WIND™

VXWORKS 7

REGRESSION TEST SUITE USER'S GUIDE

SR0630

Corporate Headquarters

Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.
Toll free (U.S.A.): +1-800-545-WIND
Telephone: +1-510-748-4100
Facsimile: +1-510-749-2010

For additional contact information, see the Wind River website:

www.windriver.com

For information on how to contact Customer Support, see:

www.windriver.com/support

VxWorks 7

Regression Test Suite User's Guide, SR0630

24 October 2019

Contents

1

About the Regression Test Suite

The Regression Test Suite allows VxWorks® users to validate that their build of VxWorks is operating correctly on their particular hardware and with the board support package (BSP) they are using.

The Regression Test Suite ships to VxWorks 7 customers, and introduces a component **INCLUDE_TM_ALL** to include all the VxWorks 6.x BSP Validation test suite (BSPVTS) in VxWorks7. Test code is shipped with the layer that it tests. The test suite validates the following operations:

- The VxWorks kernel operates correctly in UP and SMP (for multi-core CPUs) mode.
- Kernel mode and RTP mode user applications load and run as expected.
- All required BSP APIs and data structures are correct.
- Correct interoperability with command line VSB and VIP project builds based on the chosen BSP.

Test Modules, Test Cases, and Test Code Files

The *test module* is the **.c** file that includes the test cases. The file naming format is **tm***Name***.c**. An example is **tmPthreadLibTest.c**. Each test module implements some C routines that are the *test case*. The routines take either one or no arguments and have a return value of type **VXTEST_STATUS**. Each test module also implements an **API:tmXXXExec**. This API will execute all test case routines in this test module. An example of a test case routine is:

```
VXTEST_STATUS tmPthreadTest1 (void)
```

The test code files are located in a folder named **vxTest** in the layer being tested. For example, the test code for the kernel layer is under the **kernel** directory:

kernel/

|-- Makefile

|-- cdf

|-- common.vxconfig

|-- configlette

|-- core_kernel.spec

|-- genh

|-- h

|-- layer.vsbl

```
|-- pre_src
|-- src
`-- vxTest /*Test code folder*/
```

Building and Executing Tests

There are four methods for building test modules:

- building and running tests in *embedded mode*, in which test modules are statically linked to the kernel, documented in Building Test Code that is Statically-Linked to the Kernel on page 3.
- building and running tests as a DKM, documented in Building Test Code for DKM and RTP Applications on page 9.
- building and running tests as an RTP, documented in Building Test Code for DKM and RTP Applications on page 9.
- building and running tests as an RTP with shared library support, documented in Building Test Code as an RTP with Shared Library Support on page 11.

To write your own test, see Writing New Test Code Cases on page 13.

2

Building Test Code that is Statically-Linked to the Kernel

The first step in building test code is to create a VSB. To execute a test in "embedded mode", which is code that is statically linked to the kernel, build a VIP based on the VSB.

The steps for this task use fsl_imx6 as an example and build the VSB using the command line tool **wrtool**.

Procedure

1. Create a VxWorks Source Build(VSB) Project.

```
wrtool -data $workspace
prj vsb create -bsp fsl_imx6 -S vsb_fsl_imx6 -force
cd vsb_fsl_imx6
```

2. Enable vxTest building.

   To enable vxTest building, execute these commands, which enable the VSB configuration option, **_WRS_CONFIG_VXTEST_BUILD**.

```
prj vsb add FSL_IMX
prj vsb config -w -add _WRS_CONFIG_VXTEST_BUILD=y
```

3. Build the VSB Project.

```
prj build -j 32
```

```
Test code object can be found in here:
                     kernel objects:
                        $VSB_DIR/krnl/${TOOL}/objvxTest_$(TEST_TARGET)
                        Like: OS_CORE_KERNEL_POSIX testing:
                        vsb_fsl_imx6/krnl/${TOOL}/objvxTest_OS_CORE_KERNEL_POSIX/
                    RTP vxe files:
                        $VSB_DIR/usr/root/$(TOOL)/bin/static  for static RTP
                        $VSB_DIR/usr/root/$(TOOL)/bin/shared for dynamic RTP
```

4. Create a VIP based on the VSB.

```
prj vip create -vsb vsb_fsl_imx6 fsl_imx6 llvm vip_fsl_imx6 -force
cd vip_fsl_imx6
prj vip component list all INCLUDE_TM_OS
```

5. Add the test components to the VIP.

   For example, the following command selects **INCLUDE_TM_OS_CORE_KERNEL_UTIL**, the Kernel Utilities, for testing:

   ```
   prj vip component add INCLUDE_TM_OS_CORE_KERNEL_UTIL
   ```

6. Optionally, enable VxWorks to launch the test modules automatically when it boots.

   To launch the test automatically after booting, add the **INCLUDE_VXTEST_AUTORUN** component.

   ```
   prj vip component add INCLUDE_VXTEST_AUTORUN
   ```

7. Build the VIP.

   ```
   prj build
   ```

8. Run the image.

The following output is displayed; the **vxTest** command takes as arguments, any of the options shown in the output below.

```
 _________           _________
 \77777777\          /77777777/
  \77777777\        /77777777/
   \77777777\      /77777777/
    \77777777\    /77777777/
     \77777777\   \7777777/
      \77777777\   \77777/             VxWorks 7
       \77777777\   \777/
        \77777777\   \7/     Core Kernel version: 1.0.7.1
         \77777777\   -      Build date: Jan  8 2015 14:33:29
          \77777777\
           \7777777/         Copyright Wind River Systems, Inc.
            \77777/   -                1984-2015
             \777/   /7\
              \7/   /777\
               -   -------

             Board: Freescale i.MX6Q Sabre Smart Device Board - ARMv7
    OS Memory Size: 1024MB
  ED&R Policy Mode: Deployed
       Debug Agent: Not started
   Stop Mode Agent: Not started


 Adding 9794 symbols for standalone.

vxTestOptions:           -em -v 4
->
->
-> vxTest "-h"

Usage:


Options:
-em                          Driver runs in embedded mode
-v[erbose] level             Set verbosity level
-l[oops] x                   Loop over the testing x times, -1 means indefinitely
-rtpPath path                RTP test module Folder path
-kerPath path                Kernel test module Folder path
-tc testCase                 Run individual test case
-tm testModule               Run individual test module
-lm                          List all kernel test modules
-testLevel level             Run specified level testing, 0 - all, 1 - sanity, 2 - feature

value = 1 = 0x1
->
-> vxTest "-lm"
Test Module List:
    tmFfsLib
    tmHashLib
    tmHookLib
TOTAL = 3
->
value = 3 = 0x3
-> vxTest "-em -v 4"
Test Options Summary:
---------------------------
Verbosity level : 4
Loops           : 1
moduleName      :
caseName        :
TPRG tmFfsLib.ffsLsbTest1:1:cpu- BEGIN Test with different input values TIMEOUT:9000ms
TRES tmFfsLib.ffsLsbTest1:1:cpu- PASS
TPRG tmFfsLib.ffsLsbTest1:1:cpu- END EXC_TIME = 0 ticks
TPRG tmFfsLib.ffsMsbTest1:1:cpu- BEGIN Test with different input values TIMEOUT:9000ms
TRES tmFfsLib.ffsMsbTest1:1:cpu- PASS
TPRG tmFfsLib.ffsMsbTest1:1:cpu- END EXC_TIME = 0 ticks
....
```

When building test code, keep in mind the following information.

- You can add any **INCLUDE_TM**_*NAME* components for testing. Examples are **INCLUDE_TM_HASHLIB** and **INCLUDE_TM_HOOKLIB**.
- The VxWorks regression test suite in non-embedded mode needs ensure the VIP image is configured with at least one file system. That way DKM and/or RTP object modules can be loaded. For example:

```
vxTest "-kerPath ...VSB_DIR/krnl/${TOOL}/objvxTest_OS_CORE_KERNEL_UTIL/"
```

- To execute all of the test cases in a specific test module, **tm***XXX***Exec** can be run from the shell. For example:

```
tmHookLib module
       -> tmHookLibExec
       TPRG tmHookLib.hookLibTest:100:cpu- BEGIN hookLib test TIMEOUT:5000ms
       TMSG V_GENERAL Testing hookAddToTail() ........................
       TMSG PASS...
```

- To execute a specific test case within a test module, the test case name can be called from the shell. For example:

```
hookLibTest in tmHookLib module
       -> hookLibTest
       TMSG V_GENERAL Testing hookAddToTail() ........................
       TMSG PASS
       TMSG V_GENERAL Testing hookAddToTail() on a full table ..........
       TMSG PASS
```

- For **"-kerPath"** and **"-rtpPath"**, if the path name is a test module file, then **vxTest** executes all of the test cases in the test module. For example:

```
vxTest "-kerPath /romfs/tmXXX.o"
```

```
vxTest "-rtpPath /romfs/tmXXX.vxe"
```

Interpret the test output using the following as an example:

```
TPRG tmKernelLib.kernelCpuEnableTest2:1:cpu- BEGIN try to enable current CPU TIMEOUT:30000ms
TMSG tmKernelLib.kernelCpuEnableTest2:1:cpu0 Will be executed in core 0
TMSG API returned ERROR while trying to enable current CPU (which is already enabled)
TMSG tmKernelLib.kernelCpuEnableTest2:1:cpu1 Will be executed in core 1
TMSG API returned ERROR while trying to enable current CPU (which is already enabled)
TMSG tmKernelLib.kernelCpuEnableTest2:1:cpu2 Will be executed in core 2
TMSG API returned ERROR while trying to enable current CPU (which is already enabled)
TMSG tmKernelLib.kernelCpuEnableTest2:1:cpu3 Will be executed in core 3
TMSG API returned ERROR while trying to enable current CPU (which is already enabled)
TRES tmKernelLib.kernelCpuEnableTest2:1:cpu- PASS
TPRG tmKernelLib.kernelCpuEnableTest2:1:cpu- END EXC_TIME = 6 ticks
```

Message Types

TPRG

Test case progress message.

TMSG

Test case print message.

TRES

Test case result message.

MRES

Test module result summary message.

To further interpret the messages, consider the first line of the output:

```
TPRG tmKernelLib.kernelCpuEnableTest2:1:cpu- BEGIN try to enable current CPU TIMEOUT:30000ms
```

tmKernelLib

Test module name.

kernelCpuEnableTest2

Test case name.

:1

Test run loop.

cpu- BEGIN

Test case progress message.

try to enable current CPU

Test case description.

TIMEOUT:30000ms

Test case timeout value.

On lines where a test is listed as executing in a specific core, you will see:

cpu*N*

The test case will be executed on cpu*N*.

3

Building Test Code for DKM and RTP Applications

Once you have built a VSB project enabled for test code, you can use **wrtool** to build a VIP for test code as either a DKM or RTP application.

Procedure

1. Build the VxWorks Image project based on the VSB project you built with testing enabled.

   To create a VxWorks Image Project (VIP) for user space testing, create the VIP as a real-time process (RTP) application.

   ```
   wrtool -data $workspace
   prj vip create -vsb vsb_fsl_imx6 fsl_imx6 llvm vip_fsl_imx6 -profile PROFILE_DEVELOPMENT -
   force
   cd vip_fsl_imx6
   prj vip component list all INCLUDE_TM_OS
   ```

2. Add the test components to the VIP.

   For example, the following command selects **INCLUDE_TM_OS_CORE_USER_MEM** for memory testing in user space and **INCLUDE_ROMFS**.

   ```
   prj vip component add INCLUDE_TM_OS_CORE_USER_MEM
   prj vip component add INCLUDE_ROMFS
   ```

3. Create a directory for ROMFS files, and copy them to this directory.

   ```
   prj romfs create romfs
   prj romfs add -file $VSB_DIR/usr/root/llvm/bin/static/vxTest_OS_CORE_USER_MEM/tmPoolLib.vxe
   romfs/
   prj romfs add -file $VSB_DIR/usr/root/llvm/bin/static/vxTest_OS_CORE_USER_MEM/tmMemLib.vxe
   romfs/
   prj romfs add -file $VSB_DIR/usr/root/llvm/bin/static/vxTest_OS_CORE_USER_MEM/
   tmMemPartLib.vxe romfs/
   ```

4. Build the image.

   ```
   prj build
   ```

5. Run the image.

```
-> vxTest "-rtpPath /romfs"
```

The following output is displayed:

```
Path: /romfs : Total 3 files
Test Options Summary:
---------------------------
Verbosity level : 4
Loops           : 1
moduleName      :
caseName        :
vxeFile: /romfs/tmPoolLib.vxe
TPRG RTP tmPoolLib.poolCreateTest1:1:cpu- BEGIN create a pool with thread safe option
TIMEOUT:5000ms
TMSG Thread safe memory pool is created successfully
TRES RTP tmPoolLib.poolCreateTest1:1:cpu- PASS
    ....
```

4

Building Test Code as an RTP with Shared Library Support

Test code can be built as an RTP that has shared library support. This example uses the command line tool **wrtool**.

Procedure

1. At the command line, use the **cd** command to change to the appropriate directory.

   To build the **.vxe**, change to the appropriate directory under vxTest, either **user_src** or **share_src**; for example, *installDir*/**vxworks-7/pkgs_v2/os/core-***x.x.x.x*/**kernel/vxTest/user_src/posix**. Do this at the command line using the **cd** command.

2. Run **make** for the appropriate toolchain:

   Since some test cases can be used for both kernel and user space applications, use **"SPACE=user"** to run the Makefile to build a user-space application.

   ```
   make TOOL=llvm VSB_DIR=XXXX SPACE=user EXE_FORMAT=dynamic
   ```

   The dynamic **.vxe** files are created in **$VSB_DIR/usr/root/$TOOL/bin/shared/vxTest_OS_CORE_USER_POSIX**.

3. Locate shared library files in *$VSB_DIR*/**usr/root/***$TOOL*/**bin**.

4. Copy **libvxTestUtils.so.1** and **libc.so.1** to the **romfs** or the **vxe** folder.

5

Writing New Test Code Cases

In addition to using the test cases that ship with VxWorks, you can create your own test cases using a set of templates and common functions that are provided.

Procedure

1. Create a test module file and name it using the format: **tm***myTestCase***.c**.

   The example below uses the name **tm***XXXXX***.c**.

2. Add test cases to the test module file.

A new test case needs to follow the following format for code comments. The code comments provide information on the required test environment for the test case. Below is an example.

```
/*****************************************************************************
*
* taskSpawnTest - check multiple task spawning with different priority
*
* \cs
*
*    <testCase>
*         <timeout>     2000    </timeout>
*         <reentrant>   TRUE    </reentrant>
*         <memCheck>    TRUE    </memCheck>
*         <osMode>      smp     </osMode>
*         <exeMode>     kernel  </exeMode>
*         <destructive> FALSE   </destructive>
*    </testCase>
*
* \ce
*
* \h Test Case:
*       Spawns multiple tasks of different priority.
* \h Verification:
*       Verifies that all the tasks spawned get executed.
*
* INTERNAL
*
* RETURNS:
*   VXTEST_PASS if test passes.
*   VXTEST_FAIL if test fails.
*
*/

VXTEST_STATUS taskSpawnTest (void)
    {
    TASK_ID  taskId1;        /* task Id */
    TASK_ID  taskId2;        /* task Id */
...
    return VXTEST_PASS;
    }
```

<timeout>

Test case timeout in milliseconds.

<reentrant>

Indicates whether or not a test case is task-reentrant: TRUE or FALSE.

<memCheck>

If TRUE, this signals the driver to perform memory leak checks for the test case. Legal values are TRUE or FALSE.

<osMode>

Tag indicates which OS modes this test is valid for. The content can be any one or all of "up" "smp" "amp". The tag content will be compared to the current type of target bsp the kernel is running. If the kernel OS mode and tag content match, the test case will be executed.

<exeMode>

This tag tells the driver that the test case can only be executed in a specific environment. Valid values are "kernel", "rtp", or "all", signifying the environment in which it can be executed. Absence of this tag will be interpreted as "all".

<destructive>

If TRUE, this indicates that the test case can not restore the target to its original condition. The test harness will reboot the target after the test case completes. Legal values are TRUE or FALSE.

3. Create the test case entry table, as in the example below.

   Create a table to record all test cases in the test module. The **vxTestRun( )** function executes the test cases, one by one, by traversing the table entries.

   For single test case, you can simply replace the call in **main( )** with the test case function.

   The test table structure is defined below.

```
vxTestEntry info:
    char *  pTestName;    /* test case name */
    FUNCPTR func;         /* test function entry */
    void *  pArg;         /* test function argument pointer */
    UINT32  flags;        /* flags to determine what to do, use VXTEST_DISABLE_CASE to
disable the test */
    cpuset_t cpuset;      /* bitmap of applicable CPUs for SMP tests, default is 0 */
    UINT32  timeOut;      /* test case timeout in milliseconds, default and minimum is 5000
*/
    UINT8  exeMode;       /* execute in specific mode: RTP, KERNEL, or BOTH */
    UINT8  osMode;        /* UP, SMP, or AMP */
    UINT8  level;         /* test level: 0 = basic/sanity testing level, 1 = core kernel
functionality testing, etc..*/
    char *  description   /* test case description */;
    }VXTEST_ENTRY;
```

   Valid values for **exeMode** are **VXTEST_EXEMODE_KERNEL**, **VXTEST_EXEMODE_RTP**, or **VXTEST_EXEMODE_ALL**, which is the default.

   The test level is passed to the individual **tm***Name***Exec( )** routines, which determine whether any testing should occur based on the specified level. On subsequent invocations (where the level would have been incremented) the same**tm***Name***Exec( )** routine might initiate some testing .

   An example of a test table is:

```
LOCAL VXTEST_ENTRY vxTestTbl_tmXXXXX[] =
{
    /*pTestName, FUNCPTR,  pArg,   flags,   cpuSet,   timeout,   exeMode,   osMode,
level,  description*/
    {"XXXXXTest1", (FUNCPTR)XXXXXTest1, 0, 0, 0, 100, VXTEST_EXEMODE_ALL,
VXTEST_OSMODE_ALL, 0, "test for XXXXX creation"},
    {NULL, (FUNCPTR)"tmXXXXX", 0, 0, 0, 600000, 0, 0, 0}
};
```

4. Complete the test module.
   a) Define the **tmXXXXXExec( )** routine.

The test module file publishes an API (only one) to run all the test cases in the file. This API runs all test entries with the conditions defined in table.

```
/*************************************************************************
*
* tmXXXXXExec - Exec tmXXXXX test module
*
* This routine should be called to execute the test module.
*
* RETURNS: N/A
*
* NOMANUAL
*/

#ifdef _WRS_KERNEL

STATUS tmXXXXXExec
    (
    char * testCaseName,
    VXTEST_RESULT * pTestResult
    )
    {
    return vxTestRun((VXTEST_ENTRY**)&vxTestTbl_tmXXXXX, testCaseName, pTestResult);
    }

#else

STATUS tmXXXXXExec
    (
    char * testCaseName,
    VXTEST_RESULT * pTestResult,
    int    argc,
    char * argv[]
    )
    {
    return vxTestRun((VXTEST_ENTRY**)&vxTestTbl_tmXXXXX, testCaseName, pTestResult,
argc, argv);
    }
```

b) Implement the **main( )** routine, which is used for RTP applications.

```
/*************************************************************************
* main - User application entry function
*
* This routine is the entry point for user application. A real time process
* is created with the first task starting at this entry point.
*
*/
int main
    (
    int    argc,    /* number of arguments */
    char * argv[]   /* array of arguments */
    )
    {
    return tmXXXXXExec(NULL, NULL, argc, argv);
    }
#endif
```

6

Test Suite Configuration Components

When building a VxWorks Image Project (VIP) for the test suite, choose the appropriate components that include the code you want to test. Component representing Regression Suite test modules are all named **INCLUDE_TM_***.

Including Component Groups

You can include entire component groups, for example:

INCLUDE_TM_OS_CORE_KERNEL_MEM
INCLUDE_TM_OS_CORE_KERNEL_WIND
INCLUDE_TM_OS_CORE_KERNEL_POSIX

To see the full list of component groups, type:

```
wrtool -data $workspace
prj vip component list all INCLUDE_TM_OS
```

Including Individual Components

If you do not need all the components in a component group, you can choose to include only some individual components.

To see components in a particular component group, type:

```
prj vip component dtree component
```

For example:

```
$ prj vip component dtree  INCLUDE_TM_OS_CORE_KERNEL_MEM
INCLUDE_TM_OS_CORE_KERNEL_MEM
+ INCLUDE_VXTEST_DRIVER (R)
+ INCLUDE_TM_ADRSPACELIB (I)
+ INCLUDE_TM_HEAPSTRESSTEST (I)
+ INCLUDE_TM_MEMLIB (I)
+ INCLUDE_TM_MEMPARTLIB (I)
+ INCLUDE_TM_PGMGRLIB (I)
+ INCLUDE_TM_POOLLIB (I)
+ INCLUDE_TM_USERRESERVEDMEM (I)
+ INCLUDE_TM_VMCTXTEST (I)
+ INCLUDE_TM_VMTEST (I)

Where (R) REQUIRED, (I) INCLUDE_WHEN, (S) Symbol dependency
Use -all to expand the full tree for this component
```

7

Using wrtool to Build Test Code for Customized DKM and RTP Applications

Once you have built a VSB enabled for test code, you can use **wrtool** to build a VIP for test code as either a DKM or RTP application.

Building a DKM using wrtool

At the command line, run the following commands:

```
wrtool -data $workspace
prj dkm create -vsb $VSB_DIR tmDkmTest
prj build tmDkmTest
prj file delete dkm.c tmDkmTest
prj file add $WIND_BASE/pkgs_v2/os/core-x.x.x.x/kernel/vxTest/share_src/wind/tmTaskInfo.c
tmDkmTest
prj build tmDkmTest
```

Building an RTP using wrtool

At the command line, run the following commands:

```
wrtool -data $workspace
prj rtp create -force -vsb $VSB_DIR tmRtpTest
prj file delete rtp.c tmRtpTest
prj file add $WIND_BASE/pkgs_v2/os/core-x.x.x.x/kernel/vxTest/share_src/wind/tmTaskInfo.c
tmRtpTest
prj file add $WIND_BASE/pkgs_v2/os/core-x.x.x.x/kernel/vxTest/utils/vxTestUtils.c tmRtpTest
prj file add $WIND_BASE/pkgs_v2/os/core-x.x.x.x/kernel/vxTest/utils/vxTestCppUtils.cpp
tmRtpTest
prj build tmRtpTest
```

How to Run the Customized DKM and RTP Applications

Before running the DKM and RTP applications, you should create a VIP with component INCLUDE_VXTEST_DRIVER added.

To run your applications, run the following commands from the kernel shell with the command interpreter.

For DKMs:

```
ld < $WORKSPACE/tmDkmTest/vsb_fsl_imx6_ARMARCH7llvm_SMP/tmDkmTest/Debug/tmDkmTest.out
vxTest "-tm tmTaskInfo"
```

For RTPs:

```
vxTest "-rtpPath $WORKSPACE/tmRtpTest/vsb_fsl_imx6_ARMARCH7llvm/tmRtpTest/Debug/
tmRtpTest.vxe"
```