TEMU

API Reference

Version 3.0-pre, 2022-04-12

Table of Contents

Chapter 1. API

This is a placeholder.

Most API docs will be extracted with Doxygen

1.1. Functions

1.1.1. 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 1. `temu_addField` Parameters

Parameter	Type	Description
R	temu_Register *	Meta register pointer.
Name	const char *	Name of of field (must be a C-compatible identifier)
Mask	uint64_t	Mask identifying the bits in the register corresponding to the field. Mask must contain consecutive bits only.
Reset	uint64_t	Reset value.

Parameter	Type	Description
Flags	uint64_t	Flags used: TEMU_FIELD_WR means field is writable, TEMU_FIELD_WARM_RESET means the field is subject to reset actions also on warm resets.
Doc	const char *	Documentation string

1.1.2. 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. uartaiface, 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 2. `temu_addInterfaceArray` Parameters

Parameter	Type	Description
Cls	temu_Class *	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]))

Parameter	Type	Description
Doc	const char *	Documentation comment for interface

1.1.3. 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 3. `temu_addInterfaceReference` Parameters

Parameter	Type	Description
Cls	temu_Class *	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.
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

1.1.4. 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 4. `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

1.1.5. 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 5. `temu_addPort` Parameters

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

1.1.6. 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 6. `temu_addProperty` Parameters

Parameter	Type	Description
Cls	temu_Class *	The class object
PropName	const char *	Name of the property to add.

Parameter	Type	Description
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	int),temu_PropReader	Property read function. This function can have side-effects.
Doc	const char *	Documentation string

1.1.7. 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 a interface reference property.

Parameters

Table 7. `temu_addPseudoInterfaceReference` Parameters

Parameter	Type	Description
Cls	temu_Class *	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.

Parameter	Type	Description
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.
Set	temu_PropWriter	Property set function. Non-semantic.
Get	int),temu_PropReader	Property get function. Non-semantic.
Doc	const char *	Documentation string

1.1.8. 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 8. `temu_addPseudoProperty` Parameters

Parameter	Type	Description
Cls	temu_Class *	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	int),temu_PropReader	Property read function (semantic effect)
Set	temu_PropWriter	Property set function (non-semantic)
Get	int),temu_PropReader	Property get function (non-semantic)
Doc	const char *	Documentation string

1.1.9. 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 9. `temu_addRegister` Parameters

Parameter	Type	Description
Bank	temu_RegisterBank *	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	int),temu_PropReader	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.

1.1.10. 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 10. `temu_addRegisterBank` Parameters

Parameter	Type	Description
C	temu_Class *	Pointer to the TEMU class
Name	const char *	Name of the register
MemAccessIface	temu_MemAccessIface *	Pointer to the register memory access interface

1.1.11. temu_asDouble

Include

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

Signature

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

Description

temu_asDouble Converts the given property to double

Result

Returns the property as an unsigned integer

Parameters

Table 11. `temu_asDouble` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be converted

1.1.12. temu_asInteger

Include

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

Signature

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

Description

temu_asInteger Converts the given propery to integer

Result

Returns the property as integer

Parameters

Table 12. `temu_asInteger` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be converted

1.1.13. temu_asUnsigned

Include

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

Signature

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

Description

temu_asUnsigned Converts the given property to unsigned integer

Result

Returns the property as an unsigned integer

Parameters

Table 13. `temu_asUnsigned` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be converted

1.1.14. 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 14. `temu_assemble` Parameters

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

1.1.15. 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 15. `temu_assembleToMemory` Parameters

Parameter	Type	Description
Cpu	void *	processor for which to assemble.
AsmStr	const char *	String with instruction in the CPUs assembler.
Addr	uint64_t	Physical address of where to put the instruction.

1.1.16. temu_asyncSocketAdd

Include

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

Signature

```
int temu_asyncSocketAdd(void * Q, 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 16. `temu_asyncSocketAdd` Parameters

Parameter	Type	Description
Q	void *	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.

Parameter	Type	Description
Flags	unsigned int	TEMU_ASYNC_READ in case you wish to be notified about data available to read.

1.1.17. 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 17. `temu_asyncSocketRemove` Parameters

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

1.1.18. temu_asyncTimerAdd

Include

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

Signature

```
int temu_asyncTimerAdd(void * Q, 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 synchornous 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 wether 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 18. `temu_asyncTimerAdd` Parameters

Parameter	Type	Description
Q	void *	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 TEMU_ASYNC_CYCLIC 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

1.1.19. temu_asyncTimerRemove

Include

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

Signature

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

Description

Remove an async timer

Parameters

Table 19. `temu_asyncTimerRemove` Parameters

Parameter	Type	Description
Fd	int	Timer ID to remove

1.1.20. 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 20. `temu_bitreverse16` Parameters

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

1.1.21. 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 21. `temu_bitreverse32` Parameters

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

1.1.22. 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 22. `temu_buffCopy` Parameters

Parameter	Type	Description
B	const temu_Buff *	The buffer to copy.

1.1.23. 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 23. `temu_buffCreate` Parameters

Parameter	Type	Description
size	uint32_t	Size of buffer in bytes

1.1.24. 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 24. `temu_buffDispose` Parameters

Parameter	Type	Description
B	temu_Buff *	buffer to delete.

1.1.25. 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 25. `temu_buffLen` Parameters

Parameter	Type	Description
B	const temu_Buff *	buffer

1.1.26. 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 26. `temu_buffReadableData` Parameters

Parameter	Type	Description
B	const temu_Buff *	Buffer to get read pointer from

1.1.27. 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 27. `temu_buffRemoveHead` Parameters

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

1.1.28. 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 28. `temu_buffRemoveTail` Parameters

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

1.1.29. 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 29. `temu_buffWritableData` Parameters

Parameter	Type	Description
B	temu_Buff *	Buffer to get data pointer from.

1.1.30. 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 30. `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.

1.1.31. 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 31. `temu_checkpointGetLength` Parameters

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

1.1.32. temu_checkpointGetValue

Include

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

Signature

```
temu_Propval 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 32. `temu_checkpointGetValue` 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)

1.1.33. 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 33. `temu_classForName` Parameters

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

1.1.34. 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 34. `temu_classForObject` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object

1.1.35. 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 35. `temu_classHasCommand` Parameters

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

1.1.36. 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 36. `temu_clearLeftmostBit32` Parameters

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

1.1.37. 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 37. `temu_clearLeftmostBit64` Parameters

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

1.1.38. 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 38. `temu_clearMemAttr` 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 attribute should be set.
Attr	temu_MemoryAttr	The attribute to clear.

1.1.39. 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 39. `temu_clearRightMostBits32` Parameters

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

1.1.40. 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 40. `temu_clearRightMostBits64` Parameters

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

1.1.41. 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 41. `temu_clz16` Parameters

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

1.1.42. 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 42. `temu_clz32` Parameters

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

1.1.43. 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 43. `temu_clz64` Parameters

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

1.1.44. 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 44. `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

1.1.45. 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 45. `temu_cmdGetData` Parameters

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

1.1.46. 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 46. `temu_cmdGetInterpreter` Parameters

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

1.1.47. 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 47. `temu_cmdGetOptionAsInteger` Parameters

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

1.1.48. 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 48. `temu_cmdGetOptionAsObject` Parameters

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

1.1.49. 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 49. `temu_cmdGetOptionAsReal` Parameters

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

1.1.50. 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 50. `temu_cmdGetOptionAsString` Parameters

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

1.1.51. 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 51. `temu_cmdGetPosOpt` Parameters

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

1.1.52. 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 52. `temu_cmdGetPosOptSize` Parameters

Parameter	Type	Description
Ctxt	void *	Command context

1.1.53. 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 53. `temu_cmdGetVariable` Parameters

Parameter	Type	Description
Key	const char *	Variable name

1.1.54. 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 54. `temu_cmdOptionIsValid` Parameters

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

1.1.55. 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 55. `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

1.1.56. 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 56. `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

1.1.57. 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 57. `temu_componentAddDelegateProp` Parameters

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

1.1.58. 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 58. `temu_componentCreate` Parameters

Parameter	Type	Description
Name	const char *	Name of component

1.1.59. 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 59. `temu_componentDispose` Parameters

Parameter	Type	Description
Comp	void *	Component to dispose.

1.1.60. temu_componentGetDelegateIface

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 60. `temu_componentGetDelegateIface` Parameters

Parameter	Type	Description
Comp	temu_Component *	The component to query the IfaceRef from
Name	const char *	Name of the interface reference.

1.1.61. temu_componentGetDelegateProp

Include

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

Signature

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

Description

Query the compontent 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 61. `temu_componentGetDelegateProp` Parameters

Parameter	Type	Description
Comp	temu_Component *	The componten to query
Name	const char *	Name of delegated property

1.1.62. 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 62. `temu_componentGetObject` Parameters

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

1.1.63. 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 63. `temu_connect` Parameters

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

1.1.64. temu_cpuDisableTraps

Include

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

Signature

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

Description

Disable traps on processor

Parameters

Table 64. `temu_cpuDisableTraps` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer

1.1.65. temu_cpuEnableTraps

Include

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

Signature

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

Description

Enable traps on processor

Parameters

Table 65. `temu_cpuEnableTraps` Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer

1.1.66. 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 66. `temu_cpuGetFpr32` Parameters

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

1.1.67. 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 67. `temu_cpuGetFpr32Bits` Parameters

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

1.1.68. 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 68. temu_cpuGetFpr64 Parameters

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

1.1.69. 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 69. `temu_cpuGetFpr64Bits` Parameters

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

1.1.70. 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 70. `temu_cpuGetFreq` Parameters

Parameter	Type	Description
Cpu	void *	The CPU object

1.1.71. 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 71. `temu_cpuGetMachine` Parameters

Parameter	Type	Description
Cpu	void *	Pointer to the CPU object

1.1.72. 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 72. `temu_cpuGetPc` Parameters

Parameter	Type	Description
Cpu	void *	The CPU object

1.1.73. 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 73. `temu_cpuGetReg` Parameters

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

1.1.74. 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 74. `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.

Parameter	Type	Description
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.

1.1.75. 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 75. `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.

1.1.76. 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 76. temu_cpuRun Parameters

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

1.1.77. 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 77. temu_cpuSetFpr32 Parameters

Parameter	Type	Description
Cpu	void *	CPU pointer

Parameter	Type	Description
Reg	unsigned int	Floating point register number
Value	float	Floating point value to set

1.1.78. 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 78. `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

1.1.79. 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 79. temu_cpuSetFpr64 Parameters

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

1.1.80. 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 80. temu_cpuSetFpr64Bits Parameters

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

1.1.81. 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 81. `temu_cpuSetPc` Parameters

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

1.1.82. 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 82. `temu_cpuSetReg` Parameters

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

1.1.83. 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 83. `temu_cpuStep` Parameters

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

1.1.84. 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 84. `temu_cpuTranslateAddress` Parameters

Parameter	Type	Description
Cpu	void *	the CPU object
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

1.1.85. 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 85. `temu_createCmd` Parameters

Parameter	Type	Description
Name	const char *	Name of command
F	temu_CommandFunc	Command function to invoke
Doc	const char *	Documentation string

Parameter	Type	Description
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.

1.1.86. 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 indended to be used in the component constructor.

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

Result

Pointer to created object.

Parameters

Table 86. `temu_createComponentObject` Parameters

Parameter	Type	Description
Comp	temu_Component *	The component under which the object is to be created.
Class	const char *	class of the created object

Parameter	Type	Description
Args	const temu_CreateArg *	NULL terminated array of create arguments for the constructor.
ObjNameFmt	const char *	Name of created object, but it allows for printf style string names, simplifying the creation of multiple objects of the same type.

1.1.87. temu_createGdbServer

Include

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

Signature

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

Description

Create a new GDB server

Result

1.1.88. 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 87. `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.

1.1.89. 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 88. `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)

1.1.90. 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 89. `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)

1.1.91. 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 90. `temu_ctz16` Parameters

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

1.1.92. 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 91. `temu_ctz32` Parameters

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

1.1.93. 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 92. `temu_ctz64` Parameters

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

1.1.94. 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 93. `temu_cyclesToNanos` Parameters

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

1.1.95. 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 94. `temu_cyclesToSecs` Parameters

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

1.1.96. 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 95. `temu_deserialiseJSON` Parameters

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

1.1.97. 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 96. `temu_deserialiseProp` Parameters

Parameter	Type	Description
Ctxt	void *	An opaque context, this is the same context as is passed to the deserialise interface function.
Obj	temu_Object *	Object being deserialised
Name	const char *	The name of the property

1.1.98. temu_dictCreate

Include

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

Signature

```
temu_Dict * temu_dictCreate()
```

Description

temu_dictCreate Creates a dictionary object

Result

A pointer to the dictionary object

1.1.99. temu_dictDispose

Include

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

Signature

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

Description

temu_dictDispose Delete a dictionary object

Parameters

Table 97. `temu_dictDispose` Parameters

Parameter	Type	Description
Dict	temu_Dict *	Pointer to the dictionary object to be deleted

1.1.100. temu_dictGetNextKey

Include

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

Signature

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

Description

temu_dictGetNextKey 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 98. `temu_dictGetNextKey` Parameters

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

1.1.101. temu_dictGetValue

Include

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

Signature

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

Description

temu_dictGetValue retrieve a dictionary value by name

Result

The value

Parameters

Table 99. `temu_dictGetValue` Parameters

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

1.1.102. temu_dictInsertValue

Include

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

Signature

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

Description

temu_dictInsertValue 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 100. `temu_dictInsertValue` Parameters

Parameter	Type	Description
Dict	temu_Dict *	Pointer to the dictionary object
Name	const char *	The name to be inserted
Val	temu_Propval	The value to be inserted

1.1.103. temu_dictRemoveValue

Include

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

Signature

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

Description

temu_dictRemoveValue 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 101. `temu_dictRemoveValue` Parameters

Parameter	Type	Description
Dict	temu_Dict *	Pointer to the dictionary object

Parameter	Type	Description
Name	const char *	The name of the pair to be removed

1.1.104. 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.

1.1.105. 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.

1.1.106. 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 dissasmbles an instruction and retuns a string allocated on the heap. You are responsible for its release using free().

Result

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

Parameters

Table 102. `temu_disassembleMemory` Parameters

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

1.1.107. temu_disposeGdbServer

Include

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

Signature

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

Description

Dispose a GDB server

1.1.108. 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 103. `temu_disposeObject` Parameters

Parameter	Type	Description
Obj	temu_Object *	The object to delete.

1.1.109. temu_disposeSymtab

Include

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

Signature

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

Description

Dispose (deallocate) the symbol table

Parameters

Table 104. `temu_disposeSymtab` Parameters

Parameter	Type	Description
Sym	temu_Symtab *	Symbol table pointer to free.

1.1.110. 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 105. `temu_elfHasDwarf` Parameters

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

1.1.111. 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 >= 1536.

Result

1.1.112. 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

1.1.113. 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 106. `temu_eventDepublish` Parameters

Parameter	Type	Description
EvID	int64_t	Event ID

1.1.114. temu_eventDeschedule

Include

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

Signature

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

Description

Deschedule event

Parameters

Table 107. `temu_eventDeschedule` Parameters

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

1.1.115. temu_eventGetCycles

Include

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

Signature

```
int64_t temu_eventGetCycles(void * Q, 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 108. `temu_eventGetCycles` Parameters

Parameter	Type	Description
Q	void *	The time source object (normally a CPU object)
EvID	int64_t	the event to get the cycles for.

1.1.116. temu_eventGetNanos

Include

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

Signature

```
int64_t temu_eventGetNanos(void * Q, 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 109. `temu_eventGetNanos` Parameters

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

1.1.117. temu_eventGetSecs

Include

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

Signature

```
double temu_eventGetSecs(void * Q, 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 110. `temu_eventGetSecs` Parameters

Parameter	Type	Description
Q	void *	The time source object (normally a CPU object)
EvID	int64_t	the event to get the seconds for.

1.1.118. 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 111. `temu_eventIsScheduled` Parameters

Parameter	Type	Description
EvID	int64_t	The event ID to check whether it is scheduled.

1.1.119. temu_eventPostAsync

Include

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

Signature

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

Description

Post an event to an asynchronous queue

Parameters

Table 112. `temu_eventPostAsync` Parameters

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

1.1.120. temu_eventPostCycles

Include

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

Signature

```
void temu_eventPostCycles(void * Q, 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 113. `temu_eventPostCycles` Parameters

Parameter	Type	Description
Q	void *	The time source object (normally a CPU object)
EvID	int64_t	The even ID returned by one of the event publish functions.

Parameter	Type	Description
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.

1.1.121. temu_eventPostNanos

Include

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

Signature

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

Description

Post events with a relative time in nanoseconds in the future

Parameters

Table 114. temu_eventPostNanos Parameters

Parameter	Type	Description
Q	void *	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.

1.1.122. temu_eventPostSecs

Include

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

Signature

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

Description

Post events with a relative time in seconds in the future

Parameters

Table 115. `temu_eventPostSecs` Parameters

Parameter	Type	Description
Q	void *	The time source object (normally a CPU object)
EvID	int64_t	The even ID returned by one of the event publish functions.
Delta	double	The number of seconds 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.

1.1.123. temu_eventPostStack

Include

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

Signature

```
void temu_eventPostStack(void * Q, 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 116. `temu_eventPostStack` Parameters

Parameter	Type	Description
Q	void *	The time source object (normally a CPU object)
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.

1.1.124. 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 >= 0 in case of success. Negative values indicate errors.

Parameters

Table 117. `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.

1.1.125. 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 118. `temu_eventPublishStruct` Parameters

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

1.1.126. temu_eventQueueGetFreq

Include

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

Signature

```
uint64_t temu_eventQueueGetFreq(void * Q)
```

Description

Get the frequency in Hz of the given queue.

Result

1.1.127. 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 119. `temu_eventSetPeriodCycles` Parameters

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

1.1.128. 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 120. temu_eventSetRTPeriodNanos Parameters

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

1.1.129. 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 absoulute 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 121. temu_eventSetRTTime Parameters

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

1.1.130. 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 122. `temu_eventSetRealTime` Parameters

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

1.1.131. 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 123. `temu_execCommand` Parameters

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

1.1.132. 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 124. `temu_execCommandFile` Parameters

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

1.1.133. 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 125. `temu_foreachClass` Parameters

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

1.1.134. 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 126. `temu_foreachComponent` Parameters

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

1.1.135. 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 127. `temu_foreachInterface` Parameters

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

1.1.136. 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 128. `temu_foreachObject` Parameters

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

1.1.137. 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 129. `temu_foreachProcessor` Parameters

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

1.1.138. 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 130. temu_foreachProperty Parameters

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

1.1.139. 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 131. temu_foreachRootComponent Parameters

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

1.1.140. 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

1.1.141. 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

1.1.142. 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.

1.1.143. temu_gdbRun

Include

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

Signature

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

Description

Run the GDB server loop

Result

1.1.144. temu_gdbWaitForConnection

Include

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

Signature

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

Description

Wait current thread for a user to connect

1.1.145. 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 132. `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 .

1.1.146. 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

1.1.147. 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()

1.1.148. temu_getCycles

Include

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

Signature

```
int64_t temu_getCycles(const void * Q)
```

Description

Get current time in cycles.

Result

Current cycles as it is understood by the object.

Parameters

Table 133. `temu_getCycles` Parameters

Parameter	Type	Description
Q	const void *	CPU object

1.1.149. 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 134. `temu_getFieldValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	The object of the class the has the register
RegName	const char *	Register name
RegIdx	unsigned int	Register index
FieldName	const char *	Field name

1.1.150. 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 135. `temu_getInterfaceName` Parameters

Parameter	Type	Description
Obj	temu_Object *	Pointer to the object containing the interface
Iface	void *	Interface to be find w.r.t. Object

1.1.151. temu_getInterfaceRef

Include

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

Signature

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

Description

temu_getInterfaceRef Retrieve a reference to an interface identified by its name

Result

The interface reference as a temu_IfaceRef object

Parameters

Table 136. `temu_getInterfaceRef` Parameters

Parameter	Type	Description
Obj	temu_Object *	Pointer to the object containing the interface
IfaceName	const char *	Name of the interface
Idx	int	TODO

1.1.152. 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 137. `temu_getLoggingCategory` Parameters

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

1.1.153. 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 mobel reg info object with a list of all register banks.

Parameters

Table 138. temu_getModelRegisterBankInfo Parameters

Parameter	Type	Description
C	temu_Class *	The class object.

1.1.154. temu_getNanos

Include

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

Signature

```
int64_t temu_getNanos(const void * Q)
```

Description

Get current time in nanoseconds

Result

Current nanoseconds as it is understood by the object.

Parameters

Table 139. temu_getNanos Parameters

Parameter	Type	Description
Q	const void *	CPU or Machine object

1.1.155. 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

1.1.156. 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()

1.1.157. temu_getPropDynLength

Include

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

Signature

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

Description

temu_getPropDynLength TODO

Result

TODO

Parameters

Table 140. `temu_getPropDynLength` Parameters

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

1.1.158. temu_getPropLength

Include

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

Signature

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

Description

temu_getPropLength Returns property length/size if a property is an array

Result

The length of the property

Parameters

Table 141. `temu_getPropLength` Parameters

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

1.1.159. 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 142. `temu_getPropName` Parameters

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

1.1.160. temu_getPropType

Include

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

Signature

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

Description

temu_getPropType Get the type of a property

Result

Property type

Parameters

Table 143. `temu_getPropType` Parameters

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

1.1.161. 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 144. `temu_getPropref` Parameters

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

1.1.162. 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 145. `temu_getRegister` Parameters

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

1.1.163. 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 146. `temu_getRegisterBank` Parameters

Parameter	Type	Description
C	temu_Class *	Pointer to the TEMU class
Name	const char *	Name of the register bank

1.1.164. 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 147. `temu_getRegisterBankName` Parameters

Parameter	Type	Description
Bank	temu_RegisterBank *	Pointer to the register bank name

1.1.165. 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 148. `temu_getRegisterColdResetValue` Parameters

Parameter	Type	Description
Reg	temu_Register *	Pointer to the register

1.1.166. 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 149. `temu_getRegisterDocs` Parameters

Parameter	Type	Description
R	temu_Register *	Pointer to the register

1.1.167. 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 150. `temu_getRegisterInfo` Parameters

Parameter	Type	Description
C	temu_Class *	The class object.
RegName	const char *	Register name.

1.1.168. 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 151. temu_getRegisterName Parameters

Parameter	Type	Description
R	temu_Register *	Pointer to the register

1.1.169. 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 152. `temu_getRegisterReadMask` Parameters

Parameter	Type	Description
Reg	temu_Register *	Pointer to the register

1.1.170. 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 153. `temu_getRegisterWarmResetValue` Parameters

Parameter	Type	Description
Reg	temu_Register *	Pointer to the register

1.1.171. 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 154. `temu_getRegisterWriteMask` Parameters

Parameter	Type	Description
Reg	temu_Register *	Pointer to the register

1.1.172. temu_getSecs

Include

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

Signature

```
double temu_getSecs(const void * Q)
```

Description

Get current time in seconds

Result

Current seconds as it is understood by the object.

Parameters

Table 155. `temu_getSecs` Parameters

Parameter	Type	Description
Q	const void *	CPU or Machine object

1.1.173. 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 156. `temu_getVTable` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object in question

1.1.174. temu_getVTableForClass

Include

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

Signature

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

Description

temu_getVTableForClass Get the VTable pointer for a class

Result

Pointer to the vtable in question

Parameters

Table 157. `temu_getVTableForClass` Parameters

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

1.1.175. temu_getValue

Include

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

Signature

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

Description

temu_getValue Get the value of a property from an object

Result

The value of the property

Parameters

Table 158. `temu_getValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.176. temu_getValueI16

Include

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

Signature

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

Description

temu_getValueI16 Get the value of a property from an object if it is an int16

Result

The value of the property

Parameters

Table 159. `temu_getValueI16` Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.177. temu_getValueI32

Include

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

Signature

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

Description

temu_getValueI32 Get the value of a property from an object if it is an int32

Result

The value of the property

Parameters

Table 160. `temu_getValueI32` Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.178. temu_getValueI64

Include

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

Signature

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

Description

temu_getValueI64 Get the value of a property from an object if it is an int64

Result

The value of the property

Parameters

Table 161. temu_getValueI64 Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.179. temu_getValueI8

Include

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

Signature

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

Description

temu_getValueI8 Get the value of a property from an object if it is an int8

Result

The value of the property

Parameters

Table 162. `temu_getValueI8` Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.180. temu_getValueU16

Include

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

Signature

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

Description

temu_getValueU16 Get the value of a property from an object if it is an uint16

Result

The value of the property

Parameters

Table 163. `temu_getValueU16` Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.181. temu_getValueU32

Include

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

Signature

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

Description

temu_getValueU32 Get the value of a property from an object if it is an uint32

Result

The value of the property

Parameters

Table 164. `temu_getValueU32` Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.182. temu_getValueU64

Include

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

Signature

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

Description

temu_getValueU64 Get the value of a property from an object if it is an uint64

Result

The value of the property

Parameters

Table 165. temu_getValueU64 Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.183. temu_getValueU8

Include

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

Signature

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

Description

temu_getValueU8 Get the value of a property from an object if it is an uint8

Result

The value of the property

Parameters

Table 166. temu_getValueU8 Parameters

Parameter	Type	Description
Obj	temu_Object *	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

1.1.184. 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 167. `temu_identifyTrailingZeroes32` Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which trailing zeros to be set

1.1.185. 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 168. `temu_identifyTrailingZeroes64` Parameters

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

1.1.186. 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 169. `temu_identifyTrailingZeroesAndFirst32` Parameters

Parameter	Type	Description
Word	uint32_t	Word to get trailing zero mask from

1.1.187. 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 170. `temu_identifyTrailingZeroesAndFirst64` Parameters

Parameter	Type	Description
Word	uint64_t	Word to get trailing zero mask from

1.1.188. 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 171. `temu_ifaceRefArrayAlloc` Parameters

Parameter	Type	Description
Reserve	unsigned int	Number of entries to reserve initially.

1.1.189. 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 172. temu_ifaceRefArrayGet Parameters

Parameter	Type	Description
Arr	temu_IfaceRefArray *	The interface reference array
idx	unsigned int	is the index of the array

1.1.190. 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 173. temu_ifaceRefArrayPush Parameters

Parameter	Type	Description
Arr	temu_IfaceRefArray *	The interface reference array
Iface	temu_IfaceRef	The interface reference (object-interface pointer pair)

1.1.191. 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 174. `temu_ifaceRefArrayPush2` Parameters

Parameter	Type	Description
Arr	temu_IfaceRefArray *	The interface reference array.
Obj	temu_Object *	The object implementing the relevant interface
Iface	void *	Pointer to the interface struct.

1.1.192. 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 175. `temu_ifaceRefArraySize` Parameters

Parameter	Type	Description
Arr	temu_IfaceRefArray *	The interface array to get the size from.

1.1.193. 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 176. `temu_imageIsELF` Parameters

Parameter	Type	Description
file	const char *	Path to file to check for ELFiness.

1.1.194. 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 177. `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

1.1.195. 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.

1.1.196. temu_initGdbServerLib

Include

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

Signature

```
void temu_initGdbServerLib()
```

Description

Initialise the GDB Server Library

1.1.197. 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.

1.1.198. 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 178. `temu_initPathSupport` Parameters

Parameter	Type	Description
Argv0	const char *	Name of TEMU executable (typically passed as argv[0] to main)

1.1.199. 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!

1.1.200. 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 179. `temu_inlineDeserialiseJSON` Parameters

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

1.1.201. 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 180. `temu_isComponent` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

1.1.202. 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 181. `temu_isCpu` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

1.1.203. temu_isDiscrete

Include

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

Signature

```
int temu_isDiscrete(temu_Propval Pv)
```

Description

temu_isDiscrete checks whether the numeric property is an integer-like

Result

0 for false, non-zero for true

Parameters

Table 182. `temu_isDiscrete` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

1.1.204. 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 183. `temu_isMachine` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

1.1.205. 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 184. `temu_isMemory` Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to test

1.1.206. 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 185. `temu_isNormalProperty` Parameters

Parameter	Type	Description
Obj	temu_Object *	The object that contains the property
PropName	const char *	The property name

1.1.207. temu_isNumber

Include

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

Signature

```
int temu_isNumber(temu_Propval Pv)
```

Description

temu_isNumber Checks whether a property is a number

Result

0 for false, non-zero for true

Parameters

Table 186. `temu_isNumber` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

1.1.208. 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 187. `temu_isPow2_32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be checked

1.1.209. 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 188. temu_isPow2_64 Parameters

Parameter	Type	Description
Word	uint64_t	The word to be checked

1.1.210. 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 189. temu_isPseudoProperty Parameters

Parameter	Type	Description
Obj	temu_Object *	The object that contains the property
PropName	const char *	The property name

1.1.211. 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 190. temu_isQualifiedAs Parameters

Parameter	Type	Description
Obj	const temu_Object *	The object to be checked
Qualifier	unsigned int	The qualification to be checked

1.1.212. temu_isReal

Include

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

Signature

```
int temu_isReal(temu_Propval Pv)
```

Description

temu_isReal Checks whether a property is a real number

Result

0 for false, non-zero for true

Parameters

Table 191. `temu_isReal` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

1.1.213. temu_isString

Include

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

Signature

```
int temu_isString(temu_Propval Pv)
```

Description

temu_isString Checks whether a property is a string

Result

0 for false, non-zero for true

Parameters

Table 192. `temu_isString` Parameters

Parameter	Type	Description
Pv	temu_Propval	The property to be checked

1.1.214. 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 193. `temu_isolateLeftmostBit32` Parameters

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

1.1.215. 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 194. `temu_isolateLeftmostBit64` Parameters

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

1.1.216. 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 195. temu_isolateRightMostZeroBit32 Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which the right most bit with value zero will be isolated

1.1.217. 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 196. temu_isolateRightMostZeroBit64 Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which the right most bit with value zero will be isolated

1.1.218. 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 197. temu_lineDataGetLine Parameters

Parameter	Type	Description
Obj	void *	Data-logging object
Line	uint64_t	Line number to read out

1.1.219. 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 198. `temu_lineDataGetLineCount` Parameters

Parameter	Type	Description
Obj	void *	Data-logging object

1.1.220. temu_listAppend

Include

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

Signature

```
void temu_listAppend(temu_List * List, temu_Propval Val)
```

Description

temu_listAppend Add an element to the end of a linked-list

Parameters

Table 199. `temu_listAppend` Parameters

Parameter	Type	Description
List	temu_List *	The list, to which the element to be added
Val	temu_Propval	The element to be added

1.1.221. temu_listCreate

Include

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

Signature

```
temu_List temu_listCreate(temu_Type Typ)
```

Description

temu_listCreate Create a linked-list object

Result

A newly created list.

Parameters

Table 200. `temu_listCreate` Parameters

Parameter	Type	Description
Typ	temu_Type	The type to be stored in the elements of the linked list

1.1.222. temu_listDispose

Include

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

Signature

```
void temu_listDispose(temu_List * List)
```

Description

temu_listDispose Delete a linked-list object

Parameters

Table 201. `temu_listDispose` Parameters

Parameter	Type	Description
List	temu_List *	The list to be deleted

1.1.223. temu_listGetHead

Include

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

Signature

```
temu_ListNode * temu_listGetHead(temu_List * List)
```

Description

temu_listGetHead Get the first element of a linked-list

Result

Pointer to the first element of the linked-list

Parameters

Table 202. `temu_listGetHead` Parameters

Parameter	Type	Description
List	temu_List *	The list, whose first element is requested

1.1.224. temu_listGetNext

Include

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

Signature

```
temu_ListNode * temu_listGetNext(temu_ListNode * Node)
```

Description

temu_listGetNext Get the next element of a linked list

Result

The next element if available, otherwise nullptr

Parameters

Table 203. `temu_listGetNext` Parameters

Parameter	Type	Description
Node	temu_ListNode *	The node, whose next element is requested

1.1.225. temu_listGetPrev

Include

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

Signature

```
temu_ListNode * temu_listGetPrev(temu_ListNode * Node)
```

Description

temu_listGetPrev Get the previous element of a linked list

Result

The previous element if available, otherwise nullptr

Parameters

Table 204. `temu_listGetPrev` Parameters

Parameter	Type	Description
Node	temu_ListNode *	The node, whose next element is requested

1.1.226. temu_listGetTail

Include

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

Signature

```
temu_ListNode * temu_listGetTail(temu_List * List)
```

Description

temu_listGetTail Get the last element of a linked-list

Result

Pointer to the last element of the linked-list

Parameters

Table 205. `temu_listGetTail` Parameters

Parameter	Type	Description
List	temu_List *	The list, whose last element is requested

1.1.227. temu_listNodeGetVal

Include

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

Signature

```
temu_Propval temu_listNodeGetVal(temu_ListNode * Node)
```

Description

temu_listNodeGetVal Get value of a linked-list node

Result

The value extracted from the node

Parameters

Table 206. `temu_listNodeGetVal` Parameters

Parameter	Type	Description
Node	temu_ListNode *	The node, from which the value to be extracted

1.1.228. temu_listPrepend

Include

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

Signature

```
void temu_listPrepend(temu_List * List, temu_Propval Val)
```

Description

temu_listPrepend Add an element to the beginning of a linked-list

Parameters

Table 207. `temu_listPrepend` Parameters

Parameter	Type	Description
List	temu_List *	The list, to which the element to be added
Val	temu_Propval	The element to be added

1.1.229. temu_listRemoveHead

Include

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

Signature

```
temu_Propval temu_listRemoveHead(temu_List * List)
```

Description

temu_listRemoveHead Removes the first element of a linked-list

Result

The element that was removed from the list

Parameters

Table 208. `temu_listRemoveHead` Parameters

Parameter	Type	Description
List	temu_List *	The list, from which the first element to be removed

1.1.230. temu_listRemoveTail

Include

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

Signature

```
temu_Propval temu_listRemoveTail(temu_List * List)
```

Description

temu_listRemoveHead Removes the last element of a linked-list

Result

The element that was removed from the list

Parameters

Table 209. `temu_listRemoveTail` Parameters

Parameter	Type	Description
List	temu_List *	The list, from which the last element to be removed

1.1.231. 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 210. `temu_loadBinaryImage` Parameters

Parameter	Type	Description
mem	void *	Memory object, must conform to the mem interface.

Parameter	Type	Description
file	const char *	Path to file to load.
pa	uint64_t	Physical address where to load the image.

1.1.232. 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 211. `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

1.1.233. 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 212. `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

1.1.234. 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 213. temu_loadImage Parameters

Parameter	Type	Description
mem	void *	Memory object, must conform to the mem interface.
file	const char *	Path to file to load.

1.1.235. 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 214. temu_loadImageAndStartAddr Parameters

Parameter	Type	Description
Mem	void *	Memory object, must conform to the mem interface.
FileName	const char *	Path to file to load.

Parameter	Type	Description
StartAddr	uint64_t *	Pointer to location to store the start address.

1.1.236. 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 215. `temu_loadPlugin` Parameters

Parameter	Type	Description
PluginName	const char *	A path or plugin name

1.1.237. 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 216. 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

1.1.238. 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 217. 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

1.1.239. 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 218. `temu_loadSymtab` Parameters

Parameter	Type	Description
FileName	const char *	Name of ELF file
Sym	temu_Symtab **	Address of your symtab pointer.

1.1.240. 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 219. `temu_logConfigError` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.241. 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 220. `temu_logConfigFatal` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.242. 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 221. `temu_logConfigInfo` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.243. 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 222. `temu_logConfigWarning` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.244. 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 223. `temu_logDebugFunc` Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

1.1.245. 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 224. `temu_logError` Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

1.1.246. 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 225. `temu_logFatal` Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

1.1.247. 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 226. `temu_logInfo` Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message
Msg	const char *	The message to log

1.1.248. 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 227. `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.

1.1.249. temu_logSetColour

Include

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

Signature

```
void temu_logSetColour(int Enable)
```

Description

Enable colours on logging

Parameters

Table 228. `temu_logSetColour` Parameters

Parameter	Type	Description
Enable	int	0 to disable colours, 1 to enable

1.1.250. 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 229. `temu_logSetDefaultFile` Parameters

Parameter	Type	Description
FP	FILE *	Logging file pointer, NULL to restore default.

1.1.251. 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 230. `temu_logSetFunc` Parameters

Parameter	Type	Description
LogFunc	void (*)(const char *)	Logging function to use. NULL to restore the default.

1.1.252. 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 231. `temu_logSetLevel` Parameters

Parameter	Type	Description
LogLevel	temu_LogLevel	The logging level.

1.1.253. 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 232. `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.

1.1.254. 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 233. `temu_logSimError` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.255. 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 234. `temu_logSimFatal` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.256. 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 235. `temu_logSimInfo` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.257. 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 236. `temu_logSimWarning` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.258. 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 237. `temu_logTargetError` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.259. 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 238. `temu_logTargetFatal` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.260. 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 239. `temu_logTargetInfo` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.261. 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 240. `temu_logTargetWarning` Parameters

Parameter	Type	Description
Obj	const void *	Object parm Msg Printf formatted message string

1.1.262. 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

1.1.263. 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 241. `temu_logWarning` Parameters

Parameter	Type	Description
Obj	const void *	is the source of the log message

Parameter	Type	Description
Msg	const char *	The message to log

1.1.264. 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 242. 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.

1.1.265. 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 243. `temu_machineRun` Parameters

Parameter	Type	Description
Machine	void *	The machine object
NanoSecs	uint64_t	The number of nanosecs to run each processor for.

1.1.266. 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 244. `temu_mapMemorySpace` 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 object is mapped.
MemObj	temu_Object *	The memory object. This object must correspond to the MemAccessIface

1.1.267. 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 245. 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	temu_Object *	The memory object. This object must correspond to the MemAccessIface
Flags	uint32_t	Sticky flags for memory accesses to that object (e.g. cachable)

1.1.268. 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 246. `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
Attr	temu_MemoryAttr	Attribute to clear

1.1.269. 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 247. `temu_memoryGetAttrs` Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical addres

1.1.270. 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 248. `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.

1.1.271. 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 249. `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
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

1.1.272. 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 250. `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
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

1.1.273. 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 251. 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

1.1.274. 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 252. temu_memoryReadData Parameters

Parameter	Type	Description
obj	void *	Memory space object

Parameter	Type	Description
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)

1.1.275. 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 253. temu_memoryReadPhys16 Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint16_t *	16 bit word that is read

1.1.276. 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 254. `temu_memoryReadPhys16Little` Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint16_t *	16 bit word that is read

1.1.277. 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 255. `temu_memoryReadPhys32` Parameters

Parameter	Type	Description
Obj	void *	Memory space

Parameter	Type	Description
Addr	uint64_t	Physical address
Word	uint32_t *	32 bit word that is read

1.1.278. 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 256. `temu_memoryReadPhys32Little` Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint32_t *	32 bit word that is read

1.1.279. 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 257. `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

1.1.280. 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 258. `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

Parameter	Type	Description
swap	int	Negative on failures. Other values indicate success.

1.1.281. 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 259. `temu_memoryWriteData` Parameters

Parameter	Type	Description
obj	void *	Memory space object
addr	uint64_t	Physical address inside memory space
buff	const 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)

1.1.282. 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 260. temu_memoryWritePhys32 Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint32_t	32 bit word to write

1.1.283. 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 261. temu_memoryWritePhys32Little Parameters

Parameter	Type	Description
Obj	void *	Memory space
Addr	uint64_t	Physical address
Word	uint32_t	32 bit word to write

1.1.284. 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 262. `temu_nameForClass` Parameters

Parameter	Type	Description
Cls	temu_Class *	Pointer to the object of the class in question

1.1.285. 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 263. `temu_nameForInterface` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object that contains the interface
Iface	const void *	Pointer to the interface

1.1.286. 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 264. `temu_nameForObject` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object

1.1.287. 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 265. `temu_nanosToCycles` Parameters

Parameter	Type	Description
Nanos	int64_t	Nanoseconds to convert
Freq	int64_t	Frequency in Hz

1.1.288. 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 266. `temu_nanosToCyclesRoundedUp` Parameters

Parameter	Type	Description
Nanos	int64_t	Nanoseconds to convert
Freq	int64_t	Frequency in Hz

1.1.289. 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 267. temu_nanosToSecs Parameters

Parameter	Type	Description
Nanos	int64_t	Nanoseconds to convert

1.1.290. 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 268. `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)

1.1.291. 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 269. `temu_normaliseRead32` Parameters

Parameter	Type	Description
Value	uint32_t	Register value
Sz	int	Size in log number of bytes (0, 1, or 2)
Off	int	Offset in bytes within the 32 bit word of the transaction (0-3)

1.1.292. 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 neded where we mix the written word with the old bits. So for transaction size of 8, and an offset of8, the resulting word is (old

0x00ff) | (new

8)

Result

Parameters

Table 270. `temu_normaliseWrite16` Parameters

Parameter	Type	Description
OldVal	uint16_t	Old regiseter 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.

1.1.293. 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 271. `temu_normaliseWrite32` Parameters

Parameter	Type	Description
OldVal	uint32_t	Old regiseter 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.

1.1.294. 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 272. `temu_notify` Parameters

Parameter	Type	Description
Id	int64_t	Notification ID
NotInfo	void *	Ponter to pass to notification handlers

1.1.295. 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 273. `temu_notifyFast` Parameters

Parameter	Type	Description
Id	int64_t *	Notification ID
NotInfo	void *	Pointer to pass to notification handlers

1.1.296. 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 274. `temu_objectForName` Parameters

Parameter	Type	Description
Name	const char *	The name of the object

1.1.297. temu_objectHasIface

Include

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

Signature

```
int temu_objectHasIface(const temu_Object * Obj, const char * IfaceName)
```

Description

temu_objectHasInterface Check if the object has a named interface

Result

0 if the interface does not exist, otherwise the interface exists

Parameters

Table 275. `temu_objectHasIface` Parameters

Parameter	Type	Description
Obj	const temu_Object *	Pointer to the object to query

Parameter	Type	Description
IfaceName	const char *	Interface name to look for

1.1.298. temu_objectHasProp

Include

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

Signature

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

Description

temu_objectHasProperty Check if the object has a named property

Result

0 if the property does not exist, otherwise the property is valid

Parameters

Table 276. `temu_objectHasProp` Parameters

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

1.1.299. 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.

1.1.300. temu_objsysClearObjects

Include

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

Signature

```
void temu_objsysClearObjects()
```

Description

Erase all objects, but do not delete classes.

1.1.301. 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 277. `temu_parity32` Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which parity to be computed

1.1.302. 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 278. temu_parity64 Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which parity to be computed

1.1.303. 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 embedds temu.

Result

0 on success, otherwise non-zero value.

Parameters

Table 279. `temu_parseCommandLineOptions` Parameters

Parameter	Type	Description
argc	int	Number of arguments
argv	const char **	Command line arguments

1.1.304. 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 280. `temu_pluginPathAppend` Parameters

Parameter	Type	Description
Path	const char *	The path to append

1.1.305. 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

1.1.306. 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 281. temu_pluginPathRemove Parameters

Parameter	Type	Description
Path	const char *	to remove

1.1.307. 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 282. temu_popcount32 Parameters

Parameter	Type	Description
Word	uint32_t	The word, in which bits to be counted

1.1.308. 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 283. `temu_popcount64` Parameters

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

1.1.309. temu_printCommandLineHelp

Include

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

Signature

```
void temu_printCommandLineHelp()
```

Description

Print command line help to temu stderr stream.

1.1.310. 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

1.1.311. 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

1.1.312. 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 284. `temu_propInfoForClass` Parameters

Parameter	Type	Description
Cls	temu_Class *	The class to inspect
PIIndex	unsigned int	The property index to read from.
PICount	unsigned int	Size of PI array in number of entries.
PI	temu_PropInfo *	Pointer to an array of prop info objects to fill in.

1.1.313. 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 285. `temu_propagateRightMostBit32` Parameters

Parameter	Type	Description
Word	uint32_t	Word to propagate bits

1.1.314. 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 286. `temu_propagateRightMostBit64` Parameters

Parameter	Type	Description
Word	uint64_t	Word to propagate bits

1.1.315. 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 287. `temu_publishNotification` Parameters

Parameter	Type	Description
NotName	const char *	Name of the notification
Obj	temu_Object *	Pointer to the object

1.1.316. 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 288. `temu_qualifyAs` Parameters

Parameter	Type	Description
Cls	temu_Class *	The class to be blessed

1.1.317. 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 289. `temu_qualifyAsCpu` Parameters

Parameter	Type	Description
Cls	temu_Class *	The class to be blessed

1.1.318. 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 290. temu_qualifyAsMachine Parameters

Parameter	Type	Description
Cls	temu_Class *	The class to be blessed

1.1.319. 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 291. temu_qualifyAsMemory Parameters

Parameter	Type	Description
Cls	temu_Class *	The class to be blessed

1.1.320. 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 292. `temu_readFieldValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	The object of the class the has the register
RegName	const char *	Register name
RegIdx	unsigned int	Register index
FieldName	const char *	Field name

1.1.321. temu_readPCIeConfigRegister

Include

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

Signature

```
temu_Propval temu_readPCIeConfigRegister(temu_PCIExpressConfig * DevConf, uint32_t
offset)
```

Description

Read register value from PCIe configuration

Result

return register value.

Parameters

Table 293. `temu_readPCIeConfigRegister` Parameters

Parameter	Type	Description
DevConf	temu_PCIExpressConfig *	PCIe configuration
offset	uint32_t	register offset in memory map.

1.1.322. 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 294. `temu_readValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
Idx	int	Index of property, set to 0 if it is not an array.

1.1.323. 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 295. `temu_readValueI16` 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

1.1.324. 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 296. `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

1.1.325. 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 297. `temu_readValueI64` 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

1.1.326. 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 298. `temu_readValueI8` 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

1.1.327. 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 299. `temu_readValueU16` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer

Parameter	Type	Description
PropName	const char *	Name of property to read
Idx	int	Index in property array, set to 0 if the property is not an array

1.1.328. 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 300. `temu_readValueU32` 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

1.1.329. 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 301. `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

1.1.330. 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 302. `temu_readValueU8` 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

1.1.331. 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 303. `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.

Parameter	Type	Description
Dispose	temu_ObjectDisposeFunc	A destructor, this function is responsible for deleting the objects of the class.

1.1.332. 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 304. `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

1.1.333. 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 305. `temu_requireInterface` Parameters

Parameter	Type	Description
Cls	temu_Class *	Class pointer
PropName	const char *	Name of interface reference property
IfaceType	const char *	Type required to connect the property

1.1.334. 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 306. `temu_roundDownNearestPow2_32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be round

1.1.335. 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 307. temu_roundDownNearestPow2_64 Parameters

Parameter	Type	Description
Word	uint64_t	The word to be round

1.1.336. 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 308. temu_roundDownToPow2_32 Parameters

Parameter	Type	Description
Word	uint32_t	The word to be rounded
Size	uint32_t	Power of 2 sized block

1.1.337. 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 309. temu_roundDownToPow2_64 Parameters

Parameter	Type	Description
Word	uint64_t	The word to be rounded
Size	uint64_t	Power of 2 sided block

1.1.338. 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 310. `temu_roundUpNearestPow2_32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be round

1.1.339. 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 311. `temu_roundUpNearestPow2_64` Parameters

Parameter	Type	Description
Word	uint64_t	The word to be round

1.1.340. 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 312. temu_roundUpToPow2_32 Parameters

Parameter	Type	Description
Word	uint32_t	The word to be rounded
Size	uint32_t	Power of 2 sized block

1.1.341. 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 313. temu_roundUpToPow2_64 Parameters

Parameter	Type	Description
Word	uint64_t	The word to be rounded
Size	uint64_t	Power of 2 sized block

1.1.342. 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 314. `temu_secsToCycles` Parameters

Parameter	Type	Description
Secs	double	Seconds to convert
Freq	int64_t	Frequency in Hz

1.1.343. 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 315. `temu_secsToNanos` Parameters

Parameter	Type	Description
Secs	double	Seconds to convert

1.1.344. 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 316. `temu_serialiseJSON` Parameters

Parameter	Type	Description
FileName	const char *	The filename, to which the JSON data is to be stored

1.1.345. 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 serialisiation interface. Note that the availability of pseudo properties makes this function and the serialise interface more or less obsolete.

Parameters

Table 317. `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

1.1.346. 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 318. `temu_setFieldValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	The object that contains the register
RegName	const char *	Register name
RegIdx	unsigned int	Register index
FieldName	const char *	Field name
Value	uint64_t	The value to be set

1.1.347. 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 319. `temu_setMemAttr` 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 attribute should be set.
Attr	temu_MemoryAttr	The attribute to set.

1.1.348. temu_setRightmostZeoroBit64

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 320. temu_setRightmostZeoroBit64 Parameters

Parameter	Type	Description
Word	uint64_t	The word, in which the right most set bit will be set

1.1.349. 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 321. temu_setRightmostZeroBit32 Parameters

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

1.1.350. temu_setTimeSource

Include

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

Signature

```
void temu_setTimeSource(temu_Object * Obj, temu_Object * 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 322. `temu_setTimeSource` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object to set time source in.
TS	temu_Object *	Time source object (CPU or clock model)

1.1.351. 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 323. `temu_setVTable` Parameters

Parameter	Type	Description
Cls	temu_Class *	The class, for which the vtable to be set
VTable	void *	Pointer to the vtable to be used

1.1.352. 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 324. `temu_setValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	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.

1.1.353. 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 325. `temu_setValueI16` Parameters

Parameter	Type	Description
Obj	temu_Object *	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.

1.1.354. 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 326. `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.

Parameter	Type	Description
Idx	int	Index of property, set to 0 if it is not an array.

1.1.355. 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 327. `temu_setValueI64` 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.

1.1.356. 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 328. `temu_setValueI8` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
Val	int8_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.

1.1.357. 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 329. `temu_setValueU16` Parameters

Parameter	Type	Description
Obj	temu_Object *	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.

1.1.358. 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 330. `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.

1.1.359. 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 331. `temu_setValueU64` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer

Parameter	Type	Description
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.

1.1.360. 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 332. temu_setValueU8 Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
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.

1.1.361. 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 333. `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

1.1.362. temu_simGetCpuCount

Include

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

Signature

```
int temu_simGetCpuCount()
```

Description

Get the number of CPUs in the simulator.

Result

Number of CPUs

1.1.363. temu_simGetCurrentCpu

Include

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

Signature

```
int temu_simGetCurrentCpu()
```

Description

Get the id of the current CPU for the simulator.

Result

CPU id of the current CPU

1.1.364. temu_simGetExitReason

Include

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

Signature

```
temu_CpuExitReason temu_simGetExitReason()
```

Description

Get the last exit reason for the simulator

This can be used to query the last exit result for the run and step functions.

Result

Exit reason from last call.

1.1.365. temu_simGetTime

Include

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

Signature

```
int64_t temu_simGetTime()
```

Description

Get current time of simulator in nanoseconds.

The simulator time will in normal cases be adjusted to be a multiple of the quanta, but as quantas are dynamically adjusted when executing synchronised events, and when the last quanta is executed, you cannot rely on the time being a multiple of the quanta.

Result

Current time in nanoseconds

1.1.366. temu_simIsRunning

Include

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

Signature

```
int temu_simIsRunning()
```

Description

Return 0 if sim is not running, non-zero otherwise.

Result

1.1.367. temu_simRunCallback

Include

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

Signature

```
void temu_simRunCallback(void (*)(void *) Func, void * Data)
```

Description

Execute a callback

The function will run the callback in the current thread if the simulator is not running, otherwise it will post it as a thread-safe callback which will be called a bit later during emulator event processing.

Parameters

Table 334. temu_simRunCallback Parameters

Parameter	Type	Description
Func	void (*)(void *)	Function to invoke
Data	void *	Pointer to pass to Func

1.1.368. temu_simRunNanos

Include

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

Signature

```
temu_CpuExitReason temu_simRunNanos(int64_t Nanos, _Bool Detatched)
```

Description

Run the simulator for the given amount of nanoseconds

Result

Exit reason of the last executed CPU. Under normal conditions this is teCER_Normal, but may be other values in-case breakpoints are hit.

Parameters

Table 335. temu_simRunNanos Parameters

Parameter	Type	Description
Nanos	int64_t	Nanoseconds to run the simulator
Detatched	_Bool	Set to true to run in a separate thread.

1.1.369. temu_simSetQuanta

Include

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

Signature

```
void temu_simSetQuanta(int64_t Ns)
```

Description

Set the quanta of the simulator in nanoseconds.

The quanta is used when scheduling processors. It is essentially identical to the machine class quanta property. Effectively, each processor in the system will synchronise with the other processors and ensure that their timers are identical (within a small tolerance, as a CPU will run until the quanta end plus the slack introduced by the last instruction). Note that the quantas are set in nanoseconds as cycles are inappropriate when multiple processors of different clocks frequencies are used. Note that runtime quantas are adapted based on when synchronised events occur, and for the last quanta when running the processor.

Parameters

Table 336. `temu_simSetQuanta` Parameters

Parameter	Type	Description
Ns	int64_t	Nanoseconds of quanta

1.1.370. temu_simStep

Include

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

Signature

```
temu_CpuExitReason temu_simStep(int Cpu, uint64_t Steps)
```

Description

Step the CPU for a number of steps.

The function will step the CPU with the provided CPU number for a number of steps. A step is defined to be equal to one instruction, or one time advancement in case the CPU is in idle mode or halted.

Note that it is not necessarily so that the CPU id in question is still the current CPU when the function returns. Other CPUs may for example hit a breakpoint or enter halt/error-mode.

Result

Exit reason of the last executed CPU.

Parameters

Table 337. temu_simStep Parameters

Parameter	Type	Description
Cpu	int	CPU number to step
Steps	uint64_t	Number of steps to run.

1.1.371. temu_simStop

Include

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

Signature

```
void temu_simStop()
```

Description

Stop the simulator.

Returns after the simulator has stopped.

1.1.372. 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 338. temu_snapshotGetLength Parameters

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

1.1.373. 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 339. `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)

1.1.374. 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 340. `temu_sparcGetAsr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Reg	unsigned int	ASR register ID

1.1.375. 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 341. `temu_sparcGetNPc` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

1.1.376. 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 342. `temu_sparcGetPsr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

1.1.377. 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 343. `temu_sparcGetTbr` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

1.1.378. 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 344. temu_sparcGetWim Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

1.1.379. 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 345. temu_sparcGetWindowCount Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

1.1.380. 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 346. 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

1.1.381. 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 347. temu_sparcGetY Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer

1.1.382. 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 348. temu_sparcSetAsr Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Reg	unsigned int	ASR register ID
Value	uint64_t	Value to write

1.1.383. 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 349. `temu_sparcSetAsrReader` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Asr	unsigned int	ASR register ID
Handler	temu_SparcAsrHandler	Function to call.

1.1.384. 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 350. `temu_sparcSetAsrWriter` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Asr	unsigned int	ASR register ID

Parameter	Type	Description
Handler	temu_SparcAsrHandler	Function to call.

1.1.385. 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 351. `temu_sparcSetNPc` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint32_t	Value to write to NPC

1.1.386. 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 352. temu_sparcSetPsr Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint32_t	Value to write

1.1.387. 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 353. temu_sparcSetTbr Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer,
Value	uint32_t	Value for TBR register

1.1.388. 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 354. temu_sparcSetWim Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint32_t	value to set in WIM

1.1.389. 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 355. 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
Value	uint32_t	Register value

1.1.390. 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 356. `temu_sparcSetY` Parameters

Parameter	Type	Description
Cpu	void *	SPARC CPU pointer
Value	uint64_t	Value to write to the Y register

1.1.391. temu_spwConnect

Include

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

Signature

```
void temu_spwConnect(temu_SpwPortIfaceRef Dev1PortIface, temu_SpwPortIfaceRef
Dev2PortIface)
```

Description

Connect two SpaceWire ports.

1.1.392. temu_spwDisconnect

Include

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

Signature

```
void temu_spwDisconnect(temu_SpwPortIfaceRef Dev1PortIface, temu_SpwPortIfaceRef
Dev2PortIface)
```

Description

Disconnect two SpaceWire ports.

1.1.393. 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

1.1.394. temu_spwLsmInit

Include

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

Signature

```
void temu_spwLsmInit(temu_SpwLinkState * StatePtr)
```

Description

Initialize the link state.

1.1.395. 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

1.1.396. 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

1.1.397. 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

1.1.398. 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

1.1.399. 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

1.1.400. 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

1.1.401. 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

1.1.402. 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

1.1.403. 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

1.1.404. 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 357. `temu_subscribeNotification` 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.
Arg	void *	Context argument to be passed to the callback notification function
NotFunc	temu_NotificationHandler	The callback function on notification

1.1.405. 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 358. `temu_swap16` Parameters

Parameter	Type	Description
HWord	uint16_t	The word to be swapped

1.1.406. 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 359. `temu_swap32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

1.1.407. 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 360. `temu_swap32Half` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

1.1.408. 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 361. `temu_swap64` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

1.1.409. 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 362. `temu_swap64Word` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

1.1.410. 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 363. temu_swapBigHost16 Parameters

Parameter	Type	Description
HWord	uint16_t	The word to be swapped

1.1.411. 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 364. temu_swapBigHost32 Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

1.1.412. 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 365. `temu_swapBigHost32Half` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

1.1.413. 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 366. `temu_swapBigHost64` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

1.1.414. 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 367. `temu_swapBigHost64Word` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

1.1.415. 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 368. `temu_swapLittleHost16` Parameters

Parameter	Type	Description
HWord	uint16_t	The word to be swapped

1.1.416. 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 369. `temu_swapLittleHost32` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

1.1.417. 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 370. `temu_swapLittleHost32Half` Parameters

Parameter	Type	Description
Word	uint32_t	The word to be swapped

1.1.418. 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 371. `temu_swapLittleHost64` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

1.1.419. 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 372. `temu_swapLittleHost64Word` Parameters

Parameter	Type	Description
DWord	uint64_t	The word to be swapped

1.1.420. temu_symtabGetFuncName

Include

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

Signature

```
void temu_symtabGetFuncName(temu_Symtab * 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 373. `temu_symtabGetFuncName` Parameters

Parameter	Type	Description
Sym	temu_Symtab *	Symboltable

Parameter	Type	Description
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.

1.1.421. temu_symtabGetGlobalFuncRange

Include

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

Signature

```
int temu_symtabGetGlobalFuncRange(temu_Symtab * 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 falures.

Parameters

Table 374. `temu_symtabGetGlobalFuncRange` Parameters

Parameter	Type	Description
Sym	temu_Symtab *	The Symbol table, from which the function is to be retrieved
FuncName	const char *	The function, whose range is required
Addr	uint64_t *	On success, gives the address of the function

Parameter	Type	Description
Size	uint64_t *	On success, gives the size of the function start from Addr

1.1.422. temu_symtabGetGlobalObjRange

Include

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

Signature

```
int temu_symtabGetGlobalObjRange(temu_Symtab * 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 respecivelly.

Result

0 on success (the object exists), other values indicate falures

Parameters

Table 375. `temu_symtabGetGlobalObjRange` Parameters

Parameter	Type	Description
Sym	temu_Symtab *	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

1.1.423. temu_symtabGetLocalFuncRange

Include

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

Signature

```
int temu_symtabGetLocalFuncRange(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 falures

Parameters

Table 376. `temu_symtabGetLocalFuncRange` Parameters

Parameter	Type	Description
Sym	temu_Symtab *	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
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

1.1.424. temu_symtabGetLocalObjRange

Include

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

Signature

```
int temu_symtabGetLocalObjRange(temu_Symtab * 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 respecivelly.

Result

0 on success (the object exists), other values indicate falures

Parameters

Table 377. temu_symtabGetLocalObjRange Parameters

Parameter	Type	Description
Sym	temu_Symtab *	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

1.1.425. temu_symtabGetObjName

Include

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

Signature

```
void temu_symtabGetObjName(temu_Symtab * 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 378. `temu_symtabGetObjName` Parameters

Parameter	Type	Description
Sym	temu_Symtab *	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.

1.1.426. temu_timeGetCurrentSrtNanos

Include

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

Signature

```
uint64_t temu_timeGetCurrentSrtNanos(void * Obj)
```

Description

Get the current simulated real-time in nanoseconds

Result

simulated time for the given object

Parameters

Table 379. `temu_timeGetCurrentSrtNanos` Parameters

Parameter	Type	Description
Obj	void *	The object in question

1.1.427. 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.

1.1.428. 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

1.1.429. 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 380. `temu_typeToName` Parameters

Parameter	Type	Description
Typ	temu_Type	The type, whose name is required

1.1.430. 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 381. `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

1.1.431. 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 382. `temu_unsubscribeNotification` 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.

Parameter	Type	Description
NotFunc	temu_NotificationHandler	The callback function to be called on notification

1.1.432. 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 383. `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

1.1.433. temu_vecCreate

Include

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

Signature

```
temu_Vector temu_vecCreate(temu_Type Typ)
```

Description

temu_vecCreate Create a vector object

Result

the vector object created

Parameters

Table 384. `temu_vecCreate` Parameters

Parameter	Type	Description
Typ	temu_Type	The type of the elements

1.1.434. temu_vecDispose

Include

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

Signature

```
void temu_vecDispose(temu_Vector * Vec)
```

Description

temu_vecDispose Delete a vector object

Parameters

Table 385. `temu_vecDispose` Parameters

Parameter	Type	Description
Vec	temu_Vector *	The vector object to be deleted

1.1.435. temu_vecGetData

Include

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

Signature

```
void * temu_vecGetData(temu_Vector * Vec)
```

Description

temu_vecGetData Retrieve the array of elements of a vector

Result

Returns a pointer to the first element of the vector

Parameters

Table 386. `temu_vecGetData` Parameters

Parameter	Type	Description
Vec	temu_Vector *	The vector, whose elements are requested

1.1.436. temu_vecGetSize

Include

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

Signature

```
size_t temu_vecGetSize(temu_Vector * Vec)
```

Description

temu_vecGetSize Get the number of elements of a vector

Result

The number of elements in Vec

Parameters

Table 387. `temu_vecGetSize` Parameters

Parameter	Type	Description
Vec	temu_Vector *	The vector, whose size to be retrieved

1.1.437. temu_vecPush

Include

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

Signature

```
void temu_vecPush(temu_Vector * Vec, temu_Propval Val)
```

Description

temu_vecPush Add an element to a vector object

Parameters

Table 388. temu_vecPush Parameters

Parameter	Type	Description
Vec	temu_Vector *	The vector, to which the element to be added
Val	temu_Propval	The element to be added

1.1.438. 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 389. `temu_writeFieldValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	The object that contains the register
RegName	const char *	Register name
RegIdx	unsigned int	Register index
FieldName	const char *	Field name
Value	uint64_t	The value to be set

1.1.439. temu_writePCIeConfigRegister

Include

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

Signature

```
void temu_writePCIeConfigRegister(temu_PCIExpressConfig * devConf, uint32_t offset,
uint32_t value)
```

Description

Write value to PCIe config to the register with given offset.

Parameters

Table 390. `temu_writePCIeConfigRegister` Parameters

Parameter	Type	Description
devConf	temu_PCIExpressConfig *	PCIe configuration
offset	uint32_t	register offset in memory map.
value	uint32_t	new value fot the register

1.1.440. 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 391. `temu_writeValue` Parameters

Parameter	Type	Description
Obj	temu_Object *	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.

1.1.441. 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 392. `temu_writeValueI16` Parameters

Parameter	Type	Description
Obj	temu_Object *	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.

1.1.442. 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 393. temu_writeValueI32 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.

1.1.443. 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 394. `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.

1.1.444. 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 395. `temu_writeValueI8` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.

Parameter	Type	Description
Val	int8_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.

1.1.445. 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 396. `temu_writeValueU16` Parameters

Parameter	Type	Description
Obj	temu_Object *	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.

1.1.446. 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 397. `temu_writeValueU32` 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.

1.1.447. 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 398. `temu_writeValueU64` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
Val	uint64_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.

1.1.448. 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 399. `temu_writeValueU8` Parameters

Parameter	Type	Description
Obj	temu_Object *	Object pointer
PropName	const char *	Name of property to read.
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.

1.1.449. 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 400. `xemu_getBranchCounter` Parameters

Parameter	Type	Description
src	uint64_t	Source physical address
tgt	uint64_t	Target physical address

1.1.450. 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 401. `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_ARMCpuIfaceRefArray

Include

```
#include "temu-c/Target/ARM.h"
```

Type

```
struct temu_ARMCpuIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.5. temu_AhbIfaceRef

Include

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

Type

```
struct temu_AhbIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.6. temu_AhbIfaceRefArray

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_AnalogIfaceRef

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_ApbIfaceRefArray

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_BTIfaceRef

Include

```
#include "temu-c/EmulatorManager/BinaryTranslation.h"
```

Type

```
struct temu_BTIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.14. temu_BTIfaceRefArray

Include

```
#include "temu-c/EmulatorManager/BinaryTranslation.h"
```

Type

```
struct temu_BTIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.15. temu_Buff

Include

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

Type

```
struct temu_Buff {
}
```

Description

Fields

Field	Type	Description

1.2.16. temu_CacheCtrlIfaceRef

Include

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

Type

```
struct temu_CacheCtrlIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.17. temu_CacheCtrlIfaceRefArray

Include

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

Type

```
struct temu_CacheCtrlIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.18. temu_CacheIfaceRef

Include

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

Type

```
struct temu_CacheIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.19. temu_CacheIfaceRefArray

Include

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

Type

```
struct temu_CacheIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.20. temu_CanBusIfaceRef

Include

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

Type

```
struct temu_CanBusIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.21. temu_CanBusIfaceRefArray

Include

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

Type

```
struct temu_CanBusIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.22. temu_CanBusStats

Include

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

Type

```
struct temu_CanBusStats {
}
```

Description

Fields

Field	Type	Description

1.2.23. temu_CanDevIfaceRef

Include

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

Type

```
struct temu_CanDevIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.24. temu_CanDevIfaceRefArray

Include

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

Type

```
struct temu_CanDevIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.25. temu_CanFrame

Include

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

Type

```
struct temu_CanFrame {
}
```

Description

Fields

Field	Type	Description

1.2.26. 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.

1.2.27. temu_ClockIfaceRef

Include

```
#include "temu-c/Models/Clock.h"
```

Type

```
struct temu_ClockIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.28. temu_ClockIfaceRefArray

Include

```
#include "temu-c/Models/Clock.h"
```

Type

```
struct temu_ClockIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.29. temu_ClockVTable

Include

```
#include "temu-c/Models/Clock.h"
```

Type

```
struct temu_ClockVTable {
}
```

Description

Fields

Field	Type	Description

1.2.30. temu_CmdArg

Include

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

Type

```
struct temu_CmdArg {
}
```

Description

Fields

Field	Type	Description

1.2.31. temu_CpuIfaceRef

Include

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

Type

```
struct temu_CpuIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.32. temu_CpuIfaceRefArray

Include

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

Type

```
struct temu_CpuIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.33. temu_CpuInfo

Include

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

Type

```
struct temu_CpuInfo {
}
```

Description

Fields

Field	Type	Description

1.2.34. temu_CpuVTable

Include

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

Type

```
struct temu_CpuVTable {
}
```

Description

Fields

Field	Type	Description

1.2.35. 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

1.2.36. temu_DeviceIdIfaceRef

Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

Type

```
struct temu_DeviceIdIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.37. temu_DeviceIdIfaceRefArray

Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

Type

```
struct temu_DeviceIdIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.38. temu_DeviceIdMemAccessIfaceRef

Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

Type

```
struct temu_DeviceIdMemAccessIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.39. temu_DeviceIdMemAccessIfaceRefArray

Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

Type

```
struct temu_DeviceIdMemAccessIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.40. temu_DeviceIfaceRef

Include

```
#include "temu-c/Models/Device.h"
```

Type

```
struct temu_DeviceIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.41. temu_DeviceIfaceRefArray

Include

```
#include "temu-c/Models/Device.h"
```

Type

```
struct temu_DeviceIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.42. temu_DynCallIfaceRef

Include

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

Type

```
struct temu_DynCallIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.43. temu_DynCallIfaceRefArray

Include

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

Type

```
struct temu_DynCallIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.44. temu_DynamicResetAddressIfaceRef

Include

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

Type

```
struct temu_DynamicResetAddressIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.45. temu_DynamicResetAddressIfaceRefArray

Include

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

Type

```
struct temu_DynamicResetAddressIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.46. 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 aa]
Sfd	uint8_t	Start frame delimiter, normally 0xab

1.2.47. temu_EthernetIfaceRef

Include

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

Type

```
struct temu_EthernetIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.48. temu_EthernetIfaceRefArray

Include

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

Type

```
struct temu_EthernetIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.49. temu_Event

Include

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

Type

```
struct temu_Event {
}
```

Description

Fields

Field	Type	Description

1.2.50. temu_EventIfaceRef

Include

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

Type

```
struct temu_EventIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.51. temu_EventIfaceRefArray

Include

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

Type

```
struct temu_EventIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.52. temu_ExtIRInstruction

Include

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

Type

```
struct temu_ExtIRInstruction {
}
```

Description

Fields

Field	Type	Description

1.2.53. temu_FieldInfo

Include

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

Type

```
struct temu_FieldInfo {
}
```

Description

Fields

Field	Type	Description

1.2.54. temu_GpioBusIfaceRef

Include

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

Type

```
struct temu_GpioBusIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.55. temu_GpioBusIfaceRefArray

Include

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

Type

```
struct temu_GpioBusIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.56. temu_GpioClientIfaceRef

Include

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

Type

```
struct temu_GpioClientIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.57. temu_GpioClientIfaceRefArray

Include

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

Type

```
struct temu_GpioClientIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.58. temu_IRInstruction

Include

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

Type

```
struct temu_IRInstruction {
}
```

Description

Fields

Field	Type	Description

1.2.59. 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	temu_Object *	Object pointer (first field of interface reference)

Field	Type	Description
Iface	void *	Type erased interface pointer

1.2.60. temu_IfaceRefArray

Include

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

Type

```
struct temu_IfaceRefArray {
}
```

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

1.2.61. temu_InstrumenterIfaceRef

Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

Type

```
struct temu_InstrumenterIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.62. temu_InstrumenterIfaceRefArray

Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

Type

```
struct temu_InstrumenterIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.63. temu_IrqClientIfaceRef

Include

```
#include "temu-c/Models/IrqController.h"
```

Type

```
struct temu_IrqClientIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.64. temu_IrqClientIfaceRefArray

Include

```
#include "temu-c/Models/IrqController.h"
```

Type

```
struct temu_IrqClientIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.65. temu_IrqCtrlIfaceRef

Include

```
#include "temu-c/Models/IrqController.h"
```

Type

```
struct temu_IrqCtrlIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.66. temu_IrqCtrlIfaceRefArray

Include

```
#include "temu-c/Models/IrqController.h"
```

Type

```
struct temu_IrqCtrlIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.67. temu_LineDataLoggerIfaceRef

Include

```
#include "temu-c/Models/LineDataLogger.h"
```

Type

```
struct temu_LineDataLoggerIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.68. temu_LineDataLoggerIfaceRefArray

Include

```
#include "temu-c/Models/LineDataLogger.h"
```

Type

```
struct temu_LineDataLoggerIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.69. temu_List

Include

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

Type

```
struct temu_List {
}
```

Description

Fields

Field	Type	Description

1.2.70. temu_MACIfaceRef

Include

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

Type

```
struct temu_MACIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.71. temu_MACIfaceRefArray

Include

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

Type

```
struct temu_MACIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.72. temu_MDIOIfaceRef

Include

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

Type

```
struct temu_MDIOIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.73. temu_MDIOIfaceRefArray

Include

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

Type

```
struct temu_MDIOIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.74. temu_MachineIfaceRef

Include

```
#include "temu-c/Models/Machine.h"
```

Type

```
struct temu_MachineIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.75. temu_MachineIfaceRefArray

Include

```
#include "temu-c/Models/Machine.h"
```

Type

```
struct temu_MachineIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.76. temu_MachineVTable

Include

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

Type

```
struct temu_MachineVTable {
}
```

Description

Fields

Field	Type	Description

1.2.77. temu_MemAccessCapabilities

Include

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

Type

```
struct temu_MemAccessCapabilities {
  uint16_t TransactionBaseSizes;
  uint16_t LargeTransactions;
}
```

Description

Fields

Field	Type	Description
TransactionBaseSizes	uint16_t	Flags indicating legal transaction sizes
LargeTransactions	uint16_t	Supports large transactions (e.g. RAM/ROM)

1.2.78. temu_MemAccessIfaceRef

Include

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

Type

```
struct temu_MemAccessIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.79. temu_MemAccessIfaceRefArray

Include

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

Type

```
struct temu_MemAccessIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.80. 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

Field	Type	Description
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	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. 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. The memory space must thus have a way of determining which type of transaction is legal. The MemoryAccessIface has been extended with a

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 initator. This field is new in TEMU 2.2. The field was introduced to support the implementation of I/O MMUs.
Initiator	temu_Object *	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)

1.2.81. temu_MemVTable

Include

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

Type

```
struct temu_MemVTable {
}
```

Description

Fields

Field	Type	Description

1.2.82. temu_MemoryIfaceRef

Include

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

Type

```
struct temu_MemoryIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.83. temu_MemoryIfaceRefArray

Include

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

Type

```
struct temu_MemoryIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.84. temu_MemorySpaceIfaceRef

Include

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

Type

```
struct temu_MemorySpaceIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.85. temu_MemorySpaceIfaceRefArray

Include

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

Type

```
struct temu_MemorySpaceIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.86. temu_Mil1553BusIdleInfo

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553BusIdleInfo {
}
```

Description

Fields

Field	Type	Description

1.2.87. temu_Mil1553BusIfaceRef

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553BusIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.88. temu_Mil1553BusIfaceRefArray

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553BusIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.89. temu_Mil1553DevIfaceRef

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553DevIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.90. temu_Mil1553DevIfaceRefArray

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553DevIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.91. temu_Mil1553Msg

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553Msg {
}
```

Description

Fields

Field	Type	Description

1.2.92. temu_Mil1553Stats

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553Stats {
}
```

Description

Fields

Field	Type	Description

1.2.93. temu_ModeSwitchInfo

Include

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

Type

```
struct temu_ModeSwitchInfo {
}
```

Description

Fields

Field	Type	Description

1.2.94. temu_ModelRegInfo

Include

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

Type

```
struct temu_ModelRegInfo {
}
```

Description

Fields

Field	Type	Description

1.2.95. temu_Object

Include

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

Type

```
struct temu_Object {
  temu_Class * Class;
  char * Name;
  struct temu_Object * TimeSource;
  temu_Component * Component;
  uint64_t LoggingFlags;
  int64_t WillDisposeNotification;
  int64_t DisposedNotification;
  void * UserData;
  uint64_t IsClassObject;
  uint64_t IsCheckpointable;
}
```

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	temu_Class *	Class pointer
Name	char *	Object name
TimeSource	struct temu_Object *	Timesource object
Component	temu_Component *	Parent component (null for root comp)
LoggingFlags	uint64_t	Log category enabled/disabled
WillDisposeNotification	int64_t	Notification emitted before object is deleted
DisposedNotification	int64_t	Notification emitted after object has been deleted
UserData	void *	User data pointer. This is not saved in snapshots.
IsClassObject	uint64_t	The object is a class object
IsCheckpointable	uint64_t	The object is snapshottable

1.2.96. temu_ObjectIfaceRef

Include

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

Type

```
struct temu_ObjectIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.97. temu_ObjectIfaceRefArray

Include

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

Type

```
struct temu_ObjectIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.98. temu_PCIBridgeIfaceRef

Include

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

Type

```
struct temu_PCIBridgeIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.99. temu_PCIBridgeIfaceRefArray

Include

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

Type

```
struct temu_PCIBridgeIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.100. temu_PCIBusIfaceRef

Include

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

Type

```
struct temu_PCIBusIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.101. temu_PCIBusIfaceRefArray

Include

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

Type

```
struct temu_PCIBusIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.102. temu_PCIConfig

Include

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

Type

```
struct temu_PCIConfig {
}
```

Description

Fields

Field	Type	Description

1.2.103. temu_PCIDevice

Include

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

Type

```
struct temu_PCIDevice {
}
```

Description

Fields

Field	Type	Description

1.2.104. temu_PCIDeviceIfaceRef

Include

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

Type

```
struct temu_PCIDeviceIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.105. temu_PCIDeviceIfaceRefArray

Include

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

Type

```
struct temu_PCIDeviceIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.106. temu_PCIDeviceVTable

Include

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

Type

```
struct temu_PCIDeviceVTable {
}
```

Description

Fields

Field	Type	Description

1.2.107. temu_PCIExpressBridge

Include

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

Type

```
struct temu_PCIExpressBridge {
}
```

Description

Fields

Field	Type	Description

1.2.108. temu_PCIExpressBridgeIfaceRef

Include

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

Type

```
struct temu_PCIExpressBridgeIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.109. temu_PCIExpressBridgeIfaceRefArray

Include

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

Type

```
struct temu_PCIExpressBridgeIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.110. temu_PCIExpressBus

Include

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

Type

```
struct temu_PCIExpressBus {
}
```

Description

Fields

Field	Type	Description

1.2.111. temu_PCIExpressBusIfaceRef

Include

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

Type

```
struct temu_PCIExpressBusIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.112. temu_PCIExpressBusIfaceRefArray

Include

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

Type

```
struct temu_PCIExpressBusIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.113. 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

1.2.114. temu_PCIExpressDevice

Include

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

Type

```
struct temu_PCIExpressDevice {
}
```

Description

Fields

Field	Type	Description

1.2.115. temu_PCIExpressDeviceIfaceRef

Include

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

Type

```
struct temu_PCIExpressDeviceIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.116. temu_PCIExpressDeviceIfaceRefArray

Include

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

Type

```
struct temu_PCIExpressDeviceIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.117. temu_PCIeConfigType0

Include

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

Type

```
struct temu_PCIeConfigType0 {
}
```

Description

temu_PCIeConfigType0: Fields that exists only in config type 0

Fields

Field	Type	Description

1.2.118. 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

1.2.119. temu_PCIeControllerInternalCSRs

Include

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

Type

```
struct temu_PCIeControllerInternalCSRs {
}
```

Description

Fields

Field	Type	Description

1.2.120. temu_PCIeDeviceSpecificConfigSpace

Include

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

Type

```
struct temu_PCIeDeviceSpecificConfigSpace {
}
```

Description

Fields

Field	Type	Description

1.2.121. temu_PDCIfaceRef

Include

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

Type

```
struct temu_PDCIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.122. temu_PDCIfaceRefArray

Include

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

Type

```
struct temu_PDCIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.123. temu_PHYIfaceRef

Include

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

Type

```
struct temu_PHYIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.124. temu_PHYIfaceRefArray

Include

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

Type

```
struct temu_PHYIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.125. temu_PowerIfaceRef

Include

```
#include "temu-c/Models/Power.h"
```

Type

```
struct temu_PowerIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.126. temu_PowerIfaceRefArray

Include

```
#include "temu-c/Models/Power.h"
```

Type

```
struct temu_PowerIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.127. temu_PowerLineIfaceRef

Include

```
#include "temu-c/Models/Power.h"
```

Type

```
struct temu_PowerLineIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.128. temu_PowerLineIfaceRefArray

Include

```
#include "temu-c/Models/Power.h"
```

Type

```
struct temu_PowerLineIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.129. temu_PowerPCIfaceRef

Include

```
#include "temu-c/Target/PowerPC.h"
```

Type

```
struct temu_PowerPCIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.130. temu_PowerPCIfaceRefArray

Include

```
#include "temu-c/Target/PowerPC.h"
```

Type

```
struct temu_PowerPCIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.131. 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
Count	size_t	Number of elements in property
Offset	uintptr_t	Offset from struct start

1.2.132. temu_PropName

Include

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

Type

```
struct temu_PropName {
}
```

Description

Fields

Field	Type	Description

1.2.133. temu_Propref

Include

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

Type

```
struct temu_Propref {
}
```

Description

Typed property reference.

The property reference contains a type tag and a raw pointer to the property.

Fields

Field	Type	Description

1.2.134. temu_Propval

Include

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

Type

```
struct temu_Propval {
}
```

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

1.2.135. temu_RegisterBankInfo

Include

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

Type

```
struct temu_RegisterBankInfo {
}
```

Description

Fields

Field	Type	Description

1.2.136. temu_RegisterInfo

Include

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

Type

```
struct temu_RegisterInfo {
}
```

Description

Fields

Field	Type	Description

1.2.137. temu_ResetIfaceRef

Include

```
#include "temu-c/Models/Reset.h"
```

Type

```
struct temu_ResetIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.138. temu_ResetIfaceRefArray

Include

```
#include "temu-c/Models/Reset.h"
```

Type

```
struct temu_ResetIfaceRefArray {
```

```
}
```

Description

Fields

Field	Type	Description

1.2.139. temu_SerialIfaceRef

Include

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

Type

```
struct temu_SerialIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.140. temu_SerialIfaceRefArray

Include

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

Type

```
struct temu_SerialIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.141. temu_SignalIfaceRef

Include

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

Type

```
struct temu_SignalIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.142. temu_SignalIfaceRefArray

Include

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

Type

```
struct temu_SignalIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.143. temu_SparcV8IfaceRef

Include

```
#include "temu-c/Target/Sparc.h"
```

Type

```
struct temu_SparcV8IfaceRef {
```

```
}
```

Description

Fields

Field	Type	Description

1.2.144. temu_SparcV8IfaceRefArray

Include

```
#include "temu-c/Target/Sparc.h"
```

Type

```
struct temu_SparcV8IfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.145. temu_SpiBus

Include

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

Type

```
struct temu_SpiBus {
}
```

Description

Fields

Field	Type	Description

1.2.146. temu_SpiBusIfaceRef

Include

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

Type

```
struct temu_SpiBusIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.147. temu_SpiBusIfaceRefArray

Include

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

Type

```
struct temu_SpiBusIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.148. temu_SpiDevConfig

Include

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

Type

```
struct temu_SpiDevConfig {
```

```
}
```

Description

Fields

Field	Type	Description

1.2.149. temu_SpiMasterDeviceIfaceRef

Include

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

Type

```
struct temu_SpiMasterDeviceIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.150. temu_SpiMasterDeviceIfaceRefArray

Include

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

Type

```
struct temu_SpiMasterDeviceIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.151. temu_SpiSlaveDevice

Include

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

Type

```
struct temu_SpiSlaveDevice {
}
```

Description

Fields

Field	Type	Description

1.2.152. temu_SpiSlaveDeviceIfaceRef

Include

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

Type

```
struct temu_SpiSlaveDeviceIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.153. temu_SpiSlaveDeviceIfaceRefArray

Include

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

Type

```
struct temu_SpiSlaveDeviceIfaceRefArray {
```

```
}
```

Description

Fields

Field	Type	Description

1.2.154. temu_SpwPacket

Include

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

Type

```
struct temu_SpwPacket {
}
```

Description

Fields

Field	Type	Description

1.2.155. temu_SpwPortIfaceRef

Include

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

Type

```
struct temu_SpwPortIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.156. temu_SpwPortIfaceRefArray

Include

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

Type

```
struct temu_SpwPortIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.157. temu_SpwRmapDecodedCmdField

Include

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

Type

```
struct temu_SpwRmapDecodedCmdField {
}
```

Description

Fields

Field	Type	Description

1.2.158. temu_SpwRmapDecodedCommandHeader

Include

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

Type

```
struct temu_SpwRmapDecodedCommandHeader {
```

```
}
```

Description

Fields

Field	Type	Description

1.2.159. 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.
CmdField	temu_SpwRmapDecodedCmdField	The command field, bits [5,4,3,2] in instruction.
RawHeader	temu_SpwRmapRawHeader	Raw header data access.

1.2.160. 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.

1.2.161. temu_SpwRmapDecodedReadReplyHeader

Include

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

Type

```
struct temu_SpwRmapDecodedReadReplyHeader {
}
```

Description

Fields

Field	Type	Description

1.2.162. 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.

1.2.163. temu_SpwRmapDecodedWriteReply

Include

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

Type

```
struct temu_SpwRmapDecodedWriteReply {
}
```

Description

Fields

Field	Type	Description

1.2.164. temu_SpwRmapDecodedWriteReplyHeader

Include

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

Type

```
struct temu_SpwRmapDecodedWriteReplyHeader {
}
```

Description

Fields

Field	Type	Description

1.2.165. temu_SpwRmapRawHeader

Include

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

Type

```
struct temu_SpwRmapRawHeader {
}
```

Description

Fields

Field	Type	Description

1.2.166. temu_SpwRmapReadCmdPacket

Include

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

Type

```
struct temu_SpwRmapReadCmdPacket {
}
```

Description

Fields

Field	Type	Description

1.2.167. 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.

1.2.168. 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.

1.2.169. temu_TargetExecutionIfaceRef

Include

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

Type

```
struct temu_TargetExecutionIfaceRef {
}
```

Description

Fields

Field	Type	Description

1.2.170. temu_TargetExecutionIfaceRefArray

Include

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

Type

```
struct temu_TargetExecutionIfaceRefArray {
}
```

Description

Fields

Field	Type	Description

1.2.171. temu_TrapEventInfo

Include

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

Type

```
struct temu_TrapEventInfo {
  uint64_t PC;
  uint64_t nPC;
}
```

Description

Fields

Field	Type	Description
PC	uint64_t	Program counter when trap occurred
nPC	uint64_t	Only valid for targets with delay slots

1.2.172. temu_Vector

Include

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

Type

```
struct temu_Vector {
}
```

Description

Fields

Field	Type	Description

1.2.173. temu_i16Array

Include

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

Type

```
struct temu_i16Array {
}
```

Description

Fields

Field	Type	Description

1.2.174. temu_i32Array

Include

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

Type

```
struct temu_i32Array {
}
```

Description

Fields

Field	Type	Description

1.2.175. temu_i64Array

Include

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

Type

```
struct temu_i64Array {
}
```

Description

Fields

Field	Type	Description

1.2.176. temu_i8Array

Include

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

Type

```
struct temu_i8Array {
}
```

Description

Fields

Field	Type	Description

1.2.177. temu_objectArray

Include

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

Type

```
struct temu_objectArray {
}
```

Description

Fields

Field	Type	Description

1.2.178. temu_u16Array

Include

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

Type

```
struct temu_u16Array {
}
```

Description

Fields

Field	Type	Description

1.2.179. temu_u32Array

Include

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

Type

```
struct temu_u32Array {
}
```

Description

Fields

Field	Type	Description

1.2.180. temu_u64Array

Include

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

Type

```
struct temu_u64Array {
}
```

Description

Fields

Field	Type	Description

1.2.181. 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 {
}
```

Description

Enumerators

Name	Value	Description

1.3.2. temu_ARMMode

Include

```
#include "temu-c/Target/ARM.h"
```

Type

```
enum temu_ARMMode {
}
```

Description

Enumerators

Name	Value	Description

1.3.3. temu_BTStatID

Include

```
#include "temu-c/EmulatorManager/BinaryTranslation.h"
```

Type

```
enum temu_BTStatID {
}
```

Description

Enumerators

Name	Value	Description

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,
}
```

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

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,
}
```

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

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,
}
```

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)

Name	Value	Description
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)

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
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 {
}
```

Description

Enumerators

Name	Value	Description

1.3.9. temu_InitiatorType

Include

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

Type

```
enum temu_InitiatorType {
}
```

Description

Enumerators

Name	Value	Description

1.3.10. temu_InstructionFlags

Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

Type

```
enum temu_InstructionFlags {
```

```
}
```

Description

Enumerators

Name	Value	Description

1.3.11. 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_Debug = 4,
}
```

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_Debug	4	Debug

1.3.12. 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

1.3.13. temu_MemoryEndianness

Include

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

Type

```
enum temu_MemoryEndianness {
}
```

Description

Enumerators

Name	Value	Description

1.3.14. temu_MemoryKind

Include

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

Type

```
enum temu_MemoryKind {
}
```

Description

Enumerators

Name	Value	Description

1.3.15. temu_Mil1553BusResetType

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
enum temu_Mil1553BusResetType {
}
```

Description

Enumerators

Name	Value	Description

1.3.16. temu_Mil1553Error

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
enum temu_Mil1553Error {
}
```

Description

Enumerators

Name	Value	Description

1.3.17. temu_Mil1553MsgType

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
enum temu_Mil1553MsgType {
}
```

Description

Enumerators

Name	Value	Description

1.3.18. 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

1.3.19. 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

1.3.20. temu_SpwLinkState

Include

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

Type

```
enum temu_SpwLinkState {
}
```

Description

Enumerators

Name	Value	Description

1.3.21. temu_SpwPacketType

Include

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

Type

```
enum temu_SpwPacketType {
}
```

Description

Enumerators

Name	Value	Description

1.3.22. temu_SpwRmapCommandType

Include

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

Type

```
enum temu_SpwRmapCommandType {
}
```

Description

Enumerators

Name	Value	Description

1.3.23. temu_SpwRmapDecodedPacketType

Include

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

Type

```
enum temu_SpwRmapDecodedPacketType {
}
```

Description

Enumerators

Name	Value	Description

1.3.24. temu_SpwRmapDecodingOutcome

Include

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

Type

```
enum temu_SpwRmapDecodingOutcome {
}
```

Description

Enumerators

Name	Value	Description

1.3.25. temu_SpwRmapPacketType

Include

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

Type

```
enum temu_SpwRmapPacketType {
}
```

Description

Enumerators

Name	Value	Description

1.3.26. temu_SyncEvent

Include

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

Type

```
enum temu_SyncEvent {
}
```

Description

Enumerators

Name	Value	Description

1.3.27. 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

Name	Value	Description
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
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_AhbIface

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_ApbIface

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_BTIface

Include

```
#include "temu-c/EmulatorManager/BinaryTranslation.h"
```

Type

```
struct temu_BTIface {
}
```

Description

Fields

Field	Type	Description

1.4.7. temu_CacheCtrlIface

Include

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

Type

```
struct temu_CacheCtrlIface {
}
```

Description

Fields

Field	Type	Description

1.4.8. temu_CacheIface

Include

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

Type

```
struct temu_CacheIface {
}
```

Description

Fields

Field	Type	Description

1.4.9. temu_CanBusIface

Include

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

Type

```
struct temu_CanBusIface {
}
```

Description

Fields

Field	Type	Description

1.4.10. temu_CanDevIface

Include

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

Type

```
struct temu_CanDevIface {
}
```

Description

Fields

Field	Type	Description

1.4.11. temu_ClockIface

Include

```
#include "temu-c/Models/Clock.h"
```

Type

```
struct temu_ClockIface {
}
```

Description

Fields

Field	Type	Description

1.4.12. temu_CpuIface

Include

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

Type

```
struct temu_CpuIface {
  int (*)(void *) wakeUp;
  void (*)(void *) forceEarlyExit;
}
```

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 ~(page size-1). Otherwise, the return is the translated page address and in the case of failure -1.

field raiseTrapNoJmp 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
wakeUp	int (*)(void *)	Wake up CPU if it is idling, return 1 if woken up 0 if no state change occurred

Field	Type	Description
forceEarlyExit	void (*)(void *)	Force early return of core (after event cleanup), unless the core returns for normal reasons, the emulator will return teCER_Early after the current instruction.

1.4.13. temu_DeviceIdIface

Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

Type

```
struct temu_DeviceIdIface {
}
```

Description

Fields

Field	Type	Description

1.4.14. temu_DeviceIdMemAccessIface

Include

```
#include "temu-c/Models/IntegrationSupport.h"
```

Type

```
struct temu_DeviceIdMemAccessIface {
}
```

Description

Fields

Field	Type	Description

1.4.15. temu_DeviceIface

Include

```
#include "temu-c/Models/Device.h"
```

Type

```
struct temu_DeviceIface {
}
```

Description

Fields

Field	Type	Description

1.4.16. temu_DynCallIface

Include

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

Type

```
struct temu_DynCallIface {
}
```

Description

Fields

Field	Type	Description

1.4.17. temu_DynamicResetAddressIface

Include

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

Type

```
struct temu_DynamicResetAddressIface {
}
```

Description

Fields

Field	Type	Description

1.4.18. temu_EthernetIface

Include

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

Type

```
struct temu_EthernetIface {
}
```

Description

Fields

Field	Type	Description

1.4.19. temu_EventIface

Include

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

Type

```
struct temu_EventIface {
}
```

Description

Fields

Field	Type	Description

1.4.20. 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.

1.4.21. 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.

1.4.22. temu_InstrumenterIface

Include

```
#include "temu-c/EmulatorManager/Instrumenter.h"
```

Type

```
struct temu_InstrumenterIface {
}
```

Description

Fields

Field	Type	Description

1.4.23. temu_IrqClientIface

Include

```
#include "temu-c/Models/IrqController.h"
```

Type

```
struct temu_IrqClientIface {
  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
updateInterrupts	void (*)(void *)	Called in case IRQ re-issuing is needed eg. when the IRQ controlling registers have been modified.

1.4.24. temu_IrqControllerIface

Include

```
#include "temu-c/Models/IrqController.h"
```

Type

```
struct temu_IrqControllerIface {
```

```
}
```

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

1.4.25. temu_LineDataLoggerIface

Include

```
#include "temu-c/Models/LineDataLogger.h"
```

Type

```
struct temu_LineDataLoggerIface {
}
```

Description

Fields

Field	Type	Description

1.4.26. temu_MACIface

Include

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

Type

```
struct temu_MACIface {
}
```

Description

Fields

Field	Type	Description

1.4.27. temu_MDIOIface

Include

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

Type

```
struct temu_MDIOIface {
}
```

Description

Fields

Field	Type	Description

1.4.28. temu_MachineIface

Include

```
#include "temu-c/Models/Machine.h"
```

Type

```
struct temu_MachineIface {
}
```

Description

Fields

Field	Type	Description

1.4.29. 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;
  const temu_MemAccessCapabilities *(*)(void *) getCapabilities;
}
```

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.
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

1.4.30. 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 endianess. 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

1.4.31. temu_MemorySpaceIface

Include

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

Type

```
struct temu_MemorySpaceIface {
}
```

Description

Fields

Field	Type	Description

1.4.32. temu_Mil1553BusIface

Include

```
#include "temu-c/Bus/MilStd1553.h"
```

Type

```
struct temu_Mil1553BusIface {
}
```

Description

Fields

Field	Type	Description

1.4.33. 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

Field	Type	Description
receive	void (*)(void *, temu_Mil1553Msg *)	Receive of 1553 message
busEnteredIdle	void (*)(void *, temu_Mil1553BusIdleInfo *)	Notifies the bus controller the bus enters an idle

1.4.34. 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.

Field	Type	Description
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.
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.

1.4.35. temu_PCIBridgeIface

Include

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

Type

```
struct temu_PCIBridgeIface {
}
```

Description

Fields

Field	Type	Description

1.4.36. temu_PCIBusIface

Include

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

Type

```
struct temu_PCIBusIface {
}
```

Description

Fields

Field	Type	Description

1.4.37. temu_PCIDeviceIface

Include

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

Type

```
struct temu_PCIDeviceIface {
}
```

Description

Fields

Field	Type	Description

1.4.38. temu_PCIExpressBridgeIface

Include

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

Type

```
struct temu_PCIExpressBridgeIface {
}
```

Description

Fields

Field	Type	Description

1.4.39. temu_PCIExpressBusIface

Include

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

Type

```
struct temu_PCIExpressBusIface {
}
```

Description

Fields

Field	Type	Description

1.4.40. temu_PCIExpressDeviceIface

Include

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

Type

```
struct temu_PCIExpressDeviceIface {
}
```

Description

Fields

Field	Type	Description

1.4.41. temu_PDCIface

Include

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

Type

```
struct temu_PDCIface {
}
```

Description

Fields

Field	Type	Description

1.4.42. temu_PHYIface

Include

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

Type

```
struct temu_PHYIface {
}
```

Description

Fields

Field	Type	Description

1.4.43. temu_PowerIface

Include

```
#include "temu-c/Models/Power.h"
```

Type

```
struct temu_PowerIface {
}
```

Description

Fields

Field	Type	Description

1.4.44. temu_PowerLineIface

Include

```
#include "temu-c/Models/Power.h"
```

Type

```
struct temu_PowerLineIface {
}
```

Description

Fields

Field	Type	Description

1.4.45. temu_PowerPCIface

Include

```
#include "temu-c/Target/PowerPC.h"
```

Type

```
struct temu_PowerPCIface {
}
```

Description

Interface for PowerPC specific functionality

Fields

Field	Type	Description

1.4.46. temu_ResetIface

Include

```
#include "temu-c/Models/Reset.h"
```

Type

```
struct temu_ResetIface {
}
```

Description

Fields

Field	Type	Description

1.4.47. 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.

1.4.48. temu_SignalIface

Include

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

Type

```
struct temu_SignalIface {
}
```

Description

Digital signal interface

Fields

Field	Type	Description

1.4.49. temu_SparcV8Iface

Include

```
#include "temu-c/Target/Sparc.h"
```

Type

```
struct temu_SparcV8Iface {
}
```

Description

Interface for SPARC specific functionality

Fields

Field	Type	Description

1.4.50. temu_SpiBusIface

Include

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

Type

```
struct temu_SpiBusIface {
}
```

Description

Fields

Field	Type	Description

1.4.51. temu_SpiMasterDeviceIface

Include

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

Type

```
struct temu_SpiMasterDeviceIface {
}
```

Description

Fields

Field	Type	Description

1.4.52. temu_SpiSlaveDeviceIface

Include

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

Type

```
struct temu_SpiSlaveDeviceIface {
}
```

Description

Fields

Field	Type	Description

1.4.53. 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.

1.4.54. temu_TargetExecutionIface

Include

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

Type

```
struct temu_TargetExecutionIface {
}
```

Description

Fields

Field	Type	Description