSize	UDINT	R	The number of bytes required for each element in the queue.
MaxSize	UDINT	R	The number of elements this queue can hold before a reallocation for additional memory is required.
Size	UDINT	R	The number of elements in the queue.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Clear (Method)

Deallocates all memory associated with the queue. Call this method only if the queue is instantiated with limited scope (i.e., if it is instantiated as a local variable of a function or method).

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the queue successfully deallocates its internal memory. FALSE if an error occurs.

EraseFront (Method)

This method deletes the specified number of elements from the front of the queue.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the front of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the queue.

Processing

- Removes as many as *numToErase* elements from the front of the queue.
- If the *Size* of the queue is less than *numToErase*, only *Size* elements are removed from the queue.

Front (Method)

This method copies the specified number of elements from the front of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the front of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the front of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

PopFront (Method)

This method copies the specified number of elements from the front of the queue to the provided pointer location and deletes them from the queue.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the front of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the queue. Zero if the queue was not modified.

Processing

- Copies as many as *numToPop* elements from the front of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the queue.
- If *pt_destination* is invalid or the elements cannot be copied, the queue is not modified and zero is returned.

PushBack (Method)

This method copies elements from the specified pointer location and pushes them onto the back of the queue.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the queue. Zero if an error occurred and the queue was not modified.

Processing

- If the queue is not large enough to contain the new elements, additional memory is allocated to enlarge the queue.
- Copies the elements from *pt_source* and pushes them onto the back of the queue.
- If *pt_source* is invalid or *numToPush* is zero, the queue is not modified and zero is returned.

Recycle (Method)

This method removes all elements from the queue without modifying the memory allocated to the queue.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the queue successfully removes all elements. FALSE if an error occurs.

Processing

All elements are removed from the queue.

- After recycling, the *Size* of the queue is zero.
- The *MaxSize* is unchanged after a call to `Recycle()`.
- This method neither allocates nor frees any memory.

Classes

class_Deque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque can handle objects of arbitrary size, so long as the number of bytes required for each element is the same.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
elementSize	UDINT	The number of bytes required for each element that this deque can hold. If zero, defaults to one.
numElements	UDINT	The number of elements to allocate initially. If zero, *g_p_-DefaultQueueSize* is used.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

class_ByteDeque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque operates only on elements of type BYTE.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
size	UDINT	The number of elements to allocate initially. If zero, use *g_p_DefaultQueueSize*.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

BackByte (Method)

This method provides the element at the back of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	BYTE	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

FrontByte (Method)

This method provides the element at the front of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	BYTE	A copy of the element at the front of the deque. If the return value is false, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PopBackByte (Method)

This method provides a copy of the element at the back of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	BYTE	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PopFrontByte (Method)

This method provides a copy of the element at the front of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	BYTE	A copy of the element at the front of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PushBackByte (Method)

This method appends a copy of the provided element to the back of the deque.

Inputs

Name	IEC 61131 Type	Description
element	BYTE	The element to append to the back of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

PushFrontByte (Method)

This method appends a copy of the provided element to the front of the deque.

Inputs

Name	IEC 61131 Type	Description
element	BYTE	The element to append to the front of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

class_DwordDeque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque operates only on elements of type DWORD.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
size	UDINT	The number of elements to allocate initially. If zero, use *g_p_DefaultQueueSize*.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

BackDword (Method)

This method provides the element at the back of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	DWORD	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

FrontDword (Method)

This method provides the element at the front of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	DWORD	A copy of the element at the front of the deque. If the return value is false, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PopBackDword (Method)

This method provides a copy of the element at the back of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	DWORD	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PopFrontDword (Method)

This method provides a copy of the element at the front of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	DWORD	A copy of the element at the front of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PushBackDword (Method)

This method appends a copy of the provided element to the back of the deque.

Inputs

Name	IEC 61131 Type	Description
element	DWORD	The element to append to the back of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

PushFrontDword (Method)

This method appends a copy of the provided element to the front of the deque.

Inputs

Name	IEC 61131 Type	Description
element	DWORD	The element to append to the front of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

class_LwordDeque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque operates only on elements of type DWORD.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
size	UDINT	The number of elements to allocate initially. If zero, use *g_p_DefaultQueueSize*.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

BackLword (Method)

This method provides the element at the back of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	LWORD	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

FrontLword (Method)

This method provides the element at the front of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	LWORD	A copy of the element at the front of the deque. If the return value is false, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PopBackLword (Method)

This method provides a copy of the element at the back of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	LWORD	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PopFrontLword (Method)

This method provides a copy of the element at the front of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	LWORD	A copy of the element at the front of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PushBackLword (Method)

This method appends a copy of the provided element to the back of the deque.

Inputs

Name	IEC 61131 Type	Description
element	LWORD	The element to append to the back of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

PushFrontLword (Method)

This method appends a copy of the provided element to the front of the deque.

Inputs

Name	IEC 61131 Type	Description
element	LWORD	The element to append to the front of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

class_LrealDeque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque operates only on elements of type LREAL.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
size	UDINT	The number of elements to allocate initially. If zero, use *g_p_DefaultQueueSize*.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

BackLreal (Method)

This method provides the element at the back of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	LREAL	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

FrontLreal (Method)

This method provides the element at the front of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	LREAL	A copy of the element at the front of the deque. If the return value is false, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PopBackLreal (Method)

This method provides a copy of the element at the back of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	LREAL	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PopFrontLreal (Method)

This method provides a copy of the element at the front of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	LREAL	A copy of the element at the front of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PushBackLreal (Method)

This method appends a copy of the provided element to the back of the deque.

Inputs

Name	IEC 61131 Type	Description
element	LREAL	The element to append to the back of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

PushFrontLreal (Method)

This method appends a copy of the provided element to the front of the deque.

Inputs

Name	IEC 61131 Type	Description
element	LREAL	The element to append to the front of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

class_PointerDeque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque operates only on elements of type POINTER TO ANY.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
size	UDINT	The number of elements to allocate initially. If zero, use *g_p_DefaultQueueSize*.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

BackPointer (Method)

This method provides the element at the back of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

FrontPointer (Method)

This method provides the element at the front of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	A copy of the element at the front of the deque. If the return value is false, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PopBackPointer (Method)

This method provides a copy of the element at the back of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PopFrontPointer (Method)

This method provides a copy of the element at the front of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	A copy of the element at the front of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PushBackPointer (Method)

This method appends a copy of the provided element to the back of the deque.

Inputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	The element to append to the back of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

PushFrontPointer (Method)

This method appends a copy of the provided element to the front of the deque.

Inputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	The element to append to the front of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

class_RealDeque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque operates only on elements of type REAL.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
size	UDINT	The number of elements to allocate initially. If zero, use *g_p_DefaultQueueSize*.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

BackReal (Method)

This method provides the element at the back of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	REAL	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

FrontReal (Method)

This method provides the element at the front of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	REAL	A copy of the element at the front of the deque. If the return value is false, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PopBackReal (Method)

This method provides a copy of the element at the back of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	REAL	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PopFrontReal (Method)

This method provides a copy of the element at the front of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	REAL	A copy of the element at the front of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PushBackReal (Method)

This method appends a copy of the provided element to the back of the deque.

Inputs

Name	IEC 61131 Type	Description
element	REAL	The element to append to the back of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

PushFrontReal (Method)

This method appends a copy of the provided element to the front of the deque.

Inputs

Name	IEC 61131 Type	Description
element	REAL	The element to append to the front of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

class_WordDeque

This class implements a double-ended queue that internally handles dynamic allocation of memory. This deque operates only on elements of type WORD.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Queue

Initialization Inputs

Name	IEC 61131 Type	Description
size	UDINT	The number of elements to allocate initially. If zero, use *g_p_DefaultQueueSize*.

Properties

Name	IEC 61131 Type	Access	Description
pt_Data	UDINT	R	A pointer to the first element in the deque. This implementation of deque ensures that all elements are in contiguous memory. The pointer value returned by this method is only valid until the next operation is performed on the deque, since any modification to the contents of this object can cause the storage location to move.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

Back (Method)

This method copies the specified number of elements from the back of the queue to the provided pointer location. The queue is not modified.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToCopy	UDINT	The number of elements to copy from the back of the queue.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied.

Processing

- Copies as many as *numToCopy* elements from the back of the queue to *pt_destination*.
- If the *Size* of the queue is less than *numToCopy*, only *Size* elements are copied to *pt_destination*.
- If *pt_destination* is invalid or the elements cannot be copied, zero is returned.

BackWord (Method)

This method provides the element at the back of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	WORD	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

EraseBack (Method)

This method deletes the specified number of elements from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
numToErase	UDINT	The number of elements to erase from the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully removed from the back of the deque.

Processing

- Removes as many as *numToErase* elements from the back of the deque.
- If the *Size* of the deque is less than *numToErase*, then only *Size* elements are removed from the deque.

FrontWord (Method)

This method provides the element at the front of the deque without modifying the deque.

Outputs

Name	IEC 61131 Type	Description
element	WORD	A copy of the element at the front of the deque. If the return value is false, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied. FALSE if the size is zero or an error occurs.

PopBack (Method)

This method copies the specified number of elements from the back of the deque to the provided pointer location and deletes them from the deque.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the elements are copied.
numToPop	UDINT	The number of elements to pop off the back of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully copied and removed from the deque. Zero if the deque was not modified.

Processing

- Copies as many as *numToPop* elements from the back of the deque to *pt_destination*.
- If the *Size* of the deque is less than *numToPop*, only *Size* elements are copied to *pt_destination*.
- Removes the copied elements from the deque.
- If *pt_destination* is invalid or the elements cannot be copied, the deque is not modified and zero is returned.

PopBackWord (Method)

This method provides a copy of the element at the back of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	WORD	A copy of the element at the back of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PopFrontWord (Method)

This method provides a copy of the element at the front of the deque and removes that element from the deque.

Outputs

Name	IEC 61131 Type	Description
element	WORD	A copy of the element at the front of the deque. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully copied and removed from the deque. FALSE if the size is zero or an error occurs.

PushBackWord (Method)

This method appends a copy of the provided element to the back of the deque.

Inputs

Name	IEC 61131 Type	Description
element	WORD	The element to append to the back of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

PushFront (Method)

This method copies elements from the specified pointer location and pushes them onto the front of the deque.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the elements are copied.
numToPush	UDINT	The number of elements to push onto the front of the deque.

Return Value

IEC 61131 Type	Description
UDINT	The number of elements successfully pushed onto the deque. Zero if an error occurred and the deque was not modified.

Processing

- If the deque is not large enough to contain the new elements, additional memory is allocated to enlarge the deque.
- Copies the elements from the specified pointer and pushes them onto the front of the deque.
- If *pt_source* is invalid or *numToPush* is zero, the deque is not modified and zero is returned.

PushFrontWord (Method)

This method appends a copy of the provided element to the front of the deque.

Inputs

Name	IEC 61131 Type	Description
element	WORD	The element to append to the front of the deque.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the *element* is successfully added to the deque. FALSE if an error occurs.

Processing

If pushing *element* to the deque requires more memory than is currently available in the deque, the library allocates additional memory.

Resize (Method)

This method resizes the deque so that it can hold the specified number of elements. If reducing the size of the deque to less than *Size*, elements are deleted from the back of the deque.

Inputs

Name	IEC 61131 Type	Description
newMaxSize	UDINT	The new maximum number of elements the deque can hold before a memory allocation is required.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the deque was resized. FALSE if an error occurred and the deque was not modified.

Processing

- If *newMaxSize* is zero, all elements in the deque are deleted. This is the same functionality as the `Clear()` method.
- If *newMaxSize* is equal to *MaxSize*, the deque is not modified.
- If *newMaxSize* is smaller than *Size*, elements are removed from the back of the deque in order to resize the deque.
- If *newMaxSize* is greater than or equal to *Size*, the deque is resized and retains all existing elements.

Benchmarks

Benchmark Platforms

The benchmarking tests recorded for this library are performed on the following platforms.

- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R134 firmware
- SEL-3530
  - R134 firmware
- SEL-3505
  - R134 firmware

Benchmark Test Descriptions

Benchmarks are only performed on the Typed queues as the performance of the generic queues will depend entirely on the size of an individual element of the created queue. Tests postfixed with _TYPE_ will call the typed version of the method dealing with only a single element.

Back_TYPE_

The posted time is the average execution time of 100 calls when the `Back_TYPE_()` method copies one element from a deque containing 100 elements.

Back - 100 Elements

The posted time is the average execution time of 100 calls when the `Back()` method copies 100 elements from a deque containing 100 elements.

Clear

The posted time is the average execution time of 100 calls when clearing a deque containing 100 elements.

EraseBack - 1 Element

The posted time is the average execution time of 100 calls when the EraseBack method removes one element from a deque containing 100 elements.

EraseBack - 100 Elements

The posted time is the average execution time of 100 calls when the EraseBack() method removes 100 elements from a deque containing 100 elements.

EraseFront - 1 Element

The posted time is the average execution time of 100 calls when the EraseFront() method removes one element from a deque containing 100 elements.

EraseFront - 100 Elements

The posted time is the average execution time of 100 calls when the EraseFront() method removes 100 elements from a deque containing 100 elements.

Front_TYPE_

The posted time is the average execution time of 100 calls when the `Front_TYPE_()` method copies one element from a deque containing 100 elements.

Front - 100 Elements

The posted time is the average execution time of 100 calls when the `Front()` method copies 100 elements from a deque containing 100 elements.

PopBack_TYPE_

The posted time is the average execution time of 100 calls when popping one element from a deque containing 100 elements.

PopBack - 100 Elements

The posted time is the average execution time of 100 calls when popping 100 elements from the deque in a single method call. The deque is constructed such that it has at least one resize caused by pushing prior to starting the pop measurement.

PopFront_TYPE_

The posted time is the average execution time of 100 calls when popping one element from a deque containing 100 elements.

PopFront - 100 Elements

The posted time is the average execution time of 100 calls when popping 100 elements from the deque in a single method call. The deque is constructed such that it has at least one resize caused by pushing prior to starting the pop measurement.

PushBack_TYPE_

The posted time is the average execution time of 100 calls when pushing an element onto a deque without causing a resize.

PushBack - 100 Elements

The posted time is the average execution time of 100 calls when pushing 100 elements onto a deque and causing a resize.

PushFront_TYPE_

The posted time is the average execution time of 100 calls when pushing an element onto a deque without causing a resize.

PushFront - 100 Elements

The posted time is the average execution time of 100 calls when pushing 100 elements onto a deque and causing a resize.

Recycle

The posted time is the average execution time of 100 calls when recycling a deque containing 100 elements.

Resize - Enlarging

The posted time is the average execution time of 100 calls when resizing a full deque from 32 elements to 64 elements.

Resize - Shrinking

The posted time is the average execution time of 100 calls when resizing a full deque from 64 elements to 32 elements.

Benchmark Results

Operation Tested	Platform (time in μs)		
	SEL-3555	SEL-3530	SEL-3505
Back - Byte	1	7	53
Back - Dword	1	5	16
Back - LReal	1	6	17
Back - Lword	1	4	23
Back - Pointer	1	5	26
Back - Real	1	6	28
Back - Word	1	6	19
Back 100 - Byte	1	3	16
Back 100 - Dword	1	3	8
Back 100 - LReal	1	6	10
Back 100 - Lword	1	4	14
Back 100 - Pointer	1	3	13
Back 100 - Real	1	5	14
Back 100 - Word	1	3	9
Clear - Byte	3	24	57
Clear - Dword	2	15	69
Clear - LReal	2	33	55
Clear - Lword	2	15	61
Clear - Pointer	2	16	61
Clear - Real	2	24	60
Clear - Word	2	17	58
EraseBack 1 - Byte	1	2	3
EraseBack 1 - Dword	1	2	3
EraseBack 1 - LReal	1	2	3
EraseBack 1 - Lword	1	2	2
EraseBack 1 - Pointer	1	2	3
EraseBack 1 - Real	1	2	2
EraseBack 1 - Word	1	1	2
EraseBack 100 - Byte	1	1	1
EraseBack 100 - Dword	1	1	1
EraseBack 100 - LReal	1	1	2
EraseBack 100 - Lword	1	1	2
EraseBack 100 - Pointer	1	1	1
EraseBack 100 - Real	1	1	1
EraseBack 100 - Word	1	1	1
EraseFront 1 - Byte	1	3	12
EraseFront 1 - Dword	1	2	10
EraseFront 1 - LReal	1	5	7
EraseFront 1 - Lword	1	4	10
EraseFront 1 - Pointer	1	3	10
EraseFront 1 - Real	1	4	13
EraseFront 1 - Word	1	3	8
EraseFront 100 - Byte	1	2	2
EraseFront 100 - Dword	1	1	6
EraseFront 100 - LReal	1	2	2
EraseFront 100 - Lword	1	2	3

Operation Tested	Platform (time in μs)		
	SEL-3555	SEL-3530	SEL-3505
EraseFront 100 - Pointer	1	2	4
EraseFront 100 - Real	1	2	4
EraseFront 100 - Word	1	1	5
Front - Byte	1	10	11
Front - Dword	1	4	12
Front - LReal	1	6	16
Front - Lword	1	4	14
Front - Pointer	1	4	13
Front - Real	1	6	13
Front - Word	1	4	12
Front 100 - Byte	1	4	4
Front 100 - Dword	1	3	6
Front 100 - LReal	1	5	11
Front 100 - Lword	1	4	9
Front 100 - Pointer	1	3	7
Front 100 - Real	1	3	6
Front 100 - Word	1	3	5
PopBack - Byte	1	4	6
PopBack - Dword	1	3	15
PopBack - LReal	1	7	7
PopBack - Lword	1	3	7
PopBack - Pointer	1	4	9
PopBack - Real	1	5	10
PopBack - Word	1	4	19
PopBack 100 - Byte	1	3	4
PopBack 100 - Dword	1	4	16
PopBack 100 - LReal	1	9	7
PopBack 100 - Lword	1	4	8
PopBack 100 - Pointer	1	4	7
PopBack 100 - Real	1	5	7
PopBack 100 - Word	1	3	13
PopFront - Byte	1	5	8
PopFront - Dword	1	5	10
PopFront - LReal	1	7	16
PopFront - Lword	1	6	13
PopFront - Pointer	1	5	11
PopFront - Real	1	6	11
PopFront - Word	1	5	8
PopFront 100 - Byte	1	3	5
PopFront 100 - Dword	1	4	6
PopFront 100 - LReal	1	5	10
PopFront 100 - Lword	1	4	8
PopFront 100 - Pointer	1	4	7
PopFront 100 - Real	1	4	7
PopFront 100 - Word	1	3	5
PushBack - Byte	1	4	6
PushBack - Dword	1	3	7
PushBack - LReal	1	4	7

Operation Tested	Platform (time in μs)		
	SEL-3555	SEL-3530	SEL-3505
PushBack - Lword	1	3	6
PushBack - Pointer	1	3	6
PushBack - Real	1	3	6
PushBack - Word	1	3	5
PushBack 100 - Byte	4	86	84
PushBack 100 - Dword	4	29	89
PushBack 100 - LReal	4	40	106
PushBack 100 - Lword	4	29	98
PushBack 100 - Pointer	4	29	89
PushBack 100 - Real	4	63	88
PushBack 100 - Word	4	29	85
PushFront - Byte	1	5	15
PushFront - Dword	1	5	19
PushFront - LReal	1	8	11
PushFront - Lword	1	6	16
PushFront - Pointer	1	5	20
PushFront - Real	1	8	19
PushFront - Word	1	5	15
PushFront 100 - Byte	6	49	137
PushFront 100 - Dword	4	34	117
PushFront 100 - LReal	4	40	165
PushFront 100 - Lword	4	32	152
PushFront 100 - Pointer	4	40	123
PushFront 100 - Real	4	32	124
PushFront 100 - Word	4	61	104
Recycle - Byte	1	1	2
Recycle - Dword	1	1	2
Recycle - LReal	1	1	2
Recycle - Lword	1	1	2
Recycle - Pointer	1	1	2
Recycle - Real	1	1	2
Recycle - Word	1	1	1
Resize Down - Byte	3	30	103
Resize Down - Dword	3	22	124
Resize Down - LReal	3	44	80
Resize Down - Lword	3	23	78
Resize Down - Pointer	3	22	89
Resize Down - Real	3	31	92
Resize Down - Word	3	22	153
Resize Up - Byte	3	24	117
Resize Up - Dword	3	22	102
Resize Up - LReal	3	34	84
Resize Up - Lword	3	23	125
Resize Up - Pointer	3	22	133
Resize Up - Real	3	28	133
Resize Up - Word	3	22	79

Examples

These examples demonstrate the capabilities of this library. Do not mistake them as suggestions or recommendations from SEL.

Implement the best practices of your organization when using these libraries. As the user of this library, you are responsible for ensuring correct implementation and verifying that the project using these libraries performs as expected.

Simple Deque Operation

This example demonstrates basic operation of a deque.

Objective

This example comprises the following steps:

Step 1. Add a series of UINT values to a UINT deque; the first as a single value, and the second from an array of values.

Step 2. Remove one value from the front.

Step 3. Remove a group of five values from the front.

Step 4. Remove one value from the back.

Step 5. Push four values onto the front.

Step 6. Remove six values from the back.

Step 7. Remove the remaining value individually from the front until the deque is empty again.

Sequence of Operations

The operations that make up the solution are outlined here in detail. After each operation, the expected state of the deque is shown. The notation used in this example assumes the front of the deque is on the left and the back is on the right.

1. The deque begins empty

$$Front\ [\ ]\ Back$$

2. The first value pushed to the back of the deque is 0.

$$Front\ [0]\ Back$$

3. An array of values, $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, is pushed onto the back of the deque.

$$Front\ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]\ Back$$

4. Five values are popped off of the front of the deque and into an array.

$$Front\ [5, 6, 7, 8, 9, 10]\ Back$$

The resulting array contains $[0, 1, 2, 3, 4]$.

5. The value at the back, 10, is then popped off.

$$Front\ [5, 6, 7, 8, 9]\ Back$$

6. An array with four values, $[11, 12, 13, 14]$, are then pushed onto the front.

$$Front\ [11, 12, 13, 14, 5, 6, 7, 8, 9]\ Back$$

7. Six values are then popped off the back into an array.

$$Front\ [11, 12, 13]\ Back$$

   The resulting array that was popped off contains $[14, 5, 6, 7, 8, 9]$.

8. The values 11, 12, and 13 are then obtained by popping the remaining values off the front of the deque one at a time, leaving it empty again.

$$Front\ [\ ]\ Back$$

Solution

The implementation of this example is shown in *Code Snippet 1*.

Code Snippet 1 prg_BasicDeque

```
PROGRAM prg_BasicDeque
VAR CONSTANT
    c_NumPushStep3 : UDINT := 10;
    c_NumPopStep4 : UDINT := 5;
    c_NumPushStep6 : UDINT := 4;
    c_NumPopStep7 : UDINT := 6;
END_VAR
VAR
    Run : BOOL := TRUE; // Used to force the steps to only run once.
    Ok : BOOL := FALSE; // Is true if the operation completed
    MyDeque : class_WordDeque(numElements := 0); // Uses default internal
        allocation

    (* Arrays to push*)
    Step3ArrayToPush : ARRAY[1..c_NumPushStep3] OF UINT :=
        [1,2,3,4,5,6,7,8,9,10];
    Step6ArrayToPush : ARRAY[1..c_NumPushStep6] OF UINT := [11,12,13,14];

    (* Results *)
    Step4ArrayPopped : ARRAY[1..c_NumPopStep4] OF UINT; // Expect :
        [0,1,2,3,4]
    Step5Popped : UINT; // Expect : 10
    Step7ArrayPopped : ARRAY[1..c_NumPopStep7] OF UINT; // Expect :
        [14,5,6,7,8,9]
    Step8val1, Step8val2, Step8val3 : UINT; // Expect: 11, 12, 13
END_VAR
```

Code Snippet 1 prg_BasicDeque (Continued)

```
IF Run THEN
    (*Step 1: The queue begins empty.*)
    Ok := TRUE;
    (*Step 2: Push 0 into back of deque. Ok if push returns TRUE.*)
    Ok := Ok AND MyDeque.PushBackWord(0);
    (*Step 3: Push array to back of deque. Ok if the number popped is as
        requested.*)
    Ok := Ok AND (c_NumPushStep3 =
        MyDeque.PushBack(ADR(Step3ArrayToPush),c_NumPushStep3));
    (*Step 4: Pop 5 values from front. Ok if number popped is as
        requested.*)
    Ok := Ok AND (c_NumPopStep4 =
        MyDeque.PopFront(ADR(Step4ArrayPopped),c_NumPopStep4));
    (*Step 5: Pop one number of the back of the queue.*)
    Ok := Ok AND MyDeque.PopBackWord(element => Step5Popped);
    (*Step 6: Add 4 new values to the front. Ok if push adds number
        requested.*)
    Ok := Ok AND (c_NumPushStep6 =
        MyDeque.PushFront(ADR(Step6ArrayToPush),c_NumPushStep6));
    (*Step 7: Pop 6 values from the back to an array. Ok if number popped
        is as requested.*)
    Ok := Ok AND (c_NumPopStep7 =
        MyDeque.PopBack(ADR(Step7ArrayPopped),c_NumPopStep7));
    (*Step 8 : Pop the last 3 values from the front. Each operation OK if
        it returns TRUE.*)
    Ok := Ok AND MyDeque.PopFrontWord(element => Step8val1);
    Ok := Ok AND MyDeque.PopFrontWord(element => Step8val2);
    Ok := Ok AND MyDeque.PopFrontWord(element => Step8val3);
    Run := FALSE; // Only run 1 time.
END_IF
```

Release Notes

Version	Summary of Revisions	Date Code
3.5.1.0	➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types "Cannot convert" messages. ➤ Replaced the deprecated POINTER_TO_ANY type with POINTER_TO_BYTE. ➤ Must be used with R143 firmware or later.	20180921
3.5.0.0	➤ Initial release.	20150511