

# TEMU

## *API Reference*

Version 4.0, 2024-05-14



# Table of Contents

1. API .....	3
1.1. Functions .....	3
1.2. Structs and Unions .....	320
1.3. Enumerations .....	411
1.4. Interfaces .....	431

# Chapter 1. API

This is a placeholder.

Most API docs will be extracted with Doxygen

## 1.1. Functions

### 1.1.1. temu\_addArrayProperty

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_addArrayProperty(temu_Class * Cls, const char * Name, int offset, temu_Type  
T, int NumElems, const char * Doc)
```

#### Description

Add a fixed length array property without setters / getters

#### Result

0 if the property was added

#### Parameters

Table 1. temu\_addArrayProperty Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	Class pointer
Name	const char *	Name of property
offset	int	Offset from object pointer
T	temu_Type	Property type
NumElems	int	Array length
Doc	const char *	Documentation string

### 1.1.2. temu\_addArrayPseudoProperty

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_addArrayPseudoProperty(temu_Class * Cls, const char * Name, temu_Type T, int NumElems, const char * Doc)
```

## Description

Add an array pseudo-property

## Result

0 if the property was added

## Parameters

Table 2. `temu_addArrayPseudoProperty` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	Class pointer
Name	<code>const char *</code>	Name of property
T	<code>temu_Type</code>	Property type
NumElems	<code>int</code>	Array length
Doc	<code>const char *</code>	Documentation string

### 1.1.3. `temu_addField`

#### Include

```
#include "temu-c/Support/Register.h"
```

## Signature

```
void temu_addField(temu_Register * R, const char * Name, uint64_t Mask, uint64_t Reset, uint64_t Flags, const char * Doc)
```

## Description

Add field to meta register

Adds a field to the meta register.



## Parameters

Table 3. `temu_addField` Parameters

Parameter	Type	Description
R	<code>temu_Register *</code>	Meta register pointer.
Name	<code>const char *</code>	Name of of field (must be a C-compatible identifier)
Mask	<code>uint64_t</code>	Mask identifying the bits in the register corresponding to the field. Mask must contain consecutive bits only.
Reset	<code>uint64_t</code>	Reset value.
Flags	<code>uint64_t</code>	Flags used: <code>TEMU_FIELD_WR</code> means field is writable, <code>TEMU_FIELD_WARM_RESET</code> means the field is subject to reset actions also on warm resets.
Doc	<code>const char *</code>	Documentation string

### 1.1.4. `temu_addInterfaceArray`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_addInterfaceArray(temu_Class * Cls, const char * IfaceName, const char * IfaceType, void * Iface, size_t Count, size_t Size, const char * Doc)
```

#### Description

Add an array of interfaces to a class

Interface arrays are especially useful when e.g. multiple "network ports" are available, although they can be added manually with distinct names (e.g. `uartiface`, `uartbiface`, etc), the interface array registration allows for the addition of several interfaces at once. Individual interfaces are then referred to with normal index syntax, e.g. `obj:uartiface[0]`.

#### Parameters

Table 4. `temu_addInterfaceArray` Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	Class pointer
IfaceName	const char *	name of interface
IfaceType	const char *	name of interface type
Iface	void *	Pointer to array of interfaces
Count	size_t	Number of interfaces in array
Size	size_t	Size of one interface (e.g. sizeof(uartiface[0]))
Doc	const char *	Documentation comment for interface

### 1.1.5. temu\_addInterfaceReference

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_addInterfaceReference(temu_Class * Cls, const char * PropName, int Offset,
const char * TypeName, int Count, unsigned int Flags, temu_PropWriter Wr,
temu_PropReader Rd, const char * Doc)
```

#### Description

Add a interface reference property.

#### Parameters

Table 5. temu\_addInterfaceReference Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	The class object
PropName	const char *	Name of the property to add.
Offset	int	Offset in bytes from the start of the class to the value the property refers to.
TypeName	const char *	Interface type name.

Parameter	Type	Description
Count	int	Set to > 1 to indicate that the property is an array, the value is the length of the array.
Flags	unsigned int	Flags for property, 1 implies property is mandatory to set.
Wr	temu_PropWriter	Property write function. This function can have side-effects.
Rd	int),temu_PropReader	Property read function. This function can have side-effects.
Doc	const char *	Documentation string

### 1.1.6. temu\_addLoggingCategory

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_addLoggingCategory(temu_Class * Cls, unsigned int CategoryId, const char * Category)
```

#### Description

temu\_addLogginCategory adds a logging category to the class

#### Result

0 for success, 1 indicates failure.

#### Parameters

Table 6. temu\_addLoggingCategory Parameters

Parameter	Type	Description
Cls	temu_Class *	the class
CategoryId	unsigned int	The category id. Valid range is [8,16)
Category	const char *	The category name

### 1.1.7. temu\_addPort

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_addPort(temu_Class * C, const char * IfaceRefName, const char * IfaceName,  
const char * Doc)
```

## Description

Make interface reference property and an interface inverses

A port is a bidirectional interface. Thus, if an interface reference is connected to another objects interface, the remote object will be told to issue a reverse connection. This is convenient as it simplifies device connection during system configuration and construction. That is, instead of calling the connect two times with the inverse connections explicitly, only one call is needed.

## Result

0 on success, non-zero otherwise.

## Parameters

Table 7. `temu_addPort` Parameters

Parameter	Type	Description
C	<code>temu_Class *</code>	The class object
IfaceRefName	<code>const char *</code>	the name of the interface reference property
IfaceName	<code>const char *</code>	Name of interface that is the inverse of the interface ref property.
Doc	<code>const char *</code>	Documentation string

### 1.1.8. `temu_addProperty`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_addProperty(temu_Class * Cls, const char * PropName, int Offset, temu_Type
```

```
Typ, int Count, temu_PropWriter Wr, temu_PropReader Rd, const char * Doc)
```

## Description

Add a property to a class.

## Parameters

Table 8. `temu_addProperty` Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class *</a>	The class object
PropName	const char *	Name of the property to add.
Offset	int	Offset in bytes from the start of the class to the value the property refers to.
Typ	temu_Type	Type of the property.
Count	int	Set to > 1 to indicate that the property is an array, the value is the length of the array.
Wr	temu_PropWriter	Property write function. This function can have side-effects.
Rd	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Property read function. This function can have side-effects.
Doc	const char *	Documentation string

### 1.1.9. temu\_addPseudoInterfaceReference

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_addPseudoInterfaceReference(temu_Class * Cls, const char * PropName, const char * TypeName, int Count, unsigned int Flags, temu_PropWriter Wr, temu_PropReader Rd, temu_PropWriter Set, temu_PropReader Get, const char * Doc)
```

## Description

Add an interface reference property.

## Parameters

Table 9. `temu_addPseudoInterfaceReference` Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	The class object
PropName	const char *	Name of the property to add.
TypeName	const char *	Interface type name.
Count	int	Set to > 1 to indicate that the property is an array, the value is the length of the array.
Flags	unsigned int	Flags for property, 1 implies property is mandatory to set.
Wr	temu_PropWriter	Property write function. This function can have side-effects.
Rd	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Property read function. This function can have side-effects.
Set	temu_PropWriter	Property set function. Non-semantic.
Get	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Property get function. Non-semantic.
Doc	const char *	Documentation string

### 1.1.10. `temu_addPseudoProperty`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_addPseudoProperty(temu_Class * Cls, const char * PropName, temu_Type Typ,
int Count, temu_PropWriter Wr, temu_PropReader Rd, temu_PropWriter Set,
temu_PropReader Get, const char * Doc)
```

#### Description

Add property without storage

A pseudo property does not have backing storage (and therefore no offset). They exist to provide access to computed properties.

Pseudo properties also work fine with C++ types with non-standard layout.

Pseudo properties are snapshotted if the set and get functions are implemented. The set and get should implement read and write without semantic effect. That is, if the property is a register, then the read and write should have the effect of an actual register access (including event rescheduling etc), while for a get/set, only the value of the register is modified, event restoration is done differently.

## Parameters

Table 10. `temu_addPseudoProperty` Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	class to add pseudo property to
PropName	const char *	name of pseudo property
Typ	temu_Type	Type of property
Count	int	number of elements, 1 for a scalar.
Wr	temu_PropWriter	Property write function (semantic effect)
Rd	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Property read function (semantic effect)
Set	temu_PropWriter	Property set function (non-semantic)
Get	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Property get function (non-semantic)
Doc	const char *	Documentation string

### 1.1.11. `temu_addRegister`

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
temu_Register * temu_addRegister(temu_RegisterBank * Bank, const char * Name, int
Offset, temu_Type Typ, int Count, temu_PropWriter Wr, temu_PropReader Rd, const char *
Doc, uint32_t DeviceOffset, uint32_t Stride)
```

#### Description

## Add register property to class

Adds a register with the given name to a class. It returns a reference to a meta register which can be used to add fields.

### Result

A reference to a meta register which can be used to add fields.

### Parameters

Table 11. `temu_addRegister` Parameters

Parameter	Type	Description
Bank	<a href="#">temu_RegisterBank</a> *	Register bank to add register to
Name	const char *	Name of register (must be a valid C-identifier)
Offset	int	Offset to storage element in the device struct.
Typ	temu_Type	Type of register, note that registers are limited to unsigned integer types with fixed width.
Count	int	Number of registers (normally 1)
Wr	temu_PropWriter	Register write function
Rd	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Register read function
Doc	const char *	Documentation string for register.
DeviceOffset	uint32_t	Offset of register in memory system. This is the same offset that is used in the memory transaction interface.
Stride	uint32_t	In case the register is an array of registers, then the stride is used for the offset in physical address space between each register.

## 1.1.12. `temu_addRegisterBank`

### Include



```
#include "temu-c/Support/Register.h"
```

### Signature

```
temu_RegisterBank * temu_addRegisterBank(temu_Class * C, const char * Name,  

  temu_MemAccessIface * MemAccessIface)
```

### Description

Adds a register bank to a TEMU class

### Result

Pointer to the new register bank

### Parameters

Table 12. `temu_addRegisterBank` Parameters

Parameter	Type	Description
C	<a href="#">temu_Class</a> *	Pointer to the TEMU class
Name	const char *	Name of the register
MemAccessIface	<a href="#">temu_MemAccessIface</a> *	Pointer to the register memory access interface

## 1.1.13. temu\_addScalarProperty

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_addScalarProperty(temu_Class * Cls, const char * Name, int offset, temu_Type  

  T, const char * Doc)
```

### Description

Add a scalar (one element) property without setters / getters

### Result

0 if the property was added

## Parameters

Table 13. `temu_addScalarProperty` Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	Class pointer
Name	const char *	Name of property
offset	int	Offset from object pointer
T	temu_Type	Property type
Doc	const char *	Documentation string

### 1.1.14. `temu_addScalarPseudoProperty`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_addScalarPseudoProperty(temu_Class * Cls, const char * Name, temu_Type T,
const char * Doc)
```

#### Description

Add a scalar (one element) pseudo-property

#### Result

0 if the property was added

#### Parameters

Table 14. `temu_addScalarPseudoProperty` Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	Class pointer
Name	const char *	Name of property
T	temu_Type	Property type
Doc	const char *	Documentation string

### 1.1.15. `temu_armGetAPSR`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_armGetAPSR(void * Cpu)
```

### Description

Get the ARM processor state register

### Result

APSR value

### Parameters

Table 15. `temu_armGetAPSR` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer

## 1.1.16. `temu_armGetExecMode`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
unsigned int temu_armGetExecMode(void * Cpu)
```

### Description

Get the ARM execution mode

### Result

Execution mode

### Parameters

Table 16. `temu_armGetExecMode` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer

## 1.1.17. temu\_armGetFPEXC

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_armGetFPEXC(void * Cpu)
```

### Description

Get the ARM FPEXC register

### Result

FPEXC value

### Parameters

Table 17. temu\_armGetFPEXC Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer

## 1.1.18. temu\_armGetFPINST

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_armGetFPINST(void * Cpu, int Idx)
```

### Description

Get the ARM FPINST register

### Result

FPINST value

### Parameters

Table 18. `temu_armGetFPINST` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer
Idx	int	FPINST register number

### 1.1.19. `temu_armGetFPSCR`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint32_t temu_armGetFPSCR(void * Cpu)
```

#### Description

Get the ARM FPSCR register

#### Result

FPSCR value

#### Parameters

Table 19. `temu_armGetFPSCR` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer

### 1.1.20. `temu_armSetAPSR`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_armSetAPSR(void * Cpu, uint32_t Value)
```

#### Description

Set the ARM processor state register

## Parameters

Table 20. `temu_armSetAPSR` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer
Value	uint32_t	Value to set the APSR to

### 1.1.21. `temu_armSetExecMode`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_armSetExecMode(void * Cpu, unsigned int Mode)
```

#### Description

Set the ARM execution mode

#### Parameters

Table 21. `temu_armSetExecMode` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer
Mode	unsigned int	Execution mode

### 1.1.22. `temu_armSetFPEXC`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_armSetFPEXC(void * Cpu, uint32_t Value)
```

#### Description

Set the ARM FPEXC register

## Parameters

Table 22. `temu_armSetFPEXC` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer
Value	uint32_t	FPEXC value

### 1.1.23. `temu_armSetFPINST`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_armSetFPINST(void * Cpu, int Idx, uint32_t Value)
```

#### Description

Set the ARM FPINST register

#### Parameters

Table 23. `temu_armSetFPINST` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer
Idx	int	FPINST register number
Value	uint32_t	FPINST value

### 1.1.24. `temu_armSetFPSCR`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_armSetFPSCR(void * Cpu, uint32_t Value)
```

#### Description

## Set the ARM FPSCR register

### Parameters

Table 24. `temu_armSetFPSCR` Parameters

Parameter	Type	Description
Cpu	void *	ARM CPU pointer
Value	uint32_t	FPSCR value

### 1.1.25. `temu_asDouble`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
double temu_asDouble(temu_Propval Pv)
```

#### Description

Converts the given property to double

#### Result

Returns the property as an unsigned integer

#### Parameters

Table 25. `temu_asDouble` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be converted

### 1.1.26. `temu_asInteger`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int64_t temu_asInteger(temu_Propval Pv)
```



## Description

Converts the given property to integer

## Result

Returns the property as integer

## Parameters

Table 26. `temu_asInteger` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be converted

### 1.1.27. `temu_asUnsigned`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
uint64_t temu_asUnsigned(temu_Propval Pv)
```

#### Description

Converts the given property to unsigned integer

#### Result

Returns the property as an unsigned integer

#### Parameters

Table 27. `temu_asUnsigned` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be converted

### 1.1.28. `temu_assemble`

#### Include

```
#include "temu-c/Support/Assembler.h"
```

## Signature

```
uint32_t temu_assemble(void * Cpu, const char * AsmStr)
```

## Description

Assemble an instruction

## Result

Instruction bitpattern.

## Parameters

Table 28. `temu_assemble` Parameters

Parameter	Type	Description
Cpu	void *	processor for which to assemble.
AsmStr	const char *	String with instruction in the CPUs assembler.

### 1.1.29. `temu_assembleToMemory`

## Include

```
#include "temu-c/Support/Assembler.h"
```

## Signature

```
void temu_assembleToMemory(void * Cpu, const char * AsmStr, uint64_t Addr)
```

## Description

Assemble an instruction to memory

## Parameters

Table 29. `temu_assembleToMemory` Parameters

Parameter	Type	Description
Cpu	void *	processor for which to assemble.
AsmStr	const char *	String with instruction in the CPUs assembler.

Parameter	Type	Description
Addr	uint64_t	Physical address of where to put the instruction.

### 1.1.30. temu\_asyncSocketAdd

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
int temu_asyncSocketAdd(temu_TimeSource * TS, int Sock, unsigned int Flags, void (*)(void *) CB, void * Data)
```

#### Description

Add asynchronous event triggered by file descriptor changes

#### Result

Returns the file descriptor (Sock) if successful, otherwise -1.

#### Parameters

Table 30. temu\_asyncSocketAdd Parameters

Parameter	Type	Description
TS	temu_TimeSource *	the synchronous queue or time source object (normally a CPU object)
Sock	int	File descriptor for the socket. This can also be a pipe FD. Note that the function is likely to be renamed. Note that in case you are intending to read from the socket, make sure it is non-blocking.
Flags	unsigned int	TEMU_ASYNC_READ in case you wish to be notified about data available to read.

### 1.1.31. temu\_asyncSocketRemove

## Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_asyncSocketRemove(int Fd, unsigned int Flags)
```

## Description

Remove an asynchronous event triggered by file descriptor changes

## Parameters

Table 31. `temu_asyncSocketRemove` Parameters

Parameter	Type	Description
Fd	int	The ID of the socket to remove
Flags	unsigned int	The flags of removing

## 1.1.32. temu\_asyncStop

### Include

```
#include "temu-c/Support/Scheduler.h"
```

### Signature

```
void temu_asyncStop()
```

### Description

Stop the current scheduler

The function returns immediately, before the scheduler has stopped. The function is suitable for use in e.g. signal handlers.

## 1.1.33. temu\_asyncTimerAdd

### Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
int temu_asyncTimerAdd(temu_TimeSource * TS, double T, unsigned int Flags, void (*
)(void *) CB, void * Data)
```

## Description

Add asynchronously activated wall-clock timed events

The callback function will be called synchronously by the emulator core associated with Q. That means that when CB is executing it is safe to do anything that can be done from a normal event or MMIO handler.

Note that the event is only called when a CPU core or machine is running. When the timer has triggered, it is temporarily disabled and the CB is posted on the synchronuous event queue as an async event. This will be executed when the next normal event expires (e.g. at the end of the current quanta). After the event has been called, depending on whether the timer is cyclic or not, it will be reactivated. This means that if the emulator is paused, at most one call to the event handler will be issued, and this will be done when the emulator is resumed.

## Result

Returns a file descriptor or timer ID.

## Parameters

Table 32. `temu_asyncTimerAdd` Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource *</a>	The time source object (normally a CPU object)
T	double	Delta seconds in the future for the first event to be posted.
Flags	unsigned int	Set to <code>TEMU_ASYNC_CYCLIC</code> if the event should be executed as a cyclic event.
CB	void (*)(void *)	Call back function on when timer expires
Data	void *	Context data to be passed to the callback function

## 1.1.34. `temu_asyncTimerRemove`

### Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_asyncTimerRemove(int Fd)
```

## Description

Remove an async timer

## Parameters

Table 33. `temu_asyncTimerRemove` Parameters

Parameter	Type	Description
Fd	int	Timer ID to remove

## 1.1.35. `temu_bitreverse16`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint16_t temu_bitreverse16(uint16_t Word)
```

## Description

Reverses bits in a 16 bit word.



that this should be replaced with `__buintin_bitreverse16()`, but that is only supported in clang at present, not GCC.

## Result

Word with reversed bits.

## Parameters

Table 34. `temu_bitreverse16` Parameters

Parameter	Type	Description
Word	uint16_t	The word to reverse the bits in

## 1.1.36. temu\_bitreverse32

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
uint32_t temu_bitreverse32(uint32_t Word)
```

### Description

Reverses bits in a 32 bit word.



that this should be replaced with `__buintin_bitreverse32()`, but that is only supported in clang at present, not GCC.

### Result

Word with reversed bits.

### Parameters

Table 35. `temu_bitreverse32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to reverse the bits in

## 1.1.37. temu\_buffCopy

### Include

```
#include "temu-c/Support/Buffer.h"
```

### Signature

```
temu_Buff temu_buffCopy(const temu_Buff * B)
```

### Description

Copy COW buffer. This will make a new buff object, but the underlying data will not be copied.

### Result

Buffer object referring to the same data as B.

## Parameters

Table 36. `temu_buffCopy` Parameters

Parameter	Type	Description
B	const <code>temu_Buff *</code>	The buffer to copy.

### 1.1.38. `temu_buffCreate`

#### Include

```
#include "temu-c/Support/Buffer.h"
```

#### Signature

```
temu_Buff temu_buffCreate(uint32_t size)
```

#### Description

Create a new COW buffer of size bytes.

#### Result

Buffer object.

#### Parameters

Table 37. `temu_buffCreate` Parameters

Parameter	Type	Description
size	uint32_t	Size of buffer in bytes

### 1.1.39. `temu_buffDispose`

#### Include

```
#include "temu-c/Support/Buffer.h"
```

#### Signature

```
void temu_buffDispose(temu_Buff * B)
```

#### Description

Delete buffer. The buffer B will be deleted. If there are no more copies of the buffer data anywhere,



the data will be deallocated.

## Parameters

Table 38. `temu_buffDispose` Parameters

Parameter	Type	Description
B	<code>temu_Buff *</code>	buffer to delete.

### 1.1.40. `temu_buffLen`

#### Include

```
#include "temu-c/Support/Buffer.h"
```

#### Signature

```
uint32_t temu_buffLen(const temu_Buff * B)
```

#### Description

Get buffer length

#### Result

Length of buffer in bytes

#### Parameters

Table 39. `temu_buffLen` Parameters

Parameter	Type	Description
B	<code>const temu_Buff *</code>	buffer

### 1.1.41. `temu_buffReadableData`

#### Include

```
#include "temu-c/Support/Buffer.h"
```

#### Signature

```
const uint8_t * temu_buffReadableData(const temu_Buff * B)
```

## Description

Get a pointer to readable buffer data

The pointer to the readable data will be returned.

## Result

Pointer to data buffer.

## Parameters

Table 40. `temu_buffReadableData` Parameters

Parameter	Type	Description
B	const <code>temu_Buff *</code>	Buffer to get read pointer from

### 1.1.42. `temu_buffRemoveHead`

#### Include

```
#include "temu-c/Support/Buffer.h"
```

#### Signature

```
void temu_buffRemoveHead(temu_Buff * B, uint32_t len)
```

#### Description

Remove len bytes from the start of the buffer.

Removing bytes from the buffer start will not change the underlying buffer object. And is not seen as a write for the purpose of the data block.

This does not affect other copies of this buffer.

#### Parameters

Table 41. `temu_buffRemoveHead` Parameters

Parameter	Type	Description
B	<code>temu_Buff *</code>	buffer to remove data from
len	<code>uint32_t</code>	Number of bytes to remove.

### 1.1.43. `temu_buffRemoveTail`

#### Include

```
#include "temu-c/Support/Buffer.h"
```

### Signature

```
void temu_buffRemoveTail(temu_Buff * B, uint32_t len)
```

### Description

Remove len bytes from the end of the buffer.

Removing bytes from the buffer tail will not change the underlying buffer object. And is not seen as a write for the purpose of the data block.

This does not affect other copies of this buffer.

### Parameters

Table 42. `temu_buffRemoveTail` Parameters

Parameter	Type	Description
B	<code>temu_Buff *</code>	buffer to remove data from
len	<code>uint32_t</code>	Number of bytes to remove.

## 1.1.44. `temu_buffWritableData`

### Include

```
#include "temu-c/Support/Buffer.h"
```

### Signature

```
uint8_t * temu_buffWritableData(temu_Buff * B)
```

### Description

Get a pointer to writable data.

If the data have more than one reference, a new copy of the data will be created.

### Result

Pointer to writable data.

### Parameters

Table 43. `temu_buffWritableData` Parameters

Parameter	Type	Description
B	<code>temu_Buff *</code>	Buffer to get data pointer from.

### 1.1.45. `temu_checkSanity`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_checkSanity(int Report)
```

#### Description

Check object system sanity.

The function traverses all objects and their properties and ensures that all (scalar) interface properties are connected. An object can override the default check by implementing the `checkSanity` function in the `ObjectIface`.

#### Result

Returns 0 if the object system is connected. Returns non-zero if the object system is not properly connected.

#### Parameters

Table 44. `temu_checkSanity` Parameters

Parameter	Type	Description
Report	int	set to 1 to have the function report connectivity issues on stdout, 2 to report on stderr, and 0 to not report at all.

### 1.1.46. `temu_checkpointGetLength`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_checkpointGetLength(void * Ctxt, const char * Name)
```

### Description

Get number of entries for the serialized property

### Result

Length of the property in elements. Negative on error.

### Parameters

Table 45. `temu_checkpointGetLength` Parameters

Parameter	Type	Description
Ctxt	void *	Context passed to deserialise function
Name	const char *	Name of property / key in the serialized file

## 1.1.47. temu\_checkpointGetValue

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_Proval temu_checkpointGetValue(void * Ctxt, const char * Name, int Idx)
```

### Description

Get value for named snapshot value

### Result

Property value with the contents saved to the snapshot

### Parameters

Table 46. `temu_checkpointGetValue` Parameters

Parameter	Type	Description
Ctxt	void *	Context is passed to deserialise interface function

Parameter	Type	Description
Name	const char *	Name of the saved value
Idx	int	Index of the saved value (if array)

### 1.1.48. temu\_classCmdAddParam

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
int temu_classCmdAddParam(void * Cmd, const char * Name, temu_CmdOptionKind Type, int Required, const char * Doc)
```

#### Description

Add parameter to command

#### Result

0 on success, other values indicates errors

#### Parameters

Table 47. temu\_classCmdAddParam Parameters

Parameter	Type	Description
Cmd	void *	Command handle
Name	const char *	Name of parameter to add
Type	temu_CmdOptionKind	Type of the parameter
Required	int	Set to 1 if the parameter must be set for the command
Doc	const char *	Documentation string for the parameter

### 1.1.49. temu\_classCmdGetOption

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

## Signature

```
const temu_CmdArg * temu_classCmdGetOption(int argc, const temu_CmdArg * args, const char * OptName)
```

## Description

Get named command option

## Result

Pointer to command argument or NULL in case it is not found.

## Parameters

Table 48. `temu_classCmdGetOption` Parameters

Parameter	Type	Description
argc	int	Number of arguments
args	const temu_CmdArg *	Argument vector
OptName	const char *	Name of parameter / option to get

### 1.1.50. `temu_classCmdGetOptionAsInteger`

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

## Signature

```
int64_t temu_classCmdGetOptionAsInteger(int argc, const temu_CmdArg * args, const char * OptName)
```

## Description

Get named command option as signed integer

Can only get valid numeric parameters. The function will never fail if the type is numeric and the option is required. For optional arguments, use the generic access function `temu_classCmdGetOption()`

## Result

The value of the parameter converted to an integer.

## Parameters

Table 49. `temu_classCmdGetOptionAsInteger` Parameters

Parameter	Type	Description
<code>argc</code>	<code>int</code>	Number of arguments
<code>args</code>	<code>const temu_CmdArg *</code>	Argument vector
<code>OptName</code>	<code>const char *</code>	Name of parameter / option to get

### 1.1.51. `temu_classCmdGetOptionAsUnsigned`

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
uint64_t temu_classCmdGetOptionAsUnsigned(int argc, const temu_CmdArg * args, const char * OptName)
```

#### Description

Get named command option as unsigned integer

Can only get valid numeric parameters. The function will never fail if the type is numeric and the option is required. For optional arguments, use the generic access function `temu_classCmdGetOption()`

#### Result

The value of the parameter converted to an unsigned integer.

#### Parameters

Table 50. `temu_classCmdGetOptionAsUnsigned` Parameters

Parameter	Type	Description
<code>argc</code>	<code>int</code>	Number of arguments
<code>args</code>	<code>const temu_CmdArg *</code>	Argument vector
<code>OptName</code>	<code>const char *</code>	Name of parameter / option to get

### 1.1.52. `temu_classForName`



## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
temu_Class * temu_classForName(const char * ClsName)
```

## Description

Get the class pointer for the named class

## Result

Pointer to the class if found, otherwise NULL

## Parameters

Table 51. `temu_classForName` Parameters

Parameter	Type	Description
ClsName	const char *	The name of the class

### 1.1.53. `temu_classForObject`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
temu_Class * temu_classForObject(const temu_Object * Obj)
```

## Description

Get the class for the given object.

## Result

Pointer to the class type object

## Parameters

Table 52. `temu_classForObject` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object

### 1.1.54. temu\_classGetCmd

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
void * temu_classGetCmd(temu_Class * Cls, const char * Name)
```

#### Description

Get an already registered command handle from class

This function can be used to for example get commands registered in the metaclass, such as the new command. The command can then have arguments added.

#### Result

Opaque handle to command. NULL in case of failure.

#### Parameters

Table 53. temu\_classGetCmd Parameters

Parameter	Type	Description
Cls	temu_Class *	Class pointer
Name	const char *	Name of command to get

### 1.1.55. temu\_classHasCommand

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_classHasCommand(const temu_Class * Cls, const char * CmdName)
```

#### Description

Query whether the class has a specific command implemented

## Result

0 if command is not implemented by class, 1 if implemented.

## Parameters

Table 54. `temu_classHasCommand` Parameters

Parameter	Type	Description
Cls	const <code>temu_Class</code> *	class to query for a command.
CmdName	const char *	Command name class is expected to implement.

## 1.1.56. `temu_clearLeftmostBit32`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
uint32_t temu_clearLeftmostBit32(uint32_t Word)
```

### Description

Clear lower set bit in word

### Result

A copy of Word with ther lowest set bit cleared

### Parameters

Table 55. `temu_clearLeftmostBit32` Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which the lower set bit will be cleared

## 1.1.57. `temu_clearLeftmostBit64`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint64_t temu_clearLeftmostBit64(uint64_t Word)
```

## Description

Clear lower set bit in word

## Result

A copy of Word with ther lowest set bit cleared

## Parameters

Table 56. `temu_clearLeftmostBit64` Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which the lower set bit will be cleared

## 1.1.58. `temu_clearMemAttr`

### Include

```
#include "temu-c/Memory/Memory.h"
```

## Signature

```
void temu_clearMemAttr(void * Obj, uint64_t Addr, uint64_t Len, temu_MemoryAttr Attr)
```

## Description

Clear attribute on memory space location

## Parameters

Table 57. `temu_clearMemAttr` Parameters

Parameter	Type	Description
Obj	void *	The memory space object
Addr	uint64_t	Physical address where to map the device

Parameter	Type	Description
Len	uint64_t	Length in bytes of area where the attribute should be set.
Attr	temu_MemoryAttr	The attribute to clear.

### 1.1.59. temu\_clearRightMostBits32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_clearRightMostBits32(uint32_t Word)
```

#### Description

Clear right most bit in a word

#### Result

A copy of Word, in which right most bit to be cleared

#### Parameters

Table 58. temu\_clearRightMostBits32 Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which right most bit to be cleared

### 1.1.60. temu\_clearRightMostBits64

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_clearRightMostBits64(uint64_t Word)
```

#### Description

Clear right most bit in a word

## Result

A copy of Word, in which right most bit to be cleared

## Parameters

Table 59. `temu_clearRightMostBits64` Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which right most bit to be cleared

### 1.1.61. `temu_clz16`

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
int temu_clz16(uint16_t Word)
```

#### Description

Count leading zeroes

#### Result

Number of leading zeros in Word

#### Parameters

Table 60. `temu_clz16` Parameters

Parameter	Type	Description
Word	uint16_t	the word, in which leading zeros to be counted

### 1.1.62. `temu_clz32`

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
int temu_clz32(uint32_t Word)
```

## Description

Count leading zeroes

## Result

Number of leading zeros in Word

## Parameters

Table 61. `temu_clz32` Parameters

Parameter	Type	Description
Word	uint32_t	the word, in which leading zeros to be counted

## 1.1.63. `temu_clz64`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
int temu_clz64(uint64_t Word)
```

## Description

Count leading zeroes

## Result

The number of leading zeros in Word

## Parameters

Table 62. `temu_clz64` Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which leading zeros to be counted

## 1.1.64. temu\_cmdAddOption

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
void temu_cmdAddOption(void * Cmd, const char * OptName, temu_CmdOptionKind Type, int  

  Required, const char * Doc, const char * Default)
```

### Description

Add named argument to command

### Parameters

Table 63. temu\_cmdAddOption Parameters

Parameter	Type	Description
Cmd	void *	Pointer to the command object
OptName	const char *	Option name
Type	temu_CmdOptionKind	Option type
Required	int	Pass 0 if the option is not required, otherwise required
Doc	const char *	Option documentation
Default	const char *	Default value of the option

## 1.1.65. temu\_cmdGetData

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
void * temu_cmdGetData(void * Ctxt)
```

### Description

Get data pointer from command context This function shall be called in a command handler on the passed context.



## Result

data pointer

## Parameters

Table 64. `temu_cmdGetData` Parameters

Parameter	Type	Description
Ctxt	void *	Pointer to the context of the command

### 1.1.66. `temu_cmdGetInterpreter`

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
void * temu_cmdGetInterpreter(void * Ctxt)
```

#### Description

Get pointer to the command interpreter This function shall be called in a command handler on the passed context.

#### Result

Pointer to interpreter

#### Parameters

Table 65. `temu_cmdGetInterpreter` Parameters

Parameter	Type	Description
Ctxt	void *	Pointer to the context of the command

### 1.1.67. `temu_cmdGetOptionAsInteger`

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
int64_t temu_cmdGetOptionAsInteger(void * Ctxt, const char * OptName)
```

### Description

Get named option as integer from command context This function shall be called in a command handler on the passed context.

### Result

The integer bound to the named argument

### Parameters

Table 66. `temu_cmdGetOptionAsInteger` Parameters

Parameter	Type	Description
Ctxt	void *	Command context
OptName	const char *	Option name

## 1.1.68. temu\_cmdGetOptionAsObject

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
void * temu_cmdGetOptionAsObject(void * Ctxt, const char * OptName)
```

### Description

Get named option as object pointer from command context This function shall be called in a command handler on the passed context.

### Result

Pointer to the option object

### Parameters

Table 67. `temu_cmdGetOptionAsObject` Parameters

Parameter	Type	Description
Ctxt	void *	Command context
OptName	const char *	Option name

## 1.1.69. temu\_cmdGetOptionAsReal

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
double temu_cmdGetOptionAsReal(void * Ctxt, const char * OptName)
```

### Description

Get named option as double from command context This function shall be called in a command handler on the passed context.

### Result

The real value of the option as double

### Parameters

Table 68. temu\_cmdGetOptionAsReal Parameters

Parameter	Type	Description
Ctxt	void *	Command context
OptName	const char *	Option name

## 1.1.70. temu\_cmdGetOptionAsString

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
const char * temu_cmdGetOptionAsString(void * Ctxt, const char * OptName)
```

### Description

Get named option as string from command context This function shall be called in a command handler on the passed context.

### Result

C-string of the name of the option

## Parameters

Table 69. `temu_cmdGetOptionAsString` Parameters

Parameter	Type	Description
Ctxt	void *	Command context
OptName	const char *	Option name

### 1.1.71. `temu_cmdGetPosOpt`

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
const char * temu_cmdGetPosOpt(void * Ctxt, size_t Idx)
```

#### Description

Get positional option at index This function shall be called in a command handler on the passed context.

#### Result

The option at position Idx as a C-string

#### Parameters

Table 70. `temu_cmdGetPosOpt` Parameters

Parameter	Type	Description
Ctxt	void *	Command context
Idx	size_t	Index of the option

### 1.1.72. `temu_cmdGetPosOptSize`

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
size_t temu_cmdGetPosOptSize(void * Ctxt)
```

### Description

Get number of positional options given This function shall be called in a command handler on the passed context.

### Result

The number of position optionals in Ctxt

### Parameters

Table 71. `temu_cmdGetPosOptSize` Parameters

Parameter	Type	Description
Ctxt	void *	Command context

## 1.1.73. temu\_cmdGetVariable

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
const char * temu_cmdGetVariable(const char * Key)
```

### Description

Get a variable in the command line

### Result

In case the variable is not found NULL, otherwise a borrowed string.

### Parameters

Table 72. `temu_cmdGetVariable` Parameters

Parameter	Type	Description
Key	const char *	Variable name

## 1.1.74. temu\_cmdOptionIsValid

### Include

```
#include "temu-c/Support/CommandLine.h"
```

## Signature

```
int temu_cmdOptionIsValid(void * Ctxt, const char * OptName)
```

## Description

Return 1 if the option is valid, 0 if invalid / not set This function shall be called in a command handler on the passed context.

## Result

1 if the option is valid, 0 if invalid / not set

## Parameters

Table 73. `temu_cmdOptionIsValid` Parameters

Parameter	Type	Description
Ctxt	void *	Command context
OptName	const char *	Option name

## 1.1.75. `temu_cmdSetVariable`

### Include

```
#include "temu-c/Support/CommandLine.h"
```

## Signature

```
int temu_cmdSetVariable(const char * Key, const char * Value)
```

## Description

Set a variable in the command line

## Result

Non-zero on errors

## Parameters

Table 74. `temu_cmdSetVariable` Parameters

Parameter	Type	Description
Key	const char *	Variable name (must match [A-Za-z_][A-Za-z0-9_]*)
Value	const char *	Value to assign to variable

### 1.1.76. temu\_componentAddDelegateIface

#### Include

```
#include "temu-c/Support/Component.h"
```

#### Signature

```
void temu_componentAddDelegateIface(temu_Component * Comp, const char * Name,
temu_IfaceRef Iface)
```

#### Description

Add delegate interface to component

Delegate interfaces are interfaces in objects internal in the component. It is possible to use the connect function to attach directly to a component delegated interface (although, in practice the connection is done to the internal object's interface).

Note that delegate interface do not support interface arrays, and can only expose a single interface instance at present.

#### Parameters

Table 75. temu\_componentAddDelegateIface Parameters

Parameter	Type	Description
Comp	temu_Component *	the component to add the delegate interface to.
Name	const char *	name of the delegate interface
Iface	temu_IfaceRef	Interface reference which the delegated interface is resolved to

### 1.1.77. temu\_componentAddDelegateProp

#### Include

```
#include "temu-c/Support/Component.h"
```

## Signature

```
void temu_componentAddDelegateProp(temu_Component * Comp, const char * Name,  
temu_Object * Obj, const char * PropName)
```

## Description

Add delegate property to component

Delegate property are properties in objects internal in the component. It is possible to use the connect function to connect from a delegated property directly using the component instead of the underlying object.

## Parameters

Table 76. `temu_componentAddDelegateProp` Parameters

Parameter	Type	Description
Comp	<code>temu_Component *</code>	the component to add the delegate interface to.
Name	<code>const char *</code>	name of the delegate property
Obj	<code>temu_Object *</code>	Object to which the property resolves to.
PropName	<code>const char *</code>	Name of property in Obj

## 1.1.78. temu\_componentCreate

### Include

```
#include "temu-c/Support/Component.h"
```

### Signature

```
temu_Component * temu_componentCreate(const char * Name)
```

### Description

Allocate a component object.

The component create shall be called in the component constructor/create function (the create function passed to the registerComponent function). It will allocate an opaque component object



with the relevant name.

The returned component is what the component constructor should return.

## Result

Component pointer

## Parameters

Table 77. `temu_componentCreate` Parameters

Parameter	Type	Description
Name	const char *	Name of component

## 1.1.79. `temu_componentDispose`

### Include

```
#include "temu-c/Support/Component.h"
```

### Signature

```
void temu_componentDispose(void * Comp)
```

### Description

Deallocate a component object

The component dispose shall be called by the component destructor/dispose function registered in the registerComponent call.

Note that a component is seen as owning all the objects created with createComponentObject, and subsequently, all objects created will be recursively deleted when deleting the component.

### Parameters

Table 78. `temu_componentDispose` Parameters

Parameter	Type	Description
Comp	void *	Component to dispose.

## 1.1.80. `temu_componentGetDelegatelface`

### Include

```
#include "temu-c/Support/Component.h"
```

## Signature

```
temu_IfaceRef temu_componentGetDelegateIface(temu_Component * Comp, const char * Name)
```

## Description

Query the component for a delegated interface

Normally this function is not needed for the end user, however some usecases can be seen for exposing this. It returns the IfaceRef that has been added as a delegate interface. The main use is to redelegate delegated interfaces in a component of components.

## Result

Interface reference associated with Name, if none is found, iref type will be teTY\_Invalid.

## Parameters

Table 79. `temu_componentGetDelegateIface` Parameters

Parameter	Type	Description
Comp	<a href="#">temu_Component *</a>	The component to query the IfaceRef from
Name	const char *	Name of the interface reference.

### 1.1.81. `temu_componentGetDelegateProp`

#### Include

```
#include "temu-c/Support/Component.h"
```

## Signature

```
temu_PropName temu_componentGetDelegateProp(temu_Component * Comp, const char * Name)
```

## Description

Query the component for a delegated property

Normally this should not be called by the user. But is useful in case the user constructs components of components in which case a component can redelegate delegated properties.

## Result

Object name pair with the target object and property name.

## Parameters

Table 80. `temu_componentGetDelegatedProp` Parameters

Parameter	Type	Description
Comp	<code>temu_Component *</code>	The componen to query
Name	<code>const char *</code>	Name of delegated property

### 1.1.82. `temu_componentGetObject`

#### Include

```
#include "temu-c/Support/Component.h"
```

#### Signature

```
temu_Object * temu_componentGetObject(temu_Component * Comp, const char * Name)
```

#### Description

Get named object in component

#### Result

NULL if the object is not a member in the component

## Parameters

Table 81. `temu_componentGetObject` Parameters

Parameter	Type	Description
Comp	<code>temu_Component *</code>	Component pointer
Name	<code>const char *</code>	String without the component prefix (i.e. local name)

### 1.1.83. `temu_connect`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_connect(temu_Object * A, const char * PropName, temu_Object * B, const char *
IfaceName)
```

## Description

Connect an interface property in object A to the interface in object B. If A.PropName is an interface array, B,B:IfaceName will be appended to it. For dynamic arrays, it is pushed to the end, for static arrays, it is placed in the first null slot (unless, the indexing syntax is used in the property name).

## Result

0 on success, otherwise non-zero

## Parameters

Table 82. `temu_connect` Parameters

Parameter	Type	Description
A	<code>temu_Object *</code>	The object to connect.
PropName	<code>const char *</code>	The name of the property in object A
B	<code>temu_Object *</code>	The object to connect to.
IfaceName	<code>const char *</code>	The name of the interface in object B

### 1.1.84. `temu_cpuDisableTraps`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_cpuDisableTraps(void * Cpu)
```

#### Description

Disable traps on processor

#### Parameters

Table 83. `temu_cpuDisableTraps` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer

### 1.1.85. temu\_cpuEnableTraps

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_cpuEnableTraps(void * Cpu)
```

#### Description

Enable traps on processor

#### Parameters

Table 84. temu\_cpuEnableTraps Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer

### 1.1.86. temu\_cpuGetFpr32

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
float temu_cpuGetFpr32(void * Cpu, unsigned int Reg)
```

#### Description

Get a 32 bit floating point register

#### Result

Host float with the content of the FPU register

#### Parameters

Table 85. `temu_cpuGetFpr32` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	FPU register number

### 1.1.87. `temu_cpuGetFpr32Bits`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint32_t temu_cpuGetFpr32Bits(void * Cpu, unsigned int Reg)
```

#### Description

Get a 32 bit floating point register

#### Result

Floating point register contents

#### Parameters

Table 86. `temu_cpuGetFpr32Bits` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	FPU register number

### 1.1.88. `temu_cpuGetFpr64`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
double temu_cpuGetFpr64(void * Cpu, unsigned int Reg)
```

#### Description

Get 64 bit floating point register as double

## Result

FPU register value

## Parameters

Table 87. `temu_cpuGetFpr64` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	FPU register number

### 1.1.89. `temu_cpuGetFpr64Bits`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_cpuGetFpr64Bits(void * Cpu, unsigned int Reg)
```

#### Description

Get 64 bit floating point register contents

#### Result

FPU register value

#### Parameters

Table 88. `temu_cpuGetFpr64Bits` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	FPU register number

### 1.1.90. `temu_cpuGetFreq`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
uint64_t temu_cpuGetFreq(void * Cpu)
```

## Description

Get the clock frequency for the CPU

## Result

The configured clock frequency in Hz.

## Parameters

Table 89. `temu_cpuGetFreq` Parameters

Parameter	Type	Description
Cpu	void *	The CPU object

## 1.1.91. temu\_cpuGetMachine

### Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
void * temu_cpuGetMachine(void * Cpu)
```

## Description

Get the Machine for a given CPU

## Result

Pointer to the machine object

## Parameters

Table 90. `temu_cpuGetMachine` Parameters

Parameter	Type	Description
Cpu	void *	Pointer to the CPU object

## 1.1.92. temu\_cpuGetPc



## Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
uint64_t temu_cpuGetPc(void * Cpu)
```

## Description

Get the program counter

The program counter will be returned.

## Result

The value of the program counter register

## Parameters

Table 91. `temu_cpuGetPc` Parameters

Parameter	Type	Description
Cpu	void *	The CPU object

### 1.1.93. `temu_cpuGetReg`

## Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
uint64_t temu_cpuGetReg(void * Cpu, unsigned int Reg)
```

## Description

Gets the register in the processor

## Result

Register value

## Parameters

Table 92. `temu_cpuGetReg` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	Register number

### 1.1.94. `temu_cpuGetSpr`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_cpuGetSpr(void * Cpu, unsigned int RegId)
```

#### Description

Get the PowerPC SPR register

#### Result

Value of SPR specified in `RegId`.

#### Parameters

Table 93. `temu_cpuGetSpr` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
RegId	unsigned int	SPR register number

### 1.1.95. `temu_cpuGetStat`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_cpuGetStat(void * Obj, int Stat)
```

#### Description

## Get processor statistics

### Result

Statistics value

### Parameters

Table 94. `temu_cpuGetStat` Parameters

Parameter	Type	Description
Obj	void *	Processor pointer
Stat	int	Statistics ID

## 1.1.96. `temu_cpuRaiseTrap`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_cpuRaiseTrap(void * Cpu, int Trap, unsigned int Flags)
```

### Description

Raise a trap The function simulates a trap being raised at the current PC.

### Parameters

Table 95. `temu_cpuRaiseTrap` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
Trap	int	Trap ID. This is target dependent, for the SPARC this is the TT value of the trap.
Flags	unsigned int	set flag 1 to enable longjmp trap (this is useful in MMIO handlers to force a trap while a core is running). Set to 0 if called from outside the core or in e.g. an event handler.

## 1.1.97. `temu_cpuReset`

## Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
void temu_cpuReset(void * Cpu, int ResetType)
```

## Description

Reset the processor.

Resetting the CPU will result in a reset cascade where all connected devices are also reset.

## Parameters

Table 96. `temu_cpuReset` Parameters

Parameter	Type	Description
Cpu	void *	The CPU object
ResetType	int	The type of reset, by convention 0 means cold reset, and 1 indicates a warm reset.

### 1.1.98. `temu_cpuResetStat`

## Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
void temu_cpuResetStat(void * Obj, int Stat)
```

## Description

Clear processor statistics

## Parameters

Table 97. `temu_cpuResetStat` Parameters

Parameter	Type	Description
Obj	void *	Processor pointer

Parameter	Type	Description
Stat	int	Statistics ID

### 1.1.99. temu\_cpuRun

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_cpuRun(void * Cpu, uint64_t Cycles)
```

#### Description

Run the processor for a number of cycles

The function runs the processor for a number of cycles. If you wish to run the processor with another time unit, you can compute the cycles from the clock frequency of the emulated processor.

In case the processor halts or enters idle mode and there are no pending events the function will return early.

#### Result

The number of executed cycles.

#### Parameters

Table 98. temu\_cpuRun Parameters

Parameter	Type	Description
Cpu	void *	The CPU object
Cycles	uint64_t	The number of cycles to run the processor for.

### 1.1.100. temu\_cpuSetFpr32

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_cpuSetFpr32(void * Cpu, unsigned int Reg, float Value)
```

### Description

Set floating point register value

### Parameters

Table 99. `temu_cpuSetFpr32` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	Floating point register number
Value	float	Floating point value to set

### 1.1.101. `temu_cpuSetFpr32Bits`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_cpuSetFpr32Bits(void * Cpu, unsigned int Reg, uint32_t Value)
```

### Description

Set 32 bit floating point register value

### Parameters

Table 100. `temu_cpuSetFpr32Bits` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	Floating point register number
Value	uint32_t	Floating point value to set

### 1.1.102. `temu_cpuSetFpr64`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
void temu_cpuSetFpr64(void * Cpu, unsigned int Reg, double Value)
```

## Description

Set FPU register value

## Parameters

Table 101. `temu_cpuSetFpr64` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	FPU register number
Value	double	Value to set to register

### 1.1.103. `temu_cpuSetFpr64Bits`

## Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
void temu_cpuSetFpr64Bits(void * Cpu, unsigned int Reg, uint64_t Value)
```

## Description

Set FPU register value

## Parameters

Table 102. `temu_cpuSetFpr64Bits` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	FPU register number
Value	uint64_t	Value to set to register

### 1.1.104. temu\_cpuSetPc

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_cpuSetPc(void * Cpu, uint64_t Pc)
```

#### Description

Set the program counter

The program counter will be set to the supplied value.



For targets with delay slots (SPARC, MIPS etc), the function will also set the next PC to PC + sizeof(instruction).

#### Parameters

Table 103. temu\_cpuSetPc Parameters

Parameter	Type	Description
Cpu	void *	The CPU object
Pc	uint64_t	The program counter.

### 1.1.105. temu\_cpuSetReg

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_cpuSetReg(void * Cpu, unsigned int Reg, uint64_t Value)
```

#### Description

Set the register in the processor

#### Parameters

Table 104. temu\_cpuSetReg Parameters



Parameter	Type	Description
Cpu	void *	CPU pointer
Reg	unsigned int	Register number
Value	uint64_t	Value to write to register

### 1.1.106. temu\_cpuSetSpr

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_cpuSetSpr(void * Cpu, unsigned int RegId, uint64_t Value)
```

#### Description

Set the PowerPC SPR register

#### Parameters

Table 105. temu\_cpuSetSpr Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
RegId	unsigned int	SPR register number
Value	uint64_t	New SPR value

### 1.1.107. temu\_cpuStep

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_cpuStep(void * Cpu, uint64_t Steps)
```

#### Description

Run the processor for a number of steps

This function is different from `temu_cpuRun`, which runs for a time. The steps here indicates instructions executed (including trapping instructions). This can be contrasted to the `run` function which may advance the cycle counter by more than one for an instruction (depending on the timing models).

The function may return early in case the processor halts its execution or has entered idle mode and there are no events pending.

## Result

The number of executed steps.

## Parameters

Table 106. `temu_cpuStep` Parameters

Parameter	Type	Description
Cpu	void *	The CPU object
Steps	uint64_t	The number of steps to run the processor for.

## 1.1.108. `temu_cpuTranslateAddress`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
int temu_cpuTranslateAddress(void * Cpu, uint64_t Va, uint32_t flags, uint64_t *
physAddressResult)
```

### Description

Uses the MMU to translate the given virtual address `Va` to the physical address in the emulator

### Result

error code. 0 on success, non-zero otherwise,

### Parameters

Table 107. `temu_cpuTranslateAddress` Parameters

Parameter	Type	Description
Cpu	void *	the CPU object

Parameter	Type	Description
Va	uint64_t	the virtual address to be translated
flags	uint32_t	flags for the translation (CPU specific)
physAddressResult	uint64_t *	the result in a uint64_t pointer

### 1.1.109. temu\_createClassCmd

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
void * temu_createClassCmd(temu_Class * Cls, const char * Name, const char * Doc,
temu_ObjectCommandFunc F)
```

#### Description

Register a class command

#### Result

Opaque handle to command. NULL in case of failure.

#### Parameters

Table 108. temu\_createClassCmd Parameters

Parameter	Type	Description
Cls	temu_Class *	Class pointer
Name	const char *	Name of command
Doc	const char *	Documentation string for command
F	temu_ObjectCommandFunc	Command handler function

### 1.1.110. temu\_createCmd

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

## Signature

```
void * temu_createCmd(const char * Name, temu_CommandFunc F, const char * Doc, void *
Data)
```

## Description

Create and register global command

## Result

## Parameters

Table 109. `temu_createCmd` Parameters

Parameter	Type	Description
Name	const char *	Name of command
F	temu_CommandFunc	Command function to invoke
Doc	const char *	Documentation string
Data	void *	Data pointer. Some commands e.g. disassemble will increase the address between invocations. This must be saved in some data object which can be provided when the command is created.

### 1.1.111. `temu_createComponentObject`

## Include

```
#include "temu-c/Support/Component.h"
```

## Signature

```
temu_Object * temu_createComponentObject(temu_Component * Comp, const char * Class,
const temu_CreateArg * Args, const char * ObjNameFmt, ...)
```

## Description

Create an object in a component.

This function works as the `temu_createObject`, except that the object created is inserted in the component.

The function is intended to be used in the component constructor.

Object names are uniquely using the `objnamefmt` parameters and following `varargs`. Plus, that the name of the component itself is prefixed as `'[compname]-'`.

## Result

Pointer to created object.

## Parameters

Table 110. `temu_createComponentObject` Parameters

Parameter	Type	Description
Comp	<a href="#">temu_Component</a> *	The component under which the object is to be created.
Class	<code>const char *</code>	class of the created object
Args	<code>const temu_CreateArg *</code>	NULL terminated array of create arguments for the constructor.
ObjNameFmt	<code>const char *</code>	Name of created object, but it allows for printf style string names, simplifying the creation of multiple objects of the same type.

### 1.1.112. `temu_createGdbServer`

#### Include

```
#include "temu-c/GdbServer/GdbServer.h"
```

#### Signature

```
void * temu_createGdbServer(uint16_t Port)
```

#### Description

Create a new GDB server

#### Result

Opaque handle to GDB server.

#### Parameters

Table 111. `temu_createGdbServer` Parameters

Parameter	Type	Description
Port	uint16_t	TCP port to start server on.

### 1.1.113. `temu_createObject`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Object * temu_createObject(const char * ClsName, const char * ObjName, const
temu_CreateArg * Args)
```

#### Description

Create object with class and name.

#### Result

Allocated object. Result is NULL in case of: class does not exist, object already allocated with the given name, out of memory.

#### Parameters

Table 112. `temu_createObject` Parameters

Parameter	Type	Description
ClsName	const char *	Name of class used for instantiation
ObjName	const char *	Name of object in object system
Args	const temu_CreateArg *	Named argument pairs to pass into constructor. The list of args should be terminated by a TEMU_NULL_ARG. In case no arguments are passed, pass in NULL.

### 1.1.114. `temu_crossesBoundary32`

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
_Bool temu_crossesBoundary32(uint32_t A, uint32_t L, uint32_t Size)
```

### Description

Checks whether A + L crosses power of 2 boundary (e.g. page)

### Result

True if boundary is crossed. False otherwise.

### Parameters

Table 113. `temu_crossesBoundary32` Parameters

Parameter	Type	Description
A	uint32_t	Address value
L	uint32_t	Length / offset value
Size	uint32_t	boundary size (power of 2 size)

## 1.1.115. `temu_crossesBoundary64`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
_Bool temu_crossesBoundary64(uint64_t A, uint64_t L, uint64_t Size)
```

### Description

Checks whether A + L crosses power of 2 boundary (e.g. page)

### Result

True if boundary is crossed. False otherwise.

### Parameters

Table 114. `temu_crossesBoundary64` Parameters

Parameter	Type	Description
A	uint64_t	Address value
L	uint64_t	Length / offset value
Size	uint64_t	boundary size (power of 2 size)

### 1.1.116. temu\_ctz16

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
int temu_ctz16(uint16_t Word)
```

#### Description

Count trailing zeroes, this can be used as a square root on a power of 2 word.

#### Result

Number of trailing zeros in Word

#### Parameters

Table 115. temu\_ctz16 Parameters

Parameter	Type	Description
Word	uint16_t	the word, in which trailing zeros to be counted

### 1.1.117. temu\_ctz32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
int temu_ctz32(uint32_t Word)
```

#### Description



Count trailing zeroes, this can be used as a square root on a power of 2 word.

## Result

Number of trailing zeros in Word

## Parameters

Table 116. `temu_ctz32` Parameters

Parameter	Type	Description
Word	uint32_t	the word, in which trailing zeros to be counted

## 1.1.118. `temu_ctz64`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
int temu_ctz64(uint64_t Word)
```

### Description

Count trailing zeroes

### Result

The number of trailing zeros in Word

## Parameters

Table 117. `temu_ctz64` Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which trailing zeros to be counted

## 1.1.119. `temu_cyclesToNanos`

### Include

```
#include "temu-c/Support/Time.h"
```

## Signature

```
int64_t temu_cyclesToNanos(int64_t Cycles, int64_t Freq)
```

## Description

Convert cycles to nanoseconds

## Result

Cycles converted to nanoseconds

## Parameters

Table 118. `temu_cyclesToNanos` Parameters

Parameter	Type	Description
Cycles	int64_t	Cycle count to convert
Freq	int64_t	Frequency in Hz

### 1.1.120. `temu_cyclesToOtherFreqRoundedUp`

## Include

```
#include "temu-c/Support/Time.h"
```

## Signature

```
int64_t temu_cyclesToOtherFreqRoundedUp(int64_t Cycles, int64_t SourceFreq, int64_t TargetFreq)
```

## Description

Convert cycles to another frequency base, rounding upwards

## Result

Cycles, converted from source frequency to target frequency

## Parameters

Table 119. `temu_cyclesToOtherFreqRoundedUp` Parameters

Parameter	Type	Description
Cycles	int64_t	Cycle count in source frequency ticks

Parameter	Type	Description
SourceFreq	int64_t	Frequency in Hz of the source
TargetFreq	int64_t	Frequency in Hz for the converted value

### 1.1.121. temu\_cyclesToOtherFreqTruncated

#### Include

```
#include "temu-c/Support/Time.h"
```

#### Signature

```
int64_t temu_cyclesToOtherFreqTruncated(int64_t Cycles, int64_t SourceFreq, int64_t TargetFreq)
```

#### Description

Convert cycles to another frequency base, with truncated result

#### Result

Cycles, converted from source frequency to target frequency

#### Parameters

Table 120. temu\_cyclesToOtherFreqTruncated Parameters

Parameter	Type	Description
Cycles	int64_t	Cycle count in source frequency ticks
SourceFreq	int64_t	Frequency in Hz of the source
TargetFreq	int64_t	Frequency in Hz for the converted value

### 1.1.122. temu\_cyclesToSecs

#### Include

```
#include "temu-c/Support/Time.h"
```

#### Signature

```
double temu_cyclesToSecs(int64_t Cycles, int64_t Freq)
```

### Description

Convert cycles to seconds

### Result

Cycles converted to seconds

### Parameters

Table 121. `temu_cyclesToSecs` Parameters

Parameter	Type	Description
Cycles	int64_t	Cycle count to convert
Freq	int64_t	Frequency in Hz

## 1.1.123. `temu_cyclesToSteps`

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
uint64_t temu_cyclesToSteps(temu_TimeSource * TS, uint64_t Cycles)
```

### Description

Convert cycles to steps

### Result

Cycles converted to steps given the time source IPC/CPI configuration.

### Parameters

Table 122. `temu_cyclesToSteps` Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource *</a>	Time source object
Cycles	uint64_t	Cycle count to convert to steps

## 1.1.124. `temu_cyclesToStepsRoundedUp`

## Include

```
#include "temu-c/Support/Time.h"
```

## Signature

```
uint64_t temu_cyclesToStepsRoundedUp(temu_TimeSource * TS, uint64_t Cycles)
```

## Description

Convert cycles to steps, rounding upwards

## Result

Cycles converted to steps given the time source IPC/CPI configuration.

## Parameters

Table 123. `temu_cyclesToStepsRoundedUp` Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource</a> *	Time source object
Cycles	uint64_t	Cycle count to convert to steps

### 1.1.125. `temu_deserialiseJSON`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_deserialiseJSON(const char * FileName)
```

## Description

De-serialise the simulation state.

The function clears the whole internal object system, reads the JSON file `FileName` and creates all the objects.

If a class implements the `ObjectIface`, the (optional) deserialise procedure will be called.

## Result

0 on success, otherwise non-zero

## Parameters

Table 124. `temu_deserialiseJSON` Parameters

Parameter	Type	Description
FileName	const char *	The filename that contains the JSON data

### 1.1.126. `temu_deserialiseProp`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_deserialiseProp(void * Ctxt, temu_Object * Obj, const char * Name)
```

#### Description

`temu_deserialiseProp` Deserialize a serialized property

#### Parameters

Table 125. `temu_deserialiseProp` Parameters

Parameter	Type	Description
Ctxt	void *	An opaque context, this is the same context as is passed to the <code>deserialise</code> interface function.
Obj	<a href="#">temu_Object</a> *	Object being deserialised
Name	const char *	The name of the property

### 1.1.127. `temu_dictCreate`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Dict * temu_dictCreate()
```

**Description**

Creates a dictionary object

**Result**

A pointer to the dictionary object

**1.1.128. temu\_dictDispose****Include**

```
#include "temu-c/Support/Objsys.h"
```

**Signature**

```
void temu_dictDispose(temu_Dict * Dict)
```

**Description**

Delete a dictionary object

**Parameters**

Table 126. `temu_dictDispose` Parameters

Parameter	Type	Description
Dict	<code>temu_Dict *</code>	Pointer to the dictionary object to be deleted

**1.1.129. temu\_dictGetNextKey****Include**

```
#include "temu-c/Support/Objsys.h"
```

**Signature**

```
const char * temu_dictGetNextKey(temu_Dict * Dict, const char * Key)
```

**Description**

Given a specific key of a dictionary, the next key is provided by this function

## Result

The key that comes after Key, or nullptr on failure

## Parameters

Table 127. `temu_dictGetNextKey` Parameters

Parameter	Type	Description
Dict	<code>temu_Dict *</code>	The dictionary that contains the keys
Key	<code>const char *</code>	The key, whose next value in the dict to be provided

### 1.1.130. `temu_dictGetValue`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Propval temu_dictGetValue(temu_Dict * Dict, const char * Name)
```

#### Description

Retrieve a dictionary value by name

#### Result

The value

#### Parameters

Table 128. `temu_dictGetValue` Parameters

Parameter	Type	Description
Dict	<code>temu_Dict *</code>	Pointer to the dictionary object
Name	<code>const char *</code>	The name associated with the value to be retrieved

### 1.1.131. `temu_dictInsertValue`

#### Include



```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_dictInsertValue(temu_Dict * Dict, const char * Name, temu_Propval Val)
```

### Description

Inserts a name/value pair to a dictionary

### Result

0 on success. Other values indicate failures (e.g. the key is already used)

### Parameters

Table 129. `temu_dictInsertValue` Parameters

Parameter	Type	Description
Dict	<a href="#">temu_Dict *</a>	Pointer to the dictionary object
Name	const char *	The name to be inserted
Val	temu_Propval	The value to be inserted

## 1.1.132. temu\_dictRemoveValue

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_dictRemoveValue(temu_Dict * Dict, const char * Name)
```

### Description

Erase a name/value pair from a dictionary by name

### Result

0 on success. Other values indicate errors (e.g. key was unavailable in the dictionary)

### Parameters

Table 130. `temu_dictRemoveValue` Parameters

Parameter	Type	Description
Dict	<a href="#">temu_Dict</a> *	Pointer to the dictionary object
Name	const char *	The name of the pair to be removed

### 1.1.133. temu\_disassemble

#### Include

```
#include "temu-c/Support/Assembler.h"
```

#### Signature

```
char * temu_disassemble(void * Cpu, uint32_t Instr)
```

#### Description

Disassemble an instruction.

#### Result

A pointer to a malloced instruction string or null incase of failure.

### 1.1.134. temu\_disassembleAuto

#### Include

```
#include "temu-c/Support/Assembler.h"
```

#### Signature

```
const char * temu_disassembleAuto(void * Cpu, uint32_t Instr)
```

#### Description

Disassemble an instruction.

#### Result

A pointer to a a thread local instruction string or null incase of failure.

### 1.1.135. temu\_disassembleMemory

## Include

```
#include "temu-c/Support/Assembler.h"
```

## Signature

```
char * temu_disassembleMemory(void * Cpu, uint64_t Addr)
```

## Description

Disassemble an instruction in memory

The function disassembles an instruction and returns a string allocated on the heap. You are responsible for its release using free().

## Result

A pointer to a malloced instruction string or null in case of failure.

## Parameters

Table 131. `temu_disassembleMemory` Parameters

Parameter	Type	Description
Cpu	void *	CPU whose memory space will be used for disassembling.
Addr	uint64_t	Physical address of instruction to disassemble.

### 1.1.136. `temu_disposeGdbServer`

## Include

```
#include "temu-c/GdbServer/GdbServer.h"
```

## Signature

```
void temu_disposeGdbServer(void * Gdb)
```

## Description

Dispose a GDB server

## Parameters

Table 132. `temu_disposeGdbServer` Parameters

Parameter	Type	Description
Gdb	void *	Pointer returned by <code>temu_createGdbServer()</code>

### 1.1.137. `temu_disposeObject`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_disposeObject(temu_Object * Obj)
```

#### Description

Dispose object. The function will call the Objects dispose function.

#### Parameters

Table 133. `temu_disposeObject` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object to delete.

### 1.1.138. `temu_disposeSymtab`

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
void temu_disposeSymtab(temu_Symtab * Sym)
```

#### Description

Dispose (deallocate) the symbol table

#### Parameters

Table 134. `temu_disposeSymtab` Parameters

Parameter	Type	Description
Sym	<a href="#">temu_Symtab</a> *	Symbol table pointer to free.

### 1.1.139. temu\_elfHasDwarf

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_elfHasDwarf(const char * file)
```

#### Description

Return non-zero if file has DWARF data

#### Result

0 if file does not contain DWARF, 1 if the file contains DWARF.

#### Parameters

Table 135. `temu_elfHasDwarf` Parameters

Parameter	Type	Description
file	const char *	Path to file to check for DWARF-info.

### 1.1.140. temu\_ethGetEthTypeSizeField

#### Include

```
#include "temu-c/Bus/Ethernet.h"
```

#### Signature

```
uint16_t temu_ethGetEthTypeSizeField(temu_EthFrame * Frame)
```

#### Description

Get length / EtherType field. By ethernet standard, the size field which is 16 bit, will be a length if it is

1500 and an ethertype tag if it is  $\geq 1536$ .

## Result

### 1.1.141. temu\_ethGetPayload

#### Include

```
#include "temu-c/Bus/Ethernet.h"
```

#### Signature

```
const uint8_t * temu_ethGetPayload(temu_EthFrame * Frame)
```

#### Description

Get a readable pointer to the payload. Note that the first bytes will often be an LLC header, it may not be your actual data payload.

## Result

### 1.1.142. temu\_eventDepublish

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
void temu_eventDepublish(int64_t EvID)
```

#### Description

It is possible to depublish events explicitly, but this is rarely needed as it is managed at termination time automatically.

#### Parameters

Table 136. temu\_eventDepublish Parameters

Parameter	Type	Description
EvID	int64_t	Event ID

### 1.1.143. temu\_eventDeschedule

## Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_eventDeschedule(int64_t EvID)
```

## Description

Deschedule event

## Parameters

Table 137. `temu_eventDeschedule` Parameters

Parameter	Type	Description
EvID	int64_t	The event ID of the event to deschedule.

### 1.1.144. `temu_eventGetCycles`

## Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
int64_t temu_eventGetCycles(temu_TimeSource * TS, int64_t EvID)
```

## Description

Get delta time in cycles for the given event and queue

## Result

Number of simulated cycles in the future the event will trigger

## Parameters

Table 138. `temu_eventGetCycles` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	The time source object (normally a CPU object)

Parameter	Type	Description
EvID	int64_t	the event to get the cycles for.

### 1.1.145. temu\_eventGetNanos

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
int64_t temu_eventGetNanos(temu_TimeSource * TS, int64_t EvID)
```

#### Description

Get delta time in nanoseconds for the given event and queue

#### Result

Number of simulated nanoseconds in the future when the event will trigger

#### Parameters

Table 139. temu\_eventGetNanos Parameters

Parameter	Type	Description
TS	temu_TimeSource *	The time source object (normally a CPU object)
EvID	int64_t	the event to get the nanoseconds for.

### 1.1.146. temu\_eventGetSecs

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
double temu_eventGetSecs(temu_TimeSource * TS, int64_t EvID)
```

#### Description



Get delta time in seconds for the given event and queue

## Result

Number of simulated seconds in the future when the event will trigger

## Parameters

Table 140. `temu_eventGetSecs` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	The time source object (normally a CPU object)
EvID	<code>int64_t</code>	the event to get the seconds for.

### 1.1.147. `temu_eventIsScheduled`

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
int temu_eventIsScheduled(int64_t EvID)
```

#### Description

Check if event is scheduled

#### Result

Zero (0) in case the event is not scheduled, otherwise non-zero for scheduled events.

#### Parameters

Table 141. `temu_eventIsScheduled` Parameters

Parameter	Type	Description
EvID	<code>int64_t</code>	The event ID to check whether it is scheduled.

### 1.1.148. `temu_eventPostAsync`

#### Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_eventPostAsync(temu_TimeSource * TS, temu_ThreadSafeCb CB, void * Data,  
temu_SyncEvent Sync)
```

## Description

Post an event to an asynchronous queue

## Parameters

Table 142. `temu_eventPostAsync` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	the asynchronous queue or time source object (normally a CPU object)
CB	<code>temu_ThreadSafeCb</code>	The callback function to call when the event happens
Data	<code>void *</code>	The context data to be passed to the callback function
Sync	<code>temu_SyncEvent</code>	Execute on CPU or machine level.

### 1.1.149. `temu_eventPostCycles`

#### Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_eventPostCycles(temu_TimeSource * TS, int64_t EvID, int64_t Delta,  
temu_SyncEvent Sync)
```

## Description

Post events with a relative time in cycles in the future

Note that if posting a scheduled event, a warning will be printed and the event will be

automatically descheduled before being inserted in the event queue.

## Parameters

Table 143. `temu_eventPostCycles` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	The time source object (normally a CPU object)
EvID	<code>int64_t</code>	The even ID returned by one of the event publish functions.
Delta	<code>int64_t</code>	The number of CPU clock cycles in the future to post the event. This should be > 0.
Sync	<code>temu_SyncEvent</code>	whether the event should be executed on the CPU queue or the machine queue.

### 1.1.150. `temu_eventPostCyclesAbsolute`

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
void temu_eventPostCyclesAbsolute(temu_TimeSource * TS, int64_t EvID, int64_t AbsTime,
temu_SyncEvent Sync)
```

#### Description

Post events with an absolute time in cycles The function is only well defined for times >= current time. Timestamps

current time will have undefined behaviour.

## Parameters

Table 144. `temu_eventPostCyclesAbsolute` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	The time source object (normally a CPU object)

Parameter	Type	Description
EvID	int64_t	The even ID returned by one of the event publish functions.
AbsTime	int64_t	The absolute time stamp to run the event at, this should be > current time.
Sync	temu_SyncEvent	whether the event should be executed on the CPU queue or the machine queue.

### 1.1.151. temu\_eventPostNanos

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
void temu_eventPostNanos(temu_TimeSource * TS, int64_t EvID, int64_t Delta,
temu_SyncEvent Sync)
```

#### Description

Post events with a relative time in nanoseconds in the future

#### Parameters

Table 145. temu\_eventPostNanos Parameters

Parameter	Type	Description
TS	temu_TimeSource *	The time source object (normally a CPU object)
EvID	int64_t	The even ID returned by one of the event publish functions.
Delta	int64_t	The number of nanoseconds in the future to post the event. This should be > 0.
Sync	temu_SyncEvent	whether the event should be executed on the CPU queue or the machine queue.

### 1.1.152. temu\_eventPostNanosAbsolute

## Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_eventPostNanosAbsolute(temu_TimeSource * TS, int64_t EvID, int64_t AbsTime,
temu_SyncEvent Sync)
```

## Description

Post events with an absolute time in nanoseconds The function is only well defined for times  $\geq$  current time. Timestamps

current time will have undefined behaviour.

## Parameters

Table 146. `temu_eventPostNanosAbsolute` Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource *</a>	The time source object (normally a CPU object)
EvID	int64_t	The even ID returned by one of the event publish functions.
AbsTime	int64_t	The absolute time stamp to run the event at, this should be $>$ current time.
Sync	temu_SyncEvent	whether the event should be executed on the CPU queue or the machine queue.

### 1.1.153. temu\_eventPostSecs

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
void temu_eventPostSecs(temu_TimeSource * TS, int64_t EvID, double Delta,
temu_SyncEvent Sync)
```

## Description

Post events with a relative time in seconds in the future

## Parameters

Table 147. `temu_eventPostSecs` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	The time source object (normally a CPU object)
EvID	<code>int64_t</code>	The even ID returned by one of the event publish functions.
Delta	<code>double</code>	The number of seconds in the future to post the event. This should be > 0.
Sync	<code>temu_SyncEvent</code>	whether the event should be executed on the CPU queue or the machine queue.

### 1.1.154. `temu_eventPostStack`

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
void temu_eventPostStack(temu_TimeSource * TS, int64_t EvID, temu_SyncEvent Sync)
```

## Description

Post events in the event queue stack

Stacked events are executed either after the current instruction is finished or when the machine quanta expires in case of machine synchronised events.

## Parameters

Table 148. `temu_eventPostStack` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	The time source object (normally a CPU object)

Parameter	Type	Description
EvID	int64_t	The even ID returned by one of the event publish functions.
Sync	temu_SyncEvent	whether the event should be executed on the CPU queue or the machine queue.

### 1.1.155. temu\_eventPostSteps

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
void temu_eventPostSteps(temu_TimeSource * TS, int64_t EvID, int64_t Delta,
temu_SyncEvent Sync)
```

#### Description

Post events with a relative trigger in steps

Posting an event in steps is not effected by the IPC/CPI setting of the clock.

#### Parameters

Table 149. temu\_eventPostSteps Parameters

Parameter	Type	Description
TS	temu_TimeSource *	The time source object (normally a CPU object)
EvID	int64_t	The even ID returned by one of the event publish functions.
Delta	int64_t	The number of CPU steps in the future to post the event. This should be > 0.
Sync	temu_SyncEvent	whether the event should be executed on the CPU queue or the machine queue.

### 1.1.156. temu\_eventPublish

#### Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
int64_t temu_eventPublish(const char * EvName, void * Obj, void (*)(temu_Event *)  
Func)
```

## Description

Publish an event.

The function will allocate an event structure in the global event heap and return the event ID.

A typical use is to call the function in the object constructor and assign the event id to a field in the object. This field should not be snapshotted (i.e. it will be reassigned in the constructor) when restoring an object anyway.



This function is non thread safe. Object construction is expected to be done in a single thread (e.g. the main thread).

## Result

The event ID  $\geq 0$  in case of success. Negative values indicate errors.

## Parameters

Table 150. `temu_eventPublish` Parameters

Parameter	Type	Description
EvName	const char *	Name of the event
Obj	void *	The object associated with the event
Func	void (*)(temu_Event *)	The event callback function.

### 1.1.157. temu\_eventPublishStruct

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
int64_t temu_eventPublishStruct(const char * EvName, temu_Event * Ev, void * Obj, void
```



```
(*)(temu_Event *) Func)
```

## Description

Publish a preallocated event object.

The event objects can be embedded in your own class if needed.

## Result

The event ID.

## Parameters

Table 151. `temu_eventPublishStruct` Parameters

Parameter	Type	Description
EvName	const char *	Name of the event
Ev	<a href="#">temu_Event</a> *	Pointer to the event struct
Obj	void *	The object associated with the event
Func	void (*)(temu_Event *)	The event callback function.

### 1.1.158. `temu_eventQueueGetFreq`

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Signature

```
uint64_t temu_eventQueueGetFreq(temu_TimeSource * TS)
```

## Description

Get the frequency in Hz of the given time source.

## Result

## Parameters

Table 152. `temu_eventQueueGetFreq` Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource</a> *	Time source

## 1.1.159. temu\_eventRescheduleCycles

### Include

```
#include "temu-c/Support/Events.h"
```

### Signature

```
void temu_eventRescheduleCycles(temu_TimeSource * TS, int64_t EvID, int64_t Delta,
temu_SyncEvent Sync)
```

### Description

Reschedule event with relative time in cycles

This function automatically deschedules the event before scheduling it again. The function ensures only one lock is taken for the pattern:

- is scheduled?
- de-schedule
- post

### Parameters

Table 153. temu\_eventRescheduleCycles Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource</a> *	The time source object (normally a CPU object)
EvID	int64_t	The even ID returned by one of the event publish functions.
Delta	int64_t	The number of CPU clock cycles in the future to post the event. This should be > 0.
Sync	temu_SyncEvent	whether the event should be executed on the CPU queue or the machine queue.

## 1.1.160. temu\_eventSetPeriodCycles

### Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_eventSetPeriodCycles(int64_t EvID, int64_t Period)
```

## Description

Set the cycles property on an event.

Periodic events have the advantage that they do not slip due to reposting of event with respect to the event triggering time and behaves as if the reposting is 1, automatic and 2, is relative to the event schedule time. which differ from the triggering time by possibly a few cycles. Note, calling this function does not reschedule the event.

## Parameters

Table 154. `temu_eventSetPeriodCycles` Parameters

Parameter	Type	Description
EvID	int64_t	Event ID
Period	int64_t	Cycles to add as period.

### 1.1.161. `temu_eventSetRTPeriodNanos`

#### Include

```
#include "temu-c/Support/Events.h"
```

## Signature

```
void temu_eventSetRTPeriodNanos(int64_t EvID, int64_t Period)
```

## Description

Set the RT period (nanoseconds), this should be the same as the cycle period, but it should be in nanoseconds and tno cycles

## Parameters

Table 155. `temu_eventSetRTPeriodNanos` Parameters

Parameter	Type	Description
EvID	int64_t	Event ID
Period	int64_t	Period in nanoseconds

## 1.1.162. temu\_eventSetRTTime

### Include

```
#include "temu-c/Support/Events.h"
```

### Signature

```
void temu_eventSetRTTime(int64_t EvID, int64_t Time)
```

### Description

Set the RT time (nanoseconds absolute time computed from `temu_timeGetMonotonicWct()`). This time is used by events flagged with `TEMU_EVENT_RT` to delay the execution of the event handler function.

### Parameters

Table 156. `temu_eventSetRTTime` Parameters

Parameter	Type	Description
EvID	int64_t	Event ID
Time	int64_t	Absolute time to trigger event at in WCT

## 1.1.163. temu\_eventSetRealTime

### Include

```
#include "temu-c/Support/Events.h"
```

### Signature

```
int temu_eventSetRealTime(int64_t EvID)
```

### Description

Mark an event as real-time.

With a real-time event, it is meant that the event will execute at roughly real-time. In the case when the event is posted, it will compute a rough triggering time in wall clock. When the event is executed it will do two things:

1. Check if the event wall-clock time is before the current time, if so emit a "temu.realtime-slip"

notification".

2. Check if the event is triggered in the future (with respect to wall clock), if so sleep until the wall-clock has caught up.

Due to the behaviour, the real-time events are suitable only for relatively short event times, as long sleep times may make the emulator non responsive. Especially at present, this will block the execution of async events as the event queue is halted until the wall-clock catches up to the simulated time.

Note that RT event support is primarily used to slow down the emulator artificially, this is accomplished by enabling real-time mode on the relevant model.

This function should only be called on in-flight events.

## Result

0 On success.

## Parameters

Table 157. `temu_eventSetRealTime` Parameters

Parameter	Type	Description
EvID	int64_t	Event ID to make real-time

## 1.1.164. `temu_execCommand`

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
int temu_execCommand(const char * Cmd)
```

### Description

Executes a single command

### Result

0 on success, otherwise a non-zero value

### Parameters

Table 158. `temu_execCommand` Parameters

Parameter	Type	Description
Cmd	const char *	The command to be executed

### 1.1.165. temu\_execCommandFile

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
int temu_execCommandFile(const char * File)
```

#### Description

Executes the commands in the file "File"

#### Result

0 on success, otherwise a non-zero value

#### Parameters

Table 159. temu\_execCommandFile Parameters

Parameter	Type	Description
File	const char *	Path to the file with the commands to be executed

### 1.1.166. temu\_floatingPointPropval

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Propval temu_floatingPointPropval(temu_Type T, double V)
```

#### Description

Create a floating point propval of the given type

#### Result

Property value with the given type and value.

## Parameters

Table 160. `temu_floatingPointPropval` Parameters

Parameter	Type	Description
T	<code>temu_Type</code>	Type ID, must be a <code>teTY_Float</code> or <code>teTY_Double</code>
V	<code>double</code>	Value

## 1.1.167. `temu_foreachClass`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_foreachClass(void (*)(temu_Class *, void *) Func, void * Arg)
```

### Description

Call function on every class in the object system The function is called with the metaclass pointer and the argument. NOTE: This function is experimental.

### Parameters

Table 161. `temu_foreachClass` Parameters

Parameter	Type	Description
Func	<code>void (*)(temu_Class *, void *)</code>	Function to call
Arg	<code>void *</code>	Argument passed as the last parameter to Func

## 1.1.168. `temu_foreachComponent`

### Include

```
#include "temu-c/Support/Component.h"
```

### Signature

```
void temu_foreachComponent(void (*)(temu_Component *, void *) Func, void * Arg)
```

## Description

Iterate over all components and call a function on them

## Parameters

Table 162. `temu_foreachComponent` Parameters

Parameter	Type	Description
Func	void (*)(temu_Component *, void *)	The function to be called on each component, with the signature <b>Func)(temu_Component, void*)</b>
Arg	void *	The second argument to be passed to the function, for passing context

## 1.1.169. temu\_foreachInterface

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_foreachInterface(temu_Class * C, void (*)(temu_Class *, const char *, void *) Func, void * Arg)
```

## Description

Call function on every interface in a class. The function is called with the class pointer, interface name and the argument. NOTE: This function is experimental.

## Parameters

Table 163. `temu_foreachInterface` Parameters

Parameter	Type	Description
C	<a href="#">temu_Class</a> *	Class to iterate properties
Func	void (*)(temu_Class *, const char *, void *)	Function to call



Parameter	Type	Description
Arg	void *	Argument passed as the last parameter to Func

### 1.1.170. temu\_foreachObject

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_foreachObject(void (*)(temu_Object *, void *) Func, void * Arg)
```

#### Description

Call function on every object in the object system The function is called with the object pointer and the argument. NOTE: This function is experimental.

#### Parameters

Table 164. temu\_foreachObject Parameters

Parameter	Type	Description
Func	void (*)(temu_Object *, void *)	Function to call
Arg	void *	Argument passed as the last parameter to Func

### 1.1.171. temu\_foreachProcessor

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_foreachProcessor(void (*)(temu_Object *, void *) Func, void * Arg)
```

#### Description

Call function on every processor in the object system The function is called with the CPU pointer and the argument. NOTE: This function is experimental.

## Parameters

Table 165. `temu_foreachProcessor` Parameters

Parameter	Type	Description
Func	<code>void (*)(temu_Object *, void *)</code>	Function to call
Arg	<code>void *</code>	Argument passed as the last parameter to Func

### 1.1.172. `temu_foreachProperty`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_foreachProperty(temu_Class * C, void (*)(temu_Class *, const char *, void *)  
Func, void * Arg)
```

#### Description

Call function on every property in a class. The function is called with the class pointer, property name and the argument. NOTE: This function is experimental.

#### Parameters

Table 166. `temu_foreachProperty` Parameters

Parameter	Type	Description
C	<code>temu_Class *</code>	Class to iterate properties
Func	<code>void (*)(temu_Class *, const char *, void *)</code>	Function to call
Arg	<code>void *</code>	Argument passed as the last parameter to Func

### 1.1.173. `temu_foreachRootComponent`

#### Include

```
#include "temu-c/Support/Component.h"
```

#### Signature

```
void temu_foreachRootComponent(void (*)(temu_Component *, void *) Func, void * Arg)
```

## Description

Iterate over all root components and call a function on them

## Parameters

Table 167. `temu_foreachRootComponent` Parameters

Parameter	Type	Description
Func	void (*)(temu_Component *, void *)	The function to be called on each component, with the signature <b>Func)(temu_Component,void* )</b>
Arg	void *	The second argument to be passed to the function, for passing context

## 1.1.174. temu\_gdbAddCpu

### Include

```
#include "temu-c/GdbServer/GdbServer.h"
```

### Signature

```
void temu_gdbAddCpu(void * Gdb, const char * CpuName)
```

## Description

Add named CPU to GDB server

## Parameters

Table 168. `temu_gdbAddCpu` Parameters

Parameter	Type	Description
Gdb	void *	Pointer returned by <code>temu_createGdbServer()</code>
CpuName	const char *	Name of processor object.

## 1.1.175. temu\_gdbAddMachine

## Include

```
#include "temu-c/GdbServer/GdbServer.h"
```

## Signature

```
void temu_gdbAddMachine(void * Gdb, const char * MachineName)
```

## Description

Add named machine to GDB server

## Parameters

Table 169. `temu_gdbAddMachine` Parameters

Parameter	Type	Description
Gdb	void *	Pointer returned by <code>temu_createGdbServer()</code>
MachineName	const char *	Name of <code>Machine</code> object.

## 1.1.176. `temu_gdbAsyncStop`

### Include

```
#include "temu-c/GdbServer/GdbServer.h"
```

### Signature

```
void temu_gdbAsyncStop(void * Gdb)
```

### Description

Tell GDB server to stop at next safe point in time.

### Parameters

Table 170. `temu_gdbAsyncStop` Parameters

Parameter	Type	Description
Gdb	void *	Pointer returned by <code>temu_createGdbServer()</code>

## 1.1.177. `temu_gdbRun`

**Include**

```
#include "temu-c/GdbServer/GdbServer.h"
```

**Signature**

```
int temu_gdbRun(void * Gdb)
```

**Description**

Run the GDB server loop

**Result****Parameters**

Table 171. `temu_gdbRun` Parameters

Parameter	Type	Description
Gdb	void *	Pointer returned by <code>temu_createGdbServer()</code>

**1.1.178. temu\_gdbWaitForConnection****Include**

```
#include "temu-c/GdbServer/GdbServer.h"
```

**Signature**

```
void temu_gdbWaitForConnection(void * Gdb)
```

**Description**

Wait current thread for a user to connect

**Parameters**

Table 172. `temu_gdbWaitForConnection` Parameters

Parameter	Type	Description
Gdb	void *	Pointer returned by <code>temu_createGdbServer()</code>

**1.1.179. temu\_generateObjectGraph**

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_generateObjectGraph(const char * Path, int Display)
```

## Description

Print the current object graph as a dot file

Pass NULL (or nullptr) to generate to stdout. And set Display to non-zero to automatically display the generated dot-file.

## Result

0 if the generation was successful. Non-zero in case of failure.

## Parameters

Table 173. `temu_generateObjectGraph` Parameters

Parameter	Type	Description
Path	const char *	Destination file to write the graph to. NULL indicates stdout.
Display	int	If non-zero, the function will attempt to display the graph by running it through dot and .

## 1.1.180. temu\_getComponentCount

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
size_t temu_getComponentCount()
```

### Description

`temu_getComponentCount` Get the number of components in the current simulation

## Result

The number of components in the current simulation

### 1.1.181. temu\_getComponents

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Component ** temu_getComponents()
```

#### Description

temu\_getComponents Get the number of components in the current simulation

#### Result

An array of the pointers to the components in the current simulation, the length of the array can be obtained from temu\_getComponentCount()

### 1.1.182. temu\_getCycles

#### Include

```
#include "temu-c/Support/Time.h"
```

#### Signature

```
int64_t temu_getCycles(temu_TimeSource * TS)
```

#### Description

Get current time in cycles.

#### Result

Current cycles as it is understood by the object.

#### Parameters

Table 174. temu\_getCycles Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource</a> *	Time source object

### 1.1.183. temu\_getExecutionState

#### Include

```
#include "temu-c/Support/Scheduler.h"
```

#### Signature

```
temu_ExecutionState temu_getExecutionState()
```

#### Description

Get the current execution status of TEMU

#### Result

Current execution status

### 1.1.184. temu\_getFieldValue

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
uint64_t temu_getFieldValue(temu_Object * Obj, const char * RegName, unsigned int  

  RegIdx, const char * FieldName)
```

#### Description

Retrieve the value of a field in a register

#### Result

The value of the field

#### Parameters

Table 175. temu\_getFieldValue Parameters



Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	The object of the class the has the register
RegName	const char *	Register name
RegIdx	unsigned int	Register index
FieldName	const char *	Field name

### 1.1.185. temu\_getInterfaceName

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
const char * temu_getInterfaceName(temu_Object * Obj, void * Iface)
```

#### Description

temu\_getInterfaceName Get Interface Name of the Obj

#### Result

The name of the interface

#### Parameters

Table 176. temu\_getInterfaceName Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Pointer to the object containing the interface
Iface	void *	Interface to be find w.r.t. Object

### 1.1.186. temu\_getInterfaceRef

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_IfaceRef temu_getInterfaceRef(temu_Object * Obj, const char * IfaceName, int Idx)
```

### Description

Retrieve a reference to an interface identified by its name and index

### Result

The interface reference as a `temu_IfaceRef` object

### Parameters

Table 177. `temu_getInterfaceRef` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Pointer to the object containing the interface
IfaceName	<code>const char *</code>	Name of the interface
Idx	<code>int</code>	Interface index

## 1.1.187. `temu_getLoggingCategory`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
const char * temu_getLoggingCategory(temu_Class * Cls, unsigned int CategoryId)
```

### Description

`temu_getLogginCategory` adds a logging category to the class

### Result

Pointer to the category string, NULL for failure.

### Parameters

Table 178. `temu_getLoggingCategory` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	the class

Parameter	Type	Description
CategoryId	unsigned int	The category id. Valid range is [8,16)

### 1.1.188. temu\_getMajorVersion

#### Include

```
#include "temu-c/Support/Version.h"
```

#### Signature

```
unsigned int temu_getMajorVersion()
```

#### Description

Get TEMU major version as number.

#### Result

Major version number

### 1.1.189. temu\_getMinorVersion

#### Include

```
#include "temu-c/Support/Version.h"
```

#### Signature

```
unsigned int temu_getMinorVersion()
```

#### Description

Get TEMU minor version as number.

#### Result

Minor version number

### 1.1.190. temu\_getModelRegisterBankInfo

#### Include

```
#include "temu-c/Support/Register.h"
```

### Signature

```
const temu_ModelRegInfo * temu_getModelRegisterBankInfo(temu_Class * C)
```

### Description

Get list of class register banks The returned object is thread local. It will be destroyed on the next call in the current thread.

### Result

Thread local model reg info object with a list of all register banks.

### Parameters

Table 179. `temu_getModelRegisterBankInfo` Parameters

Parameter	Type	Description
C	<code>temu_Class *</code>	The class object.

## 1.1.191. `temu_getNanos`

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
int64_t temu_getNanos(temu_TimeSource * TS)
```

### Description

Get current time in nanoseconds

### Result

Current nanoseconds as it is understood by the object.

### Parameters

Table 180. `temu_getNanos` Parameters

Parameter	Type	Description
TS	<a href="#">temu_TimeSource</a> *	Time source object

### 1.1.192. temu\_getPatchVersion

#### Include

```
#include "temu-c/Support/Version.h"
```

#### Signature

```
unsigned int temu_getPatchVersion()
```

#### Description

Get TEMU patch version as number.

#### Result

Patch version number

### 1.1.193. temu\_getProcessorCount

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
size_t temu_getProcessorCount()
```

#### Description

temu\_getProcessorCount Get the number of processors in the current simulation

#### Result

The number of processors in the current simulation

### 1.1.194. temu\_getProcessors

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_Object ** temu_getProcessors()
```

### Description

temu\_getProcessors Get the list of processors in the current simulation

### Result

An array of processor pointers, the length can be obtained from temu\_getProcessorCount()

## 1.1.195. temu\_getPropAccessor

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_PropAccessor temu_getPropAccessor(temu_Object * Obj, const char * PropName, int PropIdx)
```

### Description

Get a reference to the named property

This is useful as a replacement of temu\_getPropref and temu\_getPropName. The resulting data structure lets you automatically disambiguate between properties and pseudo properties.

The index used will be PropIdx, or K if the PropName string has an array suffix [K].

### Result

A struct containing prop accessor data.

### Parameters

Table 181. temu\_getPropAccessor Parameters

Parameter	Type	Description
Obj	temu_Object *	An object of a class with the relevant property registered.

Parameter	Type	Description
PropName	const char *	The name of the property to be queried.
PropIdx	int	Index in property array.

### 1.1.196. temu\_getPropDynLength

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_getPropDynLength(const temu_Object * Obj, const char * PropName)
```

#### Description

Get the dynamic length of the property

Normally, the `temu_getPropLength()` would return 1 for a dynamic array. This means that there is one dynamic array. In case the length of the dynamic array is needed, this function can be used.

#### Result

Dynamic length of property

#### Parameters

Table 182. `temu_getPropDynLength` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object of the property
PropName	const char *	Name of the property

### 1.1.197. temu\_getPropLength

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_getPropLength(const temu_Object * Obj, const char * PropName)
```

### Description

Returns property length/size if a property is an array

### Result

The length of the property

### Parameters

Table 183. `temu_getPropLength` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object of the property
PropName	const char *	Name of the property

## 1.1.198. temu\_getPropName

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_PropName temu_getPropName(const temu_Object * Obj, const char * PropName)
```

### Description

Get a reference to a named property.

### Result

Prop name object, note that the returned name string is not the same as the passed string. The returned string has a lifetime bound to the class, hence it can be returned without any problems.

### Parameters

Table 184. `temu_getPropName` Parameters

Parameter	Type	Description
Obj	const temu_Object *	An object of a class with the relevant property registered.



Parameter	Type	Description
PropName	const char *	The name of the property to be queried.

### 1.1.199. temu\_getPropType

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Type temu_getPropType(const temu_Object * Obj, const char * PropName)
```

#### Description

Get the type of a property

#### Result

Property type

#### Parameters

Table 185. temu\_getPropType Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object of the property
PropName	const char *	Name of the property

### 1.1.200. temu\_getPropref

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Propref temu_getPropref(const temu_Object * Obj, const char * PropName)
```

#### Description

Get a reference to the named property. The propref is a typetag and a pointer to the raw value, as such it does not work with delegated properties in components, nor with pseudo properties.

## Result

Object of the property reference

## Parameters

Table 186. `temu_getPropref` Parameters

Parameter	Type	Description
Obj	<code>const temu_Object *</code>	An object of a class with the relevant property registered.
PropName	<code>const char *</code>	The name of the property to be queried.

### 1.1.201. `temu_getRegister`

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
temu_Register * temu_getRegister(temu_RegisterBank * Bank, const char * Name)
```

#### Description

Get a register from a register bank by name

#### Result

Pointer to the register object

#### Parameters

Table 187. `temu_getRegister` Parameters

Parameter	Type	Description
Bank	<code>temu_RegisterBank *</code>	Pointer to the register bank object
Name	<code>const char *</code>	Name of the register

### 1.1.202. `temu_getRegisterBank`

## Include

```
#include "temu-c/Support/Register.h"
```

## Signature

```
temu_RegisterBank * temu_getRegisterBank(temu_Class * C, const char * Name)
```

## Description

Get a register bank from a class by name

## Result

Pointer to the register bank

## Parameters

Table 188. *temu\_getRegisterBank Parameters*

Parameter	Type	Description
C	<a href="#">temu_Class *</a>	Pointer to the TEMU class
Name	const char *	Name of the register bank

### 1.1.203. temu\_getRegisterBankName

## Include

```
#include "temu-c/Support/Register.h"
```

## Signature

```
const char * temu_getRegisterBankName(temu_RegisterBank * Bank)
```

## Description

Retrieves the register bank name from its object

## Result

Bank name as a C-string

## Parameters

Table 189. *temu\_getRegisterBankName Parameters*

Parameter	Type	Description
Bank	<a href="#">temu_RegisterBank</a> *	Pointer to the register bank name

### 1.1.204. temu\_getRegisterColdResetValue

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
uint64_t temu_getRegisterColdResetValue(temu_Register * Reg)
```

#### Description

Returns the cold reset value of the register (computed from the field info data). Cold reset is also known as a hard reset, and implies that the power has been off for some time.

#### Result

The cold reset value

#### Parameters

Table 190. temu\_getRegisterColdResetValue Parameters

Parameter	Type	Description
Reg	<a href="#">temu_Register</a> *	Pointer to the register

### 1.1.205. temu\_getRegisterDocs

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
const char * temu_getRegisterDocs(temu_Register * R)
```

#### Description

Get the documentation of a register as a string

## Result

The documentation of the register as a C-string

## Parameters

Table 191. `temu_getRegisterDocs` Parameters

Parameter	Type	Description
R	<code>temu_Register *</code>	Pointer to the register

### 1.1.206. `temu_getRegisterInfo`

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
const temu_RegisterInfo * temu_getRegisterInfo(temu_Class * C, const char * RegName)
```

#### Description

Get register info for named register The returned object is thread local. It will be destroyed on the next call in the current thread.

#### Result

Thread local register info object.

#### Parameters

Table 192. `temu_getRegisterInfo` Parameters

Parameter	Type	Description
C	<code>temu_Class *</code>	The class object.
RegName	<code>const char *</code>	Register name.

### 1.1.207. `temu_getRegisterName`

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
const char * temu_getRegisterName(temu_Register * R)
```

### Description

Retrieve register name from pointer

### Result

The name of the register

### Parameters

Table 193. `temu_getRegisterName` Parameters

Parameter	Type	Description
R	<code>temu_Register *</code>	Pointer to the register

## 1.1.208. `temu_getRegisterReadMask`

### Include

```
#include "temu-c/Support/Register.h"
```

### Signature

```
uint64_t temu_getRegisterReadMask(temu_Register * Reg)
```

### Description

Get the current read mask of a register

### Result

The value of the mask

### Parameters

Table 194. `temu_getRegisterReadMask` Parameters

Parameter	Type	Description
Reg	<code>temu_Register *</code>	Pointer to the register

## 1.1.209. `temu_getRegisterWarmResetValue`

### Include

```
#include "temu-c/Support/Register.h"
```

### Signature

```
uint64_t temu_getRegisterWarmResetValue(temu_Register * Reg)
```

### Description

Returns the warm reset value of the register

### Result

The warm reset value

### Parameters

*Table 195. temu\_getRegisterWarmResetValue Parameters*

Parameter	Type	Description
Reg	<a href="#">temu_Register *</a>	Pointer to the register

## 1.1.210. temu\_getRegisterWriteMask

### Include

```
#include "temu-c/Support/Register.h"
```

### Signature

```
uint64_t temu_getRegisterWriteMask(temu_Register * Reg)
```

### Description

Get the current write mask of a register

### Result

The value of the mask

### Parameters

*Table 196. temu\_getRegisterWriteMask Parameters*

Parameter	Type	Description
Reg	<a href="#">temu_Register *</a>	Pointer to the register

## 1.1.211. temu\_getSecs

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
double temu_getSecs(temu_TimeSource * TS)
```

### Description

Get current time in seconds

### Result

Current seconds as it is understood by the object.

### Parameters

Table 197. temu\_getSecs Parameters

Parameter	Type	Description
TS	temu_TimeSource *	Time source object

## 1.1.212. temu\_getSteps

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
int64_t temu_getSteps(temu_TimeSource * TS)
```

### Description

Get current step count

### Result

Current steps as it is understood by the object.

### Parameters



Table 198. `temu_getSteps` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	Time source object

### 1.1.213. `temu_getVTable`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void * temu_getVTable(const temu_Object * Obj)
```

#### Description

Get the VTable pointer for an object. This only works for internal objects that inherits from `temu_Object`.

#### Result

Pointer to the vtable in question

#### Parameters

Table 199. `temu_getVTable` Parameters

Parameter	Type	Description
Obj	<code>const temu_Object *</code>	The object in question

### 1.1.214. `temu_getVTableForClass`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void * temu_getVTableForClass(temu_Class * Cls)
```

#### Description

Get the VTable pointer for a class

## Result

Pointer to the vtable in question

## Parameters

Table 200. `temu_getVTableForClass` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	The class, for which the vtable is to be retrieved

### 1.1.215. `temu_getValue`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Propval temu_getValue(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Get the value of a property from an object

#### Result

The value of the property

#### Parameters

Table 201. `temu_getValue` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the property
PropName	<code>const char *</code>	Property name
Idx	<code>int</code>	Index of the property. Set to 0 if the property is not an array

### 1.1.216. `temu_getValueDouble`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
double temu_getValueDouble(temu_Object * Obj, const char * PropName, int Idx)
```

### Description

Get property (without side effects)

### Result

The property value converted to double

### Parameters

Table 202. `temu_getValueDouble` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property
Idx	<code>int</code>	Index in property array (set to 0 if scalar)

## 1.1.217. `temu_getValueI16`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int16_t temu_getValueI16(temu_Object * Obj, const char * PropName, int Idx)
```

### Description

Get the value of a property from an object if it is an int16

### Result

The value of the property

### Parameters

Table 203. `temu_getValueI16` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the property
PropName	<code>const char *</code>	Property name
Idx	<code>int</code>	Index of the property. Set to 0 if the property is not an array

### 1.1.218. `temu_getValueI32`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int32_t temu_getValueI32(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Get the value of a property from an object if it is an int32

#### Result

The value of the property

#### Parameters

Table 204. `temu_getValueI32` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the property
PropName	<code>const char *</code>	Property name
Idx	<code>int</code>	Index of the property. Set to 0 if the property is not an array

### 1.1.219. `temu_getValueI64`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int64_t temu_getValueI64(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Get the value of a property from an object if it is an int64

## Result

The value of the property

## Parameters

Table 205. `temu_getValueI64` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the property
PropName	<code>const char *</code>	Property name
Idx	<code>int</code>	Index of the property. Set to 0 if the property is not an array

## 1.1.220. temu\_getValueI8

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int8_t temu_getValueI8(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Get the value of a property from an object if it is an int8

## Result

The value of the property

## Parameters

Table 206. `temu_getValueI8` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	The object that contains the property
PropName	const char *	Property name
Idx	int	Index of the property. Set to 0 if the property is not an array

### 1.1.221. temu\_getValueSigned

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int64_t temu_getValueSigned(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Get property (without side effects)

#### Result

The property value converted to signed

#### Parameters

Table 207. temu\_getValueSigned Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.222. temu\_getValueU16

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
uint16_t temu_getValueU16(temu_Object * Obj, const char * PropName, int Idx)
```

### Description

Get the value of a property from an object if it is an uint16

### Result

The value of the property

### Parameters

Table 208. `temu_getValueU16` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the property
PropName	<code>const char *</code>	Property name
Idx	<code>int</code>	Index of the property. Set to 0 if the property is not an array

## 1.1.223. `temu_getValueU32`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
uint32_t temu_getValueU32(temu_Object * Obj, const char * PropName, int Idx)
```

### Description

Get the value of a property from an object if it is an uint32

### Result

The value of the property

### Parameters

Table 209. `temu_getValueU32` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	The object that contains the property
PropName	const char *	Property name
Idx	int	Index of the property. Set to 0 if the property is not an array

### 1.1.224. temu\_getValueU64

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
uint64_t temu_getValueU64(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Get the value of a property from an object if it is an uint64

#### Result

The value of the property

#### Parameters

Table 210. temu\_getValueU64 Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	The object that contains the property
PropName	const char *	Property name
Idx	int	Index of the property. Set to 0 if the property is not an array

### 1.1.225. temu\_getValueU8

#### Include

```
#include "temu-c/Support/Objsys.h"
```



## Signature

```
uint8_t temu_getValueU8(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Get the value of a property from an object if it is an uint8

## Result

The value of the property

## Parameters

Table 211. `temu_getValueU8` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the property
PropName	<code>const char *</code>	Property name
Idx	<code>int</code>	Index of the property. Set to 0 if the property is not an array

## 1.1.226. temu\_getValueUnsigned

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
uint64_t temu_getValueUnsigned(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Get property (without side effects)

## Result

The property value converted to unsigned

## Parameters

Table 212. `temu_getValueUnsigned` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object</a> *	Object pointer
PropName	const char *	Name of property
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.227. temu\_getVersionString

#### Include

```
#include "temu-c/Support/Version.h"
```

#### Signature

```
const char * temu_getVersionString()
```

#### Description

Get TEMU version string.

#### Result

Version string in format like: MAJOR . MINOR . PATCH ( REVISION )

### 1.1.228. temu\_identifyTrailingZeroes32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_identifyTrailingZeroes32(uint32_t Word)
```

#### Description

Get word with all trailing zeroes set

#### Result

A copy of word, with trailing zeros set

#### Parameters

Table 213. `temu_identifyTrailingZeroes32` Parameters

Parameter	Type	Description
Word	<code>uint32_t</code>	The word, in which trailing zeros to be set

### 1.1.229. `temu_identifyTrailingZeroes64`

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_identifyTrailingZeroes64(uint64_t Word)
```

#### Description

Get word with all trailing zeroes set

#### Result

A copy of word, with trailing zeros set

#### Parameters

Table 214. `temu_identifyTrailingZeroes64` Parameters

Parameter	Type	Description
Word	<code>uint64_t</code>	The word, in which trailing zeros to be set

### 1.1.230. `temu_identifyTrailingZeroesAndFirst32`

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_identifyTrailingZeroesAndFirst32(uint32_t Word)
```

#### Description

Get word with all trailing zeroes set, and the fist bit set

## Result

Word, with all trailing zeroes set, and the fist bit set

## Parameters

Table 215. `temu_identifyTrailingZeroesAndFirst32` Parameters

Parameter	Type	Description
Word	uint32_t	Word to get trailing zero mask from

## 1.1.231. `temu_identifyTrailingZeroesAndFirst64`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
uint64_t temu_identifyTrailingZeroesAndFirst64(uint64_t Word)
```

### Description

Get word with all trailing zeroes set, and the fist bit set

## Result

Word, with all trailing zeroes set, and the fist bit set

## Parameters

Table 216. `temu_identifyTrailingZeroesAndFirst64` Parameters

Parameter	Type	Description
Word	uint64_t	Word to get trailing zero mask from

## 1.1.232. `temu_ifaceRefArrayAlloc`

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
temu_IfaceRefArray temu_ifaceRefArrayAlloc(unsigned int Reserve)
```

## Description

Allocate interface array.

## Result

The allocated interface array.

## Parameters

Table 217. `temu_ifaceRefArrayAlloc` Parameters

Parameter	Type	Description
Reserve	unsigned int	Number of entries to reserve initially.

### 1.1.233. `temu_ifaceRefArrayGet`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
temu_IfaceRef temu_ifaceRefArrayGet(temu_IfaceRefArray * Arr, unsigned int idx)
```

## Description

Get the index of the interface reference array.

## Result

The interface array with in bounds.

## Parameters

Table 218. `temu_ifaceRefArrayGet` Parameters

Parameter	Type	Description
Arr	<a href="#">temu_IfaceRefArray *</a>	The interface reference array
idx	unsigned int	is the index of the array

### 1.1.234. temu\_ifaceRefArrayPush

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_ifaceRefArrayPush(temu_IfaceRefArray * Arr, temu_IfaceRef Iface)
```

#### Description

Push an interface reference to the interface array.

#### Parameters

Table 219. temu\_ifaceRefArrayPush Parameters

Parameter	Type	Description
Arr	<a href="#">temu_IfaceRefArray *</a>	The interface reference array
Iface	temu_IfaceRef	The interface reference (object-interface pointer pair)

### 1.1.235. temu\_ifaceRefArrayPush2

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_ifaceRefArrayPush2(temu_IfaceRefArray * Arr, temu_Object * Obj, void * Iface)
```

#### Description

Push an object and raw interface pointer to an interface array.

#### Parameters

Table 220. temu\_ifaceRefArrayPush2 Parameters

Parameter	Type	Description
Arr	<a href="#">temu_IfaceRefArray *</a>	The interface reference array.

Parameter	Type	Description
Obj	<a href="#">temu_Object</a> *	The object implementing the relevant interface
Iface	void *	Pointer to the interface struct.

### 1.1.236. temu\_ifaceRefArraySize

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
unsigned int temu_ifaceRefArraySize(temu_IfaceRefArray * Arr)
```

#### Description

Get the length of a dynamically allocated interface array.

#### Result

The length in number of entries.

#### Parameters

Table 221. temu\_ifaceRefArraySize Parameters

Parameter	Type	Description
Arr	<a href="#">temu_IfaceRefArray</a> *	The interface array to get the size from.

### 1.1.237. temu\_imageIsELF

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_imageIsELF(const char * file)
```

#### Description

Return non-zero if file is an ELF file

## Result

0 if file is not an ELF file, 1 if the file is an ELF file.

## Parameters

Table 222. `temu_imageIsELF` Parameters

Parameter	Type	Description
file	const char *	Path to file to check for ELFiness.

## 1.1.238. `temu_indexForInterface`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_indexForInterface(const temu_Object * Obj, const void * Iface)
```

### Description

Get index for interface

### Result

Index of interface if it is in an iface array. Otherwise it returns -1. Consequently the function only have valid result for interface arrays.

### Parameters

Table 223. `temu_indexForInterface` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object that contains the interface
Iface	const void *	Pointer to the interface

## 1.1.239. `temu_initConfigDirs`

### Include



```
#include "temu-c/Support/Init.h"
```

### Signature

```
void temu_initConfigDirs()
```

### Description

Create the temu config directories (typically ~/.config/temu/)

This function is invoked by the init function normally, but it is useful in some cases to create the config dirs without calling init.

## 1.1.240. temu\_initGdbServerLib

### Include

```
#include "temu-c/GdbServer/GdbServer.h"
```

### Signature

```
void temu_initGdbServerLib()
```

### Description

Initialise the GDB Server Library

## 1.1.241. temu\_initLicense

### Include

```
#include "temu-c/Support/Init.h"
```

### Signature

```
int temu_initLicense()
```

### Description

Init only license system.

Initialise the license manager only. This can be used to check for whether the license is valid before

proceeding with rest of the system initialisation. The function does not terminate the application on invalid license, but returns a non-zero value.

## Result

0 if the license is valid, other values indicate errors.

### 1.1.242. temu\_initPathSupport

#### Include

```
#include "temu-c/Support/Init.h"
```

#### Signature

```
void temu_initPathSupport(const char * Argv0)
```

#### Description

Initialise path finding support

This function initialises auxillary path support needed by TEMU for locating supporting tools (e.g. stuff in libexec) and automatically appending of scripting search paths.

In the future, this will likely be merged into `initSupportLib` for a unified `init` function, but this will break API compatibility, so this change is deferred to TEMU 3.

Call the Function passing `Argv0` of whatever you would do to invoke the `temu` command line tool. E.g. if `temu` is in your path and this is the installation you wish to use, the parameter should be `"temu"`. However, if you wish to use a custom installation, pass the full path to the `temu` program. The base path can contain `~` for the current home dir and `~foo` for the home dir of user `foo`.

Note that it is not an error to omit this initialisation, but omitting it will imply that `temu` cannot launch UI models such as the `ConsoleUI` model.

#### Parameters

Table 224. `temu_initPathSupport` Parameters

Parameter	Type	Description
<code>Argv0</code>	<code>const char *</code>	Name of TEMU executable (typically passed as <code>argv[0]</code> to <code>main</code> )

### 1.1.243. temu\_initSupportLib

#### Include

```
#include "temu-c/Support/Init.h"
```

## Signature

```
void temu_initSupportLib()
```

## Description

Initialisation of TEMU support library.

Call this function before any other use of TEMU. The function does not return failures, but halts on failed initialisation.

The function should be called as early as possible in the application's execution. Preferably in the main function (though there is no strict requirement for this).

The initialisation function is at present not thread safe, call it in the main thread only!

## 1.1.244. temu\_inlineDeserialiseJSON

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_inlineDeserialiseJSON(const char * FileName)
```

## Description

De-serialise the simulation state.

This function directly restores properties on already created objects. It does not clear the objects system. Thus pointers to objects remains stable after a snapshot restore.

If a class implements the ObjectIface, the (optional) deserialise procedure will be called.

## Result

0 on success, otherwise non-zero

## Parameters

Table 225. temu\_inlineDeserialiseJSON Parameters

Parameter	Type	Description
FileName	const char *	The filename that contains the JSON data

### 1.1.245. temu\_isComponent

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_isComponent(const temu_Object * Obj)
```

#### Description

temu\_isComponent Checks if an object is a component

#### Result

0 for false, non-zero for true

#### Parameters

Table 226. temu\_isComponent Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

### 1.1.246. temu\_isCpu

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_isCpu(const temu_Object * Obj)
```

#### Description

Predicate for identifying CPU classes. In release without assert builds this runs in O(1) time.

#### Result

0 for false, non-zero for true

## Parameters

Table 227. `temu_isCpu` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

## 1.1.247. `temu_isDiscrete`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_isDiscrete(temu_Propval Pv)
```

### Description

Checks whether the numeric property is an integer-like

### Result

0 for false, non-zero for true

## Parameters

Table 228. `temu_isDiscrete` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

## 1.1.248. `temu_isMachine`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_isMachine(const temu_Object * Obj)
```

## Description

Predicate for identifying CPU classes. In release without assert builds this runs in O(1) time..

## Result

0 for false, non-zero for true

## Parameters

Table 229. `temu_isMachine` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

## 1.1.249. temu\_isMemory

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_isMemory(const temu_Object * Obj)
```

### Description

temu\_isMemory Checks whether an object is a memory object

### Result

0 for false, non-zero for true

### Parameters

Table 230. `temu_isMemory` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

## 1.1.250. temu\_isNormalProperty

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_isNormalProperty(temu_Object * Obj, const char * PropName)
```

## Description

temu\_isNormalProperty Checks whether a property is a normal property (not a pseudo-property)

## Result

0 for false, non-zero for true

## Parameters

Table 231. temu\_isNormalProperty Parameters

Parameter	Type	Description
Obj	temu_Object *	The object that contains the property
PropName	const char *	The property name

## 1.1.251. temu\_isNumber

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_isNumber(temu_Propval Pv)
```

## Description

Checks whether a property is a number

## Result

0 for false, non-zero for true

## Parameters

Table 232. temu\_isNumber Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

## 1.1.252. temu\_isPow2\_32

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
int temu_isPow2_32(uint32_t Word)
```

### Description

Predicate for checking if word is a power of 2 See Hacker's Delight, 1st edition, p11

### Result

0 for false, otherwise true

### Parameters

Table 233. temu\_isPow2\_32 Parameters

Parameter	Type	Description
Word	uint32_t	The word to be checked

## 1.1.253. temu\_isPow2\_64

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
int temu_isPow2_64(uint64_t Word)
```

### Description

Predicate for checking if word is a power of 2

### Result

0 for false, otherwise true

### Parameters



Table 234. `temu_isPow2_64` Parameters

Parameter	Type	Description
Word	uint64_t	The word to be checked

### 1.1.254. `temu_isPseudoProperty`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_isPseudoProperty(temu_Object * Obj, const char * PropName)
```

#### Description

`temu_isPseudoProperty` Checks whether a property is a pseudo-property

#### Result

0 for false, non-zero for true

#### Parameters

Table 235. `temu_isPseudoProperty` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the property
PropName	<code>const char *</code>	The property name

### 1.1.255. `temu_isQualifiedAs`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_isQualifiedAs(const temu_Object * Obj, unsigned int Qualifier)
```

#### Description

temu\_isQualifiedAs Check if an object is qualified as Qualifier

## Result

0 for false, non-zero for true

## Parameters

Table 236. temu\_isQualifiedAs Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to be checked
Qualifier	unsigned int	The qualification to be checked

## 1.1.256. temu\_isReal

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_isReal(temu_Propval Pv)
```

### Description

Checks whether a property is a real number

### Result

0 for false, non-zero for true

### Parameters

Table 237. temu\_isReal Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

## 1.1.257. temu\_isString

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_isString(temu_Propval Pv)
```

## Description

Checks whether a property is a string

## Result

0 for false, non-zero for true

## Parameters

Table 238. `temu_isString` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

## 1.1.258. temu\_isolateLeftmostBit32

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint32_t temu_isolateLeftmostBit32(uint32_t Word)
```

## Description

Isolate left most bit

## Result

A copy of word, in which the right most set bit will be cleared

## Parameters

Table 239. `temu_isolateLeftmostBit32` Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which the right most set bit will be cleared

## 1.1.259. temu\_isolateLeftmostBit64

## Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint64_t temu_isolateLeftmostBit64(uint64_t Word)
```

## Description

Isolate left most bit

## Result

A copy of word, in which the right most set bit will be cleared

## Parameters

Table 240. `temu_isolateLeftmostBit64` Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which the right most set bit will be cleared

## 1.1.260. `temu_isolateRightMostZeroBit32`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
uint32_t temu_isolateRightMostZeroBit32(uint32_t Word)
```

### Description

Isolate right most bit with value zero (it will be 1 in the result)

### Result

A copy of Word, in which the right most bit with value zero will be isolated

### Parameters

Table 241. `temu_isolateRightMostZeroBit32` Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which the right most bit with value zero will be isolated

### 1.1.261. temu\_isolateRightMostZeroBit64

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_isolateRightMostZeroBit64(uint64_t Word)
```

#### Description

Isolate right most bit with value zero (it will be 1 in the result)

#### Result

A copy of Word, in which the right most bit with value zero will be isolated

#### Parameters

Table 242. temu\_isolateRightMostZeroBit64 Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which the right most bit with value zero will be isolated

### 1.1.262. temu\_lineDataGetLine

#### Include

```
#include "temu-c/Support/DataLoggers.h"
```

#### Signature

```
const char * temu_lineDataGetLine(void * Obj, uint64_t Line)
```

#### Description

Get line with number.

## Result

A string that is borrowed.

## Parameters

Table 243. `temu_lineDataGetLine` Parameters

Parameter	Type	Description
Obj	void *	Data-logging object
Line	uint64_t	Line number to read out

### 1.1.263. `temu_lineDataGetLineCount`

#### Include

```
#include "temu-c/Support/DataLoggers.h"
```

#### Signature

```
uint64_t temu_lineDataGetLineCount(void * Obj)
```

#### Description

Get number of lines in log.

#### Result

Number of logged lines

#### Parameters

Table 244. `temu_lineDataGetLineCount` Parameters

Parameter	Type	Description
Obj	void *	Data-logging object

### 1.1.264. `temu_listAppend`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_listAppend(temu_List * List, temu_Propval Val)
```

## Description

Add an element to the end of a linked-list

## Parameters

Table 245. `temu_listAppend` Parameters

Parameter	Type	Description
List	<code>temu_List *</code>	The list, to which the element to be added
Val	<code>temu_Propval</code>	The element to be added

## 1.1.265. `temu_listCreate`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_List temu_listCreate(temu_Type Typ)
```

### Description

Create a linked-list object

### Result

A newly created list.

### Parameters

Table 246. `temu_listCreate` Parameters

Parameter	Type	Description
Typ	<code>temu_Type</code>	The type to be stored in the elements of the linked list

## 1.1.266. `temu_listDispose`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_listDispose(temu_List * List)
```

## Description

Delete a linked-list object

## Parameters

Table 247. `temu_listDispose` Parameters

Parameter	Type	Description
List	<code>temu_List *</code>	The list to be deleted

## 1.1.267. temu\_listGetHead

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_ListNode * temu_listGetHead(temu_List * List)
```

### Description

Get the first element of a linked-list

### Result

Pointer to the first element of the linked-list

### Parameters

Table 248. `temu_listGetHead` Parameters

Parameter	Type	Description
List	<code>temu_List *</code>	The list, whose first element is requested



## 1.1.268. temu\_listGetNext

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_ListNode * temu_listGetNext(temu_ListNode * Node)
```

### Description

Get the next element of a linked list

### Result

The next element if available, otherwise nullptr

### Parameters

Table 249. temu\_listGetNext Parameters

Parameter	Type	Description
Node	temu_ListNode *	The node, whose next element is requested

## 1.1.269. temu\_listGetPrev

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_ListNode * temu_listGetPrev(temu_ListNode * Node)
```

### Description

Get the previous element of a linked list

### Result

The previous element if available, otherwise nullptr

### Parameters

Table 250. `temu_listGetPrev` Parameters

Parameter	Type	Description
Node	<code>temu_ListNode *</code>	The node, whose next element is requested

### 1.1.270. `temu_listGetTail`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_ListNode * temu_listGetTail(temu_List * List)
```

#### Description

Get the last element of a linked-list

#### Result

Pointer to the last element of the linked-list

#### Parameters

Table 251. `temu_listGetTail` Parameters

Parameter	Type	Description
List	<code>temu_List *</code>	The list, whose last element is requested

### 1.1.271. `temu_listNodeGetVal`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Propval temu_listNodeGetVal(temu_ListNode * Node)
```

#### Description

## Get value of a linked-list node

### Result

The value extracted from the node

### Parameters

Table 252. `temu_listNodeGetVal` Parameters

Parameter	Type	Description
Node	<code>temu_ListNode *</code>	The node, from which the value to be extracted

## 1.1.272. `temu_listPrepend`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_listPrepend(temu_List * List, temu_Propval Val)
```

### Description

Add an element to the beginning of a linked-list

### Parameters

Table 253. `temu_listPrepend` Parameters

Parameter	Type	Description
List	<code>temu_List *</code>	The list, to which the element to be added
Val	<code>temu_Propval</code>	The element to be added

## 1.1.273. `temu_listRemoveHead`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_Propval temu_listRemoveHead(temu_List * List)
```

### Description

Removes the first element of a linked-list

### Result

The element that was removed from the list

### Parameters

Table 254. `temu_listRemoveHead` Parameters

Parameter	Type	Description
List	<code>temu_List *</code>	The list, from which the first element to be removed

## 1.1.274. `temu_listRemoveTail`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_Propval temu_listRemoveTail(temu_List * List)
```

### Description

Removes the last element of a linked-list

### Result

The element that was removed from the list

### Parameters

Table 255. `temu_listRemoveTail` Parameters

Parameter	Type	Description
List	<code>temu_List *</code>	The list, from which the last element to be removed

## 1.1.275. `temu_loadBinaryImage`

## Include

```
#include "temu-c/Support/Loader.h"
```

## Signature

```
int temu_loadBinaryImage(void * mem, const char * file, uint64_t pa)
```

## Description

Loads a raw binary image to the memory object at the given address.

The function will assume that the file is a raw binary. This way you can load an ELF file as is to memory (e.g. where it is expected by the boot loader).

## Result

0 on success, other values indicates errors.

## Parameters

Table 256. `temu_loadBinaryImage` Parameters

Parameter	Type	Description
mem	void *	Memory object, must conform to the mem interface.
file	const char *	Path to file to load.
pa	uint64_t	Physical address where to load the image.

### 1.1.276. `temu_loadElfImage`

## Include

```
#include "temu-c/Support/Loader.h"
```

## Signature

```
int temu_loadElfImage(void * mem, const char * file, uint64_t pa)
```

## Description

Loads an ELF file without going through filetype detection

## Result

0 on success, other values indicates errors

## Parameters

Table 257. `temu_loadElfImage` Parameters

Parameter	Type	Description
mem	void *	Memory object, must conform to the mem interface.
file	const char *	Path to file to load.
pa	uint64_t	Unused argument

### 1.1.277. `temu_loadElfImageAndStartAddr`

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_loadElfImageAndStartAddr(void * Mem, const char * FileName, uint64_t * StartAddr)
```

#### Description

Loads an ELF file without going through filetype detection

#### Result

0 on success, 1 for successful load but without valid StartAddr, negative values indicate error.

#### Parameters

Table 258. `temu_loadElfImageAndStartAddr` Parameters

Parameter	Type	Description
Mem	void *	Memory object, must conform to the mem interface.
FileName	const char *	Path to file to load.
StartAddr	uint64_t *	Pointer where to place the start address of the elf file

### 1.1.278. `temu_loadImage`

## Include

```
#include "temu-c/Support/Loader.h"
```

## Signature

```
int temu_loadImage(void * mem, const char * file)
```

## Description

Loads a binary image to the memory object.

The function will autodetect the file type (based on extensions and magic headers in the file).

Binary images will be placed at address 0.

The image can be one of the supported file formats.

## Result

0 on success, other values indicates errors.

## Parameters

Table 259. `temu_loadImage` Parameters

Parameter	Type	Description
mem	void *	Memory object, must conform to the mem interface.
file	const char *	Path to file to load.

### 1.1.279. `temu_loadImageAndStartAddr`

## Include

```
#include "temu-c/Support/Loader.h"
```

## Signature

```
int temu_loadImageAndStartAddr(void * Mem, const char * FileName, uint64_t * StartAddr)
```

## Description

Loads a binary image to the memory object.

The function will autodetect the file type (based on extensions and magic headers in the file).

Binary images will be placed at address 0.

The image can be one of the supported file formats.

This specific variant, will return the start address of the binary, the StartAddr is typically embedded in the ELF or SREC files. The function therefore allows you to set the program counter to the correct start address after the function has been called.

## Result

0 on success, 1 for successful load but without valid StartAddr, negative values indicate error.

## Parameters

Table 260. `temu_loadImageAndStartAddr` Parameters

Parameter	Type	Description
Mem	void *	Memory object, must conform to the mem interface.
FileName	const char *	Path to file to load.
StartAddr	uint64_t *	Pointer to location to store the start address.

### 1.1.280. `temu_loadPlugin`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_loadPlugin(const char * PluginName)
```

#### Description

Load a plugin. When a plugin is loaded the `temu_pluginInit` function will be called. This function should create and define any classes that the plugin provides.

The function loads plugins following the operating system's loading rules. This means in particular that first, the plugin will be found in the RPATH location of `libTEMUSupport` (which refers to the TEMU lib directory). Secondly, it will look in the `LD_LIBRARY_PATH` and other standard load directories.

In practice, it will look for `lib Plugin .so` in the TEMU plugin paths, or `libTEMU Plugin .so` in the



same paths. It will then fallback on attempting to load libTEMU Plugin .so from the system load paths. If the plugin name contains a slash '/' it will be treated as a path and not a plugin name.

## Result

0 on success, non-zero otherwise.

## Parameters

Table 261. `temu_loadPlugin` Parameters

Parameter	Type	Description
PluginName	const char *	A path or plugin name

### 1.1.281. `temu_loadPluginGlobal`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_loadPluginGlobal(const char * PluginName)
```

#### Description

Load a plugin. When a plugin is loaded the `temu_pluginInit` function will be called. This function should create and define any classes that the plugin provides.

The plugin will be made available globally for the linker (e.g. using `RTLD_GLOBAL`).

This is not recommended for normal TEMU plugins.

The function loads plugins following the operating system's loading rules. This means in particular that first, the plugin will be found in the `RPATH` location of `libTEMUSupport` (which refers to the TEMU lib directory). Secondly, it will look in the `LD_LIBRARY_PATH` and other standard load directories.

In practice, it will look for lib Plugin .so in the TEMU plugin paths, or libTEMU Plugin .so in the same paths. It will then fallback on attempting to load libTEMU Plugin .so from the system load paths. If the plugin name contains a slash '/' it will be treated as a path and not a plugin name.

## Result

0 on success, non-zero otherwise.

## Parameters

Table 262. `temu_loadPluginGlobal` Parameters

Parameter	Type	Description
PluginName	const char *	A path or plugin name

### 1.1.282. `temu_loadSrecImage`

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_loadSrecImage(void * mem, const char * file, uint64_t pa)
```

#### Description

Loads an SREC file without going through filetype detection.

#### Result

0 on success, other values indicates errors

#### Parameters

Table 263. `temu_loadSrecImage` Parameters

Parameter	Type	Description
mem	void *	Memory object, must conform to the mem interface.
file	const char *	Path to file to load.
pa	uint64_t	Unused argument

### 1.1.283. `temu_loadSrecImageAndStartAddr`

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_loadSrecImageAndStartAddr(void * mem, const char * fileName, uint32_t * startAddr)
```

## Description

Loads an SREC file without going through filetype detection.

## Result

0 on success, 1 for successful load but without valid StartAddr, negative values indicate error.

## Parameters

Table 264. `temu_loadSrecImageAndStartAddr` Parameters

Parameter	Type	Description
mem	void *	Memory object, must conform to the mem interface.
fileName	const char *	Path to file to load.
startAddr	uint32_t *	Start address, will be set if result is 0

### 1.1.284. `temu_loadSymtab`

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_loadSymtab(const char * FileName, temu_Symtab ** Sym)
```

#### Description

Loads the symbol table from a given ELF file.

The symbol table is allocated and the pointer of the allocated symbol table is placed in Sym. The user is responsible for disposing the symbol table with `temu_disposeSymtab()`.

#### Result

0 on success, other values indicates errors.

#### Parameters

Table 265. `temu_loadSymtab` Parameters

Parameter	Type	Description
FileName	const char *	Name of ELF file

Parameter	Type	Description
Sym	temu_Symtab **	Address of your symtab pointer.

### 1.1.285. temu\_logConfigError

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logConfigError(const void * Obj, const char * Msg, ...)
```

#### Description

Log configuration error

#### Parameters

Table 266. temu\_logConfigError Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

### 1.1.286. temu\_logConfigFatal

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logConfigFatal(const void * Obj, const char * Msg, ...)
```

#### Description

Log configuration fatal error

#### Parameters

Table 267. temu\_logConfigFatal Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

### 1.1.287. temu\_logConfigInfo

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logConfigInfo(const void * Obj, const char * Msg, ...)
```

#### Description

Log configuration info

#### Parameters

Table 268. temu\_logConfigInfo Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

### 1.1.288. temu\_logConfigWarning

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logConfigWarning(const void * Obj, const char * Msg, ...)
```

#### Description

Log configuration warning

#### Parameters

Table 269. temu\_logConfigWarning Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

### 1.1.289. temu\_logDebugFunc

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logDebugFunc(const void * Obj, const char * Msg, ...)
```

#### Description

Log a message with debug severity

#### Parameters

Table 270. temu\_logDebugFunc Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

### 1.1.290. temu\_logError

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logError(const void * Obj, const char * Msg, ...)
```

#### Description

Log a message with error severity

#### Parameters

Table 271. temu\_logError Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

### 1.1.291. temu\_logFatal

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logFatal(const void * Obj, const char * Msg, ...)
```

#### Description

Log a message with fatal severity; The program will terminate after calling this

#### Parameters

Table 272. temu\_logFatal Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

### 1.1.292. temu\_logInfo

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logInfo(const void * Obj, const char * Msg, ...)
```

#### Description

Log a message with info severity

#### Parameters

Table 273. temu\_logInfo Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

### 1.1.293. temu\_logSetAdvancedFunc

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logSetAdvancedFunc(void (*)(void *, temu_Object *, unsigned int,  

  temu_LogLevel, const char *) LogFunc, void * UserData)
```

#### Description

Set advanced logging function.

It is possible to provide a custom function for handling logging messages from TEMU. This can be used by a simulator that provides a centralized logging facility to also handle TEMU logging messages. By default, the messages will be printed to stderr using fputs.

The advanced logging function does not get messages with a terminating linefeed. In addition, the function receives the object, category and log level as its own parameters.

The message does not contain object name / time stamps. The integrator would be expected to extract the time stamp from the objects time source if set.

LogFunc will take the following parameters:

#### User data

Void pointer passed to UserData parameter (may be NULL).

#### Object pointer

Pointer to object issuing the log message (may be NULL).

#### Category

Category id of log message.

#### Log level

Severity of logging message.

#### Message

The log message, as a '



' terminated string.

## Parameters

Table 274. `temu_logSetAdvancedFunc` Parameters

Parameter	Type	Description
LogFunc	void (*)(void *, temu_Object *, unsigned int, temu_LogLevel, const char *)	Logging function to use. NULL to restore the default.
UserData	void *	User data to pass to the logging functions first parameter.

### 1.1.294. `temu_logSetColour`

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logSetColour(int Enable)
```

#### Description

Enable colours on logging

#### Parameters

Table 275. `temu_logSetColour` Parameters

Parameter	Type	Description
Enable	int	0 to disable colours, 1 to enable

### 1.1.295. `temu_logSetDefaultFile`

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logSetDefaultFile(FILE * FP)
```

## Description

Set the logging file for the default logging function. Pass NULL to restore default (stderr).

## Parameters

Table 276. `temu_logSetDefaultFile` Parameters

Parameter	Type	Description
FP	FILE *	Logging file pointer, NULL to restore default.

## 1.1.296. `temu_logSetFunc`

### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_logSetFunc(void (*)(const char *) LogFunc)
```

### Description

Set logging function.

It is possible to provide a custom function for handling logging messages from TEMU. This can be used by a simulator that provides a centralized logging facility to also handle TEMU logging messages. By default, the messages will be printed to stderr using `fputs`. Note that messages will be terminated by " n

", so if your logging system adds a linefeed, the message may need to be transformed.

### Parameters

Table 277. `temu_logSetFunc` Parameters

Parameter	Type	Description
LogFunc	void (*)(const char *)	Logging function to use. NULL to restore the default.

## 1.1.297. `temu_logSetLevel`

### Include

```
#include "temu-c/Support/Logging.h"
```

## Signature

```
void temu_logSetLevel(temu_LogLevel LogLevel)
```

## Description

Set logging level. Messages will be logged if they are of the set level or higher priority.

## Parameters

Table 278. `temu_logSetLevel` Parameters

Parameter	Type	Description
LogLevel	temu_LogLevel	The logging level.

### 1.1.298. temu\_logSetSeverity

#### Include

```
#include "temu-c/Support/Logging.h"
```

## Signature

```
void temu_logSetSeverity(void * Obj, unsigned int Category, temu_LogLevel Severity)
```

## Description

Set severity for specific category.

## Parameters

Table 279. `temu_logSetSeverity` Parameters

Parameter	Type	Description
Obj	void *	Object for which to modify logging severity
Category	unsigned int	Category to change severity for (e.g. teLC_DefaultCat)
Severity	temu_LogLevel	The log level for which messages shall be emitted.

### 1.1.299. temu\_logSimError

#### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_logSimError(const void * Obj, const char * Msg, ...)
```

### Description

Log simulation error

### Parameters

Table 280. `temu_logSimError` Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.300. `temu_logSimFatal`

### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_logSimFatal(const void * Obj, const char * Msg, ...)
```

### Description

Log fatal issue Fatal messages result in termination of the program after the message.

### Parameters

Table 281. `temu_logSimFatal` Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.301. `temu_logSimInfo`

### Include

```
#include "temu-c/Support/Logging.h"
```

## Signature

```
void temu_logSimInfo(const void * Obj, const char * Msg, ...)
```

## Description

Log simulation info

## Parameters

Table 282. *temu\_logSimInfo* Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.302. temu\_logSimWarning

### Include

```
#include "temu-c/Support/Logging.h"
```

## Signature

```
void temu_logSimWarning(const void * Obj, const char * Msg, ...)
```

## Description

Log simulation warning

## Parameters

Table 283. *temu\_logSimWarning* Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.303. temu\_logTargetError

### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_logTargetError(const void * Obj, const char * Msg, ...)
```

### Description

Log target error

### Parameters

Table 284. `temu_logTargetError` Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.304. `temu_logTargetFatal`

### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_logTargetFatal(const void * Obj, const char * Msg, ...)
```

### Description

Log target fatal error This should typically never happen, but it may be convenient in some cases.

### Parameters

Table 285. `temu_logTargetFatal` Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.305. `temu_logTargetInfo`

### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_logTargetInfo(const void * Obj, const char * Msg, ...)
```

### Description

Log target info

### Parameters

Table 286. `temu_logTargetInfo` Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.306. `temu_logTargetWarning`

### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_logTargetWarning(const void * Obj, const char * Msg, ...)
```

### Description

Log target warning

### Parameters

Table 287. `temu_logTargetWarning` Parameters

Parameter	Type	Description
Obj	const void *	Object
Msg	const char *	Printf formatted message string

## 1.1.307. `temu_logToCategory`

### Include

```
#include "temu-c/Support/Logging.h"
```

## Signature

```
void temu_logToCategory(const void * Obj, unsigned int Category, temu_LogLevel  
Severity, const char * Msg, ...)
```

## Description

Log message in the numbered category

## Parameters

Table 288. `temu_logToCategory` Parameters

Parameter	Type	Description
Obj	const void *	Object that is the source of the log message
Category	unsigned int	Category of message (e.g. <code>teLC_DefaultCat</code> )
Severity	temu_LogLevel	The log level for which the message shall be emitted.
Msg	const char *	Printf formatted message string.

## 1.1.308. `temu_logTrace`

### Include

```
#include "temu-c/Support/Logging.h"
```

## Signature

```
void temu_logTrace(const void * Obj, const char * Msg, ...)
```

## Description

Log a message with trace severity

## Parameters

Table 289. `temu_logTrace` Parameters



Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

### 1.1.309. temu\_logWarning

#### Include

```
#include "temu-c/Support/Logging.h"
```

#### Signature

```
void temu_logWarning(const void * Obj, const char * Msg, ...)
```

#### Description

Log a message with warning severity

#### Parameters

Table 290. temu\_logWarning Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

### 1.1.310. temu\_machineReset

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_machineReset(void * Machine, int ResetType)
```

#### Description

Reset the Machine

Resetting the Machine will result in a reset cascade where all connected devices and processors are also reset.

## Parameters

Table 291. `temu_machineReset` Parameters

Parameter	Type	Description
Machine	void *	The Machine object
ResetType	int	The type of reset, by convention 0 means cold reset, and 1 indicates a warm reset.

### 1.1.311. `temu_machineRun`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_machineRun(void * Machine, uint64_t NanoSecs)
```

#### Description

Run the machine for a number of nanoseconds

The function runs the machine and for a number of nanoseconds. This will run all the CPUs in the system.

In case a processor halts or enters idle mode, the function returns early.

#### Result

The number of executed nanoseconds.

#### Parameters

Table 292. `temu_machineRun` Parameters

Parameter	Type	Description
Machine	void *	The machine object
NanoSecs	uint64_t	The number of nanosecs to run each processor for.

### 1.1.312. `temu_mapMemorySpace`

#### Include

```
#include "temu-c/Memory/Memory.h"
```

### Signature

```
int temu_mapMemorySpace(void * Obj, uint64_t Addr, uint64_t Len, temu_Object * MemObj)
```

### Description

Map memory object into address space

### Result

Zero on success, other values indicates that the mapping failed

### Parameters

Table 293. `temu_mapMemorySpace` Parameters

Parameter	Type	Description
Obj	void *	The memory space object
Addr	uint64_t	Physical address where to map the device
Len	uint64_t	Length in bytes of area where the object is mapped.
MemObj	temu_Object *	The memory object. This object must correspond to the MemAccessIface

## 1.1.313. temu\_mapMemorySpaceFlags

### Include

```
#include "temu-c/Memory/Memory.h"
```

### Signature

```
int temu_mapMemorySpaceFlags(void * Obj, uint64_t Addr, uint64_t Len, temu_Object * MemObj, uint32_t Flags)
```

### Description

Map memory object into address space with flags

## Result

Zero on success, other values indicates that the mapping failed

## Parameters

Table 294. `temu_mapMemorySpaceFlags` Parameters

Parameter	Type	Description
Obj	void *	The memory space object
Addr	uint64_t	Physical address where to map the device
Len	uint64_t	Length in bytes of area where the object is mapped.
MemObj	<a href="#">temu_Object</a> *	The memory object. This object must correspond to the MemAccessIface
Flags	uint32_t	Sticky flags for memory accesses to that object (e.g. cachable)

### 1.1.314. `temu_memoryClearAttr`

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
void temu_memoryClearAttr(void * Obj, uint64_t Addr, uint64_t Len, temu_MemoryAttr Attr)
```

#### Description

Clears an attribute on the given memory range

#### Parameters

Table 295. `temu_memoryClearAttr` Parameters

Parameter	Type	Description
Obj	void *	Memory space object
Addr	uint64_t	Physical address of start
Len	uint64_t	Length of memory range

Parameter	Type	Description
Attr	temu_MemoryAttr	Attribute to clear

### 1.1.315. temu\_memoryGetAttrs

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
temu_MemoryAttrs temu_memoryGetAttrs(void * Obj, uint64_t Addr)
```

#### Description

Get memory attributes for address

#### Result

Memory attributes for the address

#### Parameters

Table 296. temu\_memoryGetAttrs Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address

### 1.1.316. temu\_memoryInstallTrampoline

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryInstallTrampoline(void * Obj, uint64_t Addr, void (*)(void *) Tramp)
```

#### Description

Install trampoline function in PDC cache

## Result

0 on success

## Parameters

Table 297. `temu_memoryInstallTrampoline` Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Tramp	void (*)(void *)	Trampoline function, takes CPU pointer as first argument.

### 1.1.317. `temu_memoryMap`

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryMap(void * Obj, uint64_t Addr, uint64_t Len, void * MemObj, uint32_t
Flags)
```

#### Description

`temu_memoryMap` Maps an object into a memory space. The object must have an interface named `MemAccessIface`, of the type defined as `TEMU_MEM_ACCESS_IFACE_TYPE`

#### Result

Zero on success

#### Parameters

Table 298. `temu_memoryMap` Parameters

Parameter	Type	Description
Obj	void *	is the memory space object
Addr	uint64_t	the physical address
Len	uint64_t	the length in bytes

Parameter	Type	Description
MemObj	void *	is the object that is mapped into the memory space it needs to be of a class that follows the rules above
Flags	uint32_t	are flags that are copied into the memory transaction object's flag field when a transaction reaches an object

### 1.1.318. temu\_memoryMapNamedIface

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryMapNamedIface(void * Obj, uint64_t Addr, uint64_t Len, void * MemObj,
const char * IfaceName, uint32_t Flags)
```

#### Description

temu\_memoryMapNamedIface Maps an object into a memory space using the named memory access interface. The interface named by IfaceName must be of the type TEMU\_MEM\_ACCESS\_IFACE\_TYPE

#### Result

Zero on success

#### Parameters

Table 299. temu\_memoryMapNamedIface Parameters

Parameter	Type	Description
Obj	void *	is the memory space object
Addr	uint64_t	the physical address
Len	uint64_t	the length in bytes
MemObj	void *	is the object that is mapped into the memory space it needs to be of a class that follows the rules above

Parameter	Type	Description
IfaceName	const char *	Name of the memory access interface
Flags	uint32_t	are flags that are copied into the memory transaction object's flag field when a transaction reaches an object

### 1.1.319. temu\_memoryRead

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryRead(void * mem, uint8_t * buff, uint64_t addr, uint32_t size, int swap)
```

#### Description

Read block of data via memory block transfer interfaces

#### Result

Negative on failures. Other values indicate success. The convention is to return the number of bytes read which should be == size.

#### Parameters

Table 300. temu\_memoryRead Parameters

Parameter	Type	Description
mem	void *	Pointer to memory space object
buff	uint8_t *	The buffer to which the memory should be stored
addr	uint64_t	The address of the memory block to be read
size	uint32_t	The size to be read
swap	int	setting this to 0 indicates reading a byte array, 1 a uint16 array, 2 a uint32 array and 3 a uint64 array



## 1.1.320. temu\_memoryReadData

### Include

```
#include "temu-c/Support/Memory.h"
```

### Signature

```
int temu_memoryReadData(void * obj, uint64_t addr, uint8_t * buff, unsigned int  
unitSize, uint32_t size, unsigned int flags)
```

### Description

Read block of data via large memory transactions.

### Result

Negative on failures. Other values indicate success.

### Parameters

Table 301. temu\_memoryReadData Parameters

Parameter	Type	Description
obj	void *	Memory space object
addr	uint64_t	Physical address inside memory space
buff	uint8_t *	Data buffer
unitSize	unsigned int	Log size of transaction unit size (0 = u8, 1 = u16, 2 = u32, 3 = u64)
size	uint32_t	Number of units to transfer
flags	unsigned int	Memory transaction flags (e.g. TEMU_MT_LITTLE_ENDIAN)

## 1.1.321. temu\_memoryReadPhys16

### Include

```
#include "temu-c/Support/Memory.h"
```

### Signature

```
int temu_memoryReadPhys16(void * Obj, uint64_t Addr, uint16_t * Word)
```

### Description

Issue a big endian memory read transaction (without initiator)

### Result

0 on success, other values imply failure and will not update the word.

### Parameters

Table 302. `temu_memoryReadPhys16` Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint16_t *	16 bit word that is read

## 1.1.322. `temu_memoryReadPhys16Little`

### Include

```
#include "temu-c/Support/Memory.h"
```

### Signature

```
int temu_memoryReadPhys16Little(void * Obj, uint64_t Addr, uint16_t * Word)
```

### Description

Issue a little endian memory read transaction (without initiator)

### Result

0 on success, other values imply failure and will not update the word.

### Parameters

Table 303. `temu_memoryReadPhys16Little` Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address

Parameter	Type	Description
Word	uint16_t *	16 bit word that is read

### 1.1.323. temu\_memoryReadPhys32

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryReadPhys32(void * Obj, uint64_t Addr, uint32_t * Word)
```

#### Description

Issue a big endian memory read transaction (without initiator)

#### Result

0 on success, other values imply failure and will not update the word.

#### Parameters

Table 304. temu\_memoryReadPhys32 Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint32_t *	32 bit word that is read

### 1.1.324. temu\_memoryReadPhys32Little

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryReadPhys32Little(void * Obj, uint64_t Addr, uint32_t * Word)
```

#### Description

Issue a little endian memory read transaction (without initiator)

## Result

0 on success, other values imply failure and will not update the word.

## Parameters

Table 305. `temu_memoryReadPhys32Little` Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint32_t *	32 bit word that is read

### 1.1.325. `temu_memorySetAttr`

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
void temu_memorySetAttr(void * Obj, uint64_t Addr, uint64_t Len, temu_MemoryAttr Attr)
```

#### Description

Sets an attribute on the given memory range

#### Parameters

Table 306. `temu_memorySetAttr` Parameters

Parameter	Type	Description
Obj	void *	Memory space object
Addr	uint64_t	Physical address of start
Len	uint64_t	Length of memory range
Attr	temu_MemoryAttr	Attribute to set

### 1.1.326. `temu_memoryWrite`

#### Include

```
#include "temu-c/Support/Memory.h"
```

## Signature

```
int temu_memoryWrite(void * mem, uint64_t addr, uint8_t * buff, uint32_t size, int swap)
```

## Description

Write block of data via memory block transfer interfaces

## Result

Negative on failure. Other values indicate success. The convention is to return the number of bytes written, which should be == size.

## Parameters

Table 307. `temu_memoryWrite` Parameters

Parameter	Type	Description
mem	void *	Pointer to memory space object
buff	uint8_t *	The buffer to which the memory should be stored
addr	uint64_t	The address, at which the write should be done
size	uint32_t	The size to be read
swap	int	Negative on failures. Other values indicate success.

### 1.1.327. `temu_memoryWriteData`

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryWriteData(void * obj, uint64_t addr, const uint8_t * buff, unsigned int unitSize, uint32_t size, unsigned int flags)
```

#### Description

Write block of data via large memory transactions.

#### Result

Negative on failures. Other values indicate success.

## Parameters

Table 308. `temu_memoryWriteData` Parameters

Parameter	Type	Description
<code>obj</code>	<code>void *</code>	Memory space object
<code>addr</code>	<code>uint64_t</code>	Physical address inside memory space
<code>buff</code>	<code>const uint8_t *</code>	Data buffer
<code>unitSize</code>	<code>unsigned int</code>	Log size of transaction unit size (0 = u8, 1 = u16, 2 = u32, 3 = u64)
<code>size</code>	<code>uint32_t</code>	Number of units to transfer
<code>flags</code>	<code>unsigned int</code>	Memory transaction flags (e.g. <code>TEMU_MT_LITTLE_ENDIAN</code> )

### 1.1.328. `temu_memoryWritePhys32`

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
int temu_memoryWritePhys32(void * Obj, uint64_t Addr, uint32_t Word)
```

#### Description

Issue a big endian memory write transaction (without initiator)

#### Result

0 on success

#### Parameters

Table 309. `temu_memoryWritePhys32` Parameters

Parameter	Type	Description
<code>Obj</code>	<code>void *</code>	Memory space
<code>Addr</code>	<code>uint64_t</code>	Physical address
<code>Word</code>	<code>uint32_t</code>	32 bit word to write

## 1.1.329. temu\_memoryWritePhys32Little

### Include

```
#include "temu-c/Support/Memory.h"
```

### Signature

```
int temu_memoryWritePhys32Little(void * Obj, uint64_t Addr, uint32_t Word)
```

### Description

Issue a little endian memory write transaction (without initiator)

### Result

0 on success

### Parameters

Table 310. temu\_memoryWritePhys32Little Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint32_t	32 bit word to write

## 1.1.330. temu\_nameForClass

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
const char * temu_nameForClass(temu_Class * Cls)
```

### Description

temu\_nameForClass Get the class name from its object

### Result

The name of the class as a C-string

## Parameters

Table 311. `temu_nameForClass` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	Pointer to the object of the class in question

### 1.1.331. `temu_nameForInterface`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
const char * temu_nameForInterface(const temu_Object * Obj, const void * Iface)
```

#### Description

Get name for interface

#### Result

The name of the interface as C-string

#### Parameters

Table 312. `temu_nameForInterface` Parameters

Parameter	Type	Description
Obj	<code>const temu_Object *</code>	Pointer to the object that contains the interface
Iface	<code>const void *</code>	Pointer to the interface

### 1.1.332. `temu_nameForObject`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
const char * temu_nameForObject(const temu_Object * Obj)
```



## Description

Get the name of an object

## Result

Name of the object as C-string

## Parameters

Table 313. `temu_nameForObject` Parameters

Parameter	Type	Description
Obj	const <code>temu_Object *</code>	Pointer to the object

## 1.1.333. `temu_nanosToCycles`

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
int64_t temu_nanosToCycles(int64_t Nanos, int64_t Freq)
```

### Description

Convert nanoseconds to cycles

This function truncates the result in case of non-exact conversion.

### Result

Nanoseconds converted to cycles

### Parameters

Table 314. `temu_nanosToCycles` Parameters

Parameter	Type	Description
Nanos	int64_t	Nanoseconds to convert
Freq	int64_t	Frequency in Hz

## 1.1.334. `temu_nanosToCyclesRoundedUp`

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
int64_t temu_nanosToCyclesRoundedUp(int64_t Nanos, int64_t Freq)
```

### Description

Convert nanoseconds to cycles rounded upwards

This function rounds up the result in case of non-exact conversion.

### Result

Nanoseconds converted to cycles

### Parameters

Table 315. `temu_nanosToCyclesRoundedUp` Parameters

Parameter	Type	Description
Nanos	int64_t	Nanoseconds to convert
Freq	int64_t	Frequency in Hz

## 1.1.335. `temu_nanosToSecs`

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
double temu_nanosToSecs(int64_t Nanos)
```

### Description

Convert nanoseconds to seconds

### Result

Nanoseconds converted to seconds

### Parameters

Table 316. `temu_nanosToSecs` Parameters

Parameter	Type	Description
Nanos	int64_t	Nanoseconds to convert

### 1.1.336. temu\_normaliseRead16

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
uint16_t temu_normaliseRead16(uint16_t Value, int Sz, int Off)
```

#### Description

Normalise a value for reads where only 16 bit transactions are supported by the device model, but the target is allowed to read non-16 bit quantities.

Given a 32 bit register value, the function extracts the bits from it that was actually requested.

#### Result

#### Parameters

Table 317. temu\_normaliseRead16 Parameters

Parameter	Type	Description
Value	uint16_t	Register value
Sz	int	Size in log number of bytes (0 or 1)
Off	int	Offset in bytes within the 32 bit word of the transaction (0 or 2)

### 1.1.337. temu\_normaliseRead32

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
uint32_t temu_normaliseRead32(uint32_t Value, int Sz, int Off)
```

## Description

Normalise a value for reads where only 32 bit transactions are supported by the device model, but the target is allowed to read non-32 bit quantities.

Given a 32 bit register value, the function extracts the bits from it that was actually requested.

## Result

## Parameters

Table 318. `temu_normaliseRead32` Parameters

Parameter	Type	Description
Value	<code>uint32_t</code>	Register value
Sz	<code>int</code>	Size in log number of bytes (0, 1, or 2)
Off	<code>int</code>	Offset in bytes within the 32 bit word of the transaction (0-3)

### 1.1.338. `temu_normaliseWrite16`

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
uint16_t temu_normaliseWrite16(uint16_t OldVal, uint16_t NewVal, int Sz, int Off)
```

## Description

Normalise a value for writes where only 16 bit transactions are supported by the device model, but the target is allowed to write non-16 bit quantity. The function mixes the old register value with the new one based on the size and offset parameter. The problem exists because the memory access interface has a value entry, which always ends up in the lower bits, so if we write to the higher bits in in a 16 bit register, then by just forwarding the the value as is to the write handler, will result in a write of the lower bits and a clear of the upper bits (for a store unsigned).

Thus a normalisation is needed where we mix the written word with the old bits. So for transaction size of 8, and an offset of 8, the resulting word is (old

`0x00ff`) | (new

8)

## Result

## Parameters

Table 319. `temu_normaliseWrite16` Parameters

Parameter	Type	Description
OldVal	uint16_t	Old register value.
NewVal	uint16_t	Content in memory transaction (new value).
Sz	int	Size in log number of bytes
Off	int	Offset in bytes within the 16 bit word of the transaction.

### 1.1.339. `temu_normaliseWrite32`

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Signature

```
uint32_t temu_normaliseWrite32(uint32_t OldVal, uint32_t NewVal, int Sz, int Off)
```

#### Description

Normalise a value for writes where only 32 bit transactions are supported by the device model, but the target is allowed to write non-32 bit quantity. The function mixes the old register value with the new one based on the size and offset parameter. The problem exists because the memory access interface has a value entry, which always ends up in the lower bits, so if we write to the higher bits in in a 32 bit register, then by just forwarding the the value as is to the write handler, will result in a write of the lower bits and a clear of the upper bits (for a store unsigned).

Thus a normalisation is needed where we mix the written word with the old bits. So for transaction size of 16, and an offset of 16, the resulting word is (old

`0x0000ffff`) | (new

16)

## Result

## Parameters

Table 320. `temu_normaliseWrite32` Parameters

Parameter	Type	Description
OldVal	uint32_t	Old register value.
NewVal	uint32_t	Content in memory transaction (new value).
Sz	int	Size in log number of bytes
Off	int	Offset in bytes within the 32 bit word of the transaction.

### 1.1.340. temu\_notify

#### Include

```
#include "temu-c/Support/Notifications.h"
```

#### Signature

```
void temu_notify(int64_t Id, void * NotInfo)
```

#### Description

Call event subscriber, EvInfo is a per event specific struct the event handler must cast this to the appropriate type.

#### Parameters

Table 321. temu\_notify Parameters

Parameter	Type	Description
Id	int64_t	Notification ID
NotInfo	void *	Ponter to pass to notification handlers

### 1.1.341. temu\_notifyFast

#### Include

```
#include "temu-c/Support/Notifications.h"
```

#### Signature

```
void temu_notifyFast(int64_t * Id, void * NotInfo)
```

## Description

Quick inline call macro that do not call the function for an invalid event id. This saves the function call to the notify function, minimising the cost for unset event handlers.

## Parameters

Table 322. `temu_notifyFast` Parameters

Parameter	Type	Description
Id	<code>int64_t *</code>	Notification ID
NotInfo	<code>void *</code>	Pointer to pass to notification handlers

### 1.1.342. `temu_objectForName`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Object * temu_objectForName(const char * Name)
```

#### Description

Get object for name

#### Result

Pointer to the object

#### Parameters

Table 323. `temu_objectForName` Parameters

Parameter	Type	Description
Name	<code>const char *</code>	The name of the object

### 1.1.343. `temu_objectGetLogLevel`

#### Include

```
#include "temu-c/Support/Logging.h"
```

## Signature

```
temu_LogLevel temu_objectGetLogLevel(void * Obj, unsigned int Category)
```

## Description

Get object specific logging level for category

## Result

Logging level for the given **Category**

## Parameters

Table 324. `temu_objectGetLogLevel` Parameters

Parameter	Type	Description
Obj	void *	Object
Category	unsigned int	Category

## 1.1.344. temu\_objectHasCmd

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_objectHasCmd(const temu_Object * Obj, const char * CmdName)
```

## Description

Check if object has a command

## Result

0 in case command does not exist. Otherwise 1.

## Parameters

Table 325. `temu_objectHasCmd` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Object pointer
CmdName	const char *	Name of command



## 1.1.345. temu\_objectHasIface

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_objectHasIface(const temu_Object * Obj, const char * IfaceName)
```

### Description

Check if the object has a named interface

### Result

0 if the interface does not exist, otherwise the interface exists

### Parameters

Table 326. temu\_objectHasIface Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object to query
IfaceName	const char *	Interface name to look for

## 1.1.346. temu\_objectHasProp

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_objectHasProp(const temu_Object * Obj, const char * PropName)
```

### Description

Check if the object has a named property

### Result

0 if the property does not exist, otherwise the property is valid

## Parameters

Table 327. `temu_objectHasProp` Parameters

Parameter	Type	Description
Obj	<code>const temu_Object *</code>	Pointer to the object of the property
PropName	<code>const char *</code>	Name of the property

### 1.1.347. `temu_objectInvokeCmd`

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Signature

```
int temu_objectInvokeCmd(temu_Object * Obj, void * I, const char * Name, int Argc,
temu_CmdArg * Argv)
```

#### Description

Invoke command on object

#### Result

Command result. 0 means success.

#### Parameters

Table 328. `temu_objectInvokeCmd` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object
I	<code>void *</code>	Interpreter context
Name	<code>const char *</code>	Name of command
Argc	<code>int</code>	Number of arguments
Argv	<code>temu_CmdArg *</code>	Argument vector

### 1.1.348. `temu_objectSetLogLevel`

#### Include

```
#include "temu-c/Support/Logging.h"
```

### Signature

```
void temu_objectSetLogLevel(void * Obj, unsigned int Category, temu_LogLevel LogLevel)
```

### Description

Set object specific logging level for category

### Parameters

Table 329. `temu_objectSetLogLevel` Parameters

Parameter	Type	Description
Obj	void *	Object
Category	unsigned int	Category
LogLevel	temu_LogLevel	Level to set for the given Category

## 1.1.349. temu\_objsysClear

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_objsysClear()
```

### Description

Erase all classes, interfaces, properties and objects registered in the object system.

## 1.1.350. temu\_objsysClearObjects

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_objsysClearObjects()
```

### Description

Erase all objects, but do not delete classes.

## 1.1.351. temu\_parity32

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
int temu_parity32(uint32_t Word)
```

### Description

Compute parity of word

### Result

The parity of the word

### Parameters

Table 330. `temu_parity32` Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which parity to be computed

## 1.1.352. temu\_parity64

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
int temu_parity64(uint64_t Word)
```

### Description

## Compute parity

### Result

The parity of word

### Parameters

Table 331. `temu_parity64` Parameters

Parameter	Type	Description
Word	<code>uint64_t</code>	The word, in which parity to be computed

## 1.1.353. `temu_parseCommandLineOptions`

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
int temu_parseCommandLineOptions(int argc, const char ** argv)
```

### Description

Parses command line options The TEMU command supports a number of built in command line options. A simulator embedding TEMU may want initialise these options in the same way that the TEMU CLI does. Note that not all options are supported this way as some options will only be registered by the temu CLI application itself.

In general interactive options will not work (options that can be interpreted as a command). It is possible to use `temu_printCommandLineHelp()` to list currently registered options from an application that embeds temu.

### Result

0 on success, otherwise non-zero value.

### Parameters

Table 332. `temu_parseCommandLineOptions` Parameters

Parameter	Type	Description
<code>argc</code>	<code>int</code>	Number of arguments
<code>argv</code>	<code>const char **</code>	Command line arguments

### 1.1.354. temu\_pluginPathAppend

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_pluginPathAppend(const char * Path)
```

#### Description

temu\_pluginPathAppend Add a path to the list of paths, where T-emu searches for plugins

#### Parameters

Table 333. temu\_pluginPathAppend Parameters

Parameter	Type	Description
Path	const char *	The path to append

### 1.1.355. temu\_pluginPathPrint

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_pluginPathPrint()
```

#### Description

temu\_pluginPathPrint Print the list of paths, where T-emu searches for plugins

### 1.1.356. temu\_pluginPathRemove

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_pluginPathRemove(const char * Path)
```

### Description

temu\_pluginPathRemove Remove a path from the list of paths, where T-emu searches for plugins

### Parameters

Table 334. temu\_pluginPathRemove Parameters

Parameter	Type	Description
Path	const char *	to remove

## 1.1.357. temu\_popcount32

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
int temu_popcount32(uint32_t Word)
```

### Description

Count number of bits set in word

### Result

The number of bits

### Parameters

Table 335. temu\_popcount32 Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which bits to be counted

## 1.1.358. temu\_popcount64

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
int temu_popcount64(uint64_t Word)
```

## Description

Count count number of bit set

## Result

The number of bits set in Word

## Parameters

Table 336. `temu_popcount64` Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which bits to be counted

### 1.1.359. `temu_postCallback`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
void temu_postCallback(void * Obj, temu_SafeCb Cb, void * Arg)
```

## Description

Post a callback in a time source

The posting will be thread-safe and the callback will be executed by the main thread (the one calling the `cpu` or `machine run / step` functions). The main thread is the thread calling `cpu run` if there is only a single `cpu` in the system, and the thread calling `machine run` if there are multiple CPUs in the system.

The callback will be executed as soon as possible, when the event queue is checked for events. Which in practice mean:

- At the end of the current time quanta for multi-cpu systems.
- At regular intervals for single CPU systems (a CPU runs a null-event periodically. Note that the callback will be called as soon as either a normal or the null event is executed).



The posting of this callback is thread-safe but not async / signal safe. I.e. do not do this from signal handler or async io callbacks.

When the event is executed, it is safe to do most emulator operations, including posting of events, calling API functions, etc.

The callback must however not:

- Manipulate the object system meta state (i.e. creating or modifying classes)
- Replace signal handlers (e.g. SIGINT)
- Run or step the emulator in any way (e.g. by calling `machineRun()`, `cpuRun()` or using the `CpuIface` or `MachineIface` interfaces.

## Parameters

Table 337. `temu_postCallback` Parameters

Parameter	Type	Description
Obj	<code>void *</code>	Time source
Cb	<code>temu_SafeCb</code>	Callback
Arg	<code>void *</code>	Argument passed to callback

## 1.1.360. `temu_ppcClearAddressReservation`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_ppcClearAddressReservation(void * Cpu)
```

### Description

Clear the PowerPC reserve address used by `lwarx` and `stwcx`

### Parameters

Table 338. `temu_ppcClearAddressReservation` Parameters

Parameter	Type	Description
Cpu	<code>void *</code>	Processor pointer

## 1.1.361. `temu_ppcGetCrReg`

**Include**

```
#include "temu-c/Support/Cpu.h"
```

**Signature**

```
uint32_t temu_ppcGetCrReg(void * Cpu)
```

**Description**

Get the PowerPC CR register

**Result**

CR value

**Parameters**

*Table 339. temu\_ppcGetCrReg Parameters*

Parameter	Type	Description
Cpu	void *	Processor pointer

**1.1.362. temu\_ppcGetCtrReg****Include**

```
#include "temu-c/Support/Cpu.h"
```

**Signature**

```
uint32_t temu_ppcGetCtrReg(void * Cpu)
```

**Description**

Get the PowerPC CTR register

**Result**

The CTR value

**Parameters**

*Table 340. temu\_ppcGetCtrReg Parameters*

Parameter	Type	Description
Cpu	void *	Processor pointer

### 1.1.363. temu\_ppcGetFpscrReg

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint32_t temu_ppcGetFpscrReg(void * Cpu)
```

#### Description

Get the PowerPC FPSCR register

#### Result

The FPSCR value

#### Parameters

Table 341. temu\_ppcGetFpscrReg Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer

### 1.1.364. temu\_ppcGetLrReg

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_ppcGetLrReg(void * Cpu)
```

#### Description

Get the PowerPC LR register

#### Result

The LR value

## Parameters

Table 342. `temu_ppcGetLrReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer

### 1.1.365. `temu_ppcGetMsrReg`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_ppcGetMsrReg(void * Cpu)
```

#### Description

Get the PowerPC MSR register

#### Result

MSR value

#### Parameters

Table 343. `temu_ppcGetMsrReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer

### 1.1.366. `temu_ppcGetReserveAddress`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_ppcGetReserveAddress(void * Cpu)
```

## Description

Get the PowerPC reserve address used by `lwarx` and `stwcx`

## Result

The reserved address value

## Parameters

Table 344. `temu_ppcGetReserveAddress` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer

## 1.1.367. `temu_ppcGetXerReg`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_ppcGetXerReg(void * Cpu)
```

### Description

Get the PowerPC XER register

### Result

XER value

### Parameters

Table 345. `temu_ppcGetXerReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer

## 1.1.368. `temu_ppcIsReservationBitSet`

### Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
int temu_ppcIsReservationBitSet(void * Cpu)
```

## Description

Check if PowerPC reserve address is set `lwarx` and `stwcx`

## Result

1 if reservation is set, 0 if cleared.

## Parameters

Table 346. `temu_ppcIsReservationBitSet` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer

### 1.1.369. `temu_ppcSetCrReg`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

## Signature

```
void temu_ppcSetCrReg(void * Cpu, uint32_t Value)
```

## Description

Set the PowerPC CR register

## Parameters

Table 347. `temu_ppcSetCrReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
Value	uint32_t	New CR value

### 1.1.370. `temu_ppcSetCtrReg`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_ppcSetCtrReg(void * Cpu, uint32_t Value)
```

### Description

Set the PowerPC CTR register

### Parameters

Table 348. `temu_ppcSetCtrReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
Value	uint32_t	New CTR value

## 1.1.371. `temu_ppcSetFpscrReg`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_ppcSetFpscrReg(void * Cpu, uint32_t Value)
```

### Description

Set the PowerPC FPSCR register

### Parameters

Table 349. `temu_ppcSetFpscrReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
Value	uint32_t	New FPSCR value

## 1.1.372. `temu_ppcSetLrReg`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_ppcSetLrReg(void * Cpu, uint64_t Value)
```

### Description

Set the PowerPC LR register

### Parameters

Table 350. `temu_ppcSetLrReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
Value	uint64_t	New LR value

## 1.1.373. `temu_ppcSetMsrReg`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_ppcSetMsrReg(void * Cpu, uint64_t Value)
```

### Description

Set the PowerPC MSR register

### Parameters

Table 351. `temu_ppcSetMsrReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
Value	uint64_t	New MSR value

## 1.1.374. `temu_ppcSetReserveAddress`

### Include



```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_ppcSetReserveAddress(void * Cpu, uint64_t reserveAddress)
```

### Description

Set the PowerPC reserve address used by `lwarx` and `stwcx`

### Parameters

Table 352. `temu_ppcSetReserveAddress` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
reserveAddress	uint64_t	New reserved address value

## 1.1.375. `temu_ppcSetXerReg`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_ppcSetXerReg(void * Cpu, uint32_t Value)
```

### Description

Set the PowerPC XER register

### Parameters

Table 353. `temu_ppcSetXerReg` Parameters

Parameter	Type	Description
Cpu	void *	Processor pointer
Value	uint32_t	New XER value

## 1.1.376. `temu_printCommandLineHelp`

### Include

```
#include "temu-c/Support/CommandLine.h"
```

### Signature

```
void temu_printCommandLineHelp()
```

### Description

Print command line help to temu stderr stream.

## 1.1.377. temu\_printerr

### Include

```
#include "temu-c/Support/Console.h"
```

### Signature

```
int temu_printerr(const char * Fmt, ...)
```

### Description

fprintf(stderr, ...) wrapper. The function prints a formatted message to what TEMU considers to be the standard error. This function respects internal rerouting rules. Unlike plain fprintf, which writes to an explicit file.

The function is limited in that it keeps a local fixed length buffer for printing. This buffer is currently 1024 bytes, hence it is not possible to print messages longer than 1023 bytes.

### Result

Number of bytes written

## 1.1.378. temu\_printf

### Include

```
#include "temu-c/Support/Console.h"
```

### Signature

```
int temu_printf(const char * Fmt, ...)
```

## Description

Printf wrapper. The function prints a formatted message to what TEMU considers to be the standard output. This function respects internal rerouting rules. Unlike plain printf, which writes only to stdout.

The function is limited in that it keeps a local fixed length buffer for printing. This buffer is 1024 bytes, hence it is not possible to print messages longer than 1023 bytes.

## Result

Number of bytes written

## 1.1.379. temu\_propInfoForClass

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_propInfoForClass(temu_Class * Cls, unsigned int PIIndex, unsigned int
PICount, temu_PropInfo * PI)
```

### Description

Get property info for class

It is possible to extract low level property info data using this function. This can be useful if integrating in another simulator or if integrating the system in e.g. a GUI in which you need to provide object introspection.

The function can be called with the PI array set to NULL to return the number of properties in the class.

You can read out all property fields using the following sequence:

### Result

Number of read PI entries. In case PI is NULL, the total number of PIs for the class.

### Parameters

Table 354. `temu_propInfoForClass` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	The class to inspect

Parameter	Type	Description
PIIndex	unsigned int	The property index to read from.
PICount	unsigned int	Size of PI array in number of entries.
PI	<a href="#">temu_PropInfo</a> *	Pointer to an array of prop info objects to fill in.

### 1.1.380. temu\_propagateRightMostBit32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_propagateRightMostBit32(uint32_t Word)
```

#### Description

Propagate the right most set bit to the bits below it

#### Result

Word with lower zero-bits set

#### Parameters

Table 355. `temu_propagateRightMostBit32` Parameters

Parameter	Type	Description
Word	uint32_t	Word to propagate bits

### 1.1.381. temu\_propagateRightMostBit64

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_propagateRightMostBit64(uint64_t Word)
```

## Description

Propagate the right most set bit to the bits below it

## Result

Word with lower zero-bits set

## Parameters

Table 356. `temu_propagateRightMostBit64` Parameters

Parameter	Type	Description
Word	uint64_t	Word to propagate bits

## 1.1.382. `temu_publishNotification`

### Include

```
#include "temu-c/Support/Notifications.h"
```

### Signature

```
int64_t temu_publishNotification(const char * NotName, temu_Object * Obj)
```

### Description

Publish a notification source A notification source is identified by an event name and an object pointer

### Result

Notification ID of the published event.

### Parameters

Table 357. `temu_publishNotification` Parameters

Parameter	Type	Description
NotName	const char *	Name of the notification
Obj	<code>temu_Object *</code>	Pointer to the object

## 1.1.383. `temu_qualifyAs`

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_qualifyAs(temu_Class * Cls, unsigned int Qualifier)
```

## Description

Bless the class so that the `isQualified` predicate returns 1. Qualifiers with the MSb cleared are reserved for TEMU internals.

## Parameters

Table 358. `temu_qualifyAs` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	The class to be blessed

### 1.1.384. `temu_qualifyAsCpu`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_qualifyAsCpu(temu_Class * Cls)
```

## Description

Bless the class so that the `isCpu` predicate returns 1. There are certain assumptions that CPU classes must meet. These assumptions are not stable yet, so do not use this in user code.

## Parameters

Table 359. `temu_qualifyAsCpu` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	The class to be blessed

### 1.1.385. `temu_qualifyAsMachine`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_qualifyAsMachine(temu_Class * Cls)
```

### Description

Bless the class so that the `isMachine` predicate returns 1. There are certain assumptions that machine classes must meet. These assumptions are not stable yet, so do not use this in user code.

### Parameters

Table 360. `temu_qualifyAsMachine` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	The class to be blessed

## 1.1.386. `temu_qualifyAsMemory`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_qualifyAsMemory(temu_Class * Cls)
```

### Description

Bless the class so that the `isMemory` predicate returns 1

### Parameters

Table 361. `temu_qualifyAsMemory` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	The class to be blessed

## 1.1.387. `temu_readFieldValue`

### Include

```
#include "temu-c/Support/Register.h"
```

## Signature

```
uint64_t temu_readFieldValue(temu_Object * Obj, const char * RegName, unsigned int  
RegIdx, const char * FieldName)
```

## Description

Retrieve the value of a field in a register with side-effects.

## Result

The value of the field

## Parameters

Table 362. `temu_readFieldValue` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object of the class the has the register
RegName	<code>const char *</code>	Register name
RegIdx	<code>unsigned int</code>	Register index
FieldName	<code>const char *</code>	Field name

## 1.1.388. `temu_readPCIeConfigRegister`

### Include

```
#include "temu-c/Support/PCIeHelper.h"
```

### Signature

```
temu_Proval temu_readPCIeConfigRegister(temu_PCIEsspressConfig * DevConf, uint32_t  
offset)
```

### Description

Read register value from PCIe configuration

### Result



return register value.

## Parameters

Table 363. `temu_readPCIeConfigRegister` Parameters

Parameter	Type	Description
DevConf	<code>temu_PCIExpressConfig *</code>	PCIe configuration
offset	<code>uint32_t</code>	register offset in memory map.

### 1.1.389. `temu_readProp`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Propval temu_readProp(temu_Object * Obj, const char * Name, int idx)
```

#### Description

Read property (with side effects)

#### Result

The read value, set to `teTY_Invalid` for failures

#### Parameters

Table 364. `temu_readProp` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
Name	<code>const char *</code>	Name of property
idx	<code>int</code>	Index in property array (set to 0 if scalar)

### 1.1.390. `temu_readValue`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
temu_Propval temu_readValue(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Read a property value with side-effects

## Result

The property value corresponding to the name. In case of the property not being found, the Typ field of the returned property will be teTY\_Invalid.

## Parameters

Table 365. `temu_readValue` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.391. `temu_readValueDouble`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
double temu_readValueDouble(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Read property (with side effects)

## Result

The property value converted to double

## Parameters

Table 366. `temu_readValueDouble` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.392. temu\_readValueI16

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int16_t temu_readValueI16(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Read typed properties.

The readValue[U|I][8|16|32|64] function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the temu\_readValue function instead.

#### Result

The read out property value.

#### Parameters

Table 367. temu\_readValueI16 Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read
Idx	int	Index in property array, set to 0 if the property is not an array

### 1.1.393. temu\_readValueI32

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int32_t temu_readValueI32(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Read typed properties.

The readValue[U|I][8|16|32|64] function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the temu\_readValue function instead.

## Result

The read out property value.

## Parameters

Table 368. temu\_readValueI32 Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read
Idx	int	Index in property array, set to 0 if the property is not an array

## 1.1.394. temu\_readValueI64

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int64_t temu_readValueI64(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Read typed properties.

The readValue[U|I][8|16|32|64] function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the temu\_readValue function instead.

## Result

The read out property value.

## Parameters

Table 369. `temu_readValueI64` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read
Idx	<code>int</code>	Index in property array, set to 0 if the property is not an array

### 1.1.395. `temu_readValueI8`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int8_t temu_readValueI8(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Read typed properties.

The `readValue[U|I][8|16|32|64]` function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the `temu_readValue` function instead.

#### Result

The read out property value.

## Parameters

Table 370. `temu_readValueI8` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read
Idx	<code>int</code>	Index in property array, set to 0 if the property is not an array

### 1.1.396. `temu_readValueSigned`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int64_t temu_readValueSigned(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Read property (with side effects)

## Result

The property value converted to signed

## Parameters

Table 371. `temu_readValueSigned` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.397. `temu_readValueU16`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
uint16_t temu_readValueU16(temu_Object * Obj, const char * PropName, int Idx)
```

## Description

Read typed properties.

The `readValue[U|I][8|16|32|64]` function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the `temu_readValue` function instead.

## Result

The read out property value.

## Parameters

Table 372. `temu_readValueU16` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read
Idx	<code>int</code>	Index in property array, set to 0 if the property is not an array

### 1.1.398. `temu_readValueU32`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
uint32_t temu_readValueU32(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Read typed properties.

The `readValue[U|I][8|16|32|64]` function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the `temu_readValue` function instead.

## Result

The read out property value.

## Parameters

Table 373. `temu_readValueU32` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read
Idx	<code>int</code>	Index in property array, set to 0 if the property is not an array

## 1.1.399. temu\_readValueU64

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
uint64_t temu_readValueU64(temu_Object * Obj, const char * PropName, int Idx)
```

### Description

Read typed properties.

The readValue[U|I][8|16|32|64] function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the temu\_readValue function instead.

### Result

The read out property value.

### Parameters

Table 374. temu\_readValueU64 Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read
Idx	int	Index in property array, set to 0 if the property is not an array

## 1.1.400. temu\_readValueU8

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
uint8_t temu_readValueU8(temu_Object * Obj, const char * PropName, int Idx)
```

### Description



Read typed properties.

The `readValue[U | I][8 | 16 | 32 | 64]` function reads a typed property. The accessor will fail with a fatal error if the read type is not the same as the property type. If the property type is unknown, use the `temu_readValue` function instead.

## Result

The read out property value.

## Parameters

Table 375. `temu_readValueU8` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read
Idx	<code>int</code>	Index in property array, set to 0 if the property is not an array

### 1.1.401. `temu_readValueUnsigned`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
uint64_t temu_readValueUnsigned(temu_Object * Obj, const char * PropName, int Idx)
```

#### Description

Read property (with side effects)

#### Result

The property value converted to unsigned

#### Parameters

Table 376. `temu_readValueUnsigned` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property

Parameter	Type	Description
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.402. temu\_registerClass

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Class * temu_registerClass(const char * ClsName, temu_ObjectCreateFunc Create,
temu_ObjectDisposeFunc Dispose)
```

#### Description

Register a class in the TEMU object system. With these classes, the TEMU object system is responsible for allocation and de-allocation.

#### Result

A pointer to the class object that can be further used to register properties and interfaces.

#### Parameters

Table 377. temu\_registerClass Parameters

Parameter	Type	Description
ClsName	const char *	The name of the class that is to be created.
Create	temu_ObjectCreateFunc	A constructor, the constructor is responsible for allocation and initialisation of the object.
Dispose	temu_ObjectDisposeFunc	A destructor, this function is responsible for deleting the objects of the class.

### 1.1.403. temu\_registerComponent

#### Include

```
#include "temu-c/Support/Component.h"
```

## Signature

```
temu_Class * temu_registerComponent(const char * CompClass, temu_ObjectCreateFunc
Create, temu_ObjectDisposeFunc Dispose)
```

## Description

Register a new component class in

## Result

Pointer to the new class

## Parameters

Table 378. `temu_registerComponent` Parameters

Parameter	Type	Description
CompClass	const char *	Name of the component to be registered
Create	temu_ObjectCreateFunc	The constructor function that allocates the component
Dispose	temu_ObjectDisposeFunc	The destructor function

### 1.1.404. `temu_requireInterface`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_requireInterface(temu_Class * Cls, const char * PropName, const char *
IfaceType)
```

## Description

Associate the interface reference property with an interface type.

Setting the type, prevents the connect command / function to assign the connection if the target interface type is not the same as the property interface reference type.

## Parameters

Table 379. `temu_requireInterface` Parameters

Parameter	Type	Description
Cls	<a href="#">temu_Class</a> *	Class pointer
PropName	const char *	Name of interface reference property
IfaceType	const char *	Type required to connect the property

### 1.1.405. temu\_roundDownNearestPow2\_32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_roundDownNearestPow2_32(uint32_t Word)
```

#### Description

Round down to nearest power of 2

#### Result

A copy of Word rounded down to nearest power of 2

#### Parameters

Table 380. temu\_roundDownNearestPow2\_32 Parameters

Parameter	Type	Description
Word	uint32_t	The word to be round

### 1.1.406. temu\_roundDownNearestPow2\_64

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_roundDownNearestPow2_64(uint64_t Word)
```

## Description

Round up to nearest power of 2

## Result

A copy of Word rounded down to nearest power of 2

## Parameters

Table 381. `temu_roundDownNearestPow2_64` Parameters

Parameter	Type	Description
Word	uint64_t	The word to be round

## 1.1.407. `temu_roundDownToPow2_32`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
uint32_t temu_roundDownToPow2_32(uint32_t Word, uint32_t Size)
```

### Description

Round down to multiples of specified power of 2

### Result

A rounded copy of Word

### Parameters

Table 382. `temu_roundDownToPow2_32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be rounded
Size	uint32_t	Power of 2 sized block

## 1.1.408. `temu_roundDownToPow2_64`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint64_t temu_roundDownToPow2_64(uint64_t Word, uint64_t Size)
```

## Description

Round down to multiples of specified power of 2

## Result

A rounded copy of Word

## Parameters

Table 383. `temu_roundDownToPow2_64` Parameters

Parameter	Type	Description
Word	uint64_t	The word to be rounded
Size	uint64_t	Power of 2 sided block

## 1.1.409. `temu_roundUpNearestPow2_32`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint32_t temu_roundUpNearestPow2_32(uint32_t Word)
```

## Description

Round up to nearest power of 2

## Result

A copy of Word rounded up to nearest power of 2

## Parameters

Table 384. `temu_roundUpNearestPow2_32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be round

## 1.1.410. `temu_roundUpNearestPow2_64`

**Include**

```
#include "temu-c/Support/Bitmanip.h"
```

**Signature**

```
uint64_t temu_roundUpNearestPow2_64(uint64_t Word)
```

**Description**

Round up to nearest power of 2

**Result**

A copy of Word rounded up to nearest power of 2

**Parameters**

Table 385. `temu_roundUpNearestPow2_64` Parameters

Parameter	Type	Description
Word	uint64_t	The word to be round

**1.1.411. temu\_roundUpToPow2\_32****Include**

```
#include "temu-c/Support/Bitmanip.h"
```

**Signature**

```
uint32_t temu_roundUpToPow2_32(uint32_t Word, uint32_t Size)
```

**Description**

Round up to multiples of specified power of 2

**Result**

A rounded copy of Word

**Parameters**

Table 386. `temu_roundUpToPow2_32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be rounded
Size	uint32_t	Power of 2 sized block

### 1.1.412. temu\_roundUpToPow2\_64

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_roundUpToPow2_64(uint64_t Word, uint64_t Size)
```

#### Description

Round up to multiples of specified power of 2

#### Result

A rounded copy of Word

#### Parameters

Table 387. temu\_roundUpToPow2\_64 Parameters

Parameter	Type	Description
Word	uint64_t	The word to be rounded
Size	uint64_t	Power of 2 sized block

### 1.1.413. temu\_run

#### Include

```
#include "temu-c/Support/Scheduler.h"
```

#### Signature

```
void temu_run()
```

#### Description

Run the current scheduler forever



## 1.1.414. temu\_runForSteps

### Include

```
#include "temu-c/Support/Scheduler.h"
```

### Signature

```
void temu_runForSteps(int64_t steps)
```

### Description

Run the current scheduler for a fixed number of steps

### Parameters

Table 388. temu\_runForSteps Parameters

Parameter	Type	Description
steps	int64_t	Time in steps

## 1.1.415. temu\_runForTime

### Include

```
#include "temu-c/Support/Scheduler.h"
```

### Signature

```
void temu_runForTime(double secs)
```

### Description

Run the current scheduler for a fixed time

### Parameters

Table 389. temu\_runForTime Parameters

Parameter	Type	Description
secs	double	Time in simulated seconds

## 1.1.416. temu\_secsToCycles

**Include**

```
#include "temu-c/Support/Time.h"
```

**Signature**

```
int64_t temu_secsToCycles(double Secs, int64_t Freq)
```

**Description**

Convert seconds to cycles

**Result**

Seconds converted to an integral number of cycles

**Parameters**

Table 390. `temu_secsToCycles` Parameters

Parameter	Type	Description
Secs	double	Seconds to convert
Freq	int64_t	Frequency in Hz

**1.1.417. temu\_secsToNanos****Include**

```
#include "temu-c/Support/Time.h"
```

**Signature**

```
int64_t temu_secsToNanos(double Secs)
```

**Description**

Convert seconds to nanoseconds

**Result**

Seconds converted to an integral number of nanoseconds

**Parameters**

Table 391. `temu_secsToNanos` Parameters

Parameter	Type	Description
Secs	double	Seconds to convert

### 1.1.418. temu\_serialiseJSON

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_serialiseJSON(const char * FileName)
```

#### Description

Serialise the simulation state to the given file name.

The serialisation writes a JSON formatted file and calls the serialise procedure in the (optional) ObjectIface if it is implemented.

The serialisation interface can for example be used to write out memory buffers to separate files as such data should not be written to a JSON file.

#### Result

0 on success, otherwise non-zero

#### Parameters

Table 392. temu\_serialiseJSON Parameters

Parameter	Type	Description
FileName	const char *	The filename, to which the JSON data is to be stored

### 1.1.419. temu\_serialiseProp

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_serialiseProp(void * Ctxt, const char * Name, temu_Type Typ, int Count, void
```

\* Data)

## Description

Serialise an explicit property, this can be called from the serialisation interface. Note that the availability of pseudo properties makes this function and the serialise interface more or less obsolete.

## Parameters

Table 393. `temu_serialiseProp` Parameters

Parameter	Type	Description
Ctxt	void *	The context object that has the property (same as passed to serialise function in the interface)
Name	const char *	The name of the property
Typ	temu_Type	The type of the property
Count	int	0 if the property is not an array, otherwise the element number in the property
Data	void *	Serialized data

### 1.1.420. `temu_setFieldValue`

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
int temu_setFieldValue(temu_Object * Obj, const char * RegName, unsigned int RegIdx,
const char * FieldName, uint64_t Value)
```

#### Description

Set a field's value in a register

#### Result

zero on success, otherwise non-zero value

#### Parameters

Table 394. `temu_setFieldValue` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the register
RegName	<code>const char *</code>	Register name
RegIdx	<code>unsigned int</code>	Register index
FieldName	<code>const char *</code>	Field name
Value	<code>uint64_t</code>	The value to be set

### 1.1.421. `temu_setMemAttr`

#### Include

```
#include "temu-c/Memory/Memory.h"
```

#### Signature

```
void temu_setMemAttr(void * Obj, uint64_t Addr, uint64_t Len, temu_MemoryAttr Attr)
```

#### Description

Set attribute on memory space location

#### Parameters

Table 395. `temu_setMemAttr` Parameters

Parameter	Type	Description
Obj	<code>void *</code>	The memory space object
Addr	<code>uint64_t</code>	Physical address where to map the device
Len	<code>uint64_t</code>	Length in bytes of area where the attribute should be set.
Attr	<code>temu_MemoryAttr</code>	The attribute to set.

### 1.1.422. `temu_setRightmostZeroBit64`

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint64_t temu_setRightmostZeoroBit64(uint64_t Word)
```

## Description

Set right most bit that is zero

## Result

A copy of Word, with the right most bit cleared

## Parameters

Table 396. `temu_setRightmostZeoroBit64` Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which the right most set bit will be set

### 1.1.423. `temu_setRightmostZeroBit32`

## Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint32_t temu_setRightmostZeroBit32(uint32_t Word)
```

## Description

Set right most bit that is zero

## Result

A copy of Word, with the right most bit cleared

## Parameters

Table 397. `temu_setRightmostZeroBit32` Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which the right most set bit will be cleared

### 1.1.424. temu\_setTimeSource

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_setTimeSource(temu_Object * Obj, temu_TimeSource * TS)
```

#### Description

Set time source object for models. Typically TS is a CPU. You can only set the time source for an internal class.

#### Parameters

Table 398. temu\_setTimeSource Parameters

Parameter	Type	Description
Obj	temu_Object *	Object to set time source in.
TS	temu_TimeSource *	Time source object (CPU or clock model)

### 1.1.425. temu\_setVTable

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
int temu_setVTable(temu_Class * Cls, void * VTable)
```

#### Description

Set the VTable pointer.

Temu classes, provides the ability to, for internal objects query for a VTable manually in O(1) time. In practice this can be any pointer, but it is typically used to register performance sensitive interfaces. E.g. the CPU and machine classes have their own VTable types that refer to a number of interfaces they implement.

#### Result

0 on success, non-zero otherwise.

## Parameters

Table 399. `temu_setVTable` Parameters

Parameter	Type	Description
Cls	<code>temu_Class *</code>	The class, for which the vtable to be set
VTable	<code>void *</code>	Pointer to the vtable to be used

## 1.1.426. `temu_setValue`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_setValue(temu_Object * Obj, const char * PropName, temu_Propval Val, int Idx)
```

### Description

Set a raw property value without side-effects

### Parameters

Table 400. `temu_setValue` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read.
Val	<code>temu_Propval</code>	The value to set. It must be of the correct type.
Idx	<code>int</code>	Index of property, set to 0 if it is not an array.

## 1.1.427. `temu_setValueDouble`

### Include

```
#include "temu-c/Support/Objsys.h"
```



## Signature

```
void temu_setValueDouble(temu_Object * Obj, const char * PropName, double Val, int Idx)
```

## Description

Set floating point property (without side effects)

## Parameters

Table 401. `temu_setValueDouble` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property
Val	<code>double</code>	The new property value
Idx	<code>int</code>	Index in property array (set to 0 if scalar)

### 1.1.428. `temu_setValueI16`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_setValueI16(temu_Object * Obj, const char * PropName, int16_t Val, int Idx)
```

## Description

Set a raw property value without side-effects

## Parameters

Table 402. `temu_setValueI16` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read.
Val	<code>int16_t</code>	The value to set. It must be of the correct type.

Parameter	Type	Description
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.429. temu\_setValueI32

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_setValueI32(temu_Object * Obj, const char * PropName, int32_t Val, int Idx)
```

#### Description

Set a raw property value without side-effects

#### Parameters

Table 403. temu\_setValueI32 Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
Val	int32_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.430. temu\_setValueI64

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_setValueI64(temu_Object * Obj, const char * PropName, int64_t Val, int Idx)
```

#### Description

Set a raw property value without side-effects

## Parameters

Table 404. `temu_setValueI64` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read.
Val	<code>int64_t</code>	The value to set. It must be of the correct type.
Idx	<code>int</code>	Index of property, set to 0 if it is not an array.

### 1.1.431. `temu_setValueI8`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_setValueI8(temu_Object * Obj, const char * PropName, int8_t Val, int Idx)
```

#### Description

Set a raw property value without side-effects

## Parameters

Table 405. `temu_setValueI8` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read.
Val	<code>int8_t</code>	The value to set. It must be of the correct type.
Idx	<code>int</code>	Index of property, set to 0 if it is not an array.

### 1.1.432. `temu_setValueSigned`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_setValueSigned(temu_Object * Obj, const char * PropName, int64_t Val, int Idx)
```

## Description

Set signed property (without side effects)

## Parameters

Table 406. `temu_setValueSigned` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property
Val	<code>int64_t</code>	The new property value
Idx	<code>int</code>	Index in property array (set to 0 if scalar)

### 1.1.433. `temu_setValueU16`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_setValueU16(temu_Object * Obj, const char * PropName, uint16_t Val, int Idx)
```

## Description

Set a raw property value without side-effects

## Parameters

Table 407. `temu_setValueU16` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer

Parameter	Type	Description
PropName	const char *	Name of property to read.
Val	uint16_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.434. temu\_setValueU32

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_setValueU32(temu_Object * Obj, const char * PropName, uint32_t Val, int Idx)
```

#### Description

Set a raw property value without side-effects

#### Parameters

Table 408. temu\_setValueU32 Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
Val	uint32_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.435. temu\_setValueU64

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_setValueU64(temu_Object * Obj, const char * PropName, uint64_t Val, int Idx)
```

## Description

Set a raw property value without side-effects

## Parameters

Table 409. `temu_setValueU64` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	uint64_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

## 1.1.436. temu\_setValueU8

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_setValueU8(temu_Object * Obj, const char * PropName, uint8_t Val, int Idx)
```

## Description

Set a raw property value without side-effects

## Parameters

Table 410. `temu_setValueU8` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	uint8_t	The value to set. It must be of the correct type.

Parameter	Type	Description
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.437. temu\_setValueUnsigned

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_setValueUnsigned(temu_Object * Obj, const char * PropName, uint64_t Val, int Idx)
```

#### Description

Set unsigned property (without side effects)

#### Parameters

Table 411. temu\_setValueUnsigned Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property
Val	uint64_t	The new property value
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.438. temu\_signed32AddOverflows

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_signed32AddOverflows(uint32_t a, uint32_t b, uint32_t carry)
```

#### Description

Checks whether the addition of signed inputs with carry would overflow;  $a+b+carry$

### Result

true if it overflows, false otherwise

### Parameters

Table 412. `temu_signed32AddOverflows` Parameters

Parameter	Type	Description
a	uint32_t	is the first input
b	uint32_t	is the second input
carry	uint32_t	is the carry; can be either zero or one

## 1.1.439. `temu_signedPropval`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_Propval temu_signedPropval(temu_Type T, int64_t I)
```

### Description

Create a signed propval of the given type

### Result

Property value with the given type and value.

### Parameters

Table 413. `temu_signedPropval` Parameters

Parameter	Type	Description
T	temu_Type	Type ID, must be a signed integer type tag
I	int64_t	Integer value

## 1.1.440. `temu_snapshotGetLength`



## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
int temu_snapshotGetLength(void * Ctxt, const char * Name)
```

## Description

Get number of entries for the serialized property

## Result

Length of the property in elements. Negative on error.

## Parameters

Table 414. `temu_snapshotGetLength` Parameters

Parameter	Type	Description
Ctxt	void *	Context passed to deserialise function
Name	const char *	Name of property / key in the serialized file

## 1.1.441. temu\_snapshotGetValue

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_Propval temu_snapshotGetValue(void * Ctxt, const char * Name, int Idx)
```

### Description

Get value for named snapshot value

### Result

Property value with the contents saved to the snapshot

## Parameters

Table 415. `temu_snapshotGetValue` Parameters

Parameter	Type	Description
Ctxt	void *	Context is passed to deserialise interface function
Name	const char *	Name of the saved value
Idx	int	Index of the saved value (if array)

### 1.1.442. `temu_sparcGetAsr`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_sparcGetAsr(void * Cpu, unsigned int Reg)
```

#### Description

Get ASR register value

#### Result

ASR register value

#### Parameters

Table 416. `temu_sparcGetAsr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Reg	unsigned int	ASR register ID

### 1.1.443. `temu_sparcGetNPc`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint32_t temu_sparcGetNPc(void * Cpu)
```

### Description

Get the nPC value

### Result

nPC value

### Parameters

Table 417. `temu_sparcGetNPc` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

## 1.1.444. `temu_sparcGetPsr`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_sparcGetPsr(void * Cpu)
```

### Description

Get the Processor State Register

### Result

PSR value

### Parameters

Table 418. `temu_sparcGetPsr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

## 1.1.445. `temu_sparcGetTbr`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_sparcGetTbr(void * Cpu)
```

### Description

Get Trap Base Register value

### Result

TBR value

### Parameters

Table 419. `temu_sparcGetTbr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

## 1.1.446. `temu_sparcGetWim`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_sparcGetWim(void * Cpu)
```

### Description

Get window invalidation mask register

### Result

WIM value

### Parameters

Table 420. `temu_sparcGetWim` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

## 1.1.447. temu\_sparcGetWindowCount

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
int temu_sparcGetWindowCount(void * Cpu)
```

### Description

Get how many register windows are supported by the CPU

### Result

Number of register windows

### Parameters

Table 421. temu\_sparcGetWindowCount Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

## 1.1.448. temu\_sparcGetWindowedReg

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
uint32_t temu_sparcGetWindowedReg(void * Cpu, int Window, unsigned int Reg)
```

### Description

Get a windowed SPARC register

### Result

Register value

### Parameters

Table 422. `temu_sparcGetWindowedReg` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Window	int	Window number (-1 for current window)
Reg	unsigned int	Register number within window

### 1.1.449. `temu_sparcGetY`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
uint64_t temu_sparcGetY(void * Cpu)
```

#### Description

Get the Y register of the CPU,

#### Result

Current Y register value

#### Parameters

Table 423. `temu_sparcGetY` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

### 1.1.450. `temu_sparcSetAsr`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_sparcSetAsr(void * Cpu, unsigned int Reg, uint64_t Value)
```

## Description

Set the ASR register

## Parameters

Table 424. `temu_sparcSetAsr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Reg	unsigned int	ASR register ID
Value	uint64_t	Value to write

### 1.1.451. `temu_sparcSetAsrReader`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_sparcSetAsrReader(void * Cpu, unsigned int Asr, temu_SparcAsrHandler  
Handler)
```

#### Description

Install the handler to be called when an ASR is read

#### Parameters

Table 425. `temu_sparcSetAsrReader` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Asr	unsigned int	ASR register ID
Handler	temu_SparcAsrHandler	Function to call.

### 1.1.452. `temu_sparcSetAsrWriter`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_sparcSetAsrWriter(void * Cpu, unsigned int Asr, temu_SparcAsrHandler
Handler)
```

## Description

Install the handler to be called when an ASR is written

## Parameters

Table 426. `temu_sparcSetAsrWriter` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Asr	unsigned int	ASR register ID
Handler	temu_SparcAsrHandler	Function to call.

### 1.1.453. `temu_sparcSetNPc`

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_sparcSetNPc(void * Cpu, uint32_t Value)
```

## Description

Set the nPC value (next program counter)



When you use the `setPC` function, the NPC is also updated to `PC+4`. Use this to explicitly set NPC.

## Parameters

Table 427. `temu_sparcSetNPc` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint32_t	Value to write to NPC

### 1.1.454. `temu_sparcSetPsr`

#### Include



```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_sparcSetPsr(void * Cpu, uint32_t Value)
```

### Description

Set the Processor State Register

### Parameters

Table 428. `temu_sparcSetPsr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint32_t	Value to write

## 1.1.455. `temu_sparcSetTbr`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_sparcSetTbr(void * Cpu, uint32_t Value)
```

### Description

Set the Trap Base Register

### Parameters

Table 429. `temu_sparcSetTbr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer,
Value	uint32_t	Value for TBR register

## 1.1.456. `temu_sparcSetWim`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_sparcSetWim(void * Cpu, uint32_t Value)
```

### Description

Set window invalidation mask register

### Parameters

Table 430. `temu_sparcSetWim` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint32_t	value to set in WIM

## 1.1.457. `temu_sparcSetWindowedReg`

### Include

```
#include "temu-c/Support/Cpu.h"
```

### Signature

```
void temu_sparcSetWindowedReg(void * Cpu, int Window, unsigned int Reg, uint32_t Value)
```

### Description

Set a windowed SPARC register

### Parameters

Table 431. `temu_sparcSetWindowedReg` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Window	int	Window number (-1 for current window)
Reg	unsigned int	Register number within window

Parameter	Type	Description
Value	uint32_t	Register value

### 1.1.458. temu\_sparcSetY

#### Include

```
#include "temu-c/Support/Cpu.h"
```

#### Signature

```
void temu_sparcSetY(void * Cpu, uint64_t Value)
```

#### Description

Set the Y register of the CPU,

#### Parameters

Table 432. temu\_sparcSetY Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint64_t	Value to write to the Y register

### 1.1.459. temu\_spwConnect

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

#### Signature

```
void temu_spwConnect(temu_SpwPortIfaceRef Dev1PortIface, temu_SpwPortIfaceRef  
Dev2PortIface)
```

#### Description

Connect two SpaceWire ports.

### 1.1.460. temu\_spwDisconnect

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Signature

```
void temu_spwDisconnect(temu_SpwPortIfaceRef Dev1PortIface, temu_SpwPortIfaceRef  
Dev2PortIface)
```

### Description

Disconnect two SpaceWire ports.

## 1.1.461. temu\_spwLinkStateToStr

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Signature

```
const char * temu_spwLinkStateToStr(uint8_t linkState)
```

### Description

Returns the string name for the link state.

### Result

## 1.1.462. temu\_spwLsmInit

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Signature

```
void temu_spwLsmInit(temu_SpwLinkState * StatePtr)
```

### Description

Initialize the link state.

## 1.1.463. temu\_spwLsmUpdate

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Signature

```
uint8_t temu_spwLsmUpdate(temu_SpwLinkState * StatePtr, uint8_t AS, uint8_t LS,  
uint8_t LD, uint8_t PortConnect, temu_SpwLinkState otherSideLinkState)
```

## Description

Updates the link state.

## Result

### 1.1.464. temu\_spwRmapCRC

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Signature

```
uint8_t temu_spwRmapCRC(const uint8_t * Data, uint32_t DataSize)
```

## Description

Calculates the CRC over the specified data.

## Result

### 1.1.465. temu\_spwRmapCRCNextCode

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Signature

```
uint8_t temu_spwRmapCRCNextCode(uint8_t InCRC, uint8_t InByte)
```

## Description

Provided the previous calculated crc and a the current byte returns the next CRC value.

## Result

### 1.1.466. temu\_spwRmapDecodeBuffer

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

#### Signature

```
temu_SpwRmapDecodingOutcome temu_spwRmapDecodeBuffer(const temu_Buff * PktDataBuffer,  
temu_SpwRmapDecodedPacket * PktDecoded)
```

#### Description

Provided a buffer containing a SpaceWire Rmap packet attempts to decode it.

## Result

### 1.1.467. temu\_spwRmapDecodePacket

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

#### Signature

```
temu_SpwRmapDecodingOutcome temu_spwRmapDecodePacket(const temu_SpwPacket * Pkt,  
temu_SpwRmapDecodedPacket * PktDecoded)
```

#### Description

Provided a SpaceWire Rmap packet attempts to decode it.

## Result

### 1.1.468. temu\_spwRmapEncodeReadReplyHeaderForPacket

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Signature

```
uint32_t temu_spwRmapEncodeReadReplyHeaderForPacket(const temu_SpwRmapDecodedPacket *  
DCmdPkt, uint8_t * Data, uint32_t AllocatedDataSize, uint8_t Status, uint32_t  
DataLength)
```

## Description

Encodes the reply for a read command.

## Result

### 1.1.469. temu\_spwRmapEncodeRmwHeaderForPacket

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Signature

```
uint32_t temu_spwRmapEncodeRmwHeaderForPacket(const temu_SpwRmapDecodedPacket *  
DCmdPkt, uint8_t * Data, uint32_t AllocatedDataSize, uint8_t Status, uint32_t  
DataLength)
```

## Description

Encodes the reply for a rmw command.

## Result

### 1.1.470. temu\_spwRmapEncodeWriteReplyHeaderForPacket

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Signature

```
uint32_t temu_spwRmapEncodeWriteReplyHeaderForPacket(const temu_SpwRmapDecodedPacket *  
DCmdPkt, uint8_t * Data, uint32_t AllocatedDataSize, uint8_t Status)
```

## Description

Encodes the reply for a write command.

## Result

### 1.1.471. temu\_spwRmapHeaderReplySize

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

#### Signature

```
uint32_t temu_spwRmapHeaderReplySize(const temu_SpwRmapDecodedPacket * DCmdPkt)
```

#### Description

Returns the total packet-size required to reply to the command.

## Result

### 1.1.472. temu\_step

#### Include

```
#include "temu-c/Support/Scheduler.h"
```

#### Signature

```
void temu_step(temu_TimeSource * cpu, int steps)
```

#### Description

Step a processor in the current scheduler

Note that, the other processors may run if the step exceeds the quanta. The step function currently runs processors in sequence, it is not running processors in parallel. This means that stepping is always deterministic.

#### Parameters

Table 433. temu\_step Parameters

Parameter	Type	Description
cpu	temu_TimeSource *	The processor to step
steps	int	The number of steps or instructions to execute



### 1.1.473. temu\_stepsToCycles

#### Include

```
#include "temu-c/Support/Time.h"
```

#### Signature

```
uint64_t temu_stepsToCycles(temu_TimeSource * TS, uint64_t Steps)
```

#### Description

Convert steps to cycles

#### Result

Steps converted to cycles given the time source IPC/CPI configuration.

#### Parameters

Table 434. temu\_stepsToCycles Parameters

Parameter	Type	Description
TS	temu_TimeSource *	Time source object
Steps	uint64_t	Step count to convert to cycles

### 1.1.474. temu\_stepsToCyclesRoundedUp

#### Include

```
#include "temu-c/Support/Time.h"
```

#### Signature

```
uint64_t temu_stepsToCyclesRoundedUp(temu_TimeSource * TS, uint64_t Steps)
```

#### Description

Convert steps to cycles, rounding upwards

#### Result

Steps converted to cycles given the time source IPC/CPI configuration.

## Parameters

Table 435. `temu_stepsToCyclesRoundedUp` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	Time source object
Steps	<code>uint64_t</code>	Step count to convert to cycles

### 1.1.475. `temu_stop`

#### Include

```
#include "temu-c/Support/Scheduler.h"
```

#### Signature

```
void temu_stop()
```

#### Description

Stop the current scheduler

The function signals the scheduler to stop, and then waits until the scheduler is in the stopped state.

### 1.1.476. `temu_subscribeNotification`

#### Include

```
#include "temu-c/Support/Notifications.h"
```

#### Signature

```
int temu_subscribeNotification(const char * NotName, temu_Object * Source, void * Arg,
temu_NotificationHandler NotFunc)
```

#### Description

Install notification functions for the given event generated by source

#### Result

#### Parameters

Table 436. `temu_subscribeNotification` Parameters

Parameter	Type	Description
NotName	const char *	Name of the notification
Source	<a href="#">temu_Object</a> *	If source is NULL, the event subscriber will be notified by all the sources for the given name.
Arg	void *	Context argument to be passed to the callback notification function
NotFunc	temu_NotificationHandler	The callback function on notification

### 1.1.477. temu\_swap16

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint16_t temu_swap16(uint16_t HWord)
```

#### Description

Swap bytes in 16 bit word

#### Result

the swapped word

#### Parameters

Table 437. temu\_swap16 Parameters

Parameter	Type	Description
HWord	uint16_t	The word to be swapped

### 1.1.478. temu\_swap32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint32_t temu_swap32(uint32_t Word)
```

## Description

Swap bytes in 32 bit word

## Result

the swapped word

## Parameters

Table 438. `temu_swap32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

## 1.1.479. `temu_swap32Half`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint32_t temu_swap32Half(uint32_t Word)
```

## Description

Swap half words inside 32 bit word

## Result

the swapped word

## Parameters

Table 439. `temu_swap32Half` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

## 1.1.480. `temu_swap64`

## Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint64_t temu_swap64(uint64_t DWord)
```

## Description

Swap bytes in 64 bit word

## Result

the swapped word

## Parameters

Table 440. `temu_swap64` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

### 1.1.481. `temu_swap64Word`

## Include

```
#include "temu-c/Support/Bitmanip.h"
```

## Signature

```
uint64_t temu_swap64Word(uint64_t DWord)
```

## Description

Swap words inside 64 bit word

## Result

the swapped word

## Parameters

Table 441. `temu_swap64Word` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

### 1.1.482. temu\_swapBigHost16

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint16_t temu_swapBigHost16(uint16_t HWord)
```

#### Description

Swap between big endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

#### Result

the swapped word

#### Parameters

Table 442. temu\_swapBigHost16 Parameters

Parameter	Type	Description
HWord	uint16_t	The word to be swapped

### 1.1.483. temu\_swapBigHost32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_swapBigHost32(uint32_t Word)
```

## Description

Swap between big endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

## Result

the swapped word

## Parameters

Table 443. `temu_swapBigHost32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

### 1.1.484. `temu_swapBigHost32Half`

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_swapBigHost32Half(uint32_t Word)
```

## Description

Swap between big endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

## Result

the swapped word

## Parameters

Table 444. `temu_swapBigHost32Half` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

### 1.1.485. temu\_swapBigHost64

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_swapBigHost64(uint64_t DWord)
```

#### Description

Swap between big endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

#### Result

the swapped word

#### Parameters

Table 445. temu\_swapBigHost64 Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

### 1.1.486. temu\_swapBigHost64Word

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_swapBigHost64Word(uint64_t DWord)
```



## Description

Swap between big endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

## Result

the swapped word

## Parameters

Table 446. `temu_swapBigHost64Word` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

## 1.1.487. `temu_swapLittleHost16`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
uint16_t temu_swapLittleHost16(uint16_t HWord)
```

## Description

Swap between little endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

## Result

the swapped word

## Parameters

Table 447. `temu_swapLittleHost16` Parameters

Parameter	Type	Description
HWord	uint16_t	The word to be swapped

### 1.1.488. temu\_swapLittleHost32

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_swapLittleHost32(uint32_t Word)
```

#### Description

Swap between little endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

#### Result

the swapped word

#### Parameters

Table 448. temu\_swapLittleHost32 Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

### 1.1.489. temu\_swapLittleHost32Half

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint32_t temu_swapLittleHost32Half(uint32_t Word)
```

## Description

Swap between little endian to host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

## Result

the swapped word

## Parameters

Table 449. `temu_swapLittleHost32Half` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

## 1.1.490. `temu_swapLittleHost64`

### Include

```
#include "temu-c/Support/Bitmanip.h"
```

### Signature

```
uint64_t temu_swapLittleHost64(uint64_t DWord)
```

## Description

Swap between little endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

## Result

the swapped word

## Parameters

Table 450. `temu_swapLittleHost64` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

### 1.1.491. temu\_swapLittleHost64Word

#### Include

```
#include "temu-c/Support/Bitmanip.h"
```

#### Signature

```
uint64_t temu_swapLittleHost64Word(uint64_t DWord)
```

#### Description

Swap between little endian and host endian. Note that at present we only have little endian hosts, so these functions simply wrap. However, they declare intent and it is recommended that you use these for swapping when you want to go between little

>host and big

>host.

#### Result

the swapped word

#### Parameters

Table 451. temu\_swapLittleHost64Word Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

### 1.1.492. temu\_syntabGetFuncName

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
void temu_syntabGetFuncName(temu_Syntab * Sym, const char ** LocalFile, const char **  

  Symbol, uint64_t Addr)
```

## Description

Get the function name associated with the given address

The address can be any address within the range of the function, i.e. from the start to othe end of the function. Thus this serves as a way to find the name of the current function given a PC value.

The function first searches for local symbols (e.g. static symbols in C). It then searches for global symbols and last for weak symbols.

If a local symbol is found, the LocalFile pointer will point at a string with the file name it is associated to, otherwise the local file pointer will be set to null.

The returned pointers are valid until the symbol table is disposed.

## Parameters

Table 452. `temu_syntabGetFuncName` Parameters

Parameter	Type	Description
Sym	<a href="#">temu_Syntab</a> *	Symboltable
LocalFile	const char **	The pointer it refers to will be set to NULL if the symbol found is global.
Symbol	const char **	The pointer will be set to NULL if a function cannot be found at the address.
Addr	uint64_t	Virtual address to find a function name for.

### 1.1.493. `temu_syntabGetGlobalFuncRange`

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_syntabGetGlobalFuncRange(temu_Syntab * Sym, const char * FuncName, uint64_t * Addr, uint64_t * Size)
```

#### Description

Get the range of a global function

On success, the Addr and the Size references will be set to the starting virtual address of the

function and the size in bytes respectively.

## Result

0 on success (the function exists), other values indicate failures.

## Parameters

Table 453. `temu_syntabGetGlobalFuncRange` Parameters

Parameter	Type	Description
Sym	<code>temu_Syntab *</code>	The Symbol table, from which the function is to be retrieved
FuncName	<code>const char *</code>	The function, whose range is required
Addr	<code>uint64_t *</code>	On success, gives the address of the function
Size	<code>uint64_t *</code>	On success, gives the size of the function start from Addr

## 1.1.494. `temu_syntabGetGlobalObjRange`

### Include

```
#include "temu-c/Support/Loader.h"
```

### Signature

```
int temu_syntabGetGlobalObjRange(temu_Syntab * Sym, const char * ObjectName, uint64_t * Addr, uint64_t * Size)
```

### Description

Get the range of a global object

On success, the Addr and the Size references will be set to the starting virtual address of the object and the size in bytes respectively.

### Result

0 on success (the object exists), other values indicate failures

### Parameters

Table 454. `temu_syntabGetGlobalObjRange` Parameters

Parameter	Type	Description
Sym	<a href="#">temu_Symtab</a> *	The Symbol table, from which the object to be retrieved
ObjectName	const char *	Name of the object, whose address range is to be obtained
Addr	uint64_t *	On success, gives the address of the object
Size	uint64_t *	On success, gives the size of the object start from Addr

### 1.1.495. `temu_syntabGetLocalFuncRange`

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_syntabGetLocalFuncRange(temu_Symtab * Sym, const char * FileName, const char * FuncName, uint64_t * Addr, uint64_t * Size)
```

#### Description

Get the range of a local/static function

On success, the Addr and the Size references will be set to the starting virtual address of the function and the size in bytes respectively.

#### Result

0 on success (the function exists in the given file), other values indicate failures

#### Parameters

Table 455. `temu_syntabGetLocalFuncRange` Parameters

Parameter	Type	Description
Sym	<a href="#">temu_Symtab</a> *	The Symbol table, from which the function to be retrieved
FileName	const char *	The file/compilation unit that contains the function
FuncName	const char *	The function, whose range is required

Parameter	Type	Description
Addr	uint64_t *	On success, gives the address of the function
Size	uint64_t *	On success, gives the size of the function start from Addr

### 1.1.496. temu\_syntabGetLocalObjRange

#### Include

```
#include "temu-c/Support/Loader.h"
```

#### Signature

```
int temu_syntabGetLocalObjRange(temu_Syntab * Sym, const char * FileName, const char *
ObjectName, uint64_t * Addr, uint64_t * Size)
```

#### Description

Get the range of a local/static object

On success, the Addr and the Size references will be set to the starting virtual address of the object and the size in bytes respectively.

#### Result

0 on success (the object exists), other values indicate failures

#### Parameters

Table 456. temu\_syntabGetLocalObjRange Parameters

Parameter	Type	Description
Sym	temu_Syntab *	The Symbol table, from which the object to be retrieved
FileName	const char *	The file/compilation unit that contains the function
ObjectName	const char *	The object, whose range is required
Addr	uint64_t *	On success, gives the address of the object
Size	uint64_t *	On success, gives the size of the object start from Addr



## 1.1.497. temu\_syntabGetObjName

### Include

```
#include "temu-c/Support/Loader.h"
```

### Signature

```
void temu_syntabGetObjName(temu_Syntab * Sym, const char ** LocalFile, const char **  
Symbol, uint64_t Addr)
```

### Description

Get the object name associated with the given address

The address can be any address within the range of the object. Thus this serves as a way to find the name of a global or static object given an address.

The function first searches for local symbols (e.g. static symbols in C). It then searches for global symbols and last for weak symbols.

If a local symbol is found, the LocalFile pointer will point at a string with the file name it is associated to, otherwise the local file pointer will be set to NULL.

The returned pointers are valid until the symbol table is disposed.

### Parameters

Table 457. temu\_syntabGetObjName Parameters

Parameter	Type	Description
Sym	<a href="#">temu_Syntab</a> *	Symboltable
LocalFile	const char **	The pointer it refers to will be set to NULL if the symbol found is global.
Symbol	const char **	The pointer will be set to NULL if an object cannot be found at the address.
Addr	uint64_t	Virtual address to find an object name for.

## 1.1.498. temu\_timeGetCurrentSrtNanos

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
uint64_t temu_timeGetCurrentSrtNanos(temu_TimeSource * TS)
```

### Description

Get the current simulated real-time in nanoseconds

### Result

simulated time for the given object

### Parameters

Table 458. `temu_timeGetCurrentSrtNanos` Parameters

Parameter	Type	Description
TS	<code>temu_TimeSource *</code>	The object in question

## 1.1.499. `temu_timeGetMonotonicWct`

### Include

```
#include "temu-c/Support/Time.h"
```

### Signature

```
uint64_t temu_timeGetMonotonicWct()
```

### Description

Get monotonic time in nanoseconds.

The monotonic time is relative since some epoch of undefined start, but it is monotonic, unaffected by adjustments due to leap seconds, setting of the system clock, etc.

This function is primarily useful for doing performance measurements since the time returned is relative to an undefined point (although that undefined point will be consistent while the program is running).

In practice, on systems with `clock_gettime()` implemented, this function returns the timespec converted to nanoseconds, but on systems without `clock_gettime()`, e.g. Darwin and Windows, the

function will only ensure that some notion of monotonic nanoseconds are returned. In Darwin for example, this results in a monotonic time returned which is relative to the first call to the function.

## Result

Wall clock nanoseconds since an unspecified epoch.

### 1.1.500. temu\_timeGetThreadWct

#### Include

```
#include "temu-c/Support/Time.h"
```

#### Signature

```
uint64_t temu_timeGetThreadWct()
```

#### Description

Returns wall clock nanoseconds of thread time



This is how much time the thread this is called from has been scheduled.

## Result

wall clock nanoseconds of thread time

### 1.1.501. temu\_typeToName

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
const char * temu_typeToName(temu_Type Typ)
```

#### Description

Get a string representing the type tag.

## Result

The name as a C-string

## Parameters

Table 459. `temu_typeToName` Parameters

Parameter	Type	Description
Typ	temu_Type	The type, whose name is required

### 1.1.502. `temu_typenameForInterface`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
const char * temu_typenameForInterface(const temu_Object * Obj, const void * Iface)
```

#### Description

Get type name for interface

#### Result

The name of the type as C-string

#### Parameters

Table 460. `temu_typenameForInterface` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object that contains the interface
Iface	const void *	Pointer to the interface

### 1.1.503. `temu_unsignedPropval`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
temu_Propval temu_unsignedPropval(temu_Type T, uint64_t U)
```

## Description

Create an unsigned propval of the given type

## Result

Property value with the given type and value.

## Parameters

Table 461. `temu_unsignedPropval` Parameters

Parameter	Type	Description
T	<code>temu_Type</code>	Type ID, must be an unsigned integer type tag
U	<code>uint64_t</code>	Integer value

## 1.1.504. `temu_unsubscribeNotification`

### Include

```
#include "temu-c/Support/Notifications.h"
```

### Signature

```
int temu_unsubscribeNotification(const char * NotName, temu_Object * Source,  
temu_NotificationHandler NotFunc)
```

### Description

Remove notification handler for the given name and source. Note that this function runs in O(N) time. It is not meant for being used in performance critical code.

### Result

### Parameters

Table 462. `temu_unsubscribeNotification` Parameters

Parameter	Type	Description
NotName	<code>const char *</code>	Name of the notification
Source	<code>temu_Object *</code>	If source is NULL, the event subscriber will be notified by all the sources for the given name.

Parameter	Type	Description
NotFunc	temu_NotificationHandler	The callback function to be called on notification

### 1.1.505. temu\_unsubscribeNotificationArg

#### Include

```
#include "temu-c/Support/Notifications.h"
```

#### Signature

```
int temu_unsubscribeNotificationArg(const char * NotName, temu_Object * Source,  
temu_NotificationHandler NotFunc, void * Arg)
```

#### Description

Remove notification handler for the given name, source and arg. Note that this function runs in O(N) time. It is not meant for being used in performance critical code.

#### Result

zero on success, otherwise non-zero

#### Parameters

Table 463. temu\_unsubscribeNotificationArg Parameters

Parameter	Type	Description
NotName	const char *	Name of the notification
Source	temu_Object *	If source is NULL, the event subscriber will be notified by all the sources for the given name.
NotFunc	temu_NotificationHandler	The callback function to be called on notification
Arg	void *	Context argument to be passed to the callback notification function

### 1.1.506. temu\_vecCreate

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
temu_Vector temu_vecCreate(temu_Type Typ)
```

### Description

Create a vector object

### Result

the vector object created

### Parameters

Table 464. `temu_vecCreate` Parameters

Parameter	Type	Description
Typ	temu_Type	The type of the elements

## 1.1.507. temu\_vecDispose

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_vecDispose(temu_Vector * Vec)
```

### Description

Delete a vector object

### Parameters

Table 465. `temu_vecDispose` Parameters

Parameter	Type	Description
Vec	temu_Vector *	The vector object to be deleted

## 1.1.508. temu\_vecGetData

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void * temu_vecGetData(temu_Vector * Vec)
```

## Description

Retrieve the array of elements of a vector

## Result

Returns a pointer to the first element of the vector

## Parameters

Table 466. *temu\_vecGetData Parameters*

Parameter	Type	Description
Vec	<a href="#">temu_Vector *</a>	The vector, whose elements are requested

## 1.1.509. temu\_vecGetSize

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
size_t temu_vecGetSize(temu_Vector * Vec)
```

### Description

Get the number of elements of a vector

### Result

The number of elements in Vec

### Parameters

Table 467. *temu\_vecGetSize Parameters*



Parameter	Type	Description
Vec	<code>temu_Vector *</code>	The vector, whose size to be retrieved

### 1.1.510. temu\_vecPush

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_vecPush(temu_Vector * Vec, temu_Propval Val)
```

#### Description

Add an element to a vector object

#### Parameters

Table 468. temu\_vecPush Parameters

Parameter	Type	Description
Vec	<code>temu_Vector *</code>	The vector, to which the element to be added
Val	<code>temu_Propval</code>	The element to be added

### 1.1.511. temu\_writeFieldValue

#### Include

```
#include "temu-c/Support/Register.h"
```

#### Signature

```
int temu_writeFieldValue(temu_Object * Obj, const char * RegName, unsigned int RegIdx,
const char * FieldName, uint64_t Value)
```

#### Description

Set a field's value in a register with side-effects

#### Result

zero on success, otherwise non-zero value

## Parameters

Table 469. `temu_writeFieldValue` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	The object that contains the register
RegName	<code>const char *</code>	Register name
RegIdx	<code>unsigned int</code>	Register index
FieldName	<code>const char *</code>	Field name
Value	<code>uint64_t</code>	The value to be set

### 1.1.512. `temu_writePCIeConfigRegister`

#### Include

```
#include "temu-c/Support/PCIeHelper.h"
```

#### Signature

```
void temu_writePCIeConfigRegister(temu_PCIEExpressConfig * devConf, uint32_t offset,
uint32_t value)
```

#### Description

Write value to PCIe config to the register with given offset.

#### Parameters

Table 470. `temu_writePCIeConfigRegister` Parameters

Parameter	Type	Description
devConf	<code>temu_PCIEExpressConfig *</code>	PCIe configuration
offset	<code>uint32_t</code>	register offset in memory map.
value	<code>uint32_t</code>	new value for the register

### 1.1.513. `temu_writeProp`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
int temu_writeProp(temu_Object * Obj, const char * Name, int idx, temu_Propval * PV)
```

### Description

Write property (with side effects)

### Result

0 if the property was written

### Parameters

Table 471. `temu_writeProp` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
Name	const char *	Name of property
idx	int	Index in property array (set to 0 if scalar)
PV	<a href="#">temu_Propval *</a>	Pointer to propval

## 1.1.514. `temu_writeValue`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_writeValue(temu_Object * Obj, const char * PropName, temu_Propval Val, int Idx)
```

### Description

Write a property value with side-effects

### Parameters

Table 472. `temu_writeValue` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	temu_Propval	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.515. temu\_writeValueDouble

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_writeValueDouble(temu_Object * Obj, const char * PropName, double Val, int Idx)
```

#### Description

Write floating point property (with side effects)

#### Parameters

Table 473. `temu_writeValueDouble` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property
Val	double	The new property value
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.516. temu\_writeValueI16

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_writeValueI16(temu_Object * Obj, const char * PropName, int16_t Val, int
Idx)
```

## Description

Write a property value with side-effects

## Parameters

Table 474. `temu_writeValueI16` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	int16_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

## 1.1.517. `temu_writeValueI32`

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Signature

```
void temu_writeValueI32(temu_Object * Obj, const char * PropName, int32_t Val, int
Idx)
```

## Description

Write a property value with side-effects

## Parameters

Table 475. `temu_writeValueI32` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	int32_t	The value to set. It must be of the correct type.

Parameter	Type	Description
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.518. temu\_writeValueI64

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_writeValueI64(temu_Object * Obj, const char * PropName, int64_t Val, int Idx)
```

#### Description

Write a property value with side-effects

#### Parameters

Table 476. temu\_writeValueI64 Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
Val	int64_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.519. temu\_writeValueI8

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_writeValueI8(temu_Object * Obj, const char * PropName, int8_t Val, int Idx)
```

#### Description

Write a property value with side-effects

## Parameters

Table 477. `temu_writeValueI8` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read.
Val	<code>int8_t</code>	The value to set. It must be of the correct type.
Idx	<code>int</code>	Index of property, set to 0 if it is not an array.

### 1.1.520. `temu_writeValueObj`

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_writeValueObj(temu_Object * Obj, const char * PropName, temu_Object * Val, int Idx)
```

#### Description

Write a property value with side-effects

## Parameters

Table 478. `temu_writeValueObj` Parameters

Parameter	Type	Description
Obj	<code>temu_Object *</code>	Object pointer
PropName	<code>const char *</code>	Name of property to read.
Val	<code>temu_Object *</code>	The value to set. It must be of the correct type.
Idx	<code>int</code>	Index of property, set to 0 if it is not an array.

### 1.1.521. `temu_writeValueSigned`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_writeValueSigned(temu_Object * Obj, const char * PropName, int64_t Val, int Idx)
```

## Description

Write signed property (with side effects)

## Parameters

Table 479. `temu_writeValueSigned` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property
Val	int64_t	The new property value
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.522. `temu_writeValueU16`

## Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_writeValueU16(temu_Object * Obj, const char * PropName, uint16_t Val, int Idx)
```

## Description

Write a property value with side-effects

## Parameters

Table 480. `temu_writeValueU16` Parameters



Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	uint16_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.523. temu\_writeValueU32

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_writeValueU32(temu_Object * Obj, const char * PropName, uint32_t Val, int Idx)
```

#### Description

Write a property value with side-effects

#### Parameters

Table 481. temu\_writeValueU32 Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	uint32_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.524. temu\_writeValueU64

#### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_writeValueU64(temu_Object * Obj, const char * PropName, uint64_t Val, int Idx)
```

## Description

Write a property value with side-effects

## Parameters

Table 482. `temu_writeValueU64` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.
Val	uint64_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

## 1.1.525. temu\_writeValueU8

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Signature

```
void temu_writeValueU8(temu_Object * Obj, const char * PropName, uint8_t Val, int Idx)
```

## Description

Write a property value with side-effects

## Parameters

Table 483. `temu_writeValueU8` Parameters

Parameter	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
PropName	const char *	Name of property to read.

Parameter	Type	Description
Val	uint8_t	The value to set. It must be of the correct type.
Idx	int	Index of property, set to 0 if it is not an array.

### 1.1.526. temu\_writeValueUnsigned

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Signature

```
void temu_writeValueUnsigned(temu_Object * Obj, const char * PropName, uint64_t Val, int Idx)
```

#### Description

Write unsigned property (with side effects)

#### Parameters

Table 484. temu\_writeValueUnsigned Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property
Val	uint64_t	The new property value
Idx	int	Index in property array (set to 0 if scalar)

### 1.1.527. xemu\_getBranchCounter

#### Include

```
#include "temu-c/Support/Profiling.h"
```

#### Signature

```
uint64_t * xemu_getBranchCounter(uint64_t src, uint64_t tgt)
```

## Description

Get a branch counter for the specific arc

## Result

Pointer to the counter

## Parameters

Table 485. `xemu_getBranchCounter` Parameters

Parameter	Type	Description
src	uint64_t	Source physical address
tgt	uint64_t	Target physical address

## 1.1.528. `xemu_writeBranchCounters`

### Include

```
#include "temu-c/Support/Profiling.h"
```

### Signature

```
int xemu_writeBranchCounters(const char * fileName)
```

### Description

Write branch counters in YAML format

Coverage is stored using physical addresses.

### Result

0 if successful

### Parameters

Table 486. `xemu_writeBranchCounters` Parameters

Parameter	Type	Description
fileName	const char *	Name of file to write

## 1.2. Structs and Unions

### 1.2.1. `temu_ARMCoProcessorIfaceRef`

## Include

```
#include "temu-c/Target/ARM.h"
```

## Type

```
struct temu_ARMCoProcessorIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.2. temu\_ARMCoProcessorIfaceRefArray

### Include

```
#include "temu-c/Target/ARM.h"
```

## Type

```
struct temu_ARMCoProcessorIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.3. temu\_ARMCpuIfaceRef

### Include

```
#include "temu-c/Target/ARM.h"
```

## Type

```
struct temu_ARMCpuIfaceRef {
```

```
}

```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.4. temu\_ARMCpulfaceRefArray

### Include

```
#include "temu-c/Target/ARM.h"

```

### Type

```
struct temu_ARMCpuIfaceRefArray {
}

```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.5. temu\_AhblfaceRef

### Include

```
#include "temu-c/Bus/Amba.h"

```

### Type

```
struct temu_AhbIfaceRef {
}

```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.6. temu\_AhblfaceRefArray

**Include**

```
#include "temu-c/Bus/Amba.h"
```

**Type**

```
struct temu_AhbIfaceRefArray {
}
```

**Description****Fields**

Field	Type	Description
-------	------	-------------

**1.2.7. temu\_AhbPnpInfo****Include**

```
#include "temu-c/Bus/Amba.h"
```

**Type**

```
struct temu_AhbPnpInfo {
}
```

**Description**

AHB bus plug and play record

**Fields**

Field	Type	Description
-------	------	-------------

**1.2.8. temu\_AnalogfaceRef****Include**

```
#include "temu-c/Bus/Analog.h"
```

**Type**

```
struct temu_AnalogIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.9. temu\_AnalogIfaceRefArray

### Include

```
#include "temu-c/Bus/Analog.h"
```

### Type

```
struct temu_AnalogIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.10. temu\_ApbIfaceRef

### Include

```
#include "temu-c/Bus/Amba.h"
```

### Type

```
struct temu_ApbIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------



## 1.2.11. temu\_ApblfaceRefArray

### Include

```
#include "temu-c/Bus/Amba.h"
```

### Type

```
struct temu_ApbIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.12. temu\_ApbPnpInfo

### Include

```
#include "temu-c/Bus/Amba.h"
```

### Type

```
struct temu_ApbPnpInfo {  
}
```

### Description

APB bus plug and play record

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.13. temu\_BinaryTranslationControllfaceRef

### Include

```
#include "temu-c/Target/Cpu.h"
```

### Type

```
struct temu_BinaryTranslationControlIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.14. temu\_BinaryTranslationControllfaceRefArray

## Include

```
#include "temu-c/Target/Cpu.h"
```

## Type

```
struct temu_BinaryTranslationControlIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.15. temu\_BlockDisassemblerIfaceRef

## Include

```
#include "temu-c/HostDisassembler/HostDisassembler.h"
```

## Type

```
struct temu_BlockDisassemblerIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.16. temu\_BlockDisassemblerIfaceRefArray

### Include

```
#include "temu-c/HostDisassembler/HostDisassembler.h"
```

### Type

```
struct temu_BlockDisassemblerIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.17. temu\_Buff

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_Buff {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.18. temu\_CacheCtrlIfaceRef

### Include

```
#include "temu-c/Memory/Cache.h"
```

### Type

```
struct temu_CacheCtrlIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.19. temu\_CacheCtrlIfaceRefArray

### Include

```
#include "temu-c/Memory/Cache.h"
```

### Type

```
struct temu_CacheCtrlIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.20. temu\_CacheIfaceRef

### Include

```
#include "temu-c/Memory/Cache.h"
```

### Type

```
struct temu_CacheIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.21. temu\_CachefaceRefArray

### Include

```
#include "temu-c/Memory/Cache.h"
```

### Type

```
struct temu_CacheIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.22. temu\_CanBusfaceRef

### Include

```
#include "temu-c/Bus/Can.h"
```

### Type

```
struct temu_CanBusIfaceRef {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.23. temu\_CanBusfaceRefArray

### Include

```
#include "temu-c/Bus/Can.h"
```

### Type

```
struct temu_CanBusIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.24. temu\_CanBusStats

### Include

```
#include "temu-c/Bus/Can.h"
```

### Type

```
struct temu_CanBusStats {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.25. temu\_CanDevIfaceRef

### Include

```
#include "temu-c/Bus/Can.h"
```

### Type

```
struct temu_CanDevIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.26. temu\_CanDevIfaceRefArray

### Include

```
#include "temu-c/Bus/Can.h"
```

### Type

```
struct temu_CanDevIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.27. temu\_CanFrame

### Include

```
#include "temu-c/Bus/Can.h"
```

### Type

```
struct temu_CanFrame {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.28. temu\_Class

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```

struct temu_Class {
    temu_Object Super;
    void * Impl;
    void * VTable;
    temu_ObjectCreateFunc Create;
    temu_ObjectDisposeFunc Dispose;
}
  
```

## Description

Class object

The TEMU object system is a dynamic object system. Classes are themselves represented as objects (instanciated by meta-classes).

The TEMU class contains among other things the constructor/destructor. Internally, the hidden implementation provides more capabilities than is exposed by the `temu_Class` type.

## Fields

Field	Type	Description
Super	temu_Object	Super class of the class instance.
Impl	void *	Internal pointer, do not touch.
VTable	void *	Internal pointer, do not touch.
Create	temu_ObjectCreateFunc	Constructor / create function for creating instances of the class.
Dispose	temu_ObjectDisposeFunc	Destructor / dispose function for disposing instances of the class.

## 1.2.29. temu\_ClockfaceRef

### Include

```
#include "temu-c/Models/Clock.h"
```

### Type

```

struct temu_ClockIfaceRef {
}
  
```



## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.30. temu\_ClockfaceRefArray

### Include

```
#include "temu-c/Models/Clock.h"
```

### Type

```
struct temu_ClockIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.31. temu\_ClockVTable

### Include

```
#include "temu-c/Models/Clock.h"
```

### Type

```
struct temu_ClockVTable {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.32. temu\_CmdArg

### Include

```
#include "temu-c/Support/CommandLine.h"
```

## Type

```
struct temu_CmdArg {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.33. temu\_CodePattern

### Include

```
#include "temu-c/Target/CodePatterns.h"
```

## Type

```
struct temu_CodePattern {
    uint64_t PhysicalAddress;
    uint64_t PhysicalAddressMask;
    temu_PatternAction Action;
    void (*)(void *, void *) Callback;
    void * CallbackData;
    int SkipLength;
    unsigned int PatternLength;
}
```

## Description

Code pattern structure

The code pattern structure exists to make it possible to define a static list of code patterns. The patterns should be installed on a processor using the code pattern interface.

The code pattern can be used to attach custom actions, either at a specific physical address or when a sequence of instructions have been matched.

TEMU uses this mechanism internally to detect idle-loops.

A pattern is matched if the physical address of the start matches, and all instructions in the pattern

match. Zero length patterns are legal if the physical address mask is non-zero. Normally, skip length and pattern length should be the same.

It is possible to skip backwards. It is also possible to skip more or less than the matched instructions.

When the pattern is installed, it is injected into a trie. The injected structure is after that no longer used and can be disposed.

Patterns are matched at decode time, for both interpreter and binary translator.

For the interpreter, a pseudo instruction will be inserted to carry out the action.

In the case of processors with branch delay slots (e.g. SPARC), the pattern is only triggered if the nPC follows the PC.

Pattern entries match individual instruction chunks, i.e. for Thumb2 an entry matches a 16-bit part, meaning two entries will be needed for a 32-bit instruction.

Note that processors are expected to implement one pattern interface per instruction set.

Patterns are not matched if they cross page boundaries.

## Fields

Field	Type	Description
PhysicalAddress	uint64_t	Physical address matched in the decoder (currently ignored)
PhysicalAddressMask	uint64_t	Physical address mask matched in the decoder, set to 0 to ignore address (currently ignored)
Action	temu_PatternAction	Action to take on this match
Callback	void (*)(void *, void *)	Function to call for actions tePA_Call and tePA_CallAndSkip, first argument is the processor pointer (currently ignored)
CallbackData	void *	Data passed in second parameter to callback (currently ignored)
SkipLength	int	Number of instructions to skip (in case of tePA_Skip and tePA_CallAndSkip)
PatternLength	unsigned int	Number of pattern entries

## 1.2.34. temu\_CodePatternEntry

### Include

```
#include "temu-c/Target/CodePatterns.h"
```

### Type

```
struct temu_CodePatternEntry {
    uint32_t Instruction;
    uint32_t Mask;
}
```

### Description

#### Fields

Field	Type	Description
Instruction	uint32_t	Instruction value
Mask	uint32_t	Instruction mask (use to ignore register numbers etc)

## 1.2.35. temu\_CodePatternIfaceRef

### Include

```
#include "temu-c/Target/CodePatterns.h"
```

### Type

```
struct temu_CodePatternIfaceRef {
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.36. temu\_CodePatternIfaceRefArray

### Include

```
#include "temu-c/Target/CodePatterns.h"
```

## Type

```
struct temu_CodePatternIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.37. temu\_CpulfaceRef

### Include

```
#include "temu-c/Target/Cpu.h"
```

## Type

```
struct temu_CpuIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.38. temu\_CpulfaceRefArray

### Include

```
#include "temu-c/Target/Cpu.h"
```

## Type

```
struct temu_CpuIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.39. temu\_CpuInfo

### Include

```
#include "temu-c/Target/Cpu.h"
```

### Type

```
struct temu_CpuInfo {
    const char * ArchName;
    const char * ModelName;
    unsigned int VASize;
    unsigned int PASize;
    unsigned int VATypeSize;
    unsigned int PATypeSize;
    unsigned int GPRCount;
    unsigned int FPRCount;
    temu_Endian InstructionEndianness;
    temu_Endian DataEndianness;
    unsigned int NumInstructionSets;
}
```

## Description

CPU architecture info that can be queried by the user.

### Fields

Field	Type	Description
ArchName	const char *	Architecture name
ModelName	const char *	Processor model name
VASize	unsigned int	Virtual address size in bits
PASize	unsigned int	Physical address size in bits
VATypeSize	unsigned int	Virtual address type size in bytes
PATypeSize	unsigned int	Physical address type size in bytes (i.e. 8 for 36 bit PA)

Field	Type	Description
GPRCount	unsigned int	GPR register count
FPRCount	unsigned int	FPR register count
InstructionEndianness	temu_Endian	Instruction endianness
DataEndianness	temu_Endian	Data endianness
NumInstructionSets	unsigned int	Number of instruction sets

## 1.2.40. temu\_CpuVTable

### Include

```
#include "temu-c/Support/VTables.h"
```

### Type

```
struct temu_CpuVTable {
    temu_CpuIface * Cpu;
    temu_IrqCtrlIface * Irq;
    temu_MemAccessIface * InvalidAccess;
    temu_ClockIface * Clock;
    temu_BinaryTranslationControlIface * Translation;
}
```

### Description

All processors provide the CpuVTable

### Fields

Field	Type	Description
Cpu	<a href="#">temu_CpuIface *</a>	CPU interface
Irq	<a href="#">temu_IrqCtrlIface *</a>	IRQ interface
InvalidAccess	<a href="#">temu_MemAccessIface *</a>	Invalid memory access interface
Clock	<a href="#">temu_ClockIface *</a>	Clock interface
Translation	<a href="#">temu_BinaryTranslationControlIface *</a>	Binary translator control interface

## 1.2.41. temu\_CreateArg

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Type

```
struct temu_CreateArg {
  const char * Key;
  temu_Propval Val;
}
```

## Description

Generic constructor argument

The object constructors takes an array of key/value pairs as parameters. The values are passed using either:

- Registered argument in the `Class.new` command method.
- Explicitly to `temu_createObject()`.
- Args parameter to `object-create` global command.

The object create function would typically scan through the array, finding relevant matching keys.

## Fields

Field	Type	Description
Key	const char *	Name of argument
Val	temu_Propval	Value of argument

## 1.2.42. temu\_DeviceIdIfaceRef

### Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

## Type

```
struct temu_DeviceIdIfaceRef {
}
```

## Description

### Fields



Field	Type	Description
-------	------	-------------

### 1.2.43. temu\_DeviceIdIfaceRefArray

#### Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

#### Type

```
struct temu_DeviceIdIfaceRefArray {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.44. temu\_DeviceIdMemAccessIfaceRef

#### Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

#### Type

```
struct temu_DeviceIdMemAccessIfaceRef {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.45. temu\_DeviceIdMemAccessIfaceRefArray

#### Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

## Type

```
struct temu_DeviceIdMemAccessIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.46. temu\_DeviceIfaceRef

### Include

```
#include "temu-c/Models/Device.h"
```

## Type

```
struct temu_DeviceIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.47. temu\_DeviceIfaceRefArray

### Include

```
#include "temu-c/Models/Device.h"
```

## Type

```
struct temu_DeviceIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.48. temu\_DynCallifaceRef

### Include

```
#include "temu-c/Bus/DynamicInvocation.h"
```

### Type

```
struct temu_DynCallIfaceRef {
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.49. temu\_DynCallifaceRefArray

### Include

```
#include "temu-c/Bus/DynamicInvocation.h"
```

### Type

```
struct temu_DynCallIfaceRefArray {
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.50. temu\_DynamicResetAddressfaceRef

### Include

```
#include "temu-c/Target/Cpu.h"
```

## Type

```
struct temu_DynamicResetAddressIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.51. temu\_DynamicResetAddressIfaceRefArray

### Include

```
#include "temu-c/Target/Cpu.h"
```

## Type

```
struct temu_DynamicResetAddressIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.52. temu\_EthFrame

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

## Type

```
struct temu_EthFrame {
  uint32_t Flags;
  temu_Buff Data;
  uint8_t [15] Preamble;
  uint8_t Sfd;
}
```

## Description

### Fields

Field	Type	Description
Flags	uint32_t	Flags used for error injection
Data	temu_Buff	ETH frame data
Preamble	uint8_t [15]	Preamble bits, normally 0x[aa aa aa aa aa aa]
Sfd	uint8_t	Start frame delimiter, normally 0xab

### 1.2.53. temu\_EthernetfaceRef

#### Include

```
#include "temu-c/Bus/Ethernet.h"
```

#### Type

```
struct temu_EthernetIfaceRef {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.54. temu\_EthernetfaceRefArray

#### Include

```
#include "temu-c/Bus/Ethernet.h"
```

#### Type

```
struct temu_EthernetIfaceRefArray {
}
```

#### Description

## Fields

Field	Type	Description
-------	------	-------------

### 1.2.55. temu\_Event

#### Include

```
#include "temu-c/Support/Events.h"
```

#### Type

```
struct temu_Event {
    int64_t Steps;
    int64_t Time;
    int64_t EventId;
    uint32_t Flags;
    int QueueIdx;
    temu_TimeSource * Queue;
    temu_Object * Obj;
    void (*)(struct temu_Event *) Func;
    int64_t Period;
    int64_t RTTime;
    int64_t RTPeriod;
    int64_t TriggerCount;
}
```

#### Description

#### Fields

Field	Type	Description
Steps	int64_t	Managed internally, do not touch
Time	int64_t	Managed internally, do not touch
EventId	int64_t	Managed internally, do not touch
Flags	uint32_t	Internal use, do not touch
QueueIdx	int	Managed internally, do not touch
Queue	<a href="#">temu_TimeSource</a> *	Managed internally, do not touch

Field	Type	Description
Obj	<a href="#">temu_Object *</a>	Managed internally, only for reading
Func	void (*)(struct temu_Event *)	Event callback
Period	int64_t	Managed: Period (cycles) for cyclic events
RTTime	int64_t	Managed: Time (monotonic ns) for RT exec
RTPeriod	int64_t	Managed: Period (ns) for RT events
TriggerCount	int64_t	Managed: Number of time event has been triggered

### 1.2.56. temu\_ExtIRInstruction

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Type

```
struct temu_ExtIRInstruction {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.57. temu\_ExtendedIR

#### Include

```
#include "temu-c/Support/Extensions128.h"
```

#### Type

```
union temu_ExtendedIR {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.58. temu\_FieldInfo

### Include

```
#include "temu-c/Support/Register.h"
```

### Type

```
struct temu_FieldInfo {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.59. temu\_GpioBusfaceRef

### Include

```
#include "temu-c/Bus/Gpio.h"
```

### Type

```
struct temu_GpioBusIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.60. temu\_GpioBusfaceRefArray

### Include



```
#include "temu-c/Bus/Gpio.h"
```

## Type

```
struct temu_GpioBusIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.61. temu\_GpioClientIfaceRef

### Include

```
#include "temu-c/Bus/Gpio.h"
```

## Type

```
struct temu_GpioClientIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.62. temu\_GpioClientIfaceRefArray

### Include

```
#include "temu-c/Bus/Gpio.h"
```

## Type

```
struct temu_GpioClientIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

### 1.2.63. temu\_IRInstruction

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Type

```
struct temu_IRInstruction {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

### 1.2.64. temu\_IfaceRef

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Type

```
struct temu_IfaceRef {
    temu_Object * Obj;
    void * Iface;
}
```

## Description

Generic interface

In TEMU interfaces are referred to using an object / interface pointer pair. The object is in general passed as the first parameter to functions in the interface. This type provides a generic interface reference where the interface pointer itself is type erased.

While the type is rarely used by itself, it is commonly returned by the TEMU API. To convert to / from this type from / to typed interface references, it is possible to either memcopy or cast field by field.

## Fields

Field	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer (first field of interface reference)
Iface	void *	Type erased interface pointer

## 1.2.65. temu\_IfaceRefArray

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_IfaceRefArray {  
    uint32_t Size;  
    uint32_t Reserved;  
    temu_IfaceRef * Ifaces;  
}
```

### Description

Dynamic interface reference array

In some cases, the number of objects we know about is unknown. To solve that TEMU provides a vector like type.



It is not the intention that the fields in the array are manipulated directly. Instead, use the [temu\\_ifaceRefArray\\*](#) functions.

### Fields

Field	Type	Description
Size	uint32_t	Number of used items in array
Reserved	uint32_t	Number of slots allocated for array
Ifaces	<a href="#">temu_IfaceRef *</a>	Interface references

## 1.2.66. temu\_InstrumenterIfaceRef

### Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

### Type

```
struct temu_InstrumenterIfaceRef {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.67. temu\_InstrumenterIfaceRefArray

### Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

### Type

```
struct temu_InstrumenterIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.68. temu\_IrqClientIfaceRef

### Include

```
#include "temu-c/Models/IrqController.h"
```

### Type

```
struct temu_IrqClientIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.69. temu\_IrqClientIfaceRefArray

## Include

```
#include "temu-c/Models/IrqController.h"
```

## Type

```
struct temu_IrqClientIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.70. temu\_IrqCtrlIfaceRef

## Include

```
#include "temu-c/Models/IrqController.h"
```

## Type

```
struct temu_IrqCtrlIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

### 1.2.71. temu\_IrqCtrlIfaceRefArray

#### Include

```
#include "temu-c/Models/IrqController.h"
```

#### Type

```
struct temu_IrqCtrlIfaceRefArray {  
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.72. temu\_LineDataLoggerIfaceRef

#### Include

```
#include "temu-c/Models/LineDataLogger.h"
```

#### Type

```
struct temu_LineDataLoggerIfaceRef {  
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.73. temu\_LineDataLoggerIfaceRefArray

#### Include

```
#include "temu-c/Models/LineDataLogger.h"
```

#### Type

```
struct temu_LineDataLoggerIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.74. temu\_List

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_List {
    temu_Type Typ;
    temu_ListNode * Head;
    temu_ListNode * Tail;
}
```

## Description

Linked list type

## Fields

Field	Type	Description
Typ	temu_Type	Element type in list
Head	<a href="#">temu_ListNode *</a>	Managed pointer, do not use directly
Tail	<a href="#">temu_ListNode *</a>	Managed pointer, do not use directly

## 1.2.75. temu\_MACIfaceRef

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

## Type

```
struct temu_MACIfaceRef {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.76. temu\_MACIfaceRefArray

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

## Type

```
struct temu_MACIfaceRefArray {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.77. temu\_MDIOIfaceRef

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

## Type

```
struct temu_MDIOIfaceRef {  
}
```

## Description

### Fields



Field	Type	Description
-------	------	-------------

### 1.2.78. temu\_MDIOfaceRefArray

#### Include

```
#include "temu-c/Bus/Ethernet.h"
```

#### Type

```
struct temu_MDIOfaceRefArray {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.79. temu\_MachinefaceRef

#### Include

```
#include "temu-c/Models/Machine.h"
```

#### Type

```
struct temu_MachineIfaceRef {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.80. temu\_MachinefaceRefArray

#### Include

```
#include "temu-c/Models/Machine.h"
```

## Type

```
struct temu_MachineIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.81. temu\_MachineVTable

### Include

```
#include "temu-c/Support/VTables.h"
```

## Type

```
struct temu_MachineVTable {
    temu_MachineIface * Machine;
}
```

## Description

All machines must implement this VTable interface. These are used to speedup certain operations on machine objects.

### Fields

Field	Type	Description
Machine	<a href="#">temu_MachineIface *</a>	Machine interface

## 1.2.82. temu\_MemAccessCapabilities

### Include

```
#include "temu-c/Memory/Memory.h"
```

## Type

```
struct temu_MemAccessCapabilities {
    uint16_t TransactionBaseSizes;
```

```
uint16_t LargeTransactions;
temu_MemoryKind Kind;
temu_MemoryEndianness Endianness;
}
```

## Description

### Fields

Field	Type	Description
TransactionBaseSizes	uint16_t	Flags indicating legal transaction sizes
LargeTransactions	uint16_t	Supports large transactions (e.g. RAM/ROM)
Kind	temu_MemoryKind	Mapping type
Endianness	temu_MemoryEndianness	Endianess of transaction interface

## 1.2.83. temu\_MemAccessIfaceRef

### Include

```
#include "temu-c/Memory/Memory.h"
```

### Type

```
struct temu_MemAccessIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.84. temu\_MemAccessIfaceRefArray

### Include

```
#include "temu-c/Memory/Memory.h"
```

### Type

```
struct temu_MemAccessIfaceRefArray {  
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.85. temu\_MemTransaction

## Include

```
#include "temu-c/Memory/Memory.h"
```

## Type

```
struct temu_MemTransaction {  
    uint64_t Va;  
    uint64_t Pa;  
    uint64_t Value;  
    uint64_t Size;  
    uint64_t Offset;  
    temu_InitiatorType InitiatorType;  
    temu_Object * Initiator;  
    void * Page;  
    uint64_t Cycles;  
    uint32_t Flags;  
    void * IR;  
}
```

## Description

Generic memory transaction.

This type is kept in sync with the emulator core. The layout is guaranteed. and will remain as is, although more fields may be added (at the bottom).

When the emulator core issues a memory transaction (assuming no ATC hit), the core allocates this structure on the stack and fills in some of the fields with default values. The memory transaction is passed by pointer or reference. By filling in the different fields you can adapt the result of the memory transaction.

## Fields

<b>Field</b>	<b>Type</b>	<b>Description</b>
Va	uint64_t	64 bit virtual for unified 32/64 bit interface.
Pa	uint64_t	64 bit physical address
Value	uint64_t	Resulting value (or written value). On MMIO Reads the model fills in this value, and on writes the written value will be stored here.



Field	Type	Description
Size	uint64_t	<p>Two-logarithm of the size of the transaction in bytes it is at most the size of the CPUs max bus size. In case of SPARCv8, this is 4 bytes (double words are issued as two accesses). As this is the 2-log of the size in bytes, a single byte access will have a size of 0, a 2 byte transaction will have size 1, a 4 byte transaction will have size 2 and an 8 byte transaction will have size 3.</p> <p>In TEMU3 this field was changed to an uint64_t from uint8_t. This as it does not add any additional space. And we can repurpose value and size as follows: - The lower 2 bits define the base unit of the transaction (same as before), 0 ⇒ 1 byte, 1 ⇒ 2 bytes, 2 ⇒ 4 bytes, 3 ⇒ 8 bytes. - The upper bits define the number of transferred units 0 implies one unit. If the transferred units is more than 0, we are dealing with a large transaction. These can be used by e.g. RAM models for block transfers etc. In that case the Value field is to be reinterpreted as a pointer to the location of the data. This means that we can use the memory access interface to e.g. read out send lists and similar items.</p> <p>The memory space must thus have a way of determining which type of transaction is legal. The MemoryAccessIface has been extended with a</p>



Field	Type	Description
Offset	uint64_t	Used for device models, this will be filled in with the offset from the start address of the device (note it is in practice possible to add a device at multiple locations (which happens in some rare cases)).
InitiatorType	temu_InitiatorType	InitiatorType identifies the type of object starting the transaction. this is only relevant when Initiator is set to non-null, and allows for the specification of device pointers in the initiator. This field is new in TEMU 2.2. The field was introduced to support the implementation of I/O MMUs.
Initiator	<a href="#">temu_Object</a> *	Initiator of the transaction (a CPU object). It can be null, which indicate that the transaction was not initiated by normal CPU activity (e.g. fetch, read or write). When the initiator is null, models should not attempt to stop the processor core, go to idle mode or raise traps (posting events is fine). An example when the initiator is null is when an MMU does a table walk. The MMU will normally special case handle table walks which access un-mapped memory.

Field	Type	Description
Page	void *	Page pointer (for caching), this can be filled in by a memory model to have the emulator core inject the page translation in the ATC. If a model sets this, the page pointer will automatically be cleared if the page has attributes (breakpoints, etc). Models that implement normal memory mapped registers should NOT populate the page pointer.
Cycles	uint64_t	Cycle cost for memory access (initialised to 0).
Flags	uint32_t	Flags for use in the memory hierarchy.
IR	void *	Intermediate code for interpreter (internally managed do not modify)

## 1.2.86. temu\_MemVTable

### Include

```
#include "temu-c/Support/VTables.h"
```

### Type

```
struct temu_MemVTable {
    temu_MemAccessIface * MemAccess;
    temu_MemoryIface * Memory;
    temu_PDCIface * PDC;
}
```

### Description

Fast memory operations

### Fields



Field	Type	Description
MemAccess	<a href="#">temu_MemAccessIface *</a>	Memory access interface
Memory	<a href="#">temu_MemoryIface *</a>	Memory interface (deprecated)
PDC	<a href="#">temu_PDCIface *</a>	Pre-decode cache interface

## 1.2.87. temu\_MemoryIfaceRef

### Include

```
#include "temu-c/Memory/Memory.h"
```

### Type

```
struct temu_MemoryIfaceRef {
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.88. temu\_MemoryIfaceRefArray

### Include

```
#include "temu-c/Memory/Memory.h"
```

### Type

```
struct temu_MemoryIfaceRefArray {
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.89. temu\_MemorySpacelfaceRef

### Include

```
#include "temu-c/Support/Memory.h"
```

**Type**

```
struct temu_MemorySpaceIfaceRef {
}
```

**Description****Fields**

Field	Type	Description
-------	------	-------------

**1.2.90. temu\_MemorySpaceIfaceRefArray****Include**

```
#include "temu-c/Support/Memory.h"
```

**Type**

```
struct temu_MemorySpaceIfaceRefArray {
}
```

**Description****Fields**

Field	Type	Description
-------	------	-------------

**1.2.91. temu\_Mil1553BusIdleInfo****Include**

```
#include "temu-c/Bus/MilStd1553.h"
```

**Type**

```
struct temu_Mil1553BusIdleInfo {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.92. temu\_Mil1553BusfaceRef

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

### Type

```
struct temu_Mil1553BusIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.93. temu\_Mil1553BusfaceRefArray

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

### Type

```
struct temu_Mil1553BusIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.94. temu\_Mil1553DevIfaceRef

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

## Type

```
struct temu_Mil1553DevIfaceRef {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.95. temu\_Mil1553DevIfaceRefArray

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

## Type

```
struct temu_Mil1553DevIfaceRefArray {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.96. temu\_Mil1553Msg

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

## Type

```
struct temu_Mil1553Msg {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.97. temu\_Mil1553Stats

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

### Type

```
struct temu_Mil1553Stats {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.98. temu\_ModeSwitchInfo

### Include

```
#include "temu-c/Target/Cpu.h"
```

### Type

```
struct temu_ModeSwitchInfo {
    uint32_t OldMode;
    uint32_t NewMode;
}
```

## Description

### Fields

Field	Type	Description
OldMode	uint32_t	Old processor privilege level
NewMode	uint32_t	New processor privilege level

## 1.2.99. temu\_ModelRegInfo

### Include

```
#include "temu-c/Support/Register.h"
```

### Type

```
struct temu_ModelRegInfo {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.100. temu\_Object

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_Object {  
    temu_Class * Class;  
    char * Name;  
    temu_TimeSource * TimeSource;  
    temu_Component * Component;  
    uint64_t LoggingFlags;  
    int64_t WillDisposeNotification;  
    int64_t DisposedNotification;  
    void * UserData;  
    uint64_t IsClassObject;  
    uint64_t IsCheckpointable;  
    uint64_t IsTimeSource;  
    uint64_t TraceMemoryReads;  
    uint64_t TraceMemoryWrites;  
    uint64_t BreakOnMemoryRead;  
    uint64_t BreakOnMemoryWrite;  
}
```

### Description

Root object type for all TEMU objects.

The TEMU object system is used for defining models in TEMU. All models must derive from `temu_Object`.

Inheritance in C is done by placing a `temu_Object` field as the first field in the derived struct.

TEMU by convention use the name `Super` for the parent field name.

The type contains a `UserData` void pointer that can be used by for example simulator integrators. The `UserData` field can be written by the user. The field is guaranteed to not be modified by the TEMU runtime.



While the convention in C is safe, in C++ it is the responsibility of the user to ensure that the derived type is a *standard layout type*.

## Fields

Field	Type	Description
Class	<code>temu_Class *</code>	Class pointer
Name	<code>char *</code>	Object name
TimeSource	<code>temu_TimeSource *</code>	Timesource object
Component	<code>temu_Component *</code>	Parent component (null for root comp)
LoggingFlags	<code>uint64_t</code>	Log category enabled/disabled
WillDisposeNotification	<code>int64_t</code>	Notification emitted before object is deleted
DisposedNotification	<code>int64_t</code>	Notification emitted after object has been deleted
UserData	<code>void *</code>	User data pointer. This is not saved in snapshots.
IsClassObject	<code>uint64_t</code>	The object is a class object
IsCheckpointable	<code>uint64_t</code>	The object is snapshottable
IsTimeSource	<code>uint64_t</code>	The object is a time source (can be safely casted to <code>temu_TimeSource</code> )
TraceMemoryReads	<code>uint64_t</code>	Memory read accesses to this object will be traced by memory space
TraceMemoryWrites	<code>uint64_t</code>	Memory write accesses to this object will be traced by memory space

Field	Type	Description
BreakOnMemoryRead	uint64_t	Memory read accesses to this object will trigger break
BreakOnMemoryWrite	uint64_t	Memory write accesses to this object will trigger break

### 1.2.101. temu\_ObjectIfaceRef

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Type

```
struct temu_ObjectIfaceRef {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.102. temu\_ObjectIfaceRefArray

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Type

```
struct temu_ObjectIfaceRefArray {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.103. temu\_PCIBridgelfaceRef



## Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIBridgeIfaceRef {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.104. temu\_PCIBridgelfaceRefArray

### Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIBridgeIfaceRefArray {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.105. temu\_PCIBusIfaceRef

### Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIBusIfaceRef {
```

}

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.106. temu\_PCIBusfaceRefArray

### Include

```
#include "temu-c/Bus/PCI.h"
```

### Type

```
struct temu_PCIBusIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.107. temu\_PCIconfig

### Include

```
#include "temu-c/Bus/PCI.h"
```

### Type

```
struct temu_PCIconfig {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.108. temu\_PCIDevice

## Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIDevice {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.109. temu\_PCIDeviceIfaceRef

### Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIDeviceIfaceRef {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.110. temu\_PCIDeviceIfaceRefArray

### Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIDeviceIfaceRefArray {
```

```
}

```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.111. temu\_PCIDeviceVTable

### Include

```
#include "temu-c/Bus/PCI.h"

```

### Type

```
struct temu_PCIDeviceVTable {
}

```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.112. temu\_PCIExpressBridge

### Include

```
#include "temu-c/Bus/PCIExpress.h"

```

### Type

```
struct temu_PCIExpressBridge {
}

```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.113. temu\_PCIExpressBridgelfaceRef

**Include**

```
#include "temu-c/Bus/PCIExpress.h"
```

**Type**

```
struct temu_PCIExpressBridgeIfaceRef {
}
```

**Description****Fields**

Field	Type	Description
-------	------	-------------

**1.2.114. temu\_PCIExpressBridgeIfaceRefArray****Include**

```
#include "temu-c/Bus/PCIExpress.h"
```

**Type**

```
struct temu_PCIExpressBridgeIfaceRefArray {
}
```

**Description****Fields**

Field	Type	Description
-------	------	-------------

**1.2.115. temu\_PCIExpressBus****Include**

```
#include "temu-c/Bus/PCIExpress.h"
```

**Type**

```
struct temu_PCIExpressBus {
```

}

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.116. temu\_PCIExpressBusfaceRef

### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

### Type

```
struct temu_PCIExpressBusIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.117. temu\_PCIExpressBusfaceRefArray

### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

### Type

```
struct temu_PCIExpressBusIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.118. temu\_PCIExpressConfig

## Include

```
#include "temu-c/Bus/PCIExpress.h"
```

## Type

```
struct temu_PCIExpressConfig {  
}
```

## Description

temu\_PCIExpressConfig: PCI Express configuration space registers

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.119. temu\_PCIExpressDevice

### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

### Type

```
struct temu_PCIExpressDevice {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.120. temu\_PCIExpressDevicefaceRef

### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

### Type

```
struct temu_PCIEExpressDeviceIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

### 1.2.121. temu\_PCIEExpressDeviceIfaceRefArray

## Include

```
#include "temu-c/Bus/PCIEExpress.h"
```

## Type

```
struct temu_PCIEExpressDeviceIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

### 1.2.122. temu\_PCIEConfigType0

## Include

```
#include "temu-c/Bus/PCIEExpress.h"
```

## Type

```
struct temu_PCIEConfigType0 {
}
```

## Description

temu\_PCIEConfigType0: Fields that exists only in config type 0

## Fields



Field	Type	Description
-------	------	-------------

### 1.2.123. temu\_PCIEConfigType1

#### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

#### Type

```
struct temu_PCIEConfigType1 {
}
```

#### Description

temu\_PCIEConfigType1: Fields that exists only in config type 1

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.124. temu\_PCIEControllerInternalCSRs

#### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

#### Type

```
struct temu_PCIEControllerInternalCSRs {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.2.125. temu\_PCIEDeviceSpecificConfigSpace

#### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

**Type**

```
struct temu_PCIEDeviceSpecificConfigSpace {
}
```

**Description****Fields**

Field	Type	Description
-------	------	-------------

**1.2.126. temu\_PDCIfaceRef****Include**

```
#include "temu-c/Support/Memory.h"
```

**Type**

```
struct temu_PDCIfaceRef {
}
```

**Description****Fields**

Field	Type	Description
-------	------	-------------

**1.2.127. temu\_PDCIfaceRefArray****Include**

```
#include "temu-c/Support/Memory.h"
```

**Type**

```
struct temu_PDCIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.128. temu\_PHYIfaceRef

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

### Type

```
struct temu_PHYIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.129. temu\_PHYIfaceRefArray

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

### Type

```
struct temu_PHYIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.130. temu\_PowerIfaceRef

### Include

```
#include "temu-c/Models/Power.h"
```

### Type

```
struct temu_PowerIfaceRef {  
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.131. temu\_PowerIfaceRefArray

### Include

```
#include "temu-c/Models/Power.h"
```

### Type

```
struct temu_PowerIfaceRefArray {  
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.132. temu\_PowerLineIfaceRef

### Include

```
#include "temu-c/Models/Power.h"
```

### Type

```
struct temu_PowerLineIfaceRef {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.133. temu\_PowerLinelfaceRefArray

### Include

```
#include "temu-c/Models/Power.h"
```

### Type

```
struct temu_PowerLineIfaceRefArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.134. temu\_PowerPCIfaceRef

### Include

```
#include "temu-c/Target/PowerPC.h"
```

### Type

```
struct temu_PowerPCIfaceRef {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.135. temu\_PowerPCIfaceRefArray

### Include

```
#include "temu-c/Target/PowerPC.h"
```

## Type

```
struct temu_PowerPCInterfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.136. temu\_PropAccessor

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Type

```
struct temu_PropAccessor {
    temu_Type Typ;
    temu_Object * Obj;
    int Index;
    void * Data;
    temu_PropWriter Writer;
    temu_PropReader Reader;
    temu_PropWriter Setter;
    temu_PropReader Getter;
}
```

## Description

Generic property accessor

This struct serves as a generic accessor for properties. The intention for this is to be able to cache property accessors, without having to disambiguate between properties and pseudo properties.

The accessor is queried by the `temu_getPropAccessor` function, and is then used by calling the functions:

- `readProp`
- `writeProp`

- `getProp`
- `setProp`

Depending on whether read, write, get or set semantics is needed.

## Fields

Field	Type	Description
Typ	temu_Type	Type of property
Obj	<a href="#">temu_Object</a> *	Pointer to object containing property
Index	int	Index used in access, can be modified by user
Data	void *	Pointer to value if applicable
Writer	temu_PropWriter	Writer function if applicable
Reader	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Reader function if applicable
Setter	temu_PropWriter	Setter function if applicable
Getter	<a href="#">int</a> ), <a href="#">temu_PropReader</a>	Getter function if applicable

## 1.2.137. temu\_PropInfo

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_PropInfo {
    const char * Name;
    temu_Type Typ;
    size_t Count;
    uintptr_t Offset;
}
```

### Description

### Fields

Field	Type	Description
Name	const char *	Name of property
Typ	temu_Type	Type tag

Field	Type	Description
Count	size_t	Number of elements in property
Offset	uintptr_t	Offset from struct start

### 1.2.138. temu\_PropName

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Type

```
struct temu_PropName {
    temu_Object * Obj;
    const char * Name;
}
```

#### Description

Named property reference

#### Fields

Field	Type	Description
Obj	<a href="#">temu_Object *</a>	Object pointer
Name	const char *	Property name

### 1.2.139. temu\_Propref

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Type

```
struct temu_Propref {
    temu_Type Typ;
    void * Ptr;
}
```



## Description

Typed property reference.

The property reference contains a type tag and a raw pointer to the property.

## Fields

Field	Type	Description
Typ	temu_Type	Type of property
Ptr	void *	Pointer to property

## 1.2.140. temu\_Propval

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_Propval {
    temu_Type Typ;
}
```

## Description

Generic property value.

As properties can be of any normal type, the propval struct provides a sum type/tagged union which contain both the type tag and the property value.

## Fields

Field	Type	Description
Typ	temu_Type	Value type

## 1.2.141. temu\_RegisterBankInfo

### Include

```
#include "temu-c/Support/Register.h"
```

### Type

```
struct temu_RegisterBankInfo {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.142. temu\_RegisterInfo

### Include

```
#include "temu-c/Support/Register.h"
```

### Type

```
struct temu_RegisterInfo {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.143. temu\_ResetIfaceRef

### Include

```
#include "temu-c/Models/Reset.h"
```

### Type

```
struct temu_ResetIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.144. temu\_ResetIfaceRefArray

### Include

```
#include "temu-c/Models/Reset.h"
```

### Type

```
struct temu_ResetIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.145. temu\_SerialIfaceRef

### Include

```
#include "temu-c/Bus/Serial.h"
```

### Type

```
struct temu_SerialIfaceRef {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.146. temu\_SerialIfaceRefArray

### Include

```
#include "temu-c/Bus/Serial.h"
```

### Type

```
struct temu_SerialIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

### 1.2.147. temu\_SignallfaceRef

## Include

```
#include "temu-c/Bus/Signal.h"
```

## Type

```
struct temu_SignallfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

### 1.2.148. temu\_SignallfaceRefArray

## Include

```
#include "temu-c/Bus/Signal.h"
```

## Type

```
struct temu_SignallfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.149. temu\_SparcV8IfaceRef

### Include

```
#include "temu-c/Target/Sparc.h"
```

### Type

```
struct temu_SparcV8IfaceRef {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.150. temu\_SparcV8IfaceRefArray

### Include

```
#include "temu-c/Target/Sparc.h"
```

### Type

```
struct temu_SparcV8IfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.151. temu\_SpiBus

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiBus {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.152. temu\_SpiBusfaceRef

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiBusIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.153. temu\_SpiBusfaceRefArray

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiBusIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.154. temu\_SpiDevConfig

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiDevConfig {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.155. temu\_SpiMasterDeviceIfaceRef

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiMasterDeviceIfaceRef {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.156. temu\_SpiMasterDeviceIfaceRefArray

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiMasterDeviceIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.157. temu\_SpiSlaveDevice

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiSlaveDevice {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.158. temu\_SpiSlaveDeviceIfaceRef

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiSlaveDeviceIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------



## 1.2.159. temu\_SpiSlaveDeviceIfaceRefArray

### Include

```
#include "temu-c/Bus/SPI.h"
```

### Type

```
struct temu_SpiSlaveDeviceIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.160. temu\_SpwPacket

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwPacket {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.161. temu\_SpwPortIfaceRef

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwPortIfaceRef {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.162. temu\_SpwPortIfaceRefArray

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Type

```
struct temu_SpwPortIfaceRefArray {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.163. temu\_SpwRmapDecodedCmdField

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Type

```
struct temu_SpwRmapDecodedCmdField {
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.164. temu\_SpwRmapDecodedCommandHeader

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapDecodedCommandHeader {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.165. temu\_SpwRmapDecodedPacket

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapDecodedPacket {  
    uint32_t TotalSize;  
    temu_SpwRmapPacketType PacketType;  
    temu_SpwRmapDecodedCmdField CmdField;  
    temu_SpwRmapRawHeader RawHeader;  
}
```

### Description

### Fields

Field	Type	Description
TotalSize	uint32_t	Total size of the packet received.
PacketType	temu_SpwRmapPacketType	The packet type as identified by bits [7,6] in instruction.

Field	Type	Description
CmdField	temu_SpwRmapDecodedCmdField	The command field, bits [5,4,3,2] in instruction.
RawHeader	temu_SpwRmapRawHeader	Raw header data access.

## 1.2.166. temu\_SpwRmapDecodedReadReply

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapDecodedReadReply {
    const uint8_t * Data;
    uint32_t AvailableDataLength;
    uint8_t DataCrc;
}
```

### Description

### Fields

Field	Type	Description
Data	const uint8_t *	Pointer to the first data char.
AvailableDataLength	uint32_t	The amount of data available.
DataCrc	uint8_t	Data crc. Valid only if AvailableDataLength > Header.DataLength.

## 1.2.167. temu\_SpwRmapDecodedReadReplyHeader

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapDecodedReadReplyHeader {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.168. temu\_SpwRmapDecodedRmwReply

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapDecodedRmwReply {
    const uint8_t * Data;
    uint32_t AvailableDataLength;
    uint8_t DataCrc;
}
```

## Description

### Fields

Field	Type	Description
Data	const uint8_t *	Pointer to the first data char.
AvailableDataLength	uint32_t	The amount of data available.
DataCrc	uint8_t	Data crc. Valid only if AvailableDataLength > Header.DataLength.

## 1.2.169. temu\_SpwRmapDecodedWriteReply

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapDecodedWriteReply {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.170. temu\_SpwRmapDecodedWriteReplyHeader

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapDecodedWriteReplyHeader {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.171. temu\_SpwRmapRawHeader

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapRawHeader {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.172. temu\_SpwRmapReadCmdPacket

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Type

```
struct temu_SpwRmapReadCmdPacket {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.173. temu\_SpwRmapRmwCmdPacket

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Type

```
struct temu_SpwRmapRmwCmdPacket {
  uint8_t AccessSize;
  const uint8_t * Data;
  const uint8_t * Mask;
  uint32_t AvailableDataLength;
  uint8_t DataCrc;
}
```

## Description

### Fields

Field	Type	Description
AccessSize	uint8_t	Size of the access.
Data	const uint8_t *	Pointer to the first data char.
Mask	const uint8_t *	Pointer to the first mask char.
AvailableDataLength	uint32_t	The amount of data available.
DataCrc	uint8_t	Data crc. Valid only if AvailableDataLength > Header.DataLength.

## 1.2.174. temu\_SpwRmapWriteCmdPacket

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

### Type

```
struct temu_SpwRmapWriteCmdPacket {
    const uint8_t * Data;
    uint32_t AvailableDataLength;
    uint8_t DataCrc;
}
```

### Description

#### Fields

Field	Type	Description
Data	const uint8_t *	Pointer to the first data char.
AvailableDataLength	uint32_t	The amount of data available.
DataCrc	uint8_t	Data crc. Valid only if AvailableDataLength > Header.DataLength.

## 1.2.175. temu\_StringBreakpointIfaceRef

### Include

```
#include "temu-c/Support/StringBreakpoint.h"
```

### Type

```
struct temu_StringBreakpointIfaceRef {
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------



## 1.2.176. temu\_StringBreakpointIfaceRefArray

### Include

```
#include "temu-c/Support/StringBreakpoint.h"
```

### Type

```
struct temu_StringBreakpointIfaceRefArray {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.177. temu\_TargetExecutionIfaceRef

### Include

```
#include "temu-c/Target/Cpu.h"
```

### Type

```
struct temu_TargetExecutionIfaceRef {  
}
```

### Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.178. temu\_TargetExecutionIfaceRefArray

### Include

```
#include "temu-c/Target/Cpu.h"
```

### Type

```
struct temu_TargetExecutionIfaceRefArray {  
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.179. temu\_TimeSource

### Include

```
#include "temu-c/Support/Events.h"
```

### Type

```
struct temu_TimeSource {  
}
```

## Description

## Fields

Field	Type	Description
-------	------	-------------

## 1.2.180. temu\_TrapEventInfo

### Include

```
#include "temu-c/Target/Cpu.h"
```

### Type

```
struct temu_TrapEventInfo {  
    uint32_t TrapId;  
    uint64_t PC;  
    uint64_t nPC;  
}
```

## Description

## Fields

Field	Type	Description
TrapId	uint32_t	Trap number (architecture specific)
PC	uint64_t	Program counter when trap occurred
nPC	uint64_t	Only valid for targets with delay slots

## 1.2.181. temu\_Vector

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_Vector {  
    temu_Type Typ;  
    void * VecData;  
    uint32_t Size;  
    uint32_t Capacity;  
}
```

### Description

Dynamically sized array

### Fields

Field	Type	Description
Typ	temu_Type	Type of vector data
VecData	void *	Managed pointer, do not use directly
Size	uint32_t	Managed do not use directly
Capacity	uint32_t	Managed do not use directly

## 1.2.182. temu\_i16Array

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_i16Array {  
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.183. temu\_i32Array

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_i32Array {  
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.184. temu\_i64Array

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_i64Array {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.185. temu\_i8Array

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_i8Array {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.186. temu\_objectArray

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_objectArray {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.2.187. temu\_u16Array

### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_u16Array {  
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.188. temu\_u32Array

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_u32Array {  
}
```

### Description

#### Fields

Field	Type	Description
-------	------	-------------

## 1.2.189. temu\_u64Array

#### Include

```
#include "temu-c/Support/Objsys.h"
```

### Type

```
struct temu_u64Array {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

### 1.2.190. temu\_u8Array

#### Include

```
#include "temu-c/Support/Objsys.h"
```

#### Type

```
struct temu_u8Array {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.3. Enumerations

### 1.3.1. temu\_ARMExecMode

#### Include

```
#include "temu-c/Target/ARM.h"
```

#### Type

```
enum temu_ARMExecMode {
  teARM_ARM = 0,
  teARM_Thumb = 1,
  teARM_Jazelle = 2,
  teARM_ThumbEE = 3,
}
```

## Description

### Enumerators

Name	Value	Description
teARM_ARM	0	Processor is in ARM mode
teARM_Thumb	1	Processor is in Thumb mode
teARM_Jazelle	2	Processor is in Jazelle mode
teARM_ThumbEE	3	Processor is in Thumb EE mode

### 1.3.2. temu\_ARMMode

#### Include

```
#include "temu-c/Target/ARM.h"
```

#### Type

```
enum temu_ARMMode {
    teARM_Usr = 0,
    teARM_Fiq = 1,
    teARM_Irq = 2,
    teARM_Svc = 3,
    teARM_Mon = 4,
    teARM_Abt = 5,
    teARM_Hyp = 6,
    teARM_Und = 7,
    teARM_Sys = 8,
}
```

#### Description

#### Enumerators

Name	Value	Description
teARM_Usr	0	User 0b10000 0x10 16
teARM_Fiq	1	Fast irq 0b10001 0x11 17
teARM_Irq	2	Interrupt 0b10010 0x12 18
teARM_Svc	3	Supervisor 0b10011 0x13 19
teARM_Mon	4	Monitor 0b10110 0x14 20
teARM_Abt	5	Abort 0b10111 0x15 21
teARM_Hyp	6	Hypervisor 0b11010 0x1a 26
teARM_Und	7	Undefined 0b11011 0b1b 27
teARM_Sys	8	System 0b11111 0b1f 31



### 1.3.3. temu\_BTStatID

#### Include

```
#include "temu-c/Target/Cpu.h"
```

#### Type

```
enum temu_BTStatID {
    teBTS_TranslatedInstructions = 0,
    teBTS_ExecutedInstructions = 1,
    teBTS_TranslatedBlocks = 2,
    teBTS_ExecutedBlocks = 3,
    teBTS_CodeSize = 4,
}
```

#### Description

Statistics ID for controlling collection of individual statistics.

#### Enumerators

Name	Value	Description
teBTS_TranslatedInstructions	0	Number of translated instructions
teBTS_ExecutedInstructions	1	Number of executed instructions
teBTS_TranslatedBlocks	2	Number of translated blocks
teBTS_ExecutedBlocks	3	Number of executed blocks
teBTS_CodeSize	4	Translated code size in bytes

### 1.3.4. temu\_ClockStopReason

#### Include

```
#include "temu-c/Models/Clock.h"
```

#### Type

```
enum temu_ClockStopReason {
    teCSR_Normal = 0,
    teCSR_Halt = 1,
}
```

```
teCSR_BreakWatch = 2,
teCSR_Early = 3,
teCSR_Panic = 4,
teCSR_Sync = 5,
}
```

## Description

Clock stop reason, these are similar to CPU exit reason, but not identical.

## Enumerators

Name	Value	Description
teCSR_Normal	0	Normal exit (cannot be passed to early exit)
teCSR_Halt	1	Exited due to clock halting
teCSR_BreakWatch	2	Exited due to breakpoint or watchpoint hit
teCSR_Early	3	Other early stop reason
teCSR_Panic	4	Clock had a serious internal error
teCSR_Sync	5	Clock needs to synchronize

### 1.3.5. temu\_CmdOptionKind

#### Include

```
#include "temu-c/Support/CommandLine.h"
```

#### Type

```
enum temu_CmdOptionKind {
  teCOK_Path = 1,
  teCOK_Object = 2,
  teCOK_Int = 3,
  teCOK_Double = 4,
  teCOK_Prop = 5,
  teCOK_Iface = 6,
  teCOK_Reg = 7,
  teCOK_Field = 8,
  teCOK_Class = 9,
}
```

## Description

## Enumerators

Name	Value	Description
teCOK_Path	1	Path is a string, but with auto completion of file names
teCOK_Object	2	Object is a named object
teCOK_Int	3	Any integer number
teCOK_Double	4	Any floating point number
teCOK_Prop	5	Property reference
teCOK_Iface	6	Interface reference
teCOK_Reg	7	Register reference
teCOK_Field	8	Register field reference
teCOK_Class	9	Class option

### 1.3.6. temu\_CpuExitReason

#### Include

```
#include "temu-c/Target/Cpu.h"
```

#### Type

```
enum temu_CpuExitReason {
    teCER_Normal = 0,
    teCER_Trap = 2,
    teCER_Halt = 3,
    teCER_Event = 4,
    teCER_Break = 5,
    teCER_WatchR = 6,
    teCER_WatchW = 7,
    teCER_Early = 8,
    teCER_Panic = 9,
    teCER_Sync = 10,
}
```

## Description

## Enumerators

Name	Value	Description
teCER_Normal	0	Normal exit (cannot be passed to early exit)
teCER_Trap	2	Exited due to trap (sync trap)
teCER_Halt	3	Exited due to halting (e.g. sparc error mode)
teCER_Event	4	Exited due to synchronised event (internally, returned for any event)
teCER_Break	5	Exited due to breakpoint hit
teCER_WatchR	6	Exited due to watchpoint read hit
teCER_WatchW	7	Exited due to watchpoint write hit
teCER_Early	8	Other early exit reason
teCER_Panic	9	Emulator panic (e.g. illegal mode transition)
teCER_Sync	10	Instruction cache sync operation

### 1.3.7. temu\_CpuState

#### Include

```
#include "temu-c/Target/Cpu.h"
```

#### Type

```
enum temu_CpuState {
  teCS_Nominal = 0,
  teCS_Halted = 1,
  teCS_Idling = 2,
}
```

#### Description

#### Enumerators

Name	Value	Description
teCS_Nominal	0	Normal all ok CPU state

Name	Value	Description
teCS_Halted	1	Halted CPU (e.g. SPARC error mode), the CPU can go to the normal state using a reset
teCS_Idling	2	The CPU is in idle mode. It will not run instructions, only advance the CPUs event queue (until the CPU moves to another mode).

### 1.3.8. temu\_Endian

#### Include

```
#include "temu-c/Target/Cpu.h"
```

#### Type

```
enum temu_Endian {
  teEN_Little = 0,
  teEN_Big = 1,
  teEN_Dynamic = 2,
}
```

#### Description

#### Enumerators

Name	Value	Description
teEN_Little	0	Always little endian
teEN_Big	1	Always big endian
teEN_Dynamic	2	Can switch at runtime

### 1.3.9. temu\_ExecutionState

#### Include

```
#include "temu-c/Support/Scheduler.h"
```

#### Type

```
enum temu_ExecutionState {
}
```

## Description

## Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.10. temu\_InitiatorType

#### Include

```
#include "temu-c/Memory/Memory.h"
```

#### Type

```
enum temu_InitiatorType {
  teIT_Cpu = 0,
  teIT_Device = 1,
  teIT_Unknown = 2,
}
```

## Description

## Enumerators

Name	Value	Description
teIT_Cpu	0	Memory transaction initiated by CPU
teIT_Device	1	Memory transaction initiated by device
teIT_Unknown	2	Memory transaction initiator unknown

### 1.3.11. temu\_InstructionFlags

#### Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

#### Type

```
enum temu_InstructionFlags {  
    teIF_Branch = 1,  
    teIF_IndirectBranch = 2,  
    teIF_Load = 4,  
    teIF_Store = 8,  
    teIF_Integer = 16,  
    teIF_Float = 32,  
    teIF_Arithmetic = 64,  
    teIF_Annulled = 128,  
    teIF_UnconditionalTaken = 256,  
    teIF_UnconditionalNotTaken = 512,  
    teIF_OnPage = 1024,  
    teIF_ModeSwitch = 2048,  
    teIF_Call = 4096,  
}
```

## Description

Instruction classification flags

## Enumerators

Name	Value	Description
teIF_Branch	1	Instruction is a branch
teIF_IndirectBranch	2	Instruction is an indirect branch
teIF_Load	4	Instruction is a load
teIF_Store	8	Instruction is a store
teIF_Integer	16	Integer instruction
teIF_Float	32	Floating point instruction
teIF_Arithmetic	64	Arithmetic instruction
teIF_Annulled	128	Annulled branch
teIF_UnconditionalTaken	256	Unconditional taken branch
teIF_UnconditionalNotTaken	512	Unconditional not-taken branch
teIF_OnPage	1024	On page branch
teIF_ModeSwitch	2048	Privilege mode switching instruction
teIF_Call	4096	Call instruction

### 1.3.12. temu\_LogLevel

## Include

```
#include "temu-c/Support/Logging.h"
```

## Type

```
enum temu_LogLevel {  
    teLL_Fatal = 0,  
    teLL_Error = 1,  
    teLL_Warning = 2,  
    teLL_Info = 3,  
    teLL_Trace = 4,  
    teLL_Debug = 5,  
}
```

## Description

Logging levels corresponds roughly to some of the RFC 5424 severity levels.

## Enumerators

Name	Value	Description
teLL_Fatal	0	Fatal, emulator cannot keep on running
teLL_Error	1	Error happened, in principle critical but up to user
teLL_Warning	2	Warnings
teLL_Info	3	Normal messages
teLL_Trace	4	Trace messages, not compiled away
teLL_Debug	5	Debug

### 1.3.13. temu\_MemoryAttr

## Include

```
#include "temu-c/Support/Memory.h"
```

## Type

```
enum temu_MemoryAttr {  
    teMA_Break = 1,  
}
```



```

teMA_WatchRead = 2,
teMA_WatchWrite = 4,
teMA_Upset = 8,
teMA_Faulty = 16,
teMA_User1 = 32,
teMA_User2 = 64,
teMA_User3 = 128,
}

```

## Description

The emulator provides 5 standard attributes, and 3 user defined ones. The attributes are set in the memory space (not the memory models), so it is possible to set a watch point on memory mapped devices. When an attribute is set on a page, that page will get a shadow attribute page (same size as the page), enabling attributes to be set on a per byte level.

Attributes are only checked on the address being accessed, the transaction size is not taken into account.

## Enumerators

Name	Value	Description
teMA_Break	1	Breakpoint set
teMA_WatchRead	2	Read watchpoint set
teMA_WatchWrite	4	Write watchpoint set
teMA_Upset	8	Single event upset
teMA_Faulty	16	Multiple event upset / uncorrectable
teMA_User1	32	User definable
teMA_User2	64	User definable
teMA_User3	128	User definable

### 1.3.14. temu\_MemoryEndianness

#### Include

```
#include "temu-c/Memory/Memory.h"
```

#### Type

```
enum temu_MemoryEndianness {
    teME_BigEndian = 0,

```

```
teME_LittleEndian = 1,
teME_UnspecifiedEndian = 2,
}
```

## Description

## Enumerators

Name	Value	Description
teME_BigEndian	0	Memory access interface in big endian
teME_LittleEndian	1	Memory access interface in little endian
teME_UnspecifiedEndian	2	Memory access interface in undefined endian

### 1.3.15. temu\_MemoryKind

## Include

```
#include "temu-c/Support/Memory.h"
```

## Type

```
enum temu_MemoryKind {
teMK_RAM = 0,
teMK_ROM = 1,
teMK_MMIO = 2,
teMK_IO = 2,
teMK_MemSpace = 4,
teMK_Last = 4,
}
```

## Description

## Enumerators

Name	Value	Description
teMK_RAM	0	Model is a RAM
teMK_ROM	1	Model is a ROM
teMK_MMIO	2	Alias for <code>teMK_IO</code>
teMK_IO	2	Model is a normal IO device

Name	Value	Description
teMK_MemSpace	4	Model is a memory space
teMK_Last	4	DO NOT USE

### 1.3.16. temu\_MemoryStat

#### Include

```
#include "temu-c/Support/Memory.h"
```

#### Type

```
enum temu_MemoryStat {  
}
```

#### Description

#### Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.17. temu\_Mil1553BusResetType

#### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

#### Type

```
enum temu_Mil1553BusResetType {  
}
```

#### Description

#### Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.18. temu\_Mil1553Error

#### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

## Type

```
enum temu_Mil1553Error {  
}
```

## Description

### Enumerators

Name	Value	Description
------	-------	-------------

## 1.3.19. temu\_Mil1553MsgType

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

## Type

```
enum temu_Mil1553MsgType {  
}
```

## Description

### Enumerators

Name	Value	Description
------	-------	-------------

## 1.3.20. temu\_PCIEMessageTypes

### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

## Type

```
enum temu_PCIEMessageTypes {  
}
```

## Description

temu\_PCIEMessageTypes: PCI Express Messages name = code

## Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.21. temu\_PatternAction

#### Include

```
#include "temu-c/Target/CodePatterns.h"
```

#### Type

```
enum temu_PatternAction {
    tePA_Idle = 0,
    tePA_Skip = 1,
    tePA_Call = 2,
    tePA_CallAndSkip = 3,
    tePA_Invalid = 4,
}
```

## Description

## Enumerators

Name	Value	Description
tePA_Idle	0	Enter idle when running pattern
tePA_Skip	1	Skip over code pattern (currently not supported)
tePA_Call	2	Call function but run code after function returns (currently not supported)
tePA_CallAndSkip	3	Call function and skip the matched instructions (currently not supported)
tePA_Invalid	4	Invalid action, do not use! Internal API use only.

### 1.3.22. temu\_PowerState

## Include

```
#include "temu-c/Models/Power.h"
```

## Type

```
enum temu_PowerState {
    tePS_Off = 0,
    tePS_On = 1,
}
```

## Description

Used to indicate whether a model is powered on

## Enumerators

Name	Value	Description
tePS_Off	0	System is powered off
tePS_On	1	System is powered on

### 1.3.23. temu\_SpwLinkState

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Type

```
enum temu_SpwLinkState {
}
```

## Description

## Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.24. temu\_SpwPacketType

## Include

```
#include "temu-c/Bus/Spacewire.h"
```

**Type**

```
enum temu_SpwPacketType {  
}
```

**Description****Enumerators**

Name	Value	Description
------	-------	-------------

**1.3.25. temu\_SpwRmapCommandType****Include**

```
#include "temu-c/Bus/Spacewire.h"
```

**Type**

```
enum temu_SpwRmapCommandType {  
}
```

**Description****Enumerators**

Name	Value	Description
------	-------	-------------

**1.3.26. temu\_SpwRmapDecodedPacketType****Include**

```
#include "temu-c/Bus/Spacewire.h"
```

**Type**

```
enum temu_SpwRmapDecodedPacketType {  
}
```

## Description

### Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.27. temu\_SpwRmapDecodingOutcome

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

#### Type

```
enum temu_SpwRmapDecodingOutcome {  
}
```

## Description

### Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.28. temu\_SpwRmapPacketType

#### Include

```
#include "temu-c/Bus/Spacewire.h"
```

#### Type

```
enum temu_SpwRmapPacketType {  
}
```

## Description

### Enumerators

Name	Value	Description
------	-------	-------------

### 1.3.29. temu\_SyncEvent

#### Include



```
#include "temu-c/Support/Events.h"
```

## Type

```
enum temu_SyncEvent {  
}
```

## Description

## Enumerators

Name	Value	Description
------	-------	-------------

## 1.3.30. temu\_Type

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Type

```
enum temu_Type {  
    teTY_Invalid = 0,  
    teTY_Intptr = 1,  
    teTY_Uintptr = 2,  
    teTY_Float = 3,  
    teTY_Double = 4,  
    teTY_U8 = 5,  
    teTY_U16 = 6,  
    teTY_U32 = 7,  
    teTY_U64 = 8,  
    teTY_I8 = 9,  
    teTY_I16 = 10,  
    teTY_I32 = 11,  
    teTY_I64 = 12,  
    teTY_Obj = 13,  
    teTY_InternalPtr = 14,  
    teTY_IfaceRef = 15,  
    teTY_IfaceRefArray = 16,  
    teTY_String = 17,  
    teTY_Buffer = 18,  
    teTY_Dict = 19,  
    teTY_Vector = 20,  
    teTY_List = 21,  
}
```

}

## Description

### Type tag

The TEMU object system uses type tags to track which type is registered and in use at runtime. Type tags are used in for example the property registration functions.

## Enumerators

Name	Value	Description
teTY_Invalid	0	Invalid value 0
teTY_Intptr	1	Pointer sized signed integer (intptr_t)
teTY_Uintptr	2	Pointer sized unsigned integer (uintptr_t)
teTY_Float	3	Single precision floating point value
teTY_Double	4	Double precision floating point value
teTY_U8	5	8-bit fixed width unsigned integer
teTY_U16	6	16-bit fixed width unsigned integer
teTY_U32	7	32-bit fixed width unsigned integer
teTY_U64	8	64-bit fixed width unsigned integer
teTY_I8	9	8-bit fixed width signed integer
teTY_I16	10	16-bit fixed width signed integer
teTY_I32	11	32-bit fixed width signed integer
teTY_I64	12	64-bit fixed width signed integer
teTY_Obj	13	Pointer to temu_Object
teTY_InternalPtr	14	Internal pointer
teTY_IfaceRef	15	Interface reference

Name	Value	Description
teTY_IfaceRefArray	16	Dynamic object/interface array
teTY_String	17	C-string, useful for serialization
teTY_Buffer	18	Buffer (see Buffer.h)
teTY_Dict	19	Dictionary
teTY_Vector	20	Vector (i.e. dynamic array)
teTY_List	21	List

## 1.4. Interfaces

### 1.4.1. temu\_ARMCoProcessorIface

#### Include

```
#include "temu-c/Target/ARM.h"
```

#### Type

```
struct temu_ARMCoProcessorIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.2. temu\_ARMCpuIface

#### Include

```
#include "temu-c/Target/ARM.h"
```

#### Type

```
struct temu_ARMCpuIface {
}
```

#### Description

## Fields

Field	Type	Description
-------	------	-------------

### 1.4.3. temu\_Ahblface

#### Include

```
#include "temu-c/Bus/Amba.h"
```

#### Type

```
struct temu_AhbIface {
}
```

#### Description

AHB bus plug and play interface

A device providing plug and play info for the AHB bridge, should implement this interface. The recommended way is to put a temu\_AhbPnpInfo struct inside your model struct and then return a pointer to this one. This way the PNP info can be changed on a per object basis.

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.4. temu\_Analogiface

#### Include

```
#include "temu-c/Bus/Analog.h"
```

#### Type

```
struct temu_AnalogIface {
}
```

#### Description

Analog signal interface NOTE THIS IS EXPERIMENTAL

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.5. temu\_Apblface

#### Include

```
#include "temu-c/Bus/Amba.h"
```

#### Type

```
struct temu_ApbIface {
}
```

#### Description

APB bus plug and play interface

A device providing plug and play info for the APB bridge, should implement this interface. The recommended way is to put a `temu_ApbPnpInfo` struct inside your model struct and then return a pointer to this one. This way the PNP info can be changed on a per object basis.

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.6. temu\_BinaryTranslationControllface

#### Include

```
#include "temu-c/Target/Cpu.h"
```

#### Type

```
struct temu_BinaryTranslationControlIface {
    void (*)(void *) enableBinaryTranslator;
    void (*)(void *) disableBinaryTranslator;
    void (*)(void *, unsigned int) setThreshold;
    int (*)(void *, uint64_t, uint64_t, unsigned int) translateInstructions;
    int (*)(void *, uint64_t, uint64_t) translateBlock;
    int (*)(void *, uint64_t, uint64_t) translateFunc;
    int (*)(void *, uint64_t, uint64_t, int) chainBlocks;
    const char (*)(void *, uint64_t) disassembleBlock;
    int (*)(void *, uint64_t) clearBlock;
    int (*)(void *, uint64_t) clearBlocksOnPage;
}
```

```

void (*)(void *, temu_BTStatID) enableStatistics;
void (*)(void *, temu_BTStatID) disableStatistics;
uint64_t (*)(void *, temu_BTStatID) getStatistics;
void (*)(void *, temu_BTStatID) clearStatistics;
}

```

## Description

## Fields

Field	Type	Description
enableBinaryTranslator	void (*)(void *)	Enable translator for processor
disableBinaryTranslator	void (*)(void *)	Disable translator for processor
setThreshold	void (*)(void *, unsigned int)	Set threshold (of call target execution) for triggering translation
translateInstructions	int (*)(void *, uint64_t, uint64_t, unsigned int)	Translate specific instruction range
translateBlock	int (*)(void *, uint64_t, uint64_t)	Translate block (terminating with a branch, or end of page (special rules for delay slots apply)).
translateFunc	int (*)(void *, uint64_t, uint64_t)	Translate a complete function (determined to be statically reachable from the PA on this page).
chainBlocks	int (*)(void *, uint64_t, uint64_t, int)	Manually chain blocks.
disassembleBlock	const char ()(void *, uint64_t)	Disassemble block (using the host assembler)
clearBlock	int (*)(void *, uint64_t)	Remove a specific block and unlink incoming/outgoing chains
clearBlocksOnPage	int (*)(void *, uint64_t)	Remove all translated blocks on a specific physical page
enableStatistics	void (*)(void *, temu_BTStatID)	Enable collection of statistic in translated code
disableStatistics	void (*)(void *, temu_BTStatID)	Disable collection of statistic in translated code
getStatistics	uint64_t (*)(void *, temu_BTStatID)	Get statistic

Field	Type	Description
clearStatistics	void (*)(void *, temu_BTStatID)	Reset statistics

### 1.4.7. temu\_BlockDisassemblerIface

#### Include

```
#include "temu-c/HostDisassembler/HostDisassembler.h"
```

#### Type

```
struct temu_BlockDisassemblerIface {
}
```

#### Description

Block disassembler disassembles data blocks of code.

The main purpose is to disassemble host code, however the interface is generically named so it could be used for other purposes in the future.

The disassemble method is supposed to return a thread local C-string, that is valid until the next call of the function. Consequently, you do not need to free or in other ways worry of the string.

field disassemble Disassembly function.

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.8. temu\_CacheCtrlIface

#### Include

```
#include "temu-c/Memory/Cache.h"
```

#### Type

```
struct temu_CacheCtrlIface {
    void (*)(void *) evictionInProgress;
    void (*)(void *) evictionCompleted;
}
```

## Description

The cache control interface can be implemented by a cache controller. Which may be embedded in a CPU or device model. Typically, the assumption is that the cache controller will turn on and off the cache and that it may need to be notified on global eviction operations, which may or may not be reflected in the cache controllers status registers.

## Fields

Field	Type	Description
evictionInProgress	void (*)(void *)	Global evict operation ongoing
evictionCompleted	void (*)(void *)	Global evict operation completed

## 1.4.9. temu\_Cacheface

### Include

```
#include "temu-c/Memory/Cache.h"
```

### Type

```
struct temu_CacheIface {
    void (*)(void *) enable;
    void (*)(void *) disable;
    void (*)(void *) freeze;
    void (*)(void *, uint64_t) lockLine;
    void (*)(void *, uint64_t) unlockLine;
    void (*)(void *) invalidateAll;
    void (*)(void *, uint64_t) invalidateLine;
    void (*)(void *) evictAll;
    void (*)(void *, uint64_t) evictLine;
    uint32_t (*)(void *) getReplacementPolicy;
    uint32_t (*)(void *) getSets;
    uint32_t (*)(void *) getWays;
    uint32_t (*)(void *) getLineSize;
    int (*)(void *, uint64_t) isValid;
    uint64_t (*)(void *, uint64_t) readFlags;
    void (*)(void *, uint64_t, uint64_t) writeFlags;
    uint64_t (*)(void *, uint64_t) readData;
    int (*)(void *, uint64_t, uint64_t) writeData;
}
```

## Description

Invalidation: cache line will be tagged as invalid (i.e. flushed) Eviction: cache line will be



invalidated and in writeBack caches the content will be written back to memory.

## Fields

Field	Type	Description
enable	void (*)(void *)	Enable cache
disable	void (*)(void *)	Disable cache
freeze	void (*)(void *)	Freeze cache
lockLine	void (*)(void *, uint64_t)	Lock line for addr
unlockLine	void (*)(void *, uint64_t)	Unlock line
invalidateAll	void (*)(void *)	Invalidate entire cache
invalidateLine	void (*)(void *, uint64_t)	Invalidate single line
evictAll	void (*)(void *)	Evict all (i.e. spill to memory)
evictLine	void (*)(void *, uint64_t)	Evict line
getReplacementPolicy	uint32_t (*)(void *)	Interrogation of cache properties - A directly mapped cache has 1 way, but typically several sets - A fully associative cache has 1 set, but typically several ways - Total cache size in bytes is sets * ways * linesize
getSets	uint32_t (*)(void *)	Get number of sets
getWays	uint32_t (*)(void *)	Get number of ways
getLineSize	uint32_t (*)(void *)	Get line size in bytes
isValid	int (*)(void *, uint64_t)	Check if an address belong to a cached line
readFlags	uint64_t (*)(void *, uint64_t)	Access the flags data, the format is cache model specific and can for specific models include also the tag. Typically, this is used for diagnostic access for more accurate cache models. Note that addr is not a physical address, but rather an address interpreted differently for different caches. I.e. the address is in a cache local address space.
writeFlags	void (*)(void *, uint64_t, uint64_t)	Write flags data

Field	Type	Description
readData	uint64_t (*)(void *, uint64_t)	Get cached data for the given address. The data returned comes from RAM using a normal memory transaction in case the cache is modelled for timing only, otherwise it comes from the cache model. Addr is a cache local address space, which depends on the cache implementation. The functions are intended for diagnostic cache access which is implemented in some systems. Note that on writes, the data will NOT be written to memory except in the case the cache line would be evicted in an accurate cache model. All reads from valid addresses succeed, other addresses are undefined. writeData returns non-zero on failure (e.g. invalid address).
writeData	int (*)(void *, uint64_t, uint64_t)	Write data in cache

### 1.4.10. temu\_CanBusIface

#### Include

```
#include "temu-c/Bus/Can.h"
```

#### Type

```
struct temu_CanBusIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.11. temu\_CanDevIface

## Include

```
#include "temu-c/Bus/Can.h"
```

## Type

```
struct temu_CanDevIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.12. temu\_Clockiface

### Include

```
#include "temu-c/Models/Clock.h"
```

## Type

```
struct temu_ClockIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.13. temu\_CodePatterniface

### Include

```
#include "temu-c/Target/CodePatterns.h"
```

## Type

```
struct temu_CodePatternIface {
```

```
}

```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.14. temu\_Cpuiface

### Include

```
#include "temu-c/Target/Cpu.h"

```

### Type

```
struct temu_CpuIface {
    int (*)(void *) wakeUp;
    void (*)(void *) forceEarlyExit;
    void (*)(void *) enterHaltedMode;
}

```

## Description

Common CPU interface.

The CPU interface provides common functionality that all processors must implement. This includes the reset and run methods. But also different register access functions. The register access functions are functions, because registers may be banked and we want some type of common interface that can access the current registers. Note that the interface currently only support 32 bit processors.

**field reset** The function executing a reset. It takes as parameter the reset type, which is 0 for a default cold reset.

**field run** Run the main emulator loop for a given number of cycles.

**field runUntil** Run the main emulator loop until its cycle counter reach the end cycles.

**field step** Run the given number of steps. A step is one instruction, completed or not. E.g. a trapping instruction does not complete but is counted as a step.

**field stepUntil** Step the emulator, but stop if the step exceeds a certain fixed time.

**field raiseTrap** Raises a trap on the CPU. THIS FUNCTION DOES NOT RETURN!!!

field `enterIdleMode` Will call `longjmp` and enter idle mode immediately, this is useful for devices that activates power down mode.

field `exitEmuCore` Will call `longjmp` to exit the emulator core. This can be used in certain cases where the other exit functions do not suite well. One being that a reset is called from an MMIO write or read, or from a watchdog event. The reset can otherwise be called when the emulator is not running, so in a model you can call `reset` and then `exitEmuCore`.

field `getGpr` Read the currently visible general purpose registers. For banked registers (e.g. SPARC reg windows), you should look at the arch specific interface instead.

field `getFpr32` Read the currently visible floating point registers.

field `setSpr` Set special purpose registers. The indexes have been explicitly choosen to be equal to what is assumed by the GDB protocol, but re-based at zero.

field `getSpr` Read special purpose registers. The indexes have been explicitly choosen to be equal to what is assumed by GDB but rebased at zero.

field `disassemble` This function will disassemble an instruction. The function returns a heap-allocated string (allocated with `malloc()`). The caller is responsible for calling `free()` and managing the lifetime.

field `invalidateAtc` Invalidates the ATC cache for the given address range. Flags can be set to control the invalidation. Bit 0: don't invalidate fetch. Bit 1: don't Invalidate read, bit 2: don't invalidate write, bit 3 don't invalidate user, bit 4 don't invalidate super.

field `translateAddress` Does a table walk and translates the virtual address to physical page address. For MMU free systems, the function returns the Va masked with  $\sim(\text{page size}-1)$ . Otherwise, the return is the translated page address and in the case of failure -1.

field `raiseTrapNoJump` Raises a trap without `longjmp` to emulator core main loop. This can be used in e.g. timed event handlers and when the core isn't running.

## Fields

Field	Type	Description
<code>wakeUp</code>	<code>int (*)(void *)</code>	Wake up CPU if it is idling, return 1 if woken up 0 if no state change occurred
<code>forceEarlyExit</code>	<code>void (*)(void *)</code>	Force early return of core (after event cleanup), unless the core returns for normal reasons, the emulator will return <code>teCER_Early</code> after the current instruction.

Field	Type	Description
enterHaltedMode	void (*)(void *)	Make the processor enter halted mode using a stack posted event.

### 1.4.15. temu\_DeviceIdIface

#### Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

#### Type

```
struct temu_DeviceIdIface {
    void (*)(void *, uint32_t, void *, temu_DeviceIdMemAccessIface *) attach;
}
```

#### Description

Device ID Interface

This interface enables an external simulator to attach a device ID handler in the DeviceIdIntegrator.

#### Fields

Field	Type	Description
attach	void (*)(void *, uint32_t, void *, temu_DeviceIdMemAccessIface *)	Attach external device

### 1.4.16. temu\_DeviceIdMemAccessIface

#### Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

#### Type

```
struct temu_DeviceIdMemAccessIface {
    void (*)(void *, uint32_t, temu_MemTransaction *) fetch;
    void (*)(void *, uint32_t, temu_MemTransaction *) read;
    void (*)(void *, uint32_t, temu_MemTransaction *) write;
}
```

```
}

```

## Description

Device ID Memory Access Interface

This interface mirrors the memory access interface, but takes an additional device id parameter to be filled in by the DeviceIdIntegrator class.

## Fields

Field	Type	Description
fetch	void (*)(void *, uint32_t, temu_MemTransaction *)	Called when fetching an instruction
read	void (*)(void *, uint32_t, temu_MemTransaction *)	Called when reading data
write	void (*)(void *, uint32_t, temu_MemTransaction *)	Called when writing data

## 1.4.17. temu\_DeviceIface

### Include

```
#include "temu-c/Models/Device.h"

```

### Type

```
struct temu_DeviceIface {
}

```

### Description

Standard device interface

field reset Called when the device is reset.

field mapDevice If defined, called when a device is mapped.

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.18. temu\_DynCallIface

## Include

```
#include "temu-c/Bus/DynamicInvocation.h"
```

## Type

```
struct temu_DynCallIface {  
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.19. temu\_DynamicResetAddressIface

### Include

```
#include "temu-c/Target/Cpu.h"
```

## Type

```
struct temu_DynamicResetAddressIface {  
    void (*)(void *, uint64_t) setResetAddress;  
}
```

## Description

Interface for controlling the reset address used in a processor.

Processors may in some cases have their reset addresses set dynamically. This interface expose such functionality to peripherals.

### Fields

Field	Type	Description
setResetAddress	void (*)(void *, uint64_t)	Update reset address in processor

## 1.4.20. temu\_EthDelayIface

### Include



```
#include "temu-c/Bus/Ethernet.h"
```

## Type

```
struct temu_EthDelayIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.21. temu\_EthernetIface

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

## Type

```
struct temu_EthernetIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.22. temu\_GpioBusIface

### Include

```
#include "temu-c/Bus/Gpio.h"
```

## Type

```
struct temu_GpioBusIface {
  void (*)(void *, uint64_t, uint64_t) setGpioBits;
  uint64_t (*)(void *, uint64_t) getGpioBits;
}
```

```
}

```

## Description

Interface implemented by the GPIO bus class.

Normally this does not have to be implemented yourself. It exist for the bus model only. In-case you need a separate bus model, you can implement this interface.

Note that this interface is deprecated for future models and it is expected that the signal interface is used instead.

## Fields

Field	Type	Description
setGpioBits	void (*)(void *, uint64_t, uint64_t)	setGpioBits should set or clear the bits in Bits if they are set in Mask. Upon a set, the GPIO bus model should notify any connected GPIO clients about the changed bits. This is done using the temu_GpioClientIface. In the built in bus model, notifications are only delivered if any of the bits actually changed.
getGpioBits	uint64_t (*)(void *, uint64_t)	Get the gpio-bits currently on the bus. The bits in mask will be extracted from the bus and returned in the result.

### 1.4.23. temu\_GpioClientIface

#### Include

```
#include "temu-c/Bus/Gpio.h"

```

#### Type

```
struct temu_GpioClientIface {
    void (*)(void *, uint64_t, uint64_t) gpioBitsChanged;
}

```

## Description

Interface for GPIO clients.

A GPIO client is a device that interface with the GPIO bus. Such a client can poll using the GpioBusIface, but it is likely better to be lazily notified about changes to the bus values. Such notifications will be delivered to the GpioClientIface.

Note that this interface is deprecated for future models and it is expected that the signal interface is used instead.

### Fields

Field	Type	Description
gpioBitsChanged	void (*)(void *, uint64_t, uint64_t)	Notification function. A client will always be notified about changed bus values. The Bus value is indicated in the Bits param, and the values that where changed are indicated in the Mask param.

## 1.4.24. temu\_InstrumenterIface

### Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

### Type

```
struct temu_InstrumenterIface {
    int (*)(void *, uint64_t, uint64_t) beginBlock;
    int (*)(void *, uint64_t, uint64_t) endBlock;
    int (*)(void *, uint64_t, uint64_t, uint32_t, uint32_t) beginInstr;
    int (*)(void *, uint64_t, uint64_t, uint32_t, uint32_t, int) endInstr;
}
```

### Description

Binary translation instrumentation interface

### Fields

Field	Type	Description
beginBlock	int (*)(void *, uint64_t, uint64_t)	Called on start of a block
endBlock	int (*)(void *, uint64_t, uint64_t)	Called at the end of a block

Field	Type	Description
beginInstr	int (*)(void *, uint64_t, uint64_t, uint32_t, uint32_t)	Called when instruction is started
endInstr	int (*)(void *, uint64_t, uint64_t, uint32_t, uint32_t, int)	Called when instruction is finished. Arm: 0 = normal / not taken branch. 1 = taken conditional instruction.

### 1.4.25. temu\_IrqClientIface

#### Include

```
#include "temu-c/Models/IrqController.h"
```

#### Type

```
struct temu_IrqClientIface {
    void (*)(void *, uint8_t) ackInterrupt;
    void (*)(void *) updateInterrupts;
}
```

#### Description

Interface to be defined for classes that uses an IRQ controller object. An IRQ controller may acknowledge to the IRQ controller client, or it may notify the IrqClient that the underlying IRQ controller has been modified and any pending IRQs should be re-issued. Typically, an updateInterrupt call happens if the IRQ registers change in such a way that the IRQ controller does not know the next interrupt to be issued. E.g. On the sparc, the updateInterrupts function will be called on its IRQ controller whenever the ET or PIL field has changed. That is updateInterrupts are called by lazy IRQ controllers.

#### Fields

Field	Type	Description
ackInterrupt	void (*)(void *, uint8_t)	Acknowledge interrupt (IRQ controller should clear interrupt)
updateInterrupts	void (*)(void *)	Called in case IRQ re-issuing is needed eg. when the IRQ controlling registers have been modified. IRQ controller should reevaluate interrupts and reraise them upstream.

## 1.4.26. temu\_IrqControllerIface

### Include

```
#include "temu-c/Models/IrqController.h"
```

### Type

```
struct temu_IrqControllerIface {  
    void (*)(void *, uint8_t) raiseInterrupt;  
    void (*)(void *, uint8_t) lowerInterrupt;  
}
```

### Description

Interrupt controller interface. An interrupt controller can raise and lower IRQ signals. For systems which has interrupts which can be configured to be active high, low or rising or falling, the raise and lower irq have different semantics. For rising edge triggering, the raiseInterrupt function should trigger the IRQ.

### Fields

Field	Type	Description
raiseInterrupt	void (*)(void *, uint8_t)	Raise interrupt
lowerInterrupt	void (*)(void *, uint8_t)	Lower interrupt

## 1.4.27. temu\_LineDataLoggerIface

### Include

```
#include "temu-c/Models/LineDataLogger.h"
```

### Type

```
struct temu_LineDataLoggerIface {  
    uint64_t (*)(void *) getLineCount;  
    const char (*)(void *, uint64_t) getLine;  
}
```

### Description

Simple interface to get recorded line data from e.g. the console model.

## Fields

Field	Type	Description
getLineCount	uint64_t (*)(void *)	Get number of lines
getLine	const char ()(void *, uint64_t)	Get pointer to line string

### 1.4.28. temu\_MACIface

#### Include

```
#include "temu-c/Bus/Ethernet.h"
```

#### Type

```
struct temu_MACIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.29. temu\_MDIOIface

#### Include

```
#include "temu-c/Bus/Ethernet.h"
```

#### Type

```
struct temu_MDIOIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.30. temu\_Machinelface

#### Include

```
#include "temu-c/Models/Machine.h"
```

## Type

```
struct temu_MachineIface {
    void (*)(void *, int) reset;
    temu_CpuExitReason (*)(void *, uint64_t) run;
    temu_CpuExitReason (*)(void *, int, uint64_t) step;
    temu_CpuIfaceRef (*)(void *) getCurrentCpu;
    temu_CpuIfaceRef (*)(void *, unsigned int) getCpu;
    int64_t (*)(void *) getTime;
    unsigned int (*)(void *) getCurrentCpuId;
    void (*)(void *) asyncStop;
}
```

## Description

Standard machine interface

## Fields

Field	Type	Description
reset	void (*)(void *, int)	Send reset to all processors in the machine
run	temu_CpuExitReason (*)(void *, uint64_t)	Run the machine for the given time
step	temu_CpuExitReason (*)(void *, int, uint64_t)	Step the given process a number of steps.
getCurrentCpu	temu_CpuIfaceRef (*)(void *)	Get current processor.
getCpu	temu_CpuIfaceRef (*)(void *, unsigned int)	Get processor by index.
getTime	int64_t (*)(void *)	Get machine time in nanoseconds.
getCurrentCpuId	unsigned int (*)(void *)	Return the current CPU number
asyncStop	void (*)(void *)	Async stop must be async safe, it can be used to tell the emulator to stop in an async safe way, e.g. from a signal handler.

### 1.4.31. temu\_MemAccessiface

## Include

```
#include "temu-c/Memory/Memory.h"
```

## Type

```
struct temu_MemAccessIface {  
    void (*)(void *, temu_MemTransaction *) fetch;  
    void (*)(void *, temu_MemTransaction *) read;  
    void (*)(void *, temu_MemTransaction *) write;  
    void (*)(void *, temu_MemTransaction *) exchange;  
    void (*)(void *, uint64_t, uint64_t) mapped;  
    const temu_MemAccessCapabilities (*)(void *) getCapabilities;  
    void (*)(void *, temu_MemTransaction *) probe;  
}
```

## Description

Memory access interface implemented by all memory mapped devices Exposed to the emulator core by a memory object.

## Fields

Field	Type	Description
fetch	void (*)(void *, temu_MemTransaction *)	Function called on fetches. The function can be null in case fetches are not allowed from the model.
read	void (*)(void *, temu_MemTransaction *)	Function called on reads.
write	void (*)(void *, temu_MemTransaction *)	Function called on writes.
exchange	void (*)(void *, temu_MemTransaction *)	Function called on atomic exchanges, by default if this is not defined, the memory space will call read followed by write in order.
mapped	void (*)(void *, uint64_t, uint64_t)	Optional method, called when interface is mapped
getCapabilities	const temu_MemAccessCapabilities ()(void *)	Query for supported features Function is optional. By default the assumption is that the base capabilities are equal to R_ALL



Field	Type	Description
W_ALL, custom memories must explicitly register itself as fetchable memory. in addition, the default assumption is that the device does not support large transactions.	probe	void (*)(void *, temu_MemTransaction *)

### 1.4.32. temu\_MemoryIface

#### Include

```
#include "temu-c/Memory/Memory.h"
```

#### Type

```
struct temu_MemoryIface {  
}
```

#### Description

For objects which have actualm memory (not just registers) This is for the simulator (not the emu core). The procedures should write the data given in bytes to the given physical offset. The offset is a 64 bit uint to support 64 bit targets. The interface is used for example by DMA transactions.

The size argument is in bytes.

The swap argument is used to swap bytes to the host endianness. Specify the log size of the read data types. - 0: We are reading bytes (bytes will be in target memory order) - 1: We are reading half words (will be swapped to host order) - 2: We are reading words (will be swapped) - 3: We are reading double words (will be swapped) With 0 for swap, we are basically reading a byte array

readBytes and writeBytes should return the number of bytes read / written or negative on error.

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.33. temu\_MemorySpacelface

#### Include

```
#include "temu-c/Support/Memory.h"
```

## Type

```

struct temu_MemorySpaceIface {
    int (*)(void *, uint64_t, uint64_t, temu_Object *, uint32_t) mapDevice;
    void (*)(void *, uint64_t, uint64_t, temu_MemoryAttr) setAttr;
    void (*)(void *, uint64_t, uint64_t, temu_MemoryAttr) clearAttr;
    temu_MemoryAttrs (*)(void *, uint64_t) getAttrs;
    int (*)(void *, uint64_t, uint64_t, temu_Object *, const char *, uint32_t)
    mapDeviceWithNamedIface;
    int (*)(void *, uint64_t, uint64_t) unmapRange;
    int (*)(void *, uint64_t, uint64_t, temu_Object *, temu_MemAccessIface *, uint32_t)
    mapDeviceWithIface;
}
  
```

## Description

### Fields

Field	Type	Description
mapDevice	int (*)(void *, uint64_t, uint64_t, temu_Object *, uint32_t)	Map device (using the default MemAccessIface interface)
setAttr	void (*)(void *, uint64_t, uint64_t, temu_MemoryAttr)	Set attribute bit
clearAttr	void (*)(void *, uint64_t, uint64_t, temu_MemoryAttr)	Clear attribute bit
getAttrs	temu_MemoryAttrs (*)(void *, uint64_t)	Get attributes for address
mapDeviceWithNamedIface	int (*)(void *, uint64_t, uint64_t, temu_Object *, const char *, uint32_t)	Map device with named interface
unmapRange	int (*)(void *, uint64_t, uint64_t)	Unmap devices in range
mapDeviceWithIface	int (*)(void *, uint64_t, uint64_t, temu_Object *, temu_MemAccessIface *, uint32_t)	Map device with interface pointer

## 1.4.34. temu\_Mil1553Busiface

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

## Type

```
struct temu_Mil1553BusIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.35. temu\_Mil1553DevIface

### Include

```
#include "temu-c/Bus/MilStd1553.h"
```

## Type

```
struct temu_Mil1553DevIface {
    void (*)(void *, temu_Mil1553BusIfaceRef, int) connected;
    void (*)(void *, temu_Mil1553BusIfaceRef, int) disconnected;
    void (*)(void *, temu_Mil1553Msg *) receive;
    void (*)(void *, temu_Mil1553BusIdleInfo *) busEnteredIdle;
}
```

## Description

### Fields

Field	Type	Description
connected	void (*)(void *, temu_Mil1553BusIfaceRef, int)	Called after device is connected to bus
disconnected	void (*)(void *, temu_Mil1553BusIfaceRef, int)	Called after device is disconnected
receive	void (*)(void *, temu_Mil1553Msg *)	Receive of 1553 message
busEnteredIdle	void (*)(void *, temu_Mil1553BusIdleInfo *)	Notifies the bus controller the bus enters an idle

## 1.4.36. temu\_ObjectIface

### Include

```
#include "temu-c/Support/Objsys.h"
```

## Type

```
struct temu_ObjectIface {  
    void (*)(void *, const char *, void *) serialise;  
    void (*)(void *, const char *, void *) deserialise;  
    int (*)(void *, int) checkSanity;  
    void (*)(void *) timeSourceSet;  
    void (*)(void *) printObject;  
}
```

## Description

Generic object interface The object interface provides generic functionality such as serialisation and sanity checking support.

## Fields

Field	Type	Description
serialise	void (*)(void *, const char *, void *)	Optional function Called after an object has been written to the snapshot, this function can write out additional properties to the snapshot and take other actions.  Note that with the pseudoproperties supporting snapshotting, this function is very rarely needed.
deserialise	void (*)(void *, const char *, void *)	Optional function Called after an object has been restored from a snapshot, this function can read additional properties to the snapshot and take other actions.  Note that with the pseudoproperties supporting snapshotting, this function is very rarely needed.

Field	Type	Description
checkSanity	int (*)(void *, int)	Return zero if the object is connected as expected, return non-zero if the object is not fully connected the default check will ensure that all the Interface references are connected, but does not care about optional interfaces or
timeSourceSet	void (*)(void *)	Optional function  Called when the time source has been set on the object The function can for example post initial events.
printObject	void (*)(void *)	Optional function  Pretty prints the object to stdout, called by the object-print command.

### 1.4.37. temu\_PCIBridgelface

#### Include

```
#include "temu-c/Bus/PCI.h"
```

#### Type

```
struct temu_PCIBridgeIface {  
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.38. temu\_PCIBusIface

#### Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIBusIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.39. temu\_PCIDeviceIface

### Include

```
#include "temu-c/Bus/PCI.h"
```

## Type

```
struct temu_PCIDeviceIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.40. temu\_PCIExpressBridgeIface

### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

## Type

```
struct temu_PCIExpressBridgeIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

### 1.4.41. temu\_PCIExpressBusiface

#### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

#### Type

```
struct temu_PCIExpressBusIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.42. temu\_PCIExpressDeviceiface

#### Include

```
#include "temu-c/Bus/PCIExpress.h"
```

#### Type

```
struct temu_PCIExpressDeviceIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.43. temu\_PDClface

#### Include

```
#include "temu-c/Support/Memory.h"
```

## Type

```
struct temu_PDCIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.44. temu\_PHYIface

### Include

```
#include "temu-c/Bus/Ethernet.h"
```

## Type

```
struct temu_PHYIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.45. temu\_PowerIface

### Include

```
#include "temu-c/Models/Power.h"
```

## Type

```
struct temu_PowerIface {
    void (*)(void *) powerOn;
    void (*)(void *) powerOff;
    temu_PowerState (*)(void *) getPowerState;
}
```

## Description



## Power interface

### Fields

Field	Type	Description
powerOn	void (*)(void *)	Power on device
powerOff	void (*)(void *)	Power off device
getPowerState	temu_PowerState (*)(void *)	Get current power state

### 1.4.46. temu\_PowerLinelface

#### Include

```
#include "temu-c/Models/Power.h"
```

#### Type

```
struct temu_PowerLineIface {  
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.47. temu\_PowerPCIface

#### Include

```
#include "temu-c/Target/PowerPC.h"
```

#### Type

```
struct temu_PowerPCIface {  
    uint64_t (*)(const void *) getXER;  
    void (*)(void *, uint64_t) setXER;  
    uint32_t (*)(const void *) getCR;  
    void (*)(void *, uint32_t) setCR;  
    uint64_t (*)(const void *) getMSR;  
    void (*)(void *, uint64_t) setMSR;  
    uint64_t (*)(const void *) getReserveAddress;  
    void (*)(const void *, uint64_t) setReserveAddress;
```

```

int (*)(const void *) isReservationBitSet;
void (*)(const void *) clearAddressReservation;
uint32_t (*)(const void *) getCTR;
void (*)(void *, uint32_t) setCTR;
uint64_t (*)(const void *) getLR;
void (*)(void *, uint64_t) setLR;
uint32_t (*)(const void *) getFPSCR;
void (*)(void *, uint32_t) setFPSCR;
}
    
```

## Description

Interface for PowerPC specific functionality

## Fields

Field	Type	Description
getXER	uint64_t (*)(const void *)	Get XER register
setXER	void (*)(void *, uint64_t)	Set XER register
getCR	uint32_t (*)(const void *)	Get CR register
setCR	void (*)(void *, uint32_t)	Set XER register
getMSR	uint64_t (*)(const void *)	Get MSR register
setMSR	void (*)(void *, uint64_t)	Set MSR register
getReserveAddress	uint64_t (*)(const void *)	Get reserved address as used by <b>lwarx</b> and <b>stwcx</b>
setReserveAddress	void (*)(const void *, uint64_t)	Set reserved address as used by <b>lwarx</b> and <b>stwcx</b>
isReservationBitSet	int (*)(const void *)	Check if reserved address as used by <b>lwarx</b> and <b>stwcx</b> is set
clearAddressReservation	void (*)(const void *)	Clear reserved address as used by <b>lwarx</b> and <b>stwcx</b>
getCTR	uint32_t (*)(const void *)	Get CTR register
setCTR	void (*)(void *, uint32_t)	Set CTR register
getLR	uint64_t (*)(const void *)	Get LR register
setLR	void (*)(void *, uint64_t)	Set LR register
getFPSCR	uint32_t (*)(const void *)	Get FPSCR register
setFPSCR	void (*)(void *, uint32_t)	Set FPSCR register

## 1.4.48. temu\_Resetlface

## Include

```
#include "temu-c/Models/Reset.h"
```

## Type

```
struct temu_ResetIface {  
    void (*)(void *, int) reset;  
}
```

## Description

Reset interface

Alternative to the reset function in the device interface.

## Fields

Field	Type	Description
reset	void (*)(void *, int)	Reset device

## 1.4.49. temu\_Serialiface

### Include

```
#include "temu-c/Bus/Serial.h"
```

### Type

```
struct temu_SerialIface {  
    void (*)(void *, uint8_t) write;  
    void (*)(void *) cts;  
}
```

### Description

Serial communications interface

### Fields

Field	Type	Description
write	void (*)(void *, uint8_t)	This function will be called when data is written on the serial bus
cts	void (*)(void *)	Clear to send. Experimental.

### 1.4.50. temu\_Signallface

#### Include

```
#include "temu-c/Bus/Signal.h"
```

#### Type

```
struct temu_SignalIface {
}
```

#### Description

Digital signal interface

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.51. temu\_SparcV8Iface

#### Include

```
#include "temu-c/Target/Sparc.h"
```

#### Type

```
struct temu_SparcV8Iface {
}
```

#### Description

Interface for SPARC specific functionality

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.52. temu\_SpiBusiface

#### Include

```
#include "temu-c/Bus/SPI.h"
```

#### Type

```
struct temu_SpiBusIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.53. temu\_SpiMasterDevicelface

#### Include

```
#include "temu-c/Bus/SPI.h"
```

#### Type

```
struct temu_SpiMasterDeviceIface {
}
```

#### Description

#### Fields

Field	Type	Description
-------	------	-------------

### 1.4.54. temu\_SpiSlaveDevicelface

#### Include

```
#include "temu-c/Bus/SPI.h"
```

## Type

```
struct temu_SpiSlaveDeviceIface {
}
```

## Description

### Fields

Field	Type	Description
-------	------	-------------

## 1.4.55. temu\_SpwPortIface

### Include

```
#include "temu-c/Bus/Spacewire.h"
```

## Type

```
struct temu_SpwPortIface {
    void (*)(void *, void *, temu_SpwPacket *) receive;
    void (*)(void *, temu_SpwLinkState) signalLinkStateChange;
    temu_SpwLinkState (*)(void *) getOtherSidelinkState;
    void (*)(void *, temu_SpwPortIfaceRef) connect;
    void (*)(void *) disconnect;
    uint64_t (*)(void *, uint64_t) timeToSendPacketNs;
}
```

## Description

SpaceWire port interface. A model must implement this interface for each SpaceWire port.

### Fields

Field	Type	Description
receive	void (*)(void *, void *, temu_SpwPacket *)	Implement to receive and handle the SpaceWire packet. This will be called by the model on the other end that sends or forwards the packet.
signalLinkStateChange	void (*)(void *, temu_SpwLinkState)	The other end device (A) uses this to inform this device (B) about its (A) change of link state.

Field	Type	Description
getOtherSideLinkState	temu_SpwLinkState (*)(void *)	Should return the link state of the device. Called by the other end device to handle connection.
connect	void (*)(void *, temu_SpwPortIfaceRef)	Connect a device to this port.
disconnect	void (*)(void *)	Disconnects the device currently connected to this port.
timeToSendPacketNs	uint64_t (*)(void *, uint64_t)	Return the amount of time required to send a packet through the port, in nano seconds.

### 1.4.56. temu\_StringBreakpointIface

#### Include

```
#include "temu-c/Support/StringBreakpoint.h"
```

#### Type

```
struct temu_StringBreakpointIface {
    void (*)(temu_Object *) printBreakpointList;
    void (*)(temu_Object *, const char *) addBreakpoint;
    void (*)(temu_Object *, int) deleteBreakpoint;
}
```

#### Description

Simple interface to get recorded line data from e.g. the console model.

#### Fields

Field	Type	Description
printBreakpointList	void (*)(temu_Object *)	print all breakpoints with their indexes
addBreakpoint	void (*)(temu_Object *, const char *)	add log line to use for break
deleteBreakpoint	void (*)(temu_Object *, int)	delete breakpoint with index = idx, if idx = -1 delete all

## 1.4.57. temu\_TargetExecutionIface

### Include

```
#include "temu-c/Target/Cpu.h"
```

### Type

```
struct temu_TargetExecutionIface {  
    void (*)(void *, uint64_t) profileCounterOverflow;  
    void (*)(void *, uint64_t, uint64_t) wrotePage;  
    void (*)(void *) getRawRuntime;  
}
```

### Description

Internal interface for binary translator

### Fields

Field	Type	Description
profileCounterOverflow	void (*)(void *, uint64_t)	Profile counter overflow handler
wrotePage	void (*)(void *, uint64_t, uint64_t)	Page written (for flushing caches)
getRawRuntime	void ()(void *)	Internal usage