



OVP VMI View Function Reference

Imperas Software Limited

Imperas Buildings, North Weston,
Thame, Oxfordshire, OX9 2HA, UK
docs@imperas.com



Author:	Imperas Software Limited
Version:	2.0.11
Filename:	OVP_VMI_View_Function_Reference.doc
Project:	OVP VMI View Function Reference
Last Saved:	Monday, 19 January 2015
Keywords:	

Copyright Notice

Copyright © 2015 Imperas Software Limited All rights reserved. This software and documentation contain information that is the property of Imperas Software Limited. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Imperas Software Limited, or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Imperas permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

IMPERAS SOFTWARE LIMITED, AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

1	Introduction.....	4
2	View User Interface	5
2.1	VMVIEWADDVIEWOBJECTNOTIFICATIONCALLBACK	5
2.2	VMVIEWREMOVEVIEWOBJECTNOTIFICATIONCALLBACK.....	5
2.3	VMVIEWADDVIEWEVENTNOTIFICATIONCALLBACK	6
2.4	VMVIEWREMOVEVIEWEVENTNOTIFICATIONCALLBACK	6
2.5	VMVIEWADDVIEWEVENTLISTENER.....	7
2.6	VMVIEWREMOVEVIEWEVENTLISTENER	7
2.7	VMVIEWGETVIEWOBJECTNAME.....	8
2.8	VMVIEWGETVIEWOBJECTDESCRIPTION	8
2.9	VMVIEWGETVIEWOBJECTPATH.....	9
2.10	VMVIEWGETVIEWOBJECTVALUE	10
2.11	VMVIEWGETVIEWOBJECTVALUESTRING	11
2.12	VMVIEWGETVIEWOBJECTPARENT.....	12
2.13	VMVIEWGETVIEWOBJECTCHILD	13
2.14	VMVIEWGETVIEWOBJECTSIBLING.....	13
2.15	VMVIEWGETVIEWEVENTNAME.....	14
2.16	VMVIEWGETVIEWEVENTDESCRIPTION.....	14
2.17	VMVIEWGETVIEWEVENTPATH	15
2.18	VMVIEWGETVIEWEVENTPARENT	15

1 Introduction

This is reference documentation for **version 2.0.11** of the VMI *view* function interface, defined in `ImpPublic/include/host/vmi/vmiView.h`.

The functions in this interface are generally used within native code execution.

Functions in the run time interface have the prefix `vmiview`.

2 View User Interface

This section describes functions to use the view events and objects.

2.1 *vmiviewAddViewObjectNotificationCallback*

Prototype

```
void vmiviewAddViewObjectNotificationCallback(  
    vmiViewObjectNotifyFn    notifyCB,  
    void                    *userData  
);
```

Description

Install a callback to be invoked each time a view object is created or deleted

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
static VMI_VIEW_OBJECT_NOTIFY_FN(objectNotify) {  
    ...  
}  
  
vmiviewAddViewObjectNotificationCallback(objectNotify, (void*) 0x12345678);
```

Notes and Restrictions

None

2.2 *vmiviewRemoveViewObjectNotificationCallback*

Prototype

```
void vmiviewRemoveViewObjectNotificationCallback(  
    vmiViewObjectNotifyFn    notifyCB,  
    void                    *userData  
);
```

Description

Remove callback which is invoked each time a view object is created or deleted

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
vmiviewRemoveViewObjectNotificationCallback(objectNotify, (void*) 0x12345678);
```

Notes and Restrictions

None

2.3 *vmiviewAddViewEventNotificationCallback***Prototype**

```
void vmiviewAddViewEventNotificationCallback(  
    vmiViewEventNotifyFn    notifyCB,  
    void                    *userData  
);
```

Description

Install a callback to be invoked each time a view event is created or deleted

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
static VMI_VIEW_EVENT_NOTIFY_FN(eventNotify) {  
    ...  
}  
  
vmiviewAddViewEventNotificationCallback(eventNotify, (void*) -1);
```

Notes and Restrictions

None

2.4 *vmiviewRemoveViewEventNotificationCallback***Prototype**

```
void vmiviewRemoveViewEventNotificationCallback(  
    vmiViewEventNotifyFn    notifyCB,  
    void                    *userData  
);
```

Description

Remove callback which is invoked each time a view event is created or deleted

Example

```
#include "vmi/vmiview.h"  
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
vmiviewRemoveViewEventNotificationCallback(eventNotify, (void*) -1);
```

Notes and Restrictions

None

2.5 *vmiviewAddViewEventListener***Prototype**

```
void vmiviewAddViewEventListener(  
    vmiViewEventP          event,  
    vmiViewEventListenerFn listenerCB,  
    void                   *userData  
);
```

Description

Install a callback to be invoked each time a particular view event occurs

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
static VMI_VIEW_EVENT_LISTENER_FN(eventTrigger) {  
    ...  
}  
  
vmiviewAddViewEventListener(event, eventTrigger, (void*) -123);
```

Notes and Restrictions

None

2.6 *vmiviewRemoveViewEventListener***Prototype**

```
void vmiviewRemoveViewEventListener(  
    vmiViewEventP          event,  
    vmiViewEventListenerFn listenerCB,  
    void                   *userData  
);
```

Description

Remove callback which is invoked each time a particular view event occurs

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
vmiviewRemoveViewEventListener(event, eventTrigger, (void*) -123);
```

Notes and Restrictions

None

2.7 *vmiviewGetViewObjectName***Prototype**

```
const char *vmiviewGetViewObjectName(vmiViewObjectP object);
```

Description

Return a view object's name (e.g. "registerA")

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

inline static vmiViewObjectP getViewObjectChildByName(
    vmiViewObjectP object,
    const char * name) {

    if(!object){
        return NULL;
    }
    // iterate across children looking for 'name' field
    vmiViewObjectP child;
    for ( child = vmiviewGetViewObjectChild(object) ;
        child ;
        child = vmiviewGetViewObjectSibling(child) ) {

        const char *thisName = vmiviewGetViewObjectName(child);
        if(strcmp(thisName, name) == 0){
            return child;
        }
    }

    return NULL;
}
```

Notes and Restrictions

None

2.8 *vmiviewGetViewObjectDescription***Prototype**

```
const char *vmiviewGetViewObjectDescription(vmiViewObjectP object);
```

Description

Return a view object's description string (or NULL if none available)

Example


```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

const char *desc = vmiviewGetViewObjectDescription(object);
```

Notes and Restrictions

None

2.9 *vmiviewGetViewObjectPath*

Prototype

```
const char *vmiviewGetViewObjectPath(vmiViewObjectP object);
```

Description

Return a view object's full path (e.g. "/platform/peripheral0/registerA")

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

const char *path = vmiviewGetViewObjectPath(object);
```

Notes and Restrictions

None

2.10 *vmiviewGetViewObjectValue*

Prototype

```
vmiValueType vmiviewGetViewObjectValue(  
    vmiViewObjectP    object,  
    void              *buffer,  
    Uns32              *bufferSize  
);
```

Description

Return a view object's value. The caller passes in a buffer and the buffer size in bytes. If the buffer is large enough the value is written to the buffer and the value type is returned, otherwise VMI_VVT_NOSPACE is returned and bufferSize is updated with the size required. VMI_VVT_ERROR is returned if no value is available for an object.

Return Value

The *vmiViewObject* has the following values

Return Value	Description
VMI_VVT_NOSPACE	No value returned. The buffer supplied is not large enough.
VMI_VVT_ERROR	No value returned
VMI_VVT_BOOL	Primitive types
VMI_VVT_SCHAR	Signed character
VMI_VVT_UCHAR	Unsigned character
VMI_VVT_INT8	8-bit signed value
VMI_VVT_UNS8	8-bit unsigned value
VMI_VVT_INT16	16-bit signed value
VMI_VVT_UNS16	16-bit unsigned value
VMI_VVT_INT32	32-bit signed value
VMI_VVT_UNS32	32-bit unsigned value
VMI_VVT_INT64	64-bit signed value
VMI_VVT_UNS64	64-bit unsigned value
VMI_VVT_ADDR	Address type
VMI_VVT_FLT64	64-bit floating point
VMI_VVT_STRING	Single line string
VMI_VVT_MULTILINE	Multi line string

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
static void printValue(vmiViewObjectP object, Uns32 bufferSize, int depth) {  
    const char *path = vmiviewGetViewObjectPath(object);
```

```

Uns32 n = bufferSize;
char buff[n+1];
Uns32 buffSize = n;

buff[n] = (char)0xfe;
vmiViewValueType t = vmiviewGetViewObjectValue(object, buff, &buffSize);
VMI_ASSERT(buff[n] == (char)0xfe, "data after value buffer clobbered");

void *p = buff;

switch (t) {
case VMI_VVT_ERROR:
    vmiPrintf("%s: %s: OBJECT %s NO VALUE\n", TAG, __func__, path);
    break;
case VMI_VVT_NOSPACE:
    VMI_ASSERT(depth == 0,
        "vmiviewGetViewObjectValue() failed with requested buffer size");
    printValue(object, buffSize, depth+1);
    break;
case VMI_VVT_BOOL:
    if(logFile) {
        fprintf(logFile, "%s\t%s\n", path, *(Bool*) p ? "True" : "False");
    } else {
        vmiPrintf("%s: %s: OBJECT %s Uns8 VALUE %s\n", TAG, __func__,
            path,
            *(Bool*) p ? "True" : "False");
    }
    break;
case VMI_VVT_UNUS8:
    if(logFile) {
        fprintf(logFile, "%s\t%u\n", path, *(Uns8*) p);
    } else {
        vmiPrintf("%s: %s: OBJECT %s Uns8 VALUE %u\n", TAG, __func__,
            path, *(Uns8*) p);
    }
    break;
case VMI_VVT_UNUS32:
    if(logFile) {
        fprintf(logFile, "%s\t%u 0x%08x\n", path, *(Uns32*) p, *(Uns32*) p);
    } else {
        vmiPrintf("%s: %s: OBJECT %s Uns32 VALUE %u 0x%08x\n", TAG, __func__,
            path, *(Uns32*) p,
            *(Uns32*) p);
    }
    break;
case VMI_VVT_STRING:
    if(logFile) {
        fprintf(logFile, "%s\t%s\n", path, (char*)p);
    } else {
        vmiPrintf("%s: %s: OBJECT %s STRING VALUE %s\n", TAG, __func__,
            path, (char*)p);
    }
    break;
default:
    VMI_ASSERT(0, "unhandled value type %u\n", t);
    break;
}
}

```

Notes and Restrictions

None

2.11 *vmiviewGetViewObjectValueString*

Prototype

```
const char *vmiviewGetViewObjectValueString(
```

```
    vmiViewObjectP    object,  
    void              *buffer,  
    Uns32              *bufferSize  
);
```

Description

Return a view object's value as a string

The caller passes in a buffer and the buffer size in bytes. If the buffer is large enough the value string is written to the buffer and a pointer to the string is returned, otherwise NULL is returned and bufferSize is updated with the size required. NULL is returned with bufferSize of zero if no value is available for an object.

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"  
#include "vmi/vmiMessage.h"  
  
...  
// Get value string.  
// Try first with a tiny buffer.  
Uns32 n = 2;  
char buff[n+1];  
Uns32 buffSize = n;  
  
buff[n] = (char)0xcd;  
const char *value = vmiviewGetViewObjectValueString(object, buff, &buffSize);  
VMI_ASSERT(buff[n] == (char)0xcd, "data after value string buffer clobbered");  
  
if ( value ) {  
    vmiPrintf("%s: %s: OBJECT %s VALUE STRING %s\n", TAG, __func__, opath, buff);  
} else {  
    if ( buffSize == 0 ) {  
        vmiPrintf("%s: %s: OBJECT %s NO VALUE\n", TAG, __func__, opath);  
    } else {  
        // Too big for small buffer. Try again with one of the requested size.  
        VMI_ASSERT(buffSize < 1024, "requested buffer too big");  
    }  
}  
...
```

Notes and Restrictions

None

2.12 *vmiviewGetViewObjectParent*

Prototype

```
vmiViewObjectP vmiviewGetViewObjectParent(vmiViewObjectP object);
```

Description

Return a view object's parent object

Example

```
#include "vmi/vmiView.h"  
#include "vmi/vmiAttrs.h"  
#include "vmi/vmiTypes.h"
```

```
#include "vmi/vmiMessage.h"

...
// Walk up the object tree, printing any values we find.
vmiViewObjectP object;
for ( object = vmiviewGetViewEventParent(event) ;
      object ;
      object = vmiviewGetViewObjectParent(object) ) {

    const char *opath = vmiviewGetViewObjectPath(object);

    ...

    // Now get the value as a typed entity.
    printValue(object, 1, 0);
}
```

Notes and Restrictions

None

2.13 *vmiviewGetViewObjectChild*

Prototype

```
vmiViewObjectP vmiviewGetViewObjectChild(vmiViewObjectP object);
```

Description

Return a view object's child object

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

...

for(object = vmiviewGetViewObjectChild(object);
    object ;
    object = vmiviewGetViewObjectSibling(object)){

    if(vmiviewGetViewObjectChild(object)){
        // if there is a child iterate
        printAll(object);
    } else {
        // else just print this object
        printValue(object, 4, 0);
    }
}
...
```

Notes and Restrictions

None

2.14 *vmiviewGetViewObjectSibling*

Prototype

```
vmiViewObjectP vmiviewGetViewObjectSibling(vmiViewObjectP object);
```

Description

Return a view object's sibling object

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

...

// iterate across children looking for 'name' field
vmiViewObjectP child;
for ( child = vmiviewGetViewObjectChild(object) ;
      child ;
      child = vmiviewGetViewObjectSibling(child) ) {
    const char *thisName = vmiviewGetViewObjectName(child);
    if(strcmp(thisName, name) == 0){
        return child;
    }
}
...
```

Notes and Restrictions

TBD

2.15 *vmiviewGetViewEventName*

Prototype

```
const char *vmiviewGetViewEventName(vmiViewEventP event);
```

Description

Return a view event's name (e.g. "read")

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

...

const char *name = vmiviewGetViewEventName(event);
```

Notes and Restrictions

None

2.16 *vmiviewGetViewEventDescription*

Prototype

```
const char *vmiviewGetViewEventDescription(vmiViewEventP event);
```

Description

Return a view event's description string (or NULL if none available)

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

const char *desc = vmiviewGetViewEventDescription(event);
```

Notes and Restrictions

None

2.17 *vmiviewGetViewEventPath*

Prototype

```
const char *vmiviewGetViewEventPath(vmiViewEventP event);
```

Description

Return a view event's full path (e.g. "/platform/peripheral0/registerA/read")

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

const char *path = vmiviewGetViewEventPath(event);
```

Notes and Restrictions

None

2.18 *vmiviewGetViewEventParent*

Prototype

```
vmiViewObjectP vmiviewGetViewEventParent(vmiViewEventP event);
```

Description

Return a view event's parent object

Example

```
#include "vmi/vmiView.h"
#include "vmi/vmiAttrs.h"
#include "vmi/vmiTypes.h"
#include "vmi/vmiMessage.h"

vmiViewObjectP pobject = vmiviewGetViewEventParent(event);
```

Notes and Restrictions

None

##