array <typename T>

Fields

array <typename T> has no public fields.

Functions

Return Type	Function Name	Parameters	Description
void	insertAt	uint alIndex, const T &in aValue	Inserts the value at the specified index, shifting existing values to the right.
void	removeAt	uint alIndex	Removes the value at the specified index, shifting values after the index to the left.
void	insertLast	const T &in aValue	Inserts the value into a new index at the end of the array.
void	insertBack	const T &in aValue	***Using this function results in an error.***
void	removeFirst		Removes the value at the beginning of the array.
void	removeLast		Removes the value at the end of the array.
uint	length		Returns the number of elements within the array.
void	resize	uint alLength	Resizes the array to the specified size, creating or removing elements as necessary.
void	sortAsc		Sorts all elements in the array into ascending order.
void	sortAsc	uint alIndex, uint alLength	Sorts elements in the array into ascending order, affecting only the subsection at the given index to the given length.
void	sortDesc		Sorts all elements in the array into descending order.
void	sortDesc	uint alIndex, uint alLength	Sorts elements in the array into descending order, affecting only the subsection at the given index to the given length.
void	reverse		Reverses the order of elements in the array.
int	find	const T &in aValue	Returns the index of the first element in the array equal to the given value, or -1 if the value was not found.
int	find	uint alIndex, const T &in aValue	Returns the index of the first element in the array equal to the given value, or -1 if the value was not found. Only affects elements starting at the given index.
void	push_back	const T &in aValue	Inserts the given value at the end of the array.
void	push_front	const T &in aValue	Inserts the given value at the beginning of the array.
void	pop_back		Removes the element at the end of the array.
void	pop_front		Removes the element at the beginning of the array.
uint	size		Returns the number of elements within the array.

Remarks

The array class is unique in the HPL3 API in that it uses what is called a generic template. What this means is, when you create an array, you need to specify what variable type the array will be holding. This ensures that all objects within a given array are guaranteed to be of that type, eliminating the need for redundant conversions and type checks.

To declare an array, you put the type the array will be holding between a less than operator (<) and a greater than operator (>).

```
// An array of integers
array<int> mvIntegerArray;

// An array of floats
array<float> mvFloatArray;

// An array of strings
array<tString> mvStringArray;
```

There are a number of ways to add values to your array. The simplest way is to use the push_back function. This automatically puts the given value at the end of the array.

```
mvIntegerArray.push_back(5);
mvIntegerArray.push_back(2);
mvIntegerArray.push_back(11);

// The contents of mvIntegerArray: { 5, 2, 11 }
```

Another common way is to assign a value to an existing index within the array, using the square bracket ([ ]) syntax. (Indeces start at 0.)

```
mvIntegerArray[] = 29;
mvIntegerArray[2] = -12;

// The contents of mvIntegerArray: { 29, 2, -12 }
```

To retrieve an element from within the array, use the square bracket ([]) syntax with an integer index.

```
int lIntValue = mvIntegerArray[];

// The value of lIntvalue: 29
```

Trying to assign a value to or retrieve a value from an index that does not exist in the array will result in an error.

```
// If the index at [100] doesn't exist, both of these lines
// of code will throw an error.
mvIntegerArray[100] = 5;
int lIntValue = mvIntegerArray[100];

// However, the following code will work just fine
mvIntegerArray.resize(101);
mvIntegerArray[100] = 5;
```

```
int lIntValue = mvIntegerArray[100];
```