Support Library Reference

September 22, 2014

Contents

Description

Contains run-time support for the Cmajor language implementation – mainly support for exception handling.

Note: You may want to enable Previous View and Next View commands in the PDF viewer for comfortable browsing experience in this document.

Copyrights

Namespaces

Namespace	Description
System.Support	Contains run-time support classes, functions and constants for the Cmajor language implementation.

1 System.Support Namespace

Contains run-time support classes, functions and constants for the Cmajor language implementation.

1.1 Classes

Class	Description
ExCodeChecker	A class for checking the exception code of a user defined destructor just before control leaves it.
ExDeleter<T>	Exception object deleter class.
MtxGuard	A helper class for locking and unlocking a mutex in the mutex table.
StackFrame	The Cmajor compiler inserts an object of class StackFrame in the beginning of a user-defined function. This enables capturing call stacks.

1.1.1 ExCodeChecker Class

A class for checking the exception code of a user defined destructor just before control leaves it.

Syntax

```
public class ExCodeChecker;
```

1.1.1.1 Remarks

The Cmajor compiler inserts a variable of type **ExCodeChecker** in the beginning of each user defined destructor. Each destructor has an implicit exception code integer variable named $\$ex$ that contains a non-zero value if an exception that has not been handled reaches the destructor. In that case the program is terminated with EXIT_UNHANDLED_EXCEPTION status.

1.1.1.2 Member Functions

Member Function	Description
ExCodeChecker(int&)	Constructor. Stores a reference to the implicit exception code integer variable of a user defined destructor.
~ExCodeChecker()	Destructor. If the exception code integer variable of a user defined destructor is not zero, calls DestructorUnhandledException() function to exit the program with EXIT_UNHANDLED_EXCEPTION status.

1.1.1.2.1 ExCodeChecker(int&) Member Function

Constructor. Stores a reference to the implicit exception code integer variable of a user defined destructor.

Syntax

```
public nothrow ExCodeChecker(int& exCode_);
```

Parameters

Name	Type	Description
exCode_	int&	Reference to an exception code integer variable of a user defined destructor.

Implementation

eh.cm, page 2

1.1.1.2.2 ~ExCodeChecker() Member Function

Destructor. If the exception code integer variable of a user defined destructor is not zero, calls DestructorUnhandledException() function to exit the program with EXIT_UNHANDLED_EXCEPTION status.

Syntax

```
public nothrow ~ExCodeChecker();
```

Implementation

eh.cm, page 3

1.1.2 ExDeleter<T> Class

Exception object deleter class.

Syntax

```
public class ExDeleter<T>;
```

1.1.2.1 Remarks

The Cmajor compiler inserts a variable of type ExDeleter<*ExceptionClassType*> at the beginning of a catch block for the *ExceptionClassType*. The destructor of the ExDeleter destroys the exception object no matter how the catch block is exited.

1.1.2.2 Member Functions

Member Function	Description
ExDeleter(T*)	Constructor. Stores a pointer to an exception object.
~ExDeleter()	Destructor. Deletes the exception object.

1.1.2.2.1 ExDeleter(T*) Member Function

Constructor. Stores a pointer to an exception object.

Syntax

```
public nothrow ExDeleter(T* ex_);
```

Parameters

Name	Type	Description
ex_	T*	A pointer to an exception object.

Implementation

eh.cm, page 2

1.1.2.2.2 ~ExDeleter() Member Function

Destructor. Deletes the exception object.

Syntax

```
public nothrow ~ExDeleter();
```

Implementation

eh.cm, page 2

1.1.3 MtxGuard Class

A helper class for locking and unlocking a mutex in the mutex table.

Syntax

```
public class MtxGuard;
```

1.1.3.1 Member Functions

Member Function	Description
MtxGuard(int)	Locks a mutex in the mutex table.
~MtxGuard()	Unlocks a mutex in the mutex table.

1.1.3.1.1 MtxGuard(int) Member Function

Locks a mutex in the mutex table.

Syntax

```
public nothrow MtxGuard(int mid_);
```

Parameters

Name	Type	Description
mid_	int	The number of the mutex to lock.

1.1.3.1.2 ~MtxGuard() Member Function

Unlocks a mutex in the mutex table.

Syntax

```
public nothrow ~MtxGuard();
```

1.1.4 StackFrame Class

The Cmajor compiler inserts an object of class StackFrame in the beginning of a user-defined function. This enables capturing call stacks.

Syntax

```
public class StackFrame;
```

1.1.4.1 Member Functions

Member Function	Description
StackFrame(const char*, const char*, int)	Calls enter_frame() function that stores the function, file and line to call stack information table.
~StackFrame()	Calls leave_frame() function that removes the function, file and line from the call stack information table.

1.1.4.1.1 StackFrame(const char*, const char*, int) Member Function

Calls enter_frame() function that stores the function, file and line to call stack information table.

Syntax

```
public inline nothrow StackFrame(const char* fun, const char* file, int line);
```

Parameters

Name	Type	Description
fun	const char*	Function name.
file	const char*	Source file name.
line	int	Source line.

Implementation

cs.cm, page 1

1.1.4.1.2 ~StackFrame() Member Function

Calls leave_frame() function that removes the function, file and line from the call stack information table.

Syntax

```
public inline nothrow ~StackFrame();
```

Implementation

cs.cm, page 1

1.2 Functions

Function	Description
Align(ulong, ulong)	A utility function for aligning an integer value.
DebugHeapMemAlloc(ulong, const char*, int)	If MEM_LEAK_DETECT symbol is defined, Cmajor compiler replaces calls to MemAlloc function with calls to DebugHeapMemAlloc function, that calls the dbgheap_malloc() function.
DebugHeapMemFree(void*)	If MEM_LEAK_DETECT symbol is defined, Cmajor compiler replaces calls to MemFree function with calls to DebugHeapMemFree function, that calls the dbgheap_free() function.
DestructorUnhandledException()	Writes a message indicating that unhandled exception reached a destructor and terminates the program with EXIT_UNHANDLED_EXCEPTION status.
FailAssertion(const char*, const char*, const char*)	Reports a failed assertion to the standard error and terminates the program with EXIT_FAILURE status.
FailNullPointer(const char*, const char*)	Reports that the tried to call a function through null pointer to the standard error and terminates the program with EXIT_NULL_POINTER status.
GetExceptionAddr(void*, int, int)	Returns address of an exception object for the given exception identifier from the exception table.
HandleThisEx(int*, int, int, int)	Returns true if the exception class with the given exception identifier *catched* catches an exception class with the given exception identifier *thrown*.

MainUnhandledException()	Writes a message indicating that unhandled exception from the user implemented main function reached the compiler implemented main function to the standard error stream and terminates the program with EXIT_UNHANDLED_EXCEPTION status.
MemAlloc(ulong)	Allocates a memory block of the given size and returns a pointer to it. If the allocation fails, reports the insufficient memory condition and terminates the program with the EXIT_INSUFFICIENT_MEMORY status.
MemFree(void*)	Releases a memory block back to system.
MemGrow(ulong)	Implements an initially rapidly growing memory allocation policy.
SetExceptionAddr(void*, int, int, void*)	Sets the address of an exception object with the given exception identifier to the thread's exception table.
StrCopy(char*, const char*)	Copies a C-style string.
StrCopy(char*, const char*, int)	Copies at most given number of characters from a C-style string.
StrLen(const char*)	Returns the length of a C-style string.
Write(int, const char*)	Writes a C-style string to the given stream.

1.2.5 Align(ulong, ulong) Function

A utility function for aligning an integer value.

Syntax

```
public inline nothrow ulong Align(ulong n, ulong alignment);
```

Parameters

Name	Type	Description
n	ulong	An integer to align.
alignment	ulong	An integer alignment.

Returns

ulong

Returns n aligned to the specified *alignment*, that is: $(n + alignment - 1)$ & $- alignment$;

Remarks

Informally: if n is divisible with the *alignment*, returns n; otherwise returns the next multiple of *alignment* that is greater than n.

Implementation

utility.cm, page 1

1.2.6 DebugHeapMemAlloc(ulong, const char*, int) Function

If MEM_LEAK_DETECT symbol is defined, Cmajor compiler replaces calls to MemAlloc function with calls to DebugHeapMemAlloc function, that calls the dbgheap_malloc() function.

Syntax

```
public nothrow void* DebugHeapMemAlloc(ulong size, const char* file, int line);
```

Parameters

Name	Type	Description
size	ulong	Size of the memory block to allocate.
file	const char*	Source file name of the allocation.
line	int	Source line number of the allocation.

Returns

void*

Returns pointer to the allocated memory block.

Implementation

utility.cm, page 3

1.2.7 DebugHeapMemFree(void*) Function

If MEM_LEAK_DETECT symbol is defined, Cmajor compiler replaces calls to MemFree function with calls to DebugHeapMemFree function, that calls the dbgheap_free() function.

Syntax

```
public inline nothrow void DebugHeapMemFree(void* block);
```

Parameters

Name	Type	Description
block	void*	Pointer to the memory block to free.

Implementation

utility.cm, page 3

1.2.8 DestructorUnhandledException() Function

Writes a message indicating that unhandled exception reached a destructor and terminates the program with EXIT_UNHANDLED_EXCEPTION status.

Syntax

```
public nothrow void DestructorUnhandledException();
```

Remarks

Calls the Write(int, const char*) function to write the message to the standard error stream. Then calls the *exit(int)* function of the Os library (see Os.pdf, page 8) to terminate the program.

Implementation

eh.cm, page 2

1.2.9 FailAssertion(const char*, const char*, const char*) Function

Reports a failed assertion to the standard error and terminates the program with EXIT_FAILURE status.

Syntax

```
public nothrow void FailAssertion(const char* expression, const char* file, const char* line);
```

Parameters

Name	Type	Description
expression	const char*	A failed assertion expression.
file	const char*	The source file containing the failed assertion.
line	const char*	The source line number of the failed assertion.

Remarks

Writes a message indicating that the given assertion expression *expression* failed in the given source file *file* at the given source line *line* to the standard error stream of the process using the Write(int, const char*) function and then terminates the program with EXIT_FAILURE status by calling the *exit(int)* function of the Os library (see Os.pdf, page 8). The Cmajor compiler inserts a call to this function to the error branch of an **#assert** statement.

Implementation

assert.cm, page 1

1.2.10 FailNullPointer(const char*, const char*) Function

Reports that the tried to call a function through null pointer to the standard error and terminates the program with EXIT_NULL_POINTER status.

Syntax

```
public nothrow void FailNullPointer(const char* file, const char* line);
```

Parameters

Name	Type	Description
file	const char*	Source file name.
line	const char*	Source line number.

Implementation

nupo.cm, page 1

1.2.11 GetExceptionAddr(void*, int, int) Function

Returns address of an exception object for the given exception identifier from the exception table.

Syntax

```
public nothrow void* GetExceptionAddr(void* exceptionTable, int numExceptions,
int exceptionId);
```

Parameters

Name	Type	Description
exceptionTable	void*	The address of the exception table of the calling thread.
numExceptions	int	The number of entries in the exception table.
exceptionId	int	An exception identifier.

Returns

void*

Returns the address of the exception object with the specified exception identifier.

Remarks

The exception table of a thread is an array of pointers to exception objects. The exception identifier of an exception class is a serial number starting from 1 given by the Cmajor compiler for each exception class thrown or catched by the program or library it uses. Exception identifier minus one is an index to thread's exception table. The Cmajor compiler inserts a call to this function at the beginning of each catch block to resolve the address of the catched exception object.

Implementation

eh.cm, page 2

1.2.12 HandleThisEx(int*, int, int, int) Function

Returns true if the exception class with the given exception identifier *catched* catches an exception class with the given exception identifier *thrown*.

Syntax

```
public nothrow bool HandleThisEx(int* exceptionBaseIdTable, int numExceptions,
int thrown, int catched);
```

Parameters

Name	Type	Description
exceptionBaseIdTable	int*	A pointer to a exception base identifier table.
numExceptions	int	The number of exception identifiers.
thrown	int	The identifier of a thrown exception class.
catched	int	The identifier of catching exception class.

Returns

bool

Returns true if *catched* catches *thrown*, false otherwise.

Remarks

The exception base identifier table of a program is an array of exception identifiers, one for each exception class catched or thrown by the program or library it uses. Each entry gives the identifier of the immediate base class of the exception class, or is 0 if the exception class has no base class (i.e. it is *System.Exception*). The Cmajor compiler inserts a call to this function at the beginning each catch block to find out if this catch branch handles the thrown exception.

Implementation

eh.cm, page 1

1.2.13 MainUnhandledException() Function

Writes a message indicating that unhandled exception from the user implemented main function reached the compiler implemented main function to the standard error stream and terminates the program with EXIT_UNHANDLED_EXCEPTION status.

Syntax

```
public nothrow void MainUnhandledException();
```

Remarks

Calls the Write(int, const char*) function to write the message to the standard error stream. Then calls the *exit(int)* function of the Os library (see Os.pdf, page 8) to terminate the program.

Implementation

eh.cm, page 2

1.2.14 MemAlloc(ulong) Function

Allocates a memory block of the given size and returns a pointer to it. If the allocation fails, reports the insufficient memory condition and terminates the program with the EXIT_INSUFFICIENT_MEMORY status.

Syntax

```
public nothrow void* MemAlloc(ulong size);
```

Parameters

Name	Type	Description
size	ulong	The size of the memory block to allocate.

Returns

void*

Returns a pointer to the allocated memory block.

Remarks

The Cmajor compiler implements the memory allocation of the **new** expression by calling this function. The out-of-memory situation is considered so fatal in the context of Cmajor language that throwing an exception was not considered an option. This reduces also the needed exception handling code for the Cmajor programs and libraries, because **new** does not throw if the constructor of the constructed class does not throw.

Calls the *malloc(ulong)* function of the Os library (see Os.pdf, page 13) to allocate the memory block. If the allocation succeeds, returns a pointer to the allocated memory block. If the allocation fails, calls the Write(int, const char*) function to write the message to the standard error stream. Then calls the *exit(int)* function of the Os library (see Os.pdf, page 8) to terminate the program.

Implementation

utility.cm, page 3

1.2.15 MemFree(void*) Function

Releases a memory block back to system.

Syntax

```
public inline nothrow void MemFree(void* block);
```

Parameters

Name	Type	Description
block	void*	Pointer to the memory block to free.

Implementation

utility.cm, page 3

1.2.16 MemGrow(ulong) Function

Implements an initially rapidly growing memory allocation policy.

Syntax

```
public nothrow ulong MemGrow(ulong size);
```

Parameters

Name	Type	Description
size	ulong	The current number of elements.

Returns

ulong

if *size* is less than 8 returns 8, else if *size* is less than 4096, returns *size* aligned to the next multiple of 8; otherwise returns two times *Align(size,* 4096*)*. Calls the Align(ulong, ulong) function to do the aligning.

Remarks

This allocation policy is used in the System library *String* and *List* classes.

Implementation

utility.cm, page 2

1.2.17 SetExceptionAddr(void*, int, int, void*) Function

Sets the address of an exception object with the given exception identifier to the thread's exception table.

Syntax

```
public nothrow void SetExceptionAddr(void* exceptionTable, int numExceptions, int
exceptionId, void* exception);
```

Parameters

Name	Type	Description
exceptionTable	void*	The address of the exception table of the calling program.
numExceptions	int	The number of entries in the exception table.
exceptionId	int	The identifier for the exception object.
exception	void*	The address of an exception object.

Remarks

The exception table of a thread is an array of pointers to exception objects. The exception identifier of an exception class is a serial number starting from 1 given by the Cmajor compiler for each exception class thrown or catched by the program or library it uses. Exception identifier minus one is an index to the exception table. The Cmajor compiler inserts a call to this function to the implementation of a **throw** statement after it has constructed an exception object in the free store.

Implementation

eh.cm, page 2

1.2.18 StrCopy(char*, const char*) Function

Copies a C-style string.

Syntax

```
public nothrow void StrCopy(char* buf, const char* from);
```

Parameters

Name	Type	Description
buf	char*	Pointer to a buffer to which to copy the string.
from	const char*	A C-style string to copy.

Remarks

The size of *buf* must be at least *StrLen(from) + 1* bytes.

Implementation

utility.cm, page 1

1.2.19 StrCopy(char*, const char*, int) Function

Copies at most given number of characters from a C-style string.

Syntax

```
public nothrow int StrCopy(char* buf, const char* from, int length);
```

Parameters

Name	Type	Description
buf	char*	Pointer to a buffer to which to copy the string.
from	const char*	A C-style string to copy.
length	int	Maximum number of characters to copy.

Returns

int

Returns the number of characters copied.

Remarks

The size of *buf* must be at least *length + 1* bytes.

Implementation

utility.cm, page 1

1.2.20 StrLen(const char*) Function

Returns the length of a C-style string.

Syntax

```
public nothrow int StrLen(const char* s);
```

Parameters

Name	Type	Description
s	const char*	A C-style string.

Returns

int

Returns the length of *s*.

Implementation

utility.cm, page 1

1.2.21 Write(int, const char*) Function

Writes a C-style string to the given stream.

Syntax

```
public nothrow int Write(int fd, const char* s);
```

Parameters

Name	Type	Description
fd	int	The file descriptor of the stream to which to write.
s	const char*	A C-style string to write.

Returns

int

Returns the number of bytes written to the stream, or -1 if an error occurred.

Remarks

Calls the *write(int, void*, uint)* function of the Os library (see Os.pdf, page 18) to write the content of the string to the stream.

Implementation

utility.cm, page 2

1.3 Constants

Constant	Type	Value	Description
EXIT_FAILURE	int	1	A failure exit status of a program.
EXIT_INSUFFICIENT_MEMORY	int	253	An out of memory condition exit status of a program.
EXIT_MUTEX_OP_FAILED	int	249	Mutex table lock or unlock operation failure exit status of the program.
EXIT_NULL_POINTER	int	252	Null pointer call exist status of a program.
EXIT_SEGMENTATION_FAULT	int	255	Program received segmentation fault signal. Not implemented.
EXIT_SUCCESS	int	0	Successful exit status of a program.
EXIT_UNHANDLED_EXCEPTION	int	254	Unhandled exception exit status of a program.