WIND RIVER

Wind River®
System Viewer

USER'S GUIDE

4.9

Corporate Headquarters
Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

http://www.windriver.com

For information on how to contact Customer Support, please visit the following URL:

http://www.windriver.com/support

Wind River System Viewer User's Guide, 4.9

9 Oct 06
Part #: DOC-15941-ND-00

Contents

1 Overview

1.1 What Is Wind River System Viewer?

Wind River System Viewer is a logic analyzer for embedded software that lets you visualize and troubleshoot complex target activities.

Often the interactions between the operating system, the application, and the target hardware occur within specified time constraints, characterized by resolutions of microseconds or finer.

Commonly used debugging and benchmarking tools for embedded systems, such as source-level debuggers and profilers, provide only static information.

System Viewer logs activities on a running target, whereby the type of data and aspects of a system that you want to view are highly configurable, and can be saved for later analysis.

Wind River System Viewer provides the ability to:

- Detect race conditions, deadlocks, CPU starvation, and other problems relating to task interaction.
- Determine application responsiveness and performance.

- See cyclic patterns in application behavior.
- Save data for deferred analysis.

NOTE: Wind River System Viewer supports the Linux operating system by leveraging the functionality of the Linux Trace ToolKit, more commonly known as LTT. Concepts and features described in other chapters of this guide also pertain to the Linux operating system, unless otherwise mentioned. For Linux architecture, installation and configuration information, see *15. Wind River System Viewer for Linux Standalone 2.4.26 and 2.6.10*.

1.1.1 What Does System Viewer Output Look Like?

If you have never used System Viewer and would like to see what sort of output you can expect, and what you can do with that output, you can open the sample log files provided.

- To open a sample log, on the main Wind River Workbench menu, choose **File > Open** and navigate to *installDir*/**workbench-***N.N*/**wrsv/***N.N*/**samples/vxworks***N*/**logs/**

  Graphical (and tabular) presentations of logged events will be displayed in the **Log Viewer**.

- To help you get an initial understanding of what you are seeing, in the **Log Viewer**, choose **Help > Legend**.

1.2 System Viewer Tools

The main System Viewer tools are:

- **System Viewer Configuration**

  Use this to configure what you want logged and how to upload the log from the target to the host.

- **Triggering**

  Use this to precisely specify when (at which event) to start and stop logging. Most events that are logged can also be used as a trigger.

NOTE: Triggering is not available for the Linux operating system. For information on creating Linux custom events, see *15.3 Linux Custom Events*, p.140.

- **Event Receive**

  A socket listener for collecting log data uploaded by the target.

- **Log Viewer**

  Use this to analyze the logged data. You can also save the logs, with or without the current display settings, in System Viewer format or text format (including CSV for importing to third party spreadsheet applications).

1.2.1 Accessing System Viewer Tools

- You can open existing System Viewer log files in the **Log Viewer** by choosing **File > Open** on the main Wind River Workbench menu.

  System Viewer log files have either a **.wvr** extension, which is a so-called *raw* file with log data only, or a **.wva** extension, which is an *analysis* file that includes information on how the previous viewing session was configured so that you (or somebody else) can pick up from where you left off.

- To use the System Viewer configuration tools (**Configuration** and **Triggering** editors) you must be be connected to a target because these tools read information from the target.

  Once you have succesfully established a connection, you can access these tools from the Wind River Workbench main **Target** menu, or from the **Target Manager** view's right-click context menu under **Target Tools**.

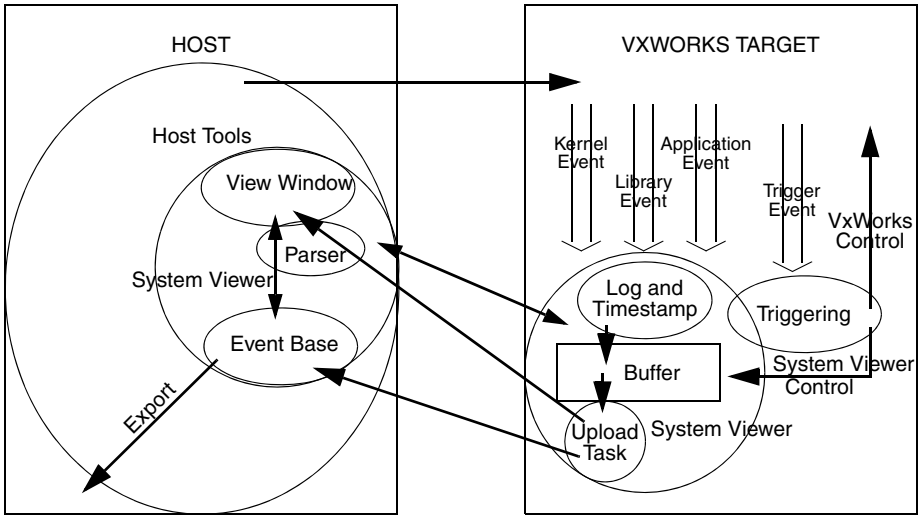
- To open **Event Recieve**, choose **Target > Event Recieve** on the main Workbench menu.

  However, usually there is no need to do this because the **System Viewer Configuration** editor automatically opens the utility when uploading to a socket.

1.3 System Viewer Architectural Overview

As you can see in Figure 1-1 (a VxWorks example) System Viewer provides both host and target side functionality, whereby as much processing as possible done on the host to minimize the effect of logging on target activities.

Figure 1-1 **Wind River System Viewer Architecture**



The host side functionaly is the main subject of the rest of this manual. The target side includes instrumentation points, buffering, event logging, timestamping, data upload, and triggering of actions such as when to start and stop logging. All of these things can be individually configured from the host. For a detailed list of which VxWorks components are needed for specific target side functionality, see

D. Configuring VxWorks for System Viewer. For information relating to Linux, see *15. Wind River System Viewer for Linux Standalone 2.4.26 and 2.6.10*.

Communication between the host and target uses the WDB protocol over a serial line or a network connection. This path is shared by all Wind River Workbench tools to communicate with a target.

2

Preparation and Distribution

2.1 Introduction

This chapter focusses on VxWorks; for information relating to Linux, see *15. Wind River System Viewer for Linux Standalone 2.4.26 and 2.6.10*.

The chapter outlines what you have to consider to be able to configure and generate System Viewer log files to debug real-time problems during the development cycle.

Once you have completed development and debugging, you will want to strip System Viewer components and instrumentation before shipping your product. The necessary procedure for doing this is outlined at the end of the chapter.

2.2 The VxWorks Image Project

Unless you specifically chose to use System Viewer instrumentation-free libraries during creation of your VxWorks Image Project in the Wind River Workbench, you can use System Viewer out of the box.

However, if, at project creation time, you *did* specify that you wanted to use System Viewer instrumentation-free libraries, the VxWorks Image Project is *not* usable for System Viewer.

2.2.1 Kernel Configuration

When you create a VxWorks Image Project in the Wind River Workbench, all commonly used System Viewer components are included in the kernel by default. For reference purposes, or in case the configuration has been modified, all System Viewer kernel components are listed in the appendix chapter, *D. Configuring VxWorks for System Viewer*.

2.3 Host/Target Communication

When you configure a logging session, System Viewer has to read information from the target. Furthermore, many configuration parameters you provide are validated against the target. You therefore must have either a target server, or a connection to the target up and running before you initiate System Viewer log configuration and generation.

System Viewer can only use communication channels that are correctly configured on the target to upload the generated log files to the host. So, if System Viewer reports invalid upload methods, or incorrect upload parameters, you may need to configure the target server/connection properties appropriately.

Configuring target server/connection properties in the Wind River Workbench **Target Manager** is described in the *Wind River Workbench User's Guide*, and will not be re-described in this manual.

2.4 Preparing for Distribution

To use System Viewer, the kernel has to include various System Viewer specific components. Furthermore, in order to be able to capture and log run-time events, *instrumentation points* are coded into the various kernel libraries.

Just as you build an optimized version, devoid of "normal" debug information, of your product, you will probably also want to remove System Viewer components and instrumentation before deployment.

Step 1: Exclude Wind River System Viewer instrumentation

Because instrumentation-free kernel libraries are not supplied with the product, you have to first build the kernel archives from source with the **OPT=-fr** (or **OPT=-inet6_fr**) option. For information on building the VxWorks kernel archives, see the source-code installation and build instructions in your getting started guide.

Step 2: Create a New VxWorks Image Project

Then, in the Wind River Workbench, create a VxWorks Image Project and be sure to select the **Use System Viewer free kernel libraries** in the project creation wizard. This will be the VxWorks image you deploy.

Step 3: Remove All System Viewer Components from the Kernel

Remove all System Viewer components as listed in the appendix chapter, *D. Configuring VxWorks for System Viewer*. The chapter also introduces you to using the **Kernel Editor** for such kernel configuration tasks.

Step 4: Add your application projects to the new VxWorks Image Project and rebuild

For more information in this respect, see the *Wind River Workbench User's Guide*.

3

Configuring a Logging Session

3.1 Configuration Workflow

All basic log configuration is done in the **System Viewer Configuration** editor. Your first step in configuring a logging session is therefore to open this editor, so right-click on a connected target and choose
Target Tools > System Viewer Configuration.

The editor opens with the **Configuration** tab uppermost, associated toolbar buttons appear on the main Workbench toolbar, and the
System Viewer Configuration menu appears on the main Workbench menu.

The order of appearance of the nodes (from top to bottom) on the **Configuration** tab of the **System Viewer Configuration** editor can be seen as a reasonable, although not prescriptive, “configuration workflow”. In practice, you will probably not always follow this workflow, but be aware that there are dependencies that flow from top to bottom.

The main thing to bear in mind when you configure a logging session is that logging is intrusive (itself influences target behavior), and that the more data you collect (the higher the **Event Logging Level**), the more intrusive it gets. Over and

above the volume of data collected, the selected **Upload Mode** and associated buffering also influences intrusiveness.

Each of the nodes (configuration categories) introduced below is individually discussed later. The following is intended merely as an overview that also illustrates a few potential inter-category dependencies.

- The topmost node, **Configuration Summary**, is a read-only summary of the the configuration settings in the other nodes below it. If everything looks correct here, forget about the rest of the nodes and start logging.
- The **Event Logging Level** node—How much information do you want to collect? What do you want log?

  Deciding what you want to log seems a logical enough place start any configuration work.

  Furthermore, because real-time debugging with System Viewer can, like any other kind of debugging, often be a question of "homing in" on the problem (or on different problems in succession), this is also the configuration category you are most likely to want to revisit.

  Modifications to the scope and type of data you want to collect can, in turn, influence your decisions in the next node down, **Upload Mode**.
- The **Upload Mode** node—When do want to upload the log? How will it be buffered on the target until that time?

  If, for example, your application can be expected to generate huge amounts of data at the **Event Logging Level** you selected above, this can influence your decisions on buffer allocation.

  Apart from the expected data volume, the type of problem you are interested in (which you will also have thought about in the context of selecting an **Event Logging Level**) will have an effect on **Upload Mode** settings.

  For example, do you want to see the events immediately preceding (and precipitating) a crash? Or, for example, are you interested in observing long-term target behavior?
- The **Upload Method** node—How do host and target communicate? Where do you want the target to put the logs on the host file system?

  This last node is mostly about aligning host and target communication settings (whereby System Viewer can only use what is available on the target) and where you want the logs to be uploaded to. Once everything is correct on this bottom node, you will normally not need to modify this configuration

category very often, although there can be a dependency on the selected **Upload Mode**, and therefore indirectly also on the **Event Logging Level**.

3.2 Beyond the Basics

Over and above the basic configuration you set **System Viewer Configuration** editor, *triggering* (see *13. Using Triggering*) lets you fine-tune exactly when to start and stop logging.

3.3 General Notes

System Viewer log configuration settings are persistently stored on the host. That is, the target knows nothing about these settings, so if you use a different host to connect to the same target, you have to re-configure for the current host.

4 The Event Logging Level

4.1 What is an Event?

Before deciding what level of events you want to log, it might be worth looking at what System Viewer sees as an *event*.

System Viewer, defines an *event* as any action undertaken by a task or an interrupt service routine (ISR) that can affect the state of a real-time system. The information logged for each event includes

- the action that occurred, such as **semGive( )**
- the context in which the event occurred, that is, the ISR, task, or idle loop
- the timestamp
- other status information as appropriate, such as the semaphore ID for a **semGive**
- parameter details
- calling routines
- and so on. For detailed information, see the *Wind River Workbench User Interface Reference: System Viewer Event Dictionary*.

Examples of events are:

- semaphore gives and takes
- task spawns and deletions
- timer expirations
- interrupts
- message queue sends and receives
- watchdog timer activity
- exceptions
- signal activity
- system calls
- I/O activity
- networking activity
- memory allocation, freeing, partitioning, and so on
- protection domain activity (VxWorks 653 only).

System Viewer provides details about the parameters logged for each library or event, the routines that call them, and so on. For more information, see the *Wind River Workbench User Interface Reference: System Viewer Event Dictionary*.

4.2 What is an Event Logging Level?

An *event logging level* determines the type of events (see *4.1 What is an Event?*, p.15) logged; that is, the breadth and depth of data collection.

In the **Event Logging Level** node of the **System Viewer Configuration** editor there is one main control for selecting logging levels, the **Event Logging Level Selection** list.

The **Event Logging Level Selection** list is ordered from lowest to highest complexity, and therefore intrusiveness. These levels are cumulative; that is, each level is the sum of all lower levels, plus whatever new options the selected level itself provides. The user interface provides descriptions of what kind of events each **Event Logging Level** will capture, so click your way through the levels to see the descriptions.

4.2.1 Which Level Do I Select?

To help you decide which level to select, use your knowledge of:

- your application and its (potential) problem areas
- how your application interacts with the operating system

If this not enough, you might try an iterative approach, especially if you are new to System Viewer:

- Start by collecting at the most simple level, **Context Switch**, then, after you have done the rest of your configuration settings and have generated a log, examine the results in the **Log Viewer**.
- If you do not see what you need, try the next highest level, **Task State Transition**.

  The **Task State Transition** level captures, in addition to the changes in execution context logged by the **Context Switch** level, the events that resulted in such changes.
- If that does not work for you, move up to the highest level, **Additional Instrumentation**.

  The **Additional Instrumentation** level is configurable and allows logging of selected operating system specific event types. Try to narrow down your selection to likely looking candidates for the problem(s) you are interested in. Note, however, that if you do not select *any* libraries, the output will be exactly the same as if you had selected **Task State Transition**.

5 The Upload Mode

5.1 Upload Mode Configuration: General Considerations

When you configure the **Upload Mode**, there are two questions you have to answer:

1. When do you want to upload the collected event log data from the target?

   This depends mainly on what kind of problem you are trying to track down:

   – If you want to see the events leading up to a crash, you have to use a **Post-Mortem Upload** mode.

   – If you want to debug non-fatal problems, you can choose between **Deferred Upload** and **Continuous Upload**.

2. How do you want to configure the target buffer that holds the data until it is uploaded?

   Normally you do not want configure the buffer at all. You can generally just accept the default buffer configurations associated with each **Upload Mode**.

Although manual buffer configuration is rare and "advanced" (you have to know something about the buffering strategies used by each **Upload Mode**), there are cases where you might want/have to modify the default configurations. Such cases range from "advanced tweaking" (for example, to minimize logging intrusiveness) to troubleshooting problems like log-collection stoppages or upload failures.

Since you will not normally need to customize the buffer configuration when you are setting up a logging session, discussion of this subject is confined to the troubleshooting section of this chapter, *5.4 Troubleshooting Upload Modes*, p.28.

5.2 Deferred Upload Or Continuous Upload?

Deferred Upload or **Continuous Upload** is the basic decision you have to make for debugging non-fatal problems.

5.2.1 Deferred Upload

This is the default mode. Data is uploaded when you issue an **Upload** command from the System Viewer user interface.

This mode also provides the option to **Uses circular buffer** on some systems (not VxWorks 653). This is a wrap-around ring of buffers where the oldest data is overwritten with the newest data when the ring is full. However, on systems that do not support this option in **Deferred Upload**, you can take advantage of the equivalent **Post Mortem Upload** feature (see *5.3.1 Using Post Mortem for Non-Fatal Problems*, p.23).

If you use a non-circular buffer, logging stops when the buffer is full. If you use a circular buffer, logging continues until stopped by a trigger, an API call, or on demand.

- **Deferred Upload Limitations**

  The volume of data collected, whether you use a circular buffer or not, is limited to whatever free memory is available on the target.

However, this may in fact not really be a limitation because, for example:

- You are looking at an application that runs its course fairly quickly (and/or you have sufficiently optimized the **Event Logging Level** configuration).
- You have localized the problem to some extent, which means that you do not need to collect data for hours or days on end.

  In such a case you might set triggers to start/stop logging, and you would probably also be able to constrain the volume of data collected via the **Event Logging Level** configuration.
- You have additional off-board memory.

- **Deferred Upload Advantages**
  - Least complex and therefore least problem-prone.
  - Minimal intrusiveness of logging.
  - Minimal configuration overhead.

5.2.2 Continuous Upload

In this mode, data is periodically uploaded from the target as buffers are filled. Logging continues until stopped by a trigger, an API call, or on demand. If the **Upload Method** is configured to **Automatically view the data on upload completion,** uploading takes place as soon as logging is stopped.

- **Continuous Upload Limitations**

  This mode is more complex than **Deferred Upload** in that, over above collecting data and allocating buffers, it has to concurrently upload filled buffers (and potentially free them). This has a number of implications:
  - More complexity may, but by no means necessarily, lead to problems and therefore also some additional configuration overhead.
  - Intrusiveness is higher because events associated with periodic uploading are reflected in the log.
  - Periodic uploading impacts target performance.

- **Continuous Upload Advantages**

  The major advantage, or reason to use, the **Continuous Upload** mode is the huge volume of data you can collect without interruption (the physical limit is the available hard-disk space on the host).

  For long-term, uninterrupted observation and situations where you do not know how to characterize the problem (and therefore cannot set triggers) this is the mode of choice.

5.2.3 Configuration Options: Deferred Upload and Continuous Upload

Both **Deferred Upload** and **Continuous Upload** modes provide the following configuration options:

Buffer size (default = **32Kb**)
: This refers to the size of individual buffers in a dynamic ring of multiple buffers. The smallest possible value (the default of **38Kb**) therefore represents the best fit in terms of how many individual buffers the available target memory can accommodate.
If target memory constraints are not an issue because the expected volume of data is relatively low and/or you have ample free target memory, you can increase the size of individual buffers. Because buffers are dynamically allocated (and in **Continuous Upload** mode also sometimes freed), this will reduce the overhead, and therefore also intrusiveness, of memory (de-)allocation.

Advanced >>
: This button opens options that you do not normally need to look at. These options are relevant only for troubleshooting or advanced tweaking; as such, they are described under *5.4 Troubleshooting Upload Modes*, p.28.

5.3 Post-Mortem Upload Modes

The **Post-Mortem Upload** mode is available on VxWorks 6.1 and higher, as well as on VxWorks 653.

The **Post-Mortem Upload (using pmLib)** mode is not supported on VxWorks 653.

Both these modes serve the same purpose, the difference lies in how non-system memory is defined.

Post-Mortem upload is used primarily to collect events leading up to a system failure. Events are stored in a circular (wrap-around) ring of buffers where the oldest data is overwritten with the newest data when the ring is full.

So even if you collect data for days or weeks, the events immediately preceding the failure will be recorded and stored in the target buffer. After a crash and a warm reboot you can then upload the buffer contents by issuing an **Upload** command from the System Viewer user interface.

This can only work if the buffer is stored in memory that is not overwritten on rebooting, which means you will usually have to first configure system memory accordingly (see *5.3.2 Preparation: Kernel Configuration for Post-Mortem Upload*, p.23) and then align the System Viewer configuration settings to match. Bear in mind that if you reconfigure kernel memory, you will have to rebuild the VxWorks image, as well as the boot loader.

5.3.1 Using Post Mortem for Non-Fatal Problems

You can also use **Post-Mortem Upload** even if you do not expect the target to crash. This is particularly useful on systems (like VxWorks 653) that do not support **Deferred Upload** with the **Use a circular buffer** option. This way, you can take advantage the corresponding circular (wrap-around) buffer feature used by **Post-Mortem**. In this case, you would not need to rebuild the boot loader.

5.3.2 Preparation: Kernel Configuration for Post-Mortem Upload

Because post-mortem mode must preserve the buffer after an application failure, the buffer cannot reside in system memory. This means you will have to configure the kernel accordingly, *unless* one of the following applies:

- You are using a BSP that does not reset system memory on a warm reboot.
- You are using shared or off-board memory.

If neither of the above apply, you will have to configure the kernel, regardless of which post-mortem mode (on systems that provide more than one) you want to use.

Your first step is therefore to open the **Kernel Configuration Editor** (accessing the **Kernel Configuration Editor** and using the **Find** dialog to locate components is described under *D.2 Configuring the Kernel*, p.168).

How you proceed from here, depends on which mode (if your system provides more that one) you want to use. Recall: both will achieve the same objective.

- *Either* follow the steps under *Post-Mortem Upload Kernel Configuration*, p.24
- *Or* follow the steps under *Post Mortem Mode (using pmLib) Kernel Configuration*, p.25

Post-Mortem Upload Kernel Configuration

To configure memory for **Post Mortem Upload** you have to set the following macro values:
LOCAL_MEM_AUTOSIZE = **FALSE**
USER_RESERVED_MEM = *required memory size* (as large as possible)

To do so:

- In the **Kernel Configuration Editor**'s **Find** dialog, start typing **LOCAL_MEM_AUTOSIZE**.

  The macro name should be matched after the first few keystrokes.
- Double-click on the match and, in the **Properties** view, set the **Value** of **LOCAL_MEM_AUTOSIZE** to **FALSE**.
- Open the **Find** dialog again, and start typing **USER_RESERVED_MEM**.

  The macro name should be matched after the first few keystrokes.
- Double-click on the match and, in the **Properties** view, set the **Value** of **USER_RESERVED_MEM** to as high a value as possible.

  For example, to reserve 512 Kb of memory for the post-mortem log buffer, set the value to **0x80000**.

  If you do not know how much memory is available for reservation, use a Target or Host Shell and:

  - To find the top of VxWorks memory, enter **`sysMemTop( )`**
  - To find the top of all local memory, enter **`sysPhysMemTop( )`**

You have now reserved memory for the post-mortem log buffer. Please continue as described under *5.3.3 Rebuild the Kernel and the Boot Loader*, p.25

Post Mortem Mode (using pmLib) Kernel Configuration

To configure memory for the **Post-Mortem Upload (using pmLib)** mode, you have to set the macro:

PM_RESERVED_MEM = *required memory size* (as large as possible)

To do so:

- In the **Kernel Configuration Editor**'s **Find** dialog, start typing **PM_RESERVED_MEM**.

  The macro name should be matched after the first few keystrokes.

- Double-click on the match and, in the **Properties** view, set the **Value** of **PM_RESERVED_MEM** to as high a value as possible.

  For example, to reserve 512 Kb of memory for the post-mortem log buffer, set the value to **0x80000**.

NOTE: The size defined by the **PM_RESERVED_MEM** value is for the whole arena, the amount of memory available for use by System Viewer depends on the amount of memory used by other components in the arena such as **ED&R**.

You have now reserved memory for the post-mortem log buffer using the persistent memory feature. Please continue as described under *5.3.3 Rebuild the Kernel and the Boot Loader*, p.25

5.3.3 Rebuild the Kernel and the Boot Loader

If you have modified the kernel in order to support post-mortem upload, you have to rebuild and reboot it.

Furthermore, unless you are using post-mortem as described under *5.3.1 Using Post Mortem for Non-Fatal Problems*, p.23, you will have to rebuild VxWorks boot loader images. This is necessary to ensure that the boot process does not clear memory reserved for the System Viewer log buffer or the system image, which could lead to changes in memory allocations.

5.3.4 Post-Mortem Upload Mode

Once you have configured the kernel as outlined under *5.3.2 Preparation: Kernel Configuration for Post-Mortem Upload*, p.23, you can select the **Post-Mortem Upload** in the **System Viewer Configuration** editor.

Normally the start and end addresses displayed in the **Post-Mortem Upload Buffer Configuration** pane can be correctly calculated by System Viewer. If not, a warning is displayed. Possible problems include:

- The target does not support user-reserved memory (for example, **VxSim**).
- You have not correctly configured user-reserved memory on the kernel, see *Post-Mortem Upload Kernel Configuration*, p.24.
- The target has ED&R or persistent memory configured (not applicable for VxWorks 653).

  In this case it is possible that the top of **USER_RESERVED_MEMORY** will be allocated and used. This prevents post-mortem from using the full range of **USER_RESERVED_MEMORY**. System Viewer attempts to detect the presence of the default ED&R and persisted memory arenas and automatically adjusts the range available in the **USER_RESERVED_MEMORY** to avoid memory conflicts.

  However, if there are custom PM or EDR arenas defined on the target, System Viewer may not be able to successfully detect them. If this is the case, you have the following options:

  – Since you have already set up such arenas, rather select the **Post-Mortem Upload (using pmLib)** option.

  – Alternatively, you can manually configure the region start and end addresses. The entered values must define an area in **USER_RESERVED_MEMORY** which is *not* used by either the default arenas or by any custom persistent memory arenas.

5.3.5 Post-Mortem Upload (using pmLib)

NOTE: This section does not apply to VxWorks 653.

System Viewer Event logs will be stored in the persisted memory arena. The size of memory reserved is configured automatically, but is limited to the free memory in the selected arena. It is therefore important to create an arena with enough

memory to allow storage of the event log, see *Post Mortem Mode (using pmLib) Kernel Configuration*, p.25.

Once the target has been configured with persisted memory, you can use the **System Viewer Configuration** editor to manage it. There are two configurable parameters:

Region Name
Enter the name of the arena in the edit box and click the **Lookup** button to locate the arena on the target. The default arena name is **pmDefaultArena**.

Region Size
Specify the amount of memory to reserve for System Viewer use, up to the maximum amount of free memory in the arena. This field will be disabled until you specify a valid arena. The field will also be disabled if an existing System Viewer reserved area is detected.

Once you have selected an arena and memory for System Viewer has been reserved, the start and end addresses of the reserved memory will be shown in the **Region Start** and **Region End** fields.

To change the size of a reserved arena, delete and recreate it.

NOTE: Deleting and recreating a reserved area may cause loss of an existing log in the reserved memory due to changes in start and end addresses. It is recommended that you upload existing logs before changing reserved memory.

5.3.6 Post-Mortem Upload: General Notes

- Switching from post mortem mode to another mode or vice-versa does not immediately destroy existing logs, however, as soon as you start collecting logs in the new mode, all previously collected logs are destroyed.
- You can not collect multiple logs using post mortem upload mode.

5.4 Troubleshooting Upload Modes

Although the System Viewer upload mode configuration defaults generally suffice, there may be extreme situations that demand custom configuration. You will probably only realize this during or after an initial logging session (failure); that is, you will have to re-configure.

The kind of problems that could be solved by upload mode (re-)configuration are:

- Logging stops prematurely.
- In **Continuous** mode only:
  - upload fails and/or
  - there is excessive buffer allocation/freeing (known as *thrashing*)

Generally speaking, such extreme situations, where System Viewer defaults do not suffice, arise if too many events are fired too rapidly to be properly handled. To overcome the problem, there are two main areas to look at:

- Constrain the volume of data collected by setting triggers (see *13. Using Triggering*) and/or by refining the **Event Logging Level** if possible (see *4. The Event Logging Level*).
- Modify the target buffer configuration, which is what the rest of this section is about.

5.4.1 The Ring Buffer

Almost all custom (re-)configuration of upload modes relates to target buffer management, so it is worth taking a brief look at the System Viewer buffering strategy.

System Viewer uses a ring of individual buffers for storing event logs on the target. Depending on the upload mode, this ring can use:

- *Dynamic* buffering

  Buffers are allocated as needed, up to a configurable maximum count, or until there is no more target memory available (whichever comes first).

  - Dynamic buffer allocation is supported in **Deferred Upload** and **Continuous Upload** modes.

    In **Continuous Upload** mode, excess buffers are freed as data is read and uploaded from filled buffers.

In both **Deferred Upload** and **Continuous Upload** modes you can suppress dynamic buffering by setting the minimum number of buffers to equal the maximum number of buffers. In this case, all available target memory is pre-allocated for use by System Viewer.

However, be aware that in these modes the buffer's memory requirements are provided by the system memory partition, the general partition from which all **malloc( )** requests are sourced.

- **Post-Mortem Upload** modes are non-dynamic. These modes pre-allocate buffers by subdividing all available user-reserved or persistent memory (depending on the post-mortem mode selected) into a ring of ten individual buffers.

- *Linear* or *Circular* buffering
  - *Linear* buffering—Buffers are allocated and filled as needed, up to a configurable maximum count, or until there is no more target memory available (whichever comes first). At this point logging stops.
    - **Deferred Upload** uses linear buffering if **Use Circular buffer** (on systems that support the option) is not selected. On VxWorks 653 **Deferred Upload** buffering is always linear.
    - **Continuous Upload** uses linear buffering. In the normal course of events, buffers are continuously emptied and uploaded, making them available for new data, so the end of the ring should never be reached. All the more so because upload timing is based on the minimum buffer allocation, and all remaining buffers serve as a reserve for catching any backlog of events generated during extreme peaks of activity.
  - *Circular* buffering—Buffers are allocated and filled as needed, up to a configurable maximum count, or until there is no more target memory available (whichever comes first). At this point the buffer wraps around; that is, the oldest buffer is reused and overwritten. This continues until stopped by a trigger, an API call, or on demand (or a full upload-host disk).
    - **Deferred Upload** uses circular buffering if **Use circular buffer** (on systems that support the option) is selected. This option is not supported on VxWorks 653.
    - **Post-Mortem Upload** modes always use circular buffering. This ensures that the events immediately preceding (precipitating) a system failure are recorded.

5.4.2 Symptoms and Solutions

This section focusses on logging problems where there is a good chance that some upload-mode related re-configuration could provide a solution.

Problem: Logging Stops Prematurely

This is observable in the **Log Manager** (see *7.2 The Configuration Editor Log Manager*, **p.40**).

Although this could potentially happen in any mode, **Continuous Upload** is most likely to be affected due to the additional overhead of concurrently uploading data.

Possible Causes

In all modes this can be because:

- Too many events are generated too quickly for buffers to be allocated in time.
- Not enough memory is allocated for buffering on the target.

Possible Solutions

All modes:

- Increase the buffer allocation task priority. You can do this either in the host or target shell, or in the **Kernel Configuration Editor** (in which case you will need to rebuild and reboot).

  The default buffer allocation task priority is **100**. So you would increase this to some higher integer, say **150**.

  – To increase buffer allocation priority, in the host or target shell, enter:
  `-> wvRBuffMgrPrioritySet(150)`

  – To increase buffer allocation priority in the **Kernel Configuration Editor**, search for **WV_RBUFF_MGR_PRIORITY** and enter a value of, say, **150**.

Post-Mortem modes:

- Allocate more user-reserved or persistent memory (depending on mode used) if possible, see *5.3.2 Preparation: Kernel Configuration for Post-Mortem Upload*, p.23.

Deferred and Continuous modes:

- Change the **Advanced Buffer Configuration** settings (click **Advanced >>**).

  Which (combination) of the following you do depends on (1) what you suspect is causing the problem, and (2) target constraints.

  Always bear in mind that log buffering in these modes uses the same memory partition as system and applications.

  – To make more total memory available for log buffering on the target, increase the **Max. Buffer Count** (default is **10**).

    This assumes there is physically more memory available on the target.

  – To reduce the frequency of buffer allocation, increase the **Buffer Size**.

    Note, however, that this can (probably) reduce total available buffer size (at the same **Max. Buffer Count**) if target memory is constrained because the bigger the individual buffers, the worse the fit is likely to be.

  – To eliminate buffer allocation overhead altogether, set **Min. Buffer Count** to equal **Max. Buffer Count**.

    This pre-allocates all available memory.

Problem: Upload Fails (Continuous Upload Only)

Continuous Upload depends, apart from buffer configuration, on a number of factors:

- The rate at which the target application generates events
- Target memory constraints
- Relative host and target performance
- Network bandwidth
- Upload method

All of the above might want looking at, possible upload mode issues are described below.

Possible Causes and Solutions

- Buffer allocation failed—see *Problem: Logging Stops Prematurely*, p.30.
- The ring buffer is filled to capacity faster than individual buffers are uploaded. In this case, the **Log Manager** will show that all buffers are full.

Possible solution: Click **Advanced >>** and increase **Max. Buffer Count** (the default is **10**).

- The upload task priority is not high enough; that is, higher-priority tasks execute so frequently that they prevent uploading (rare).

  Possible solution: Use **wvUploadTaskConfig(int** *stackSize***, int** *priority*) to change the priority (do not modify *stackSize*).

  The default upload task priority is **150**. So you would increase this to some higher integer, say **200**.

  To increase upload task priority, in the host or target shell, set the second parameter of **wvUploadTaskConfig** to, say, **200**:

  ```
  -> wvUploadTaskConfig(5000, 200)
  ```

Problem: Buffer Thrashing (Continuous Upload Only)

In a balanced system, the ring buffer is not constantly resized; it remains at the original, minimum ring size (**Min. Buffer Count** in the **Advanced Buffer Configuration**), allowing a steady upload of event data. All remaining buffers up to **Max. Buffer Count** should serve only as a reserve for temporarily holding any upload-backlog induced by sudden, massive bursts of data.

Thrashing refers to a situation where buffers are repeatedly allocated and freed. This interferes with target performance and generates intrusive event data, which will be reflected in the log.

Possible Solution

If you notice the behavior described above in the **Log Manager**, increase the minimum number of buffers in the ring:

Click **Advanced >>** and increase **Min. Buffer Count** (the default is **2**).

6 The Upload Method

6.1 The Upload Method

The **Upload Method** node lets you configure how the System Viewer event log is uploaded from the target to the host and where the uploaded log is stored.

The **Upload Method Selection** drop-list provides the following communication options:

- **Socket Via TSFS**
- **Socket Via TCP/IP**
- **File Via TSFS**
- **File Via NFS**

- **File Via netDrv**

6.1.1 Upload Method Selection Errors

If you see a red **X** icon against the **Upload Method** node in the tree at the left, it means that the current **Upload Method Selection** is invalid; uploading of log files is therefore disabled.

The error can be due to one or more of the following (the user interface will tell you which of these applies):

- The target does not support the selected communication option.

  If you want to use such an option, see *D. Configuring VxWorks for System Viewer* for information about how to add the necessary components.

- The **Upload Method Selection** configuration parameters have not yet been validated by System Viewer.

  Click the **Apply** button.

- The **Upload Method Selection** is incorrectly configured.

  See the user interface descriptions and the notes below.

NOTE: Always click the **Apply** button after changing upload method configuration parameters.

6.2 Using TSFS Upload Methods

The advantage of using the Target Server File System (TSFS) is that it does not require extra facilities. If you are on a network, TSFS uses already configured bandwidth to upload event log data. If you do not have network support, TSFS uses the serial connection.

To use either of the TSFS methods (file or socket), the target server must be configured to provide support for this. The TSFS system must be enabled and set to support read and write operations. In addition, the TSFS root directory must be specified. For information on target server configuration, see the *Wind River Workbench User's Guide*.

6.3 Automatic Upload of Logs

If log data are to be automatically uploaded and saved to files, you have to specify a valid file system location on the host the target sends the logs to.

- If you use **Socket** upload methods, you set the host file system location for the uploaded log in the **File Name** field.
  - If the target uploads data to a remote host, you would open the **Event Receive** utility on that host, and set the information there (see *6.8 The Event Receive Utility*, p.38).
  - If the target uploads data to the current host, you can set this directly in the **System Viewer Configuration** editor, without having to open the **Event Receive** utility.
- If you use **File** upload methods, you set the host file system location for the uploaded log in the **Directory containing uploaded log** field:

  This option is only available if the **Automatically view log on upload completion** checkbox is selected. The directory must be a path the host can follow to access the target's output file. For example, if you are running System Viewer on a Windows machine and the target in use is writing to a Unix machine, you have to map a Windows network drive that allows you to set a Windows path to the Unix directory holding the target's output file.

6.4 Socket Via TSFS and Socket Via TCP/IP Configuration

- **Host**

  The IP address or name of the host to which the data is to be uploaded. The default is the name of the current host.

  Specifying a remote host requires the **Event Receive** utility to be started on the remote host and ready to accept data at the specified socket number, otherwise validation of the upload method is not possible (see *6.8 The Event Receive Utility*, p.38).

- **Port Number**

  The socket port on the host to which the event log data is uploaded. The default port number is **6164**.

6.5 File Via TSFS

- **TSFS path and filename**

  You can enter a relative path segment (optional) and filename. The path segment, if used, will be relative to the target's **TSFS** root directory path (displayed further down in the current view).

- **Directory containing uploaded log**

  If the target's current TSFS root directory (displayed above the field) is a path that can be directly accessed by the current host, then enter the path exactly as displayed, plus any relative path you entered in the **TSFS path and filename** field. Otherwise, see *6.3 Automatic Upload of Logs*, p.35.

6.6 File Via NFS

- **NFS directory and file name**

  You can enter a relative path segment (optional) and filename. The path segment, if used, will be relative to the target's **NFS** directory.

- **Directory containing uploaded log**

  See *6.3 Automatic Upload of Logs*, p.35.

6.7 File Via netDrv

The netDrv driver typically uses either *ftp* or *rsh* for file transfer from the target to the host.

- **Directory containing uploaded log**

  See *6.3 Automatic Upload of Logs*, p.35.

- **netDrv directory and filename**

  You can enter a relative path segment (optional) and filename. The path segment, if used, will be relative to the target's **netDrv** directory.

- **Host**

  This is for information purposes only. If no host name or IP address can be determined, the upload method is not available for use.

6.8 The Event Receive Utility

This section is only relevant if you have socket connections on a remote host.

The **Event Receive** utility allows a remote host to listen for an incoming connection from a target that has an event buffer to upload. **Event Receive** then accepts this log and writes it to a file on the host's file system. The **.wvr** file extension is automatically appended to the saved log files.

To open the **Event Receive** utility from Wind River Workbench select **Target > Event Receive**.

The **Event Receive** utility allows you to upload data directly from the target to a socket connection. The uploaded data is then saved to a file which is specified by **Event Receive**.

From the **Event Receive** utility, you can view or adjust these options for data upload.

- **File Name**

  Specifies the path and filename to which log files are saved. The default is *userHomeDir***/eventLog.wvr**.

- **Port Number**

  Specifies the host TCP/IP port over which the **Event Receive** utility listens for event data from the upload task. The default event port number for **Event Receive** is **6164**. If your target uses a different event port, specify the port number in this field.

- **Increment Filenames**

  Multiple log files are distinguished with the naming pattern, *filename*.*num***.wvr**, whereby *num* is an integer incremented by one for each file. Although, you cannot specify the *num* segment of the filename, you can influence the integer at which it begins. If you select **Overwrite Existing Files**, the event receive session begins numbering its log files at zero, and overwrites any files from previous event receive sessions.

- **Overwrite Files**

  If not selected, the next free incremental number (files with increment numbers already exist in the directory) is used.

7 Logging and Uploading Data

7.1 Start Logging

To begin collecting data you must have a target (or target simulator) running an operating system configured for System Viewer.

If logs are to be uploaded to a remote host via a socket connection, make sure the**Event Receive** dialog is up and listening on that host. used to receive data on a socket that is sent from the target. For more information, see *6.8 The Event Receive Utility*, p.38

System Viewer provides various methods to start logging and to upload data:

Configuration

The **System Viewer Configuration** editor is the simplest and most straightforward method to manually start and stop logging. For more information, see *7.2 The Configuration Editor Log Manager*, p.40.

Triggering
With Triggering, you specify events and conditions that can be validated and used to start and stop System Viewer logging. For more information, see *13. Using Triggering*.

Wind River System Viewer API
The Wind River System Viewer API is the most precise method to determine when logging starts and stops. It is especially useful when there is a lot of activity on the target. You can issue commands from the host shellor include them in your application code. Although precise, this method is more complex than Triggering. For more information, see *7.3 Using System Viewer API to Control Logging*, p.41.

The **Event Receive** dialog is used to receive data on a socket that is sent from the target. For more information, see *6.8 The Event Receive Utility*, p.38.

If you prefer not to use a socket that listens to the target, you can also upload data using a file with TSFS, NFS, or netDrv. For more information, see *6. The Upload Method*.

7.2 The Configuration Editor Log Manager

NOTE: Using configuration and its features differ with System Viewer on Linux, see *15.2 Configuration on Linux*, p.134.

System Viewer dynamically enables or disables items to start and stop logging on the main **System Viewer Configuration** menu and the toolbar according to the status of log collection.

Refreshing information can be slightly intrusive on the target, so Wind River System Viewer provides the ability to control how and when data gets refreshed using the **Refresh Controls** that appear at the top of the **Log Manager**.

The default view of the **Log Manager** displays the total size of the ring buffer, and the amount of data currently in the buffer. The current state is also displayed in a progress bar.

Progress bar colours:

Orange
: Buffer (first five bars) has been allocated and is ready to be used.

Green
: A green buffer (first three bars) indicates the buffer has been used. If the buffer is partially green, the buffer is still being used and is not yet full.

Red
: In post mortem or using a circular buffer, red indicates that the buffer contains the oldest set of data and will be overwritten when the buffer currently in use is filled.

Clicking the **More Detail >>** button provides further information related to the buffer and its state. Note that the **Bytes Read** value is always zero in deferred or post-mortem modes because no data is read from the buffer (uploaded).

7.3 Using System Viewer API to Control Logging

Using the Wind River System Viewer API is the most precise method to determine when logging starts and stops. Use this method when there is a lot of activity on the target as using the **System Viewer Configuration** utility may generate unnecessarily large logs.

You can either issue Wind River System Viewer API commands from the Host Shell or programmatically in your application code. As described below, use one of the the following API groups.

wvOn( ) and wvOff( )

These routines provide the easiest method to control logging. The **wvOn( )** and **wvOff( )**, routines are defined in **usrWindView.c**.

These routines start a typical instance of event logging and upload. Because these routines are not used by Wind River System Viewer itself, you can safely modify them to suit your specific needs. To start event logging use **wvOn( )**, then start the target application. To close the host-target connection, use **wvOff( )** on the target.

To examine the source code and how these routines are used, refer to *installDir*/**vxWorks-***N.N*/**target/config/comps/src/usrWindView.c**.

wvLib and wvNetDLib

A more complex, yet precise method is to use the routines in **wvNetDLib**, which provide more control over event logging.

The order in which you invoke these libraries is critical. You can also use the **e( )** routine with Wind River System Viewer, which is documented in the entry for **dbgLib**. For more information, see *A. Programming Data Collection* and the reference entries for **wvLib**, **wvNetDLib**, and **dbgLib**.

8 Loading and Reading Log Files

8.1 Introduction

When you open a Wind River System Viewer log file, a main viewing window appears with primary tools to view and analyze log files. This chapter describes:

- the types of System Viewer log files, including an analysis file, that when opened initializes the Log Viewer utility to the state it was in when the analysis file was saved. This is useful for passing a log with annotations and comments to colleagues for further analysis.
- how to open, close, and save log files
- the facilities provided for viewing logs
- how to use Wind River System Viewer to read log files

8.2 Opening Logs

You can open log files using the Wind River Workbench **File > Open** menu. In addition, System Viewer can automatically open log file upon upload.

There are two ways to configure System Viewer to automatically open a log file:

- In the **Upload Method Configuration** node of the **System Viewer Configuration** editor, select **Automatically view log on upload completion**. For more information, see *6.1 The Upload Method*, p.33.
- In the **Event Receive** utility, select **View Files Automatically**. For more information, see *6.8 The Event Receive Utility*, p.38.

NOTE: If you are using triggering to start and stop System Viewer, the resulting log file may not open automatically when System Viewer is stopped. Of course, you must explicitly press upload if you are using deferred upload mode. If you are using continuous mode, the log has been uploaded but it will not open until you either refresh triggers or click **Upload**. For more information, see *5. The Upload Mode* and *13. Using Triggering*.

8.2.1 Load Progress Dialog

When you open a System Viewer log for viewing, a load progress dialog appears. You can copy and paste text from this dialog.

This dialog remains open if there are fatal errors, in which case, your log file will not open for viewing.

A log can load successfully or with warnings. If there are warnings, the **Log Viewer** opens with the **Log Load Report** in focus to draw attention to these. In these circumstances, it is possible that the the contents of the log will be truncated from the point where the warnings were encountered.

8.2.2 Errors and Warning Messages when Opening Log Files

A log file may fail to load for one of the following reasons:

1. The log file is not a System Viewer log file.
2. The log file does not exist.
3. The log file is from a target operating system not supported by the System Viewer installation.
4. The log contains unknown events.

In the first three cases, the error is fatal and log loading cannot proceed. In the final case, the log is loaded up to the point where the unrecognized event is encountered.

8.2.3 Exporting Log Files

Choose **File > Export** to save your log file in a format that may be opened by third-party tools. The two formats available are CSV (for opening in spreadsheet applications) or TXT for opening in text editors.

8.3 Understanding the Log Viewer

System Viewer provides a robust framework to view log files based on a common, underlying data model for analysis. You can view one or multiple log files in various ways, each on a different tab:

Event Graph
A graphical representation of the log's content

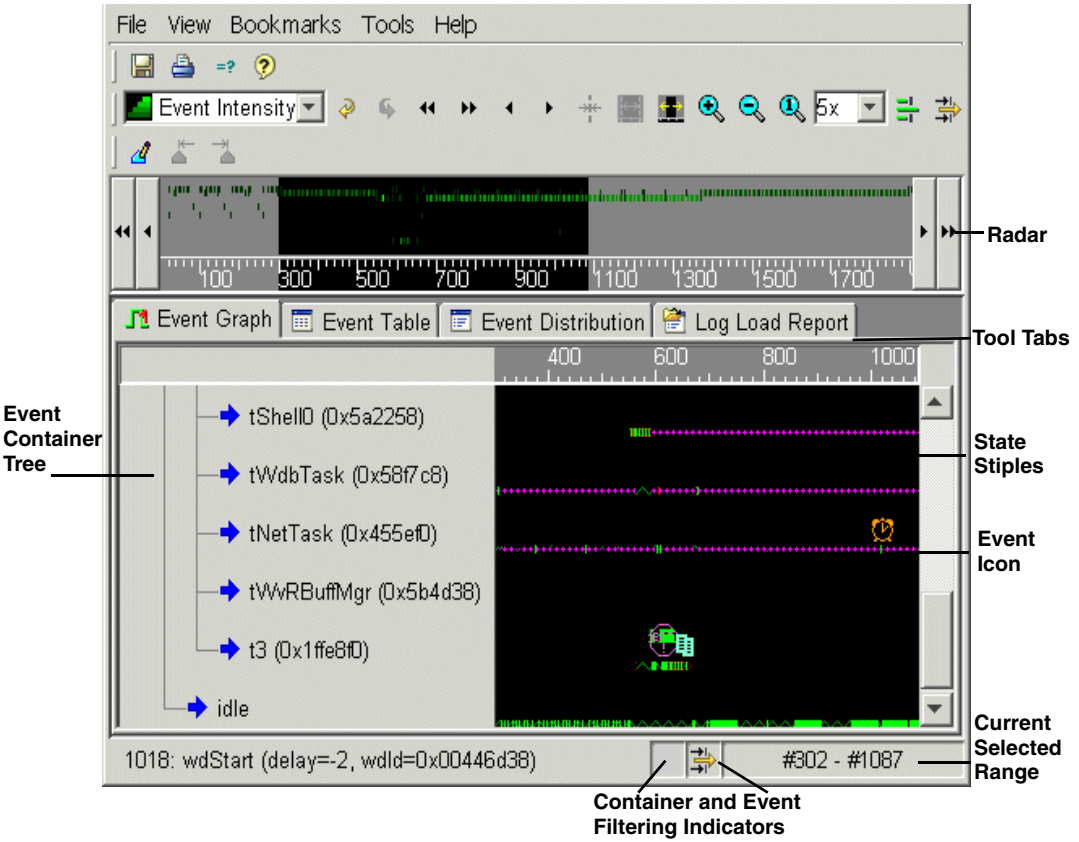
Event Table
A tabular representation of the log's content

Event Distribution
An analysis of the distribution of event types within the log

Figure 8-1 shows the Log Viewer and its component parts.

You access and close the log viewer tools from the **Tools** menu. When selected, these viewing options appear as tabs within the log viewer. The log file is displayed with the **Event Graph**, but the same log file can be viewed using all four viewing options.

Figure 8-1 **The Log Viewer Utility**



The following basic and optional components of the Log Viewer may or may not be used by any viewing tool.

8.3.1 Basic Components

Status Bar

The right section of the status bar always displays the currently selected range that is also highlighted in the Radar. Some viewing tools, such as the **Event Graph** display additional information on the status bar. The status bar also displays icons, and enables status of event class and container filtering. For more information, see *12.2 Display Filtering Options*, p.80.

Radar

The Radar presents a summary view of the profile of event activity within the log being viewed. Several graphing options are available for use in the Radar. The option which is most useful will vary depending on the characteristics of the event data contained in the log. For most purposes, the **All Events** radar suffices, since it presents a histogram of event density across the timeframe captured in the log.

Most viewing tools adjust their display to show only the time range selected in the Radar. The selected range is represented by the area with a black background. For more information, see *10. Using the Radar*.

Event Container Tree

Many viewing tools contain an Event Container Tree. Each row in the Container Tree represents an execution context (a task, an interrupt or, in some operating systems, a process). Often, groups of context types are grouped under a collapsible tree node, for example, Interrupts, Tasks, and so on.

All tools that display an event container tree reflect its current filtering configuration. If one viewing tool changes by the filtering applied to the event container tree, that change is seen across all other viewing tools.

Event Icons and State Stiples

The System Viewer Legend that can be accessed from **Help > Legend** describes the event types and states represented in some viewing tools as icons and patterned lines. The legend only displays the events and states contained in the log file being viewed.

Right-clicking on an event icon or a state stipple anywhere in the Log Viewer utility displays a context menu from which online help relating to that event or state may be accessed. For more information, see *12.7.1 Accessing the Event Dictionary*, p.88. For more information on icons, see *Event Icons*, p.54.

Analysis Pack Panel

Analysis Packs are provided for some target operating systems. Those available for the target operating systems on which the log being viewed was captured, may be viewed using the **View > Analysis Packs**

If Analysis Packs are available for your target operating system, select the desired pack in the **Inactive** pane and click **Use > > >** and **OK**. The analysis pack is then displayed below the **Event Graph**, and as an overlay in the Radar.

8.4 Reading and Configuring Timestamping

When you develop your application for a target with a supported timestamp driver, the instrumented kernel uses timestamps. The instrumented kernel can tag certain events with high-resolution timestamps, sequence numbers, or with user defined timestamps. Those events are then displayed in the Wind River System Viewer **Event Graph**, along a timeline that shows when each event occurred, based on these timestamps.

The status bar displays the exact timestamp for an event when you click on the corresponding event icon; and the **Event Table** has a **Timestamp** column.

⚠ **CAUTION:** On supported targets with limited hardware resources, Wind River System Viewer manages the system clock, the auxiliary clock, or both when the timestamp driver is enabled. In this case, these clocks are not available for the system or other application use timestamp.

Timestamp Ticks

All time measurements such as any event, bookmark, measurement marker, range selection, and so on, are aligned to a tick. The amount of time defining a tick, may or may not be a real time unit:

- In a sequential log, every event captured is assigned a tick value which increments by 1 for successive events. When displayed, these sequential timestamps are shown as **#n** where *n* is an incrementing number. This indicates that the timestamps logged against each event is not a real-time

measurement, but a sequence number. For more information, see *Sequential Timestamping*, p.49.

- In a real-time log, the time per tick is specified in the raw log as the number of ticks per second and this is related to the resolution of the clock hardware being used with the BSP for timestamping purposes. The timestamps are printed out using s, ms, us, ns, ps; whichever is most appropriate based on the formula: **time = tick / (ticks-per-second)**

  The timestamp driver's resolution is hardware dependent, but is typically between 100 KHz (10 microseconds per tick) and 1 MHz (1 microsecond per tick).

For information on the system timestamp driver for any supported board, see the entry for the board in the online BSP APIs.

High-Resolution Timestamping

Some BSPs provided by Wind River have a high-resolution timestamp driver. If this is the case, include the following component in your system image:

SYS_TIMESTAMP
BSP specific timestamping

BSP-specific drivers are in located the directory *installDir*/**vxworks-***N.N*/**target/src/drv/timer**; the corresponding header files are located in *installDir*/**vxworks-***N.N*/**target/h/drv/timer**.

If you are using a board that does not have a timestamp driver, or if you do not want to use the one it has, you can use sequential or user-defined timestamping and include the appropriate component.

Sequential Timestamping

When the real-time system runs on an unsupported board, or on a supported board without the timestamp driver included, the instrumented kernel automatically uses its *sequence-stamp driver*. This driver tags events with sequence numbers that represent the order in which the events occur on the target. The events are then spaced equidistantly from each other in the event graph and the timeline scale is in event sequence numbers rather than seconds. The support component for sequential timestamping is **SEQ_TIMESTAMP**.

Custom Timestamp Drivers

Wind River supplies the system timestamp, but you may write your own timestamp driver or use the timestamp provided by an emulator. The support component for user-defined timestamping is **USER_TIMESTAMP**. For more information, see *VxWorks Device Driver Developer's Guide:Timestamp Drivers*.

9 Using Viewing Tools

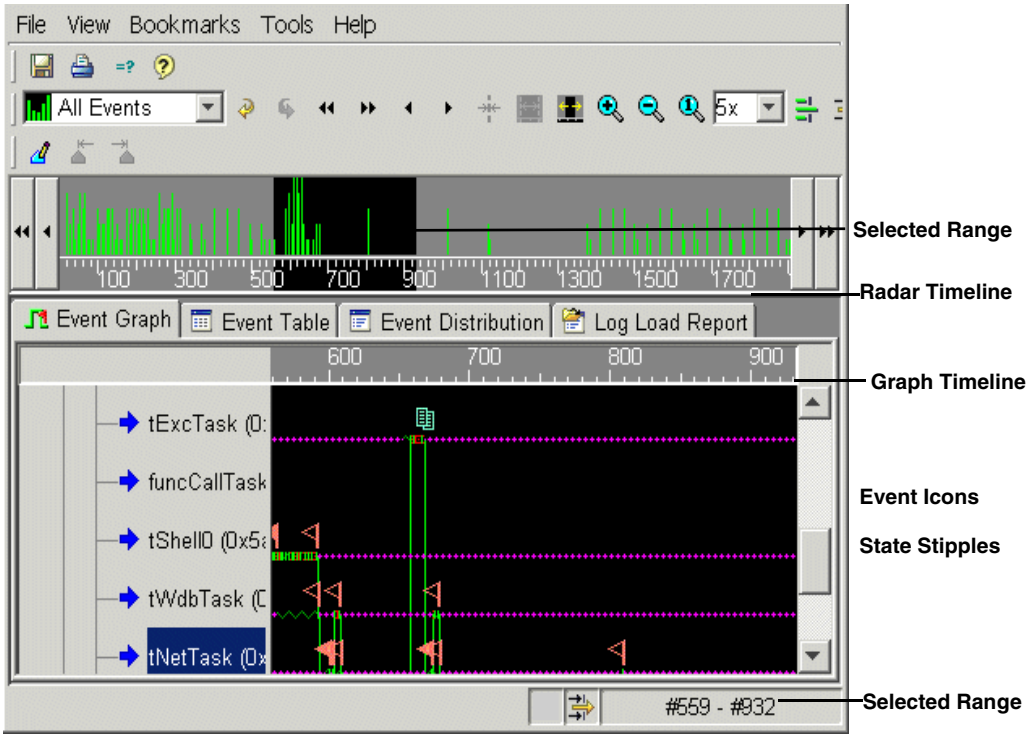
9.1 Introduction

Wind River System Viewer provides standard viewing tools and analysis pack features as options for visually presenting log information. These tools are displayed in the tool pane or analysis panel of the main Wind River System Viewer viewing window, as described in *8.3 Understanding the Log Viewer*, p.45.

9.2 Reading the Event Graph

The **Event Graph** depicts events in a log as a scrollable graph. This same sample log can also be depicted in the **Event Table**, as well as the **Event Distribution**.

Figure 9-1 **Reading Event Graph Information**



Events in the graph are laid out horizontally, from left to right, according to the graph timeline. This horizontal presentation corresponds to the time interval designated by the selected range. The timeline displays the time—either in seconds or by sequence numbers—which corresponds to the section of log activity shown in the graph.

To print out a section of the log visible in the event, select **File > Print.** This option is also available from the context menu, and described in *12.5 Event Table Context Menu*, p.86. For a description of all **Event Graph** context menus, see *12.4 Event Graph Context Menu*, p.84.

Status Bar

The left side of the status bar displays information on a selected event. When you mouse-over any event icon in the **Event Graph**, the timestamp event name, parameters, and argument values appear. The status bar displays the bounds of a measured range as you drag a new one, or if you mouse-over either boundary of an existing measured range.

The right box in the status bar displays the bounds of the currently selected range. The timestamps displayed are real time; seconds, minutes, and so on, if the target image is configured with a real-time timestamp driver, or sequence numbers, prefixed with a #, if sequential timestamping is configured. This area also displays the bounds of a new selection range whilst it is being dragged. For more information, see *8.4 Reading and Configuring Timestamping*, p.48.

The two small boxes to the left of the selected range area contain indicators which show you the various types of filtering applied to the current display. For more information, see *9. Using Viewing Tools*.

Event Graph Container Tree

The Event Graph displays an event container tree. The containers or nodes are listed in priority, from highest to lowest. The node names describe each of the horizontally corresponding event graph rows. The event container tree has context menus for its root node and sub-nodes as described in *12.3 Container Tree Context Menu*, p.81.

The log displayed in Figure 9-1 includes interrupt levels, which are listed in order, with the highest-priority interrupt at the top. Below the interrupts are tasks, listed by priority from the first task based on initial priority in the log (not within the selected range). Changes to a task priority during execution do not change its vertical position in the **Event Graph**. The last thread of execution shown in the tree represents the kernel idle loop.

The scrollbar on the right side of the event graph controls the graph and the container tree, allowing access to rows that extend vertically beyond the current viewport.

Event Icons

The Event Graph uses event icons and state stipples to identify elements in the graph. The event icons indicate the type of event that occurred at a given time. The state stiples are represented as horizontal lines and indicate the state of each task at a given time. The log in Figure 9-1 was collected at the Additional Instrumentation logging level, in which events are displayed independent of whether they result in task state transitions or context switches.

Measurement Markers

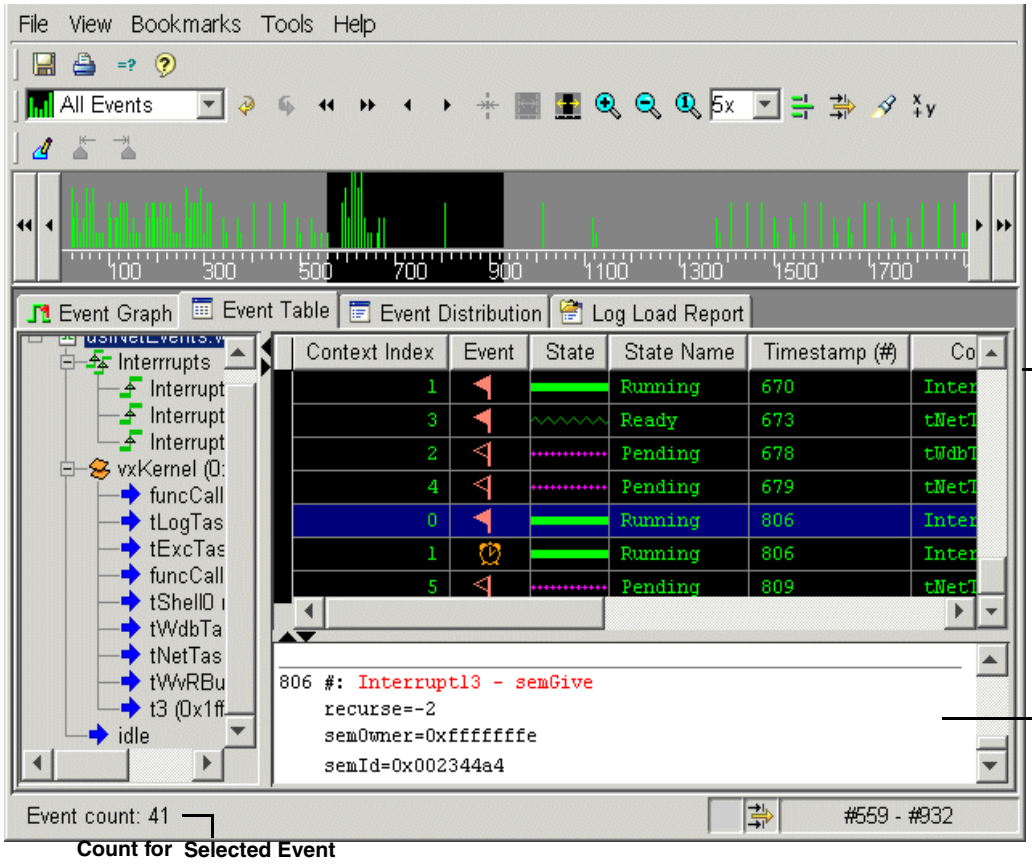
Wind River System Viewer uses measurement markers to measure the time between any two points on the **Event Graph**. The start time, end time, and duration of time between markers is shown in the status bar when you point at a measurement marker. Pointing away from a measurement marker changes the information displayed in the status bar. To view marker information again, point to the start or end measurement marker. For more information on measurement markers, see *10.2.5 Using Measurement Markers*, p.67.

9.3 Reading the Event Table

Figure 9-2 shows the **Event Table** of the same log depicted by **Event Graph** and **Event Distribution**.

The **Event Table** displays events in a log as rows of information in scrollable table format. Events are ordered according to time stamp, from top to bottom, indicated by the **Timestamp** column. The **Event Table** only depicts events that occur within the currently selected range and within nodes selected from the container tree.

Figure 9-2 **Reading Event Table Information**



The status bar displays the event count of the selected container on the left. The **Event Table** also uses the event and state icons in its information columns.

To print the **Event Table,** select **File > Print** from the main menu, and the currently displayed information in the table will print. The **Event** and **State** columns, which display icons are never printed. The lower text pane is used to copy rows from the table to the clipboard or for printing. For more information, see *12.4 Event Graph Context Menu*, p.84.

Column Information

The event table columns are described below:

- **Context Index**

  Specifies the order in which an event appears in its container. The **Context Index** is unique for each event in a container.

- **Event**

  Displays the event type as an icon. If an event type does not have an icon, nothing appears in this column.

- **State**

  Displays the context state as a state stipple.

- **State Name**

  Describes the context state.

- **Context**

  Specifies the context in which the event occurred.

- **Event Name**

  Identifies the event by name.

- **Event Class**

  Specifies the class of the event, as listed in the **Filter Events** dialog. For more information, see *12.2 Display Filtering Options*, p.80.

- **Event Type Id**

  A number between 0 and 65535 that uniquely identifies the type of event in the operating system's System Viewer instrumentation. This ID is useful only if you are interested in implementing your own application events.

- **Timestamp**

  Specifies the time at which the event occurred.

- **Time delta**

  Indicates the elapsed time between the event displayed in the previous row and the current event.

- **Parameters**

  Lists parameters, if any, and their corresponding argument values contained in the event.

Wind River System Viewer hides the **Context Index**, **State**, **State Name**, **Event Class**, **Event Type Id**, and **Time Delta** columns by default. You can toggle the columns to hide or show their content as described in *12Using Display Filtering and Context Menus*, p.79.

Event Table Container Tree

The event table also displays an event container tree. The containers are listed in order by priority from highest to lowest. Selecting the topmost or root container displays events from all visible containers in the event table. Selecting any other container, or a group of containers, displays only those events in those containers. This behavior differs from the way the event container is used in the event graph. For more information, see *12. Using Display Filtering and Context Menus*.

Text Pane and Printing

The text pane is the white area below the **Event Table** rows. Using the mouse or keyboard, you can select table rows to be copied into the lower pane. The text in the lower pane can then be cut to the clipboard or printed. Discontinuities in the selected rows are separated by lines in this pane. This is a convenient feature for comparing events that occurred at disparate times in the log. It is also allows details of all parameters attached to an event to be displayed in full, in contrast to the parameters cell in the table above which can display only the start of such information. For more information, see *12.5 Event Table Context Menu*, p.86.

9.4 Reading the Event Distribution Display

The **Event Distribution** display is a multi-column list of containers and their respective event counts.

As with all other log viewing tools, **Event Distribution** only displays information from within the bounds defined by the currently selected range, as displayed in the radar and at the bottom right of the status bar.

The **Container** column lists the contexts in the same order as the container tree. The **Event Count** column displays, in bold text, the number of events that occur in each container and, in non-bold text, the number of events that occur in each event class. The class type is the same used for the **Event Class** column in the **Event Table**. The status bar displays only the standard bounds of time interval, and no additional information. When you point to a node in the event container tree, the event count column information is displayed in the **Event Table** for containers, as described in *12.6 Event Distribution Context Menu*, p.87.

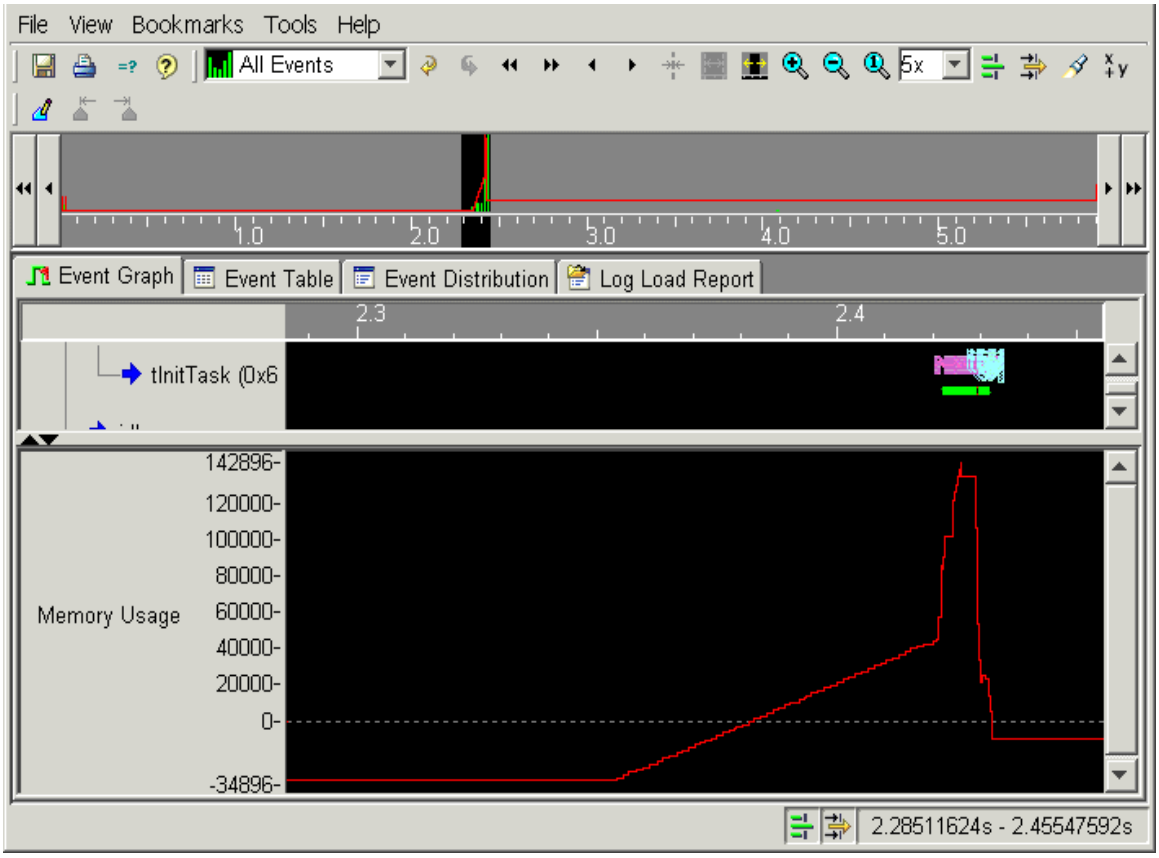
9.5 Reading the Memory Usage Analysis Pack

The memory usage analysis pack charts memory allocation and deallocation resulting from **memLib** function calls. You can see the addresses of memory blocks that get allocated and freed.

Figure 9-3 depicts a memory usage analysis pack, which is displayed as a red line graph in the analysis pack pane below the **Event Graph**. This pack uses the same horizontal timeline as the **Event Graph** and charts memory allocation and deallocation vertically. The memory graph is represented in the Radar as a red line.

While the event graph indicates when memory events occur, it does not show how much memory is allocated at any time. The memory usage analysis pack is designed specifically for viewing memory usage. The memory usage pack is only be available for applications that use the **memLib** library to allocate and free memory.

Figure 9-3 **Memory Usage Analysis Pack**



9.5.1 To Open the Memory Usage Analysis Pack

1. Open a log file.
2. Select **View > Analysis Packs...**
3. Select **Memory Usage** from the **Use/Remove Analysis Pack** panel.
4. Click **Use > > >** to add memory usage to the active list. If the log file currently in view does not contain memory data, Wind River System Viewer indicates so with a **No data** error message.

Using Filtering Tools

The memory usage analysis pack is especially useful for detecting memory leaks. The sample log in Figure 9-3 shows a gradual rise and then a sharp spike in memory usage between timestamps. To diagnose these rises in memory usage, you can use the **Filter Events** dialog to filter out all events except memory events. You can also filter out containers that have no event activity.

Figure 9-4 shows the **Event Graph** and memory usage for a log with filters applied to the first section of notable memory consumption. Note how the memory usage rises in the Radar, and how the **Event Graph** clearly displays where the memory usage activity originates. The Event Cursor information is displayed on the status bar as a call to **memPartAlignedAlloc( )**.

Figure 9-4 **Filtered for Memory Events Only**

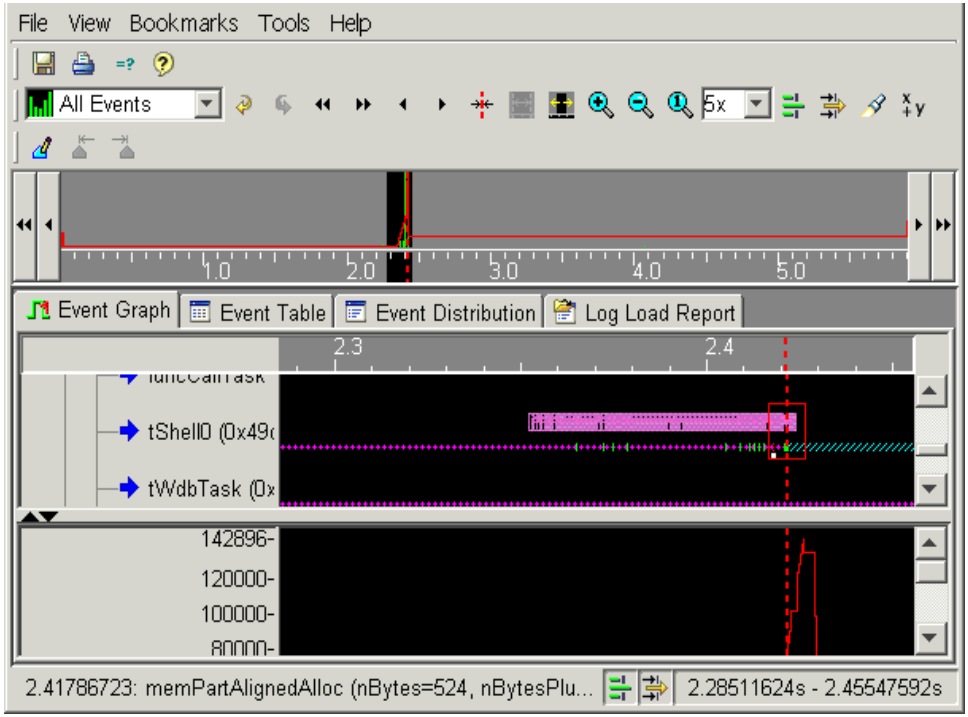
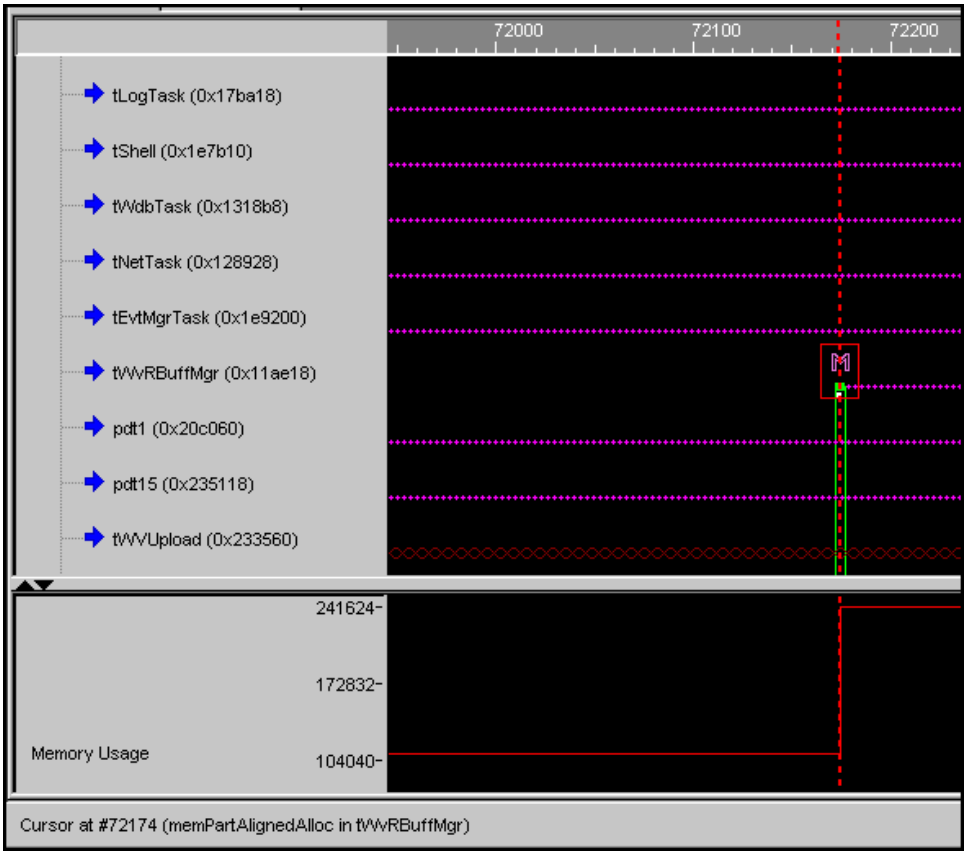


Figure 9-5 shows another log with similar filters applied. The rise in memory usage is discovered by setting the Event Cursor on the vertical red rise in the memory usage graph. The event shows a memory allocation of 131104 bytes at timestamp #72174.

Figure 9-5 **Memory Usage Spike**



10 Using the Radar

10.1 Introduction

The Radar is a way to navigate within a log file. Located at the top of the Log Viewer utility, the Radar presents a time-based view of an entire log. This feature of Wind River System Viewer allows you to view select information or a particular type of activity within the log using three major display modes.

Within the Radar, the currently selected range is represented by the area with a black background. This selected range is that portion of the content of the log file which appears within each log viewing tool.

You can change the currently selected range using these methods:

- dragging a new range in the Radar
- dragging a new range in the **Event Graph**
- defining the bounds of a new range in the Select Range dialog
- selecting a new range based on an existing pair of measurement markers
- zooming in/out using the menu, toolbar buttons, context menus or keyboard
- scrolling the range using the menu, toolbar buttons, context menus or keyboard
- jumping to bookmarked timestamps
- using the **Event Properties/Search (***filename***)** dialog to search for events within the log

10.2 Changing the Selected Range Using the Radar

The Radar always displays a currently selected range of the event log in view, signified by a black area. The gray background of the Radar are regions of the event log that fall outside the currently selected range. Changing the selected range allows you to isolate a portion of the event log which the Radar identifies as an area of activity using one of the available radar display modes.

To manually shrink or expand the current display range from within the Radar, move the mouse over either end of the Radar range. The cursor changes to a horizontal arrow. Press the left mouse button and drag an edge as required. When you release the mouse, the new black area defines a new range that is displayed by all viewing tools.

10.2.1 Moving the Selected Range with the Mouse

There are two methods to move an existing selected range using your mouse:

- Move the cursor anywhere within the selected range in the Radar, but not on an edge. The cursor changes to a four-pointed arrow. Press the left mouse button and drag the selected range left or right as required. Releasing the mouse button drops the range; and all viewing tools update accordingly.

- Single-click the mouse anywhere *outside* the currently selected range. The entire range is then centered on the point you selected, contrained by the bounds of the entire event log.

10.2.2 Defining a New Selected Range with the Mouse

To define a new range using the mouse:

1. Locate the start of the new range by moving the mouse to a desired point in the range.
2. Hold down the **SHIFT** key and simultaneously press the left mouse button, then drag the mouse to the end of the desired range.
3. Release the left mouse button before releasing the **SHIFT** key. This completes the new range selection and all viewing tools update to reflect it.

NOTE: Releasing the **SHIFT** key *before* you release the left mouse button aborts the new range selection, and the display reverts to the previously selected range.

10.2.3 Defining a New Selected Range Using the Select Range Dialog

1. Select **View > Select Range** or the corresponding toolbar button. This opens the **Select Range** dialog.
2. Enter values in **Start time** and **End time**.
3. Click **OK**. If your selection is valid, that is within the bounds of the event log, the selected range is changed as entered, otherwise you are warned that the range is invalid.

NOTE: Each time you modify or define a new selected range, the right side of the status bar dynamically updates to reflect that range. The new selected range is also reflected in the radar.

10.2.4 Zooming the Selected Range

Zooming modifies the selected range, so that the tool window shows more (zoom in) or less (zoom out) detail. Your options for zooming are accessible from keyboard shortcuts, the context menus of some tools, the main **View** menu or the toolbar. They are as follows:

- **Zoom 100%**

  Sets the range to be the entire event log. This feature is provided for completeness, but is not recommended for extensive use because rendering and displaying the large amount of information in an entire log can be time-consuming. Instead, you can use the Radar's features to identify areas of interest, and then select only those ranges for subsequent display.

- **Zoom Factor**

  Scales the range by a specified factor that you apply to subsequent zooming operations.

- **Zoom In (numpad "+")**

  Divides the range by the currently active zoom factor, displaying a smaller portion of the event log. When it is not possible to zoom in further, all means of zooming in are disabled.

- **Zoom Out (numpad "-")**

  Multiplies the range by the currently active zoom factor, displaying a larger portion of the event log. When the selected range covers the entire event log, zooming out is disabled.

Zooming always attempts to maintain the center of the range, with the following two exceptions:

- If the current range abuts one end of the log, **Zoom In** stays aligned with edge of the log. This allows you to **Zoom In** on the beginning or the end of a log.
- If the Event Cursor is out of range by zooming in and prior to zooming in, the event cursor is within the currently selected range and the zoom in would put the event cursor outside the new selected range, then the zoom in operation re-centers the new range on the event cursor. For details, see *Zooming on the Event Cursor*, p.74.

10.2.5 Using Measurement Markers

You create measurement markers to measure the time interval between two points. You can then use this measured range to define a new selected range. To create measurement markers, click the point at which the measurement should start in to the end of the measured range, then release the mouse button. Measured range markers appear as white-dashed vertical lines as you hold and drag.

As you drag the mouse, the measured range is displayed on the left side of the status bar. When you release the mouse, the measurement markers remain and the status bar indicates the measurement. The status bar then reverts to its normal mode of operation in which it displays information on whatever the mouse pointer is hovering over. To determine the bounds of a measured range following its creation, just mouse-over either of the measurement markers and the measured range bounds appear once more in the status bar. The **Event Graph** also provides information on the measured range as described in *Measurement Markers*, p.54.

Measurement markers also appear in the Radar and can be modified but not created from there by dragging either measurement marker. To set the selected range to be the same as the measured range, select **View > Select measured range** or the corresponding toolbar button. These options are disabled when you have not defined a measured range.

Once measurement markers are set, they remain until you manually delete them. In the **Event Graph**, if you create, reposition, or delete the Event Cursor when measurement markers are present, the markers remain. To delete the markers, single-click on either measurement marker in the **Event Graph** and both markers are deleted. You can also single-click in any blank area in the **Event Graph** and both the Event Cursor and the measurement markers are deleted.

Changing the range selection does not remove measurement markers. The measured range is saved along with the raw log in a **.wva** file, if desired. On opening that **.wva** file for viewing, any measured range that existed when the **.wva** file was saved is restored. and restored from a **.wva** file. For more information on saving file settings, see *8.2 Opening Logs*, p.44.

10.2.6 Nudging and Paging the Selected Range

Nudge and **Page** features allow you to scroll the currently selected range across the log for display in the viewing tools as follows:

- **Nudge** moves the selected range by 0.1x the period of the selected range in the appropriate direction, subject to the bounds of the event log.
- **Page** moves the selected range by 0.9x the period of the selected range in the appropriate direction, subject to the bounds of the event log. There is a slight overlap when paging, which lets you track where you were.

You can scroll through the log, in small or large steps. You can access these features by choosing the appropriate **View** menu item, the corresponding toolbar buttons, the scroll buttons at either end of the Radar, or the predefined keyboard shortcuts. The **View** menu also displays the appropriate keyboard shortcuts for each action:

10.2.7 Moving the Selected Range Between Markers

There are several methods to move the selected range to the Event Cursor or to define bookmarks. These are described in *Moving to the Event Cursor*, p.74 and *11.3.5 Navigating Between Bookmarks*, p.78.

10.2.8 Undoing and Redoing the Range Selection

A stack of the last 100 range selection commands is maintained so that you can go back to previous range selections. To do this, use the **View > Undo** menu item, its corresponding toolbar button, or the predefined keyboard shortcut on the menu. Having undone a range change, you can then redo any of them by using the **View > Redo** menu item, its corresponding toolbar button, or the predefined keyboard shortcut on the menu item. If you perform several undo actions, modify the selected range by any means other than **Redo**, the redo stack is deleted and the redo option is disabled.

10.3 Using Radar Modes

To choose the Radar mode, use **View > Radar** or the drop-down listbox on the toolbar. The currently selected Radar mode is indicated by a dot on the **View > Radar** menu list and in the toolbar drop-down listbox. A number of standard Radar modes are common to all operating systems.

In all modes, the (horizontal) x- axis of the Radar represents time with the extent representing the entire range of the event log. Below the main Radar display is a time scale that represents the entire range of the event log, divided into a number of equal timeslices, called *buckets*. The number of buckets allocated is automatically determined by Wind River System Viewer when an event log is loaded, and is based on the timestamping configuration. Each Radar mode differs on how it allocates events and state changes to buckets, and thus accumulates activity data in different ways.

The Radar data collection mechanism operates on an event log only after any container or event filtering as described in *12.2 Display Filtering Options*, p.80 is applied. For example, if an entire class of events is filtered out using the filtering tool, the activity profile shown by the Radar reflects this filtering. Similarly, if containers are excluded from the display using container filtering, all events and state changes that were in the excluded containers cease to contribute to the Radar's display.

The Radar mode options in Wind River System Viewer are:

10.3.1 All Events Radar Mode

The **All Events** Radar mode is a simple, time-based separation of event and state-change activity into data buckets, displayed as a bar graph. In this Radar graph, the y-axis represents magnitude—the higher the bar displayed at a given point on the time axis, the more event/state-change activity occurred during that time period. Each events and state change is given the same weighting, regardless of the container in which it occurred.

10.3.2 Peak Activity Radar Mode

The **Peak Activity** Radar mode shows how peak activity runs through the displayed containers over the course of the event log.

It is easy to locate an area of peak activity by setting this Radar mode. The y-axis represents the vertical location of a container in the container tree, as displayed alongside the **Event Graph** or the **Event Table** tools. Within each time slice, the container that exhibits the greatest number of events or state-changes is identified, and a small horizontal line is plotted at its corresponding vertical location in the tree of event containers. If the Radar shows a line in the middle of the vertical range of the Radar, this corresponds to a container that is in the middle of the vertical range of the displayed containers in the **Event Graph** or **Event Table**.

10.3.3 Event Intensity Radar Mode

The **Event Intensity** Radar mode depicts a *heat-map* of event intensity.

The y-axis represents the vertical location of a container in the container tree, as displayed alongside the **Event Graph** or the **Event Table** tools. In this mode, the number of event or state changes that occur in each container, over the given time slice, is represented by the intensity of a colored square. The brighter the square, the more activity occurred at the given container location. For example, suppose an event intensity Radar graph shows a bright green block 1/3 of the way through a log and 2/3 of the way down the Radar's vertical range. You should first select the Radar range that contains the brightest block (this changes the selected range accordingly), then scroll down in the **Event Graph** to reveal the container that is 2/3 of the way down the included event container tree. It should then be obvious where activity is occurring.

The selected Radar range matches the pattern of the main **Event Graph**. Also, the vertical green task and interrupt transition lines have been turned off for clarity. This is done using the **Event Graph** context menu as described in *12.4 Event Graph Context Menu*, p.84.

10.3.4 No Radar Mode

The **No Radar** mode removes the Radar graph area, leaving only the Radar scale, which shows the currently selected range. This mode is useful when you have defined a range of interest and you want to maximize the screen area used by a viewing tool such as the **Event Graph**.

11

Finding and Marking Events

11.1 Introduction

Wind River System Viewer allows you to search for specific events. You can mark and annotate events for future reference using:

- the **Event Cursor**, which marks the location of a single event, and can be set manually, or as a result of a search for a specific event or event type. Once set, you can re-center the selected range on the Event Cursor.
- **bookmarks**, which are named and commented markers that can be set on any location in the log.

11.2 Using the Event Cursor

The Event Cursor indicates the location of a specific event in the log. It is always aligned to an event, and its current position is represented in the Radar as a red, dashed, vertical line.

The Event Cursor appears as a result of an event search, or in some tools by clicking in the tool pane. Like the Radar, not all tools use the Event Cursor. Each tool that does use it displays it appropriately for the layout of content in that tool. For example, in the **Event Table**, the Event Cursor is represented by red highlighting of the contents in the event row; in the **Event Graph**, it is represented as it is in the Radar—by a red, dashed, vertical line. You can access the Event Cursor using menu options on either the main menu or context menus.

NOTE: The Event Cursor defines the concept of the *current* event. Its presence changes the behavior of some range selection operations, such as zooming. Also, if you set the Event Cursor in one tool, and then switch to another tool, the event cursor is attached to the same event in both tools.

11.2.1 Setting the Event Cursor

You can set the Event Cursor in several ways depending on which viewing tool is in use:

- In the **Event Graph**, set the Event Cursor by clicking on a specific event.
- In the **Event Table**, right-click any event and select the **Set cursor** entry from the context menu.
- Open the **Event Properties/Search (***filename***)** dialog, as described in *Using the Event Properties/Search (filename) Dialog*, p.73. This dialog allows you to search for specific events within your log file. As an event is located which matches your search criteria, the event cursor is moved to that event and the selected range is adjusted appropriately.

The Event Cursor appears in the Radar scale. If you hover the mouse on the Event Cursor, wherever it appears information about the corresponding event appears in the status bar. To delete the Event Cursor, click it in the **Event Graph**. Alternatively, you may click in any blank area in the **Event Graph**, but this will remove both the event cursor and any measurement markers that were defined. For more information, see *10.2.5 Using Measurement Markers*, p.67.

Using the Event Properties/Search (*filename*) Dialog

The **Event Properties/Search (*filename*)** dialog displays all known information or properties about an event, and allows you to search for other events in the log which satisfy your search criteria.

To open the **Event Properties/Search (*filename*)** dialog, select **View > Event Search**, double-click any event in the **Event Graph** or **Event Table**, or select the *event-name* properties item in the event's context menu. When initially opened the information displayed represents the properties of the event under the event cursor.

- **Event**

  A drop-down list of the event types that occur in the active log (after filtering operations).

- **Container**

  A drop-down list of the contexts that occur in the active log after filtering operations, and in which the event can occur.

- **Object**

  An edit box displaying the ID of the object related to the event currently selected (for AIL logging), if relevant.

- **Time**

  Displays the timestamp at which the selected event occurred.

- **Parameters**

  Displays the parameters for the currently selected event.

The search operates on the filtered log, after you perform event or container filtering, and any events, state changes, or containers filtered are not included in a search. For more information about filtering, see *12.2 Display Filtering Options*, p.80.

The **Lock** checkboxes limit your search to events occurring on the currently displayed parameters. As you select **Next** or **Prev**, the Event Cursor is set on the next or previous event that fits your search criteria. These buttons are enabled only if an event exists.

Moving to the Event Cursor

The current location of the Event Cursor is always displayed in the Radar's scale area. Depending on the implementation, it may also be displayed by a viewing tool in the log viewer. Wherever the Event Cursor is located, you can recenter the selected range on it by choosing **View > Move to Event Cursor** or by clicking on the corresponding toolbar button. Recentering respects the bounds of the event log, with the exception described below.

Creating or moving the Event Cursor in a tool window does not automatically change the currently selected range. Changing the selected range, by whatever means, does not effect the location of the Event Cursor. Thus, the Event Cursor can be located on an event that does not lie within the currently selected range. The exception to this is when using the **Event Properties/Search (***filename***)** dialog. When searching for events using the **Prev** and **Next** buttons, the tools scroll, as appropriate, to ensure the Event Cursor is visible. In addition, if an event search leads you to a location outside the currently selected range, the selected range is recentered around the Event Cursor.

Zooming on the Event Cursor

If you choose to zoom in on a range that contains the Event Cursor, the zoomed in range is adjusted to always contain the Event Cursor. If the zoom sends the Event Cursor outside the selected range, the new range is recentered on the Event Cursor, as subject to the bounds of the event log.

11.3 Using Bookmarks

Bookmarks are user-defined markers used to identify interesting locations in the event log. They appear in the Radar's scale region and also in the viewing tools. Bookmarks can be defined at any timestamp within the range of the event log. Optionally, they can be named and given a comment of arbitrary length. Hovering the mouse over a bookmark displays a tooltip showing its timestamp name and attached comments. All bookmark information is stored to an analysis **.wva** file with the associated event log data collected when an event log is saved.

11.3.1 Creating Bookmarks

You create a bookmark using the **Add Bookmark** dialog. You can access this dialog in several ways:

- Right-click to display a context menu and select **Add Bookmark**
- Select **Bookmarks > Manage bookmarks** and then select **New** from the **Bookmarks Maintenance** dialog
- Select the **Bookmark** button from the **Event Properties/Search (***filename***)** dialog.

Bookmarks are always based on a timestamp. If the **Add Bookmark** dialog was opened from a context menu, **Timestamp** is automatically filled in with the timestamp corresponding to the location at which the context menu was displayed. If no timestamp is automatically filled in, enter one that lies within the bounds of the event log.

When you create a bookmark, the timestamp is always aligned to a timestamp tick as described in *Timestamp Ticks*, p.48. Once you create a bookmark, it appears as a small, blue triangle in the Radar's scale area and in the viewing tools that display bookmarks.

All bookmarks appear in the Radar's scale area. As the **Event Graph** displays only the current range, only the three bookmarks that occur within the selected range are displayed.

Bookmarks are defined at timestamps rather than linked to events, therefore a bookmark can be defined at a timestamp where no event exists. If a bookmark appears on a line between events, this indicates that it was positioned at a timestamp that has no corresponding event.

The **Event Table** allows you to see bookmarks for which there is no row containing the bookmark's timestamp by displaying it between the appropriate rows.

If there are bookmarks in the selected range that exist before the timestamp of the first line in the table, the bookmark is displayed on the line above the first row. Similarly, if bookmarks exist in the selected range, but at a timestamp after the event shown in the last row, the bookmark appears on the line below the last row.

As bookmarks are set on timestamps rather than events, multiple bookmarks *can* appear, even when only one is created. For example, if several events with the same timestamp are displayed in the event table, and a bookmark is set against one of them, a bookmarks is displayed at each of those events in the table

When you point to a bookmark and hover over it with the mouse, Wind River System Viewer displays all information attached to that bookmark.

All defined bookmarks appear in the Radar's scale region, since it represents 100% of the time range of the event log. For tools displaying only the currently selected range, bookmarks defined outside of this range, do not appear in the tool.

It is possible to define bookmarks so close in time that they are indistinguishable, particularly at a coarse zoom level. In this case, they appear on top of one another. To accurately view individual details of such bookmarks, use the **Bookmark Maintenance** dialog.

11.3.2 Using the Bookmark Maintenance Dialog

All the bookmarks you define can be managed in one central place, the **Bookmark Maintenance** dialog. To open this dialog, select **Bookmarks > Manage Bookmarks** or the corresponding toolbar button. This dialog contains a table showing all currently defined bookmarks. Each bookmark is described with a timestamp, and a name that you define.

You can perform the following actions, from the **Bookmark Maintenance** dialog:

- **Sort by Column**

  To sort the contents of the bookmark table on a particular column, click a column header and the table sorts to that column. Click again, and the order of the sort is reversed.

- **Display Comments**

  To display the comment associated with a bookmark, select that bookmark in the table.

- **Create a Bookmark**

  To add a bookmark, click the **New** button and fill in the **Add Bookmark** dialog, as described in *11.3.1 Creating Bookmarks*, p.75. Click **OK** or **APPLY** to apply your changes.

- **Modify a Bookmark**

  To edit a bookmark, select a bookmark in the table, then click **Edit**, which opens the **Edit Bookmark** dialog. Modify any of the bookmark information, including adding a comment. Click **OK** to apply your changes.

- **Delete a Bookmark**

  To delete one or more bookmarks, first select any number of rows in the table. Then, click **Delete**. The bookmarks disappear from the table immediately. Click **OK** or **APPLY** to apply your changes.

- **Apply Recent Changes**

  To apply **New**, **Edit**, or **Delete** changes, click **Apply** or **OK**. Any changes that have not been finalized appear in bold, italic font. **Apply** applies the change, but leaves the dialog open. **OK** or **APPLY** applies the change and closes the dialog.

- **Cancel Recent Changes**

  To cancel and changes, which have not yet been applied, that is they will be in in italic, click **Cancel**. Even after you click **OK** in the Add or Edit Bookmark dialogs, you can still select **Cancel** from the **Bookmark Maintenance** dialog to cancel your edit.The **Cancel** button only applies to changes made since the dialog was opened and only cancels changes since the last **Apply** was clicked.

- **Center the Display on an Existing Bookmark**

  To navigate to a defined or existing bookmark, select the bookmark in the table and click **Go to**. This immediately recenters the main display on the selected bookmark, subject to the bounds of the event log.

- **Center the Display on an Incomplete Bookmark**

  To navigate to a newly created or edited bookmark, that has not been applied, that is it is still shown in italics, select the bookmark and click **Go to**. This performs an implicit **Apply** on the selected bookmark row, and then performs the **Go To** action.

11.3.3 Using the Bookmark Context Menu

Right-click a bookmark to display it's context menu. Options on this menu are:

Edit
Opens the Edit Bookmark dialog

Delete
Immediately deletes a bookmark

Go to
Immediately recenters the selected range on the bookmark, subject to the boundary of the event log.

11.3.4 Changing a Bookmark's Timestamp

You can edit an individual bookmark timestamp by dragging it. When you select a bookmark in the Radar or the **Event Graph**, the cursor changes to a hand if that bookmark is movable. If it is, press the left mouse button, and drag it anywhere, within the region in which it is displayed.

For example, in the Radar scale region, you can drag a bookmark anywhere in the event log. In the **Event Graph** you can drag a bookmark anywhere in the selected range. This method of editing a bookmark changes *only* the timestamp of the bookmark, leaving its optional name and comment unchanged.

As a bookmark is dragged, all other representations of that same bookmark are updated simultaneously wherever possible. Otherwise, as with the **Event Table**, the display is updated after the bookmark is dropped. For example, if you drag a bookmark in the Radar's scale region across and through the selected range, the bookmark appears in the **Event Graph** only while being dragged through the range displayed in the **Event Graph**.

11.3.5 Navigating Between Bookmarks

You can navigate between bookmarks using the **Previous bookmark** and **Next bookmark** items on the **Bookmarks** main menu, or the corresponding toolbar buttons. Navigating in this manner recenters the selected range on the previous or next bookmark, subject to the bounds of the event log. The Previous/Next bookmark menu items and toolbar buttons are dynamically enabled depending on the distribution of bookmarks you define relative to the currently selected range.

12

Using Display Filtering and Context Menus

12.1 Introduction

Wind River System Viewer allows you to filter information displayed to focus only on specific tasks and events that you want to study at a given time. Filtering options are available from the main menu and in some viewing tools, from the context menus.

Each of the standard viewing tools has a context menu to filter and display options, and information dialogs. These menus also provide access to the online Help. Some context menu items have counterparts on the main menu that apply across other viewing tools, while other context menus are uniquely tailored to a particular pane of a particular tool.

NOTE: Filtering only affects the displayed information, not the actual data stored in the log. To change actual information that is logged, change the event logging level or the event libraries for which you enable logging. This is done prior to starting logging, using the **System Viewer Configuration** utility.

12.2 Display Filtering Options

Wind River System Viewer provides two options for filtering, container and event class filtering.

Container Filtering allows entire containers to be removed from the event container tree and thus from the display. Event Class filtering allows groups of event types to be removed from the display. For example, in VxWorks, removing the Semaphore Events class removes all **semGive**, **semTake**, **semCreate**, **semDelete**, and so on events from the display.

NOTE: When any event filtering is applied, all viewing tools respect this filtering. In particular, the event profile displayed in the Radar adjusts accordingly and the Wind River System Viewer log viewer utility indicates filtering is applied when either or both of the two small square areas in the status bar are populated with an appropriate Event Class and/or Container filtering icon.

Filtering may be applied using the appropriate filtering dialogs:

- **Hide/Show Containers** allows you to select the container events for display. For more information, see *12.2.1 Hide and Show Containers*, p.81.
- **Filter Events** dialog allows you to select the classes of events displayed. For more information, see *12.2.2 Filter Events*, p.81.

Both the container and event filtering are data-driven. The **Hide/Show Containers** event container tree is generated from the containers that exists in current log. The **Filter Events** lists all event dictionary-defined event classes for the target operating system. For more information, see *12.7 Event Dictionary Online Help*, p.88.

12.2.1 Hide and Show Containers

The **Hide/Show Containers** dialog depicts a tree of all containers or contexts in the log file.

Checking the associated box in the tree displays all events for a corresponding container in the viewing tool. Unchecking the box filters out all events in the container. The **Hide Inactive** button unchecks containers that do not have events over the course a log.

12.2.2 Filter Events

The **Filter Events** dialog contains a checkbox list of all event classes in a log file.

When a large numbers of events types that are typically generated in high volume, such as interrupt and network events are viewed at low zoom levels, they slow the redrawing of the viewing tool option and can also obscure other events. You can unclutter the display by filtering out event classes of no interest.

Checking the associated box displays all events of that class. Unchecking the box filters or hides all events of that class. Event classes indicated here correspond to the **Event Class** column in the **Event Table** and to the **Class** subfolder in the **Containers** column of the **Event Distribution**.

12.3 Container Tree Context Menu

You access the container tree context menu by right-clicking a node in the event container tree of any tool that uses the tree. The filter settings are persisted across and affect all tools that use the filtering feature. When you filter items, the status bar displays the associated filtering icon so you are aware of the container or event classes that were filtered on a log.

12.3.1 Context Menu Items

The container tree context menu items allow you to filter containers and to obtain information about the log. For more information on container filtering, see *12.2 Display Filtering Options*, p.80. The container menu items are listed and described below:

- **Hide**

  Hide the selected container(s) from the display.This feature is not enabled for the root node.

- **Show All**

  Causes all containers to be displayed, reversing any previous "hide" actions. This feature is the same as clicking **Select All** in **Hide/Show Containers** and then **Apply**.

- **Show**

  Opens **Hide/Show Containers** as described in *12.2.1 Hide and Show Containers*, p.81.

- **Hide Inactive**

  Hides all containers which contain no events. This feature corresponds to clicking **Hide Inactive** in **Hide/Show Containers** and then **Apply**.

- **State Summary**

  Opens the **State Summary** dialog described in *State Summary Dialog*, p.83.

- **Properties**

  When selected from the root of the container tree, this opens the Log Properties dialog. When selected from any other node in the tree, opens the Container Properties dialog.

State Summary Dialog

The **State Summary** dialog displays statistical information about event states for the currently selected container(s) over the course of a certain time interval. To access the dialog, select the container node (or nodes of interest), and choose **State summar**y from the context menu.

State information relating to the selected containers over the defined time interval is displayed in columns in the lower portion of the dialog. Note that **State** lists all states that the selected contexts are in during the relevant interval. The possible states are any of the states listed in the help legend. For information on individual states, see the event dictionary in the *Wind River Workbench User Interface Reference: System Viewer Event Dictionary*.

The columns display information for selected container(s) over a time interval determined by the radio button you select. These options are:

- **Selected Range**

  The state analysis is performed over the time period defined by the current selected range.

- **Measured Range**

  The state analysis is performed over the time period specified by the currently measured range (if any). To define a measured range, refer to *10.2.5 Using Measurement Markers*, p.67.

- **Whole Log**

  The state analysis is performed over the entire log.

The **State Summary** dialog dynamically updates information, so it is possible to change the basis on which the analysis is performed, even while the dialog remains open. For example, while the dialog is open, you can click in the event container tree to change which containers are used for the analysis, you can change the selected range using any of the various methods available, or you can create a new (or edit an existing) measured range.

Log Properties Dialog

The **Log properties** dialog displays the conditions under which a log is generated.

Once a log is opened, you can review its properties by selecting **File > Properties** or **Properties** from the context menu of a root node container. Properties include the event logging level, the clock frequency, and the target BSP. The setting for each **Property** is shown under the **Value** column. You can also add information to the **Log Comment** text area, which can them be saved with the log file.

12.4 Event Graph Context Menu

Items on this menu allow you to hide and show events and containers, access details about events and states, filter the display of information, add bookmarks, and print information.

For more information on the container tree context menu, see *12.3 Container Tree Context Menu*, p.81.

NOTE: The Event Graph context menu may have different entries depending on the target operating system from which the event log was captured.

Some menu items have corresponding options on the main menu. In these cases, the menu item cross-references the main menu documentation. In particular, the zooming options correspond to the **Zoom** items on the **View** menu. For more information on zooming, see *10.2 Changing the Selected Range Using the Radar*, p.64.

The remaining context menu items are:

- **Context state**

  Opens the **Context State Information** dialog, described in *Context State Information Dialog*, p.86. This menu item is only enabled if you have right-clicked on or near a state stipple.

- *Nearest Event* **properties**

  Opens the **Event Properties/Search (***filename***)** dialog, which is described in *11.2.1 Setting the Event Cursor*, p.72, for the event nearest to the mouse pointer position. If there are no nearby events, the menu entry is disabled. As a

by-product of opening the **Event Properties/Search (***filename***)** dialog, the event cursor is always set to the focused event.

- *Nearest Event* **help**

  Opens the event dictionary, described in *12.7 Event Dictionary Online Help*, p.88. This menu item is disabled if there is no nearby event.

- **Add bookmark**

  Opens the **Add Bookmark** dialog, automatically entering the timestamp of the event at which the menu was opened. For more information, see *11.3.1 Creating Bookmarks*, p.75.

- **Print**

  Corresponds to **File > Print** and prints that portion of the log that appears in the graph pane.

- **Show gridlines**

  Displays a white vertical line at each marked time interval. For more information, see *8.4 Reading and Configuring Timestamping*, p.48.

- **Show event focus hint**

  Displays a white marker directly underneath the event that currently has focus in the event graph, that is the nearest event. When the context menu is brought up, the menu items on it pertain to the event that has focus. It also shows the event for which detailed information is displayed in the status bar.

- **Show Task Transitions**

  Toggles the display of lines that connect a previous running task to the current running. This setting controls only the display of task-task transitions.

- **Show Interrupt Transitions**

  Toggles the display of lines that connect a task to an interrupt.

NOTE: Show Task Transitions and Show Interrupt Transitions relate to VxWorks logs only. For other target operating systems, the menu entry (or entries) may say something different.

Context State Information Dialog

Selecting **Context State**, from the context menu of the **Event Graph**, opens the **Context State Information** dialog. This dialog displays information that applies to the context nearest the mouse click that brought up the dialog.

Statistics are provided about the state of the context at the time interval where the mouse click occurred, when the context entered that state, and how long it remained in that state.

12.5 Event Table Context Menu

You can access the **Event Table** context menu by right-clicking the table pane or the column headings. Both the table pane context menu and the column headings context menu are documented in this section. The container tree context menu is documented in *12.3 Container Tree Context Menu*, p.81.

12.5.1 Table Pane Context Menu

Items on this menu allow you to manipulate the Event Cursor, search for an event, count events, add bookmarks, and save the log. Some menu items have corresponding options on the main menu.

- **Set Cursor**

  Sets the Event Cursor on the selected event, as described in *11.2.1 Setting the Event Cursor*, p.72.

- **Move to Event Cursor**

  Centers the selected range on the Event Cursor, as described in *Moving to the Event Cursor*, p.74.

- **Add bookmark**

  Opens the **Add Bookmark** dialog, automatically entering the timestamp on the interval at which the menu was opened as described in *11.3.1 Creating Bookmarks*, p.75. In the **Event Table**, bookmarks appear in the left columns of the rows for the events for which they are set. For more information, see *11.3.2 Using the Bookmark Maintenance Dialog*, p.76.

- **[*event-name*] properties**

  Opens the **Event Properties/Search (*filename*)** dialog described in *11.2.1 Setting the Event Cursor*, p.72.

- **Count events**

  Lists the number of events that exist within the time interval of the current Radar range. The event count and the rest of the event table are based on only those nodes selected in the container tree.

- **Export**

  Opens the **Export as Text** dialog.

12.5.2 Column Headings Context Menu

Items on this menu allow you to hide and show columns.

The **Manage Columns** item opens the **Column Manager** dialog in which you can select columns to hide or show. Using this dialog you can also multi-select, hide or show columns all at once. To hide a single column, open the context menu on that column and select **Hide this Column**. To show all columns, select **Show All**. Finally, you can toggle the display of columns by clicking from the list on the menu.

Lower Pane Context Menu

The context menu for the lower pane of the **Event Table** provides options for working with the text in that pane. Note that **Print** prints the entire contents of the pane. When specifying **Print Setup**, note printing is set to scale, so the text width fits in one page and scrolls vertically, over as many pages as needed.

12.6 Event Distribution Context Menu

The **Event Distribution** context menu contains options for collapsing or expanding all sub-nodes of any container.

The context menu is only available on actual containers, not on non-container sub-nodes. Each visible node displays a total event count.

12.7 Event Dictionary Online Help

The event dictionary is part of the online *Wind River System Viewer User's Reference*. It provides detailed information available about every possible event type for a target operating system. Entries in the event dictionary indicate the causes and consequences of an occurring event, and the information logged about the event at each level of logging.

Wind River System Viewer's online help resources and the Event Dictionary require Wind River Workbench to be running. If Wind River System Viewer is running without Wind River Workbench, an error dialog appears on any attempt to access the online help.

The context-sensitive help is always be enabled. If System Viewer cannot communicate with the online help, an error dialog appears indicating the help system is not available. If the requested help topic does not exist, an error dialog appears indicating the help topic does not exist.

12.7.1 Accessing the Event Dictionary

You can access the event dictionary by opening the Wind River Workbench online help library and navigating to the *Wind River System Viewer User's Reference*.

There are also several ways to get help from a context menu. These options each provide a shortcut to the entry in the event dictionary for that event:

- **Help**

  There are numerous **Help** buttons throughout System Viewer. Clicking any of them opens the appropriate section in the online *Wind River System Viewer User's Guide*. Alternatively, pressing F1 accomplishes the same task.

- **Event Help**

  Detailed documentation on an event may be obtained by right-clicking that event's icon and selecting the help menu item from the event's context menu. This applies in the **Event Graph**, the **Event Table** and the **Legend**.

13 Using Triggering

13.1 Introduction

Triggering is an operating system feature that uses instrumented events, mostly found at the same point in the code as Wind River System Viewer events. As an alterative to the **System Viewer Configuration** utility, you can use triggering to start and stop the logging process. More importantly, triggering allows you to precisely control when and how to start and stop logging using instrumented events.

Most events that you can log can also be written to activate a trigger. Triggering, specifies an action to be performed when a trigger is hit. You select the actions for triggers that include controlling how Wind River System Viewer logs events.

NOTE: If your system is configured with triggering support, each time an instrumented point is hit, the operating system checks whether triggering is enabled. For more information, see *B. Triggering API*.

Wind River System Viewer provides a triggering interface that manages all aspects of triggering. Using the triggering interface you can create and run sample triggers and define them using various functions. Triggering API is also provided so you can use VxWorks triggering in conjunction with Wind River System Viewer logging.

Sample triggering files are available for your use in *installDir*/**workbench-***N.N*/**wrsv/***N.N*/**samples/vxworks***N*/**triggering**. For information, see *B. Triggering API*.

NOTE: Wind River System Viewer Triggering is not currently available for the Linux operating system. Information on creating Linux custom events is described in *15.3 Linux Custom Events*, p.140.

13.2 Getting Started

Wind River System Viewer provides a triggering interface so you can support, define, configure, save, validate, download and run trigger files.

Learn how to use triggering by performing the following the steps as per your development need and preference. Using each set of procedures, you will learn how to use the triggering interface, create and use basic triggers in coordination with log files, and the trigger functions in general.

13.2.1 To Create a Trigger

1. Define a trigger specification by defining the conditions to be checked when the trigger's specification is matched.
2. Define the action to be taken when the triggers fires and specify the trigger to enable once the trigger has fired.
3. Save the trigger to a file if required.

4. Prepare your target before downloading the trigger by declaring all objects, variables and functions on the target before a trigger can be uploaded and triggering starts.

13.2.2 Using Sample Trigger Files

To understand simple conditional and chained triggers, create and run the sample triggering examples in *installDir*/**workbench-***N.N*/**wrsv/***N.N*/**samples/vxworks***N*/**triggering**.

1. Re-create and run the simple conditional triggering example described in *13.4.1 Simple Conditional Trigger Example*, p.100.
2. Extend the simple conditional trigger to include a trigger that is chained to it, as described in *13.4.2 Chaining Simple Conditional Triggers Example*, p.102.

Understanding Functions with Triggering

As described in *13.5 Using Functions with Triggering*, p.108, Triggering allows you to use a function as a condition or as an action.

1. You can create a trigger defined with a **Condition on Function**. For more details, see *13.5.1 Using a Function as a Condition*, p.108.
2. You can create a trigger defined to use a function as an action. For more details, see *13.5.2 Writing a Call Function as an Action*, p.110.
3. Finally, you can create and upload triggers using the **Triggering** utility or the triggering API. For more information, see *B. Triggering API*, which includes an example for Wind River System Viewer logging described in *13.5.3 Starting and Stopping System Viewer with User Events*, p.111.

NOTE: An application loads events from the required dictionary, therefore if you open triggering against a live target and the target dies, you can still use triggering to create and edit triggers even with the target down.

13.3 Using Triggering

The **Triggering** utility is used to create and use triggers. It can be launched from a selected target running within the **Target Manager** view.

The **Triggering** utility depicts each trigger in table format with columns categorizing attributes of the trigger. When you first open triggering, Wind River System Viewer scans the target for triggers, and displays the ones it finds in the table.

13.3.1 Menu and Toolbar Options

File Menu

Items on the **File** menu allow you to:

- open and save triggers as **.trig** files
- set up your printer and print triggers in tabular form
- exit triggering
- import a trigger file generated from previous System Viewer (WindView) versions. For more information, see *13.6 Importing Previous Version Trigger Files*, p.113.

Trigger files have the **.trig** extension in Wind River System Viewer. These files contain definitions for triggers you create. To load a **.trig** file, choose **File > Open** and select the **.trig** file.

NOTE: File > Import should only be used to load WindView 2x and WindView 3x format trigger files into the Triggering interface. **File > Open** should only be used to load Wind River Workbench generated trigger files.

To view a trigger definition, choose the trigger from the list and double-click it or open it from the context menu. When you choose **File > Save As**, all triggers currently in the triggering list, regardless of what is highlighted, are saved to the file you specify in the **Save As** dialog. Similarly, when you choose **Print**, all triggers in the list are printed.

Edit Menu

You can create new triggers, edit, rename, delete, or duplicate existing triggers using the main **Edit** menu. To enable these features, select a specific trigger. If you do not select a trigger, only the **Edit > New Trigger** item is enabled.

The **Edit** context menu appears when you select a trigger and right-click. Before you validate and download a trigger, the text in some of the columns and in the **Trigger Maintenance** utility may display in red. For more information, see *13.3.5 Defining Variables to Validate Triggers*, p.98.

View and Action Menus

The **View** menu allows you to refresh the trigger state. From **Actions** you can start or stop triggering.

13.3.2 Columns in the Trigger Utility

The column headings list the triggers and describe their attributes. For example, once a trigger is downloaded, **Hit Count** shows how many times a trigger has fired and **Status** shows its current status. For more information, see *13.3.6 Downloading and Running Triggers*, p.98.

You manage the columns that appear and are hidden using the **Column Manager** dialog. This feature allows you to quickly view the current information on any column. To access this **Column Manager** dialog:

1. Right-click the column header on the Triggering utility to display the context menu. This menu lists all visible columns indicated with a checkmark so you know which columns are hidden and shown.
2. Select **Manage Columns**.
3. Select a column type, **Hide>>>** and **<<<Show** respectively to choose which columns appear and are hidden on the **Triggering** utility.
4. Select **OK** to return to the **Triggering** utility and view the columns.

13.3.3 Using the Trigger Maintenance Utility

You specify new triggers, edit triggers, set up conditions and actions for simple and chain triggers using four panes in the **Triggering Maintenance** utility.

To Create a Trigger

Step 1: Open the Trigger Maintenance Utility

A trigger with a default name is presented in the **Trigger Maintenance** dialog. A default name is used, however, it is advisable to specify unique names for triggers.

Step 2: Select Attributes to Define the Trigger

Specification contains drop-down list boxes from which you specify the criteria for defining a trigger. The **Context**, **Event**, and **Object** fields correspond to those used by the **Event Properties/Search (***filename***)**.

1. Select **Trigger is initially enabled** if you wish your trigger to be enabled when you download it. Wind River System Viewer enables this feature by default. If a trigger is chained, that is enabled by an earlier trigger or started from code, uncheck the checkbox.
2. On VxWorks 653 only: From D**omain**, select the domain in which the event can occur that causes the trigger to fire. Any variable or function that you specify as a condition or action must be visible from the domain selected for this field. If it is not, the system issues an exception. The option, **Any Domain**, restricts the use of variables and functions, restricts the types of action options, and disallows conditions altogether.
3. From **Context**, specify the context in which the event that causes the trigger to fire occurs. The event can occur in any context, including the system context. The **Any** or **Any Interrupt** options disable the **Call Function** option that you can select from the **Action** drop-down listbox.
4. From **Event**, specify the type of event that causes the trigger to fire. The event can occur in any specific task, or in any ISR (interrupt service routine).

   When **user#** or **intEnt** is selected from the **Event** drop-down list box, you can enter a identification number in the **#** text box. This is a user event created with the **trgEvent( )** routine. For more information, see the reference entry for **trgEvent( )** and *Creating a User Event to Fire a Trigger*, p.161.
5. From **Object**, select the object ID of all objects of the type specified in the **Event** listbox. For example, if you selected **semGive** or **semTake** as your event for the trigger, the **Object** listbox displays any semaphores on the target. If you created a semaphore on the shell as in,

   ```
   % mySem = semBCreate()
   ```

   type **mySem** in the listbox and triggering finds the appropriate semaphore for you.

6. Check **Disable trigger after firing** if the trigger should be disabled after it fires. System Viewer enables this feature by default. If you do not check this box, your trigger remains active to continue firing when conditions are met.

Step 3: Select Conditions for the Trigger to Fire

You can define a trigger to isolate an event as close as possible to the start of a sequence of interest. Once you identify unique events preceding a sequence, they can then be used as potential trigger points.

For example, you may observe that, shortly before the sequence of interest, a semaphore is given. It might be possible to identify the task context from which the **semGive( )** is performed and possibly the ID of the semaphore being given. These criteria can be used to define when the trigger that initiates logging fires.

Sometimes the criteria used to define the trigger are not sufficient to uniquely detect the start of the region of interest, if a task takes and gives the same semaphore more than once. To uniquely detect the start of the region of interest, you can refine the trigger by using a conditional qualification. For example, you can examine a variable (by symbol or address) for a specific value or range of values, or you can invoke a specific function and test the result.

The **Condition** pane contains a drop-down listbox for defining whether or not a condition is evaluated, as well as two text boxes and an operator listbox for specifying a condition to be evaluated. The text box on the left is for a function or variable name, and the text box on the right is for a constant value.

To make a trigger conditional upon a variable or function that takes a specific value or range of values, follow these steps and ensure that the correct types are used:

1. Change **Unconditionally** to **Conditional on Function** or **Conditional on Variable**. The two text boxes are enabled when **Unconditional** is not selected.
2. Enter the function or variable name in the first edit box. The function or variable name must be a known 32-bit identifier on the target.
3. Select the matching criterion by choosing one of the operators from the drop-down menu (==, <=, <, and so on).
4. Enter the constant value to test in the second text box. The value must be a 32-bit integer constant in either decimal or hexadecimal format.

The criteria specified for event, context, and object, conditions are tested for on the target. If the criteria are met, triggering proceeds. For triggering examples that use conditions, see *13.4.1 Simple Conditional Trigger Example*, p.100 and *13.5.1 Using a Function as a Condition*, p.108.

Step 4: Select Actions to be performed when the Trigger Conditions are Met

Wind River System Viewer performs actions on a trigger when that trigger is set and the event occurs. You define trigger actions and their range from controlling Wind River System Viewer to performing user-specific requests. These actions are performed when the specification criteria are met, and any specified condition is matched.

Select actions for the trigger as follows:

No Action

Select **No Action** when no action is taken on the trigger. For example, you can use this option to:

- fire the first in a sequence of chained triggers, where the first trigger has no other purpose than to enable another trigger.
- determine a specific event has occurred when the associated trigger fires. The trigger is then disabled after it fires, and you can examine the **Triggering** utility to see when it has fired.
- use as a counting mechanism. If the trigger is not disabled after firing, you can update the **Triggering** utility using the **View** menu and see a count of the number of times the trigger has fired under **Status**. For more information, see *13.3.6 Downloading and Running Triggers*, p.98.

System Viewer Logging Actions

Select **Start System Viewer Logging** to start logging when the trigger fires and **Stop System Viewer Logging** to stop logging when the trigger fires. For examples that use triggers to start and stop logging, see *13.4.3 Chaining Triggers for System Viewer Logging Example*, p.105 and *13.5.3 Starting and Stopping System Viewer with User Events*, p.111.

Call Function

Select **Call Function** to call the specified function on the target with the given integer parameter. Selecting **Call Function** enables two text box fields, where the first field specifies the function name and the second field specifies an

integer argument to that function. For more information, see *13.5.2 Writing a Call Function as an Action*, p.110.

Taskstop

Select **Taskstop** to stop the task which is currently executing. You can resume the task by using the taskResume comand on a shell.

Action Library (VxWorks 653 only)

Uses a library of functions that you set up (for use with triggering) from a shell, using the commands from the VxWorks 653 API library, **trgLib**. When **Action Library** is selected, only the first field editbox is enabled. It specifies the library function by index. The second field editbox simply *displays* the argument associated with it. This argument was provided when the function was added to the library. For more information about when to use an action library, see *13.5.4 VxWorks 653 Only: The Action Library Manager*, p.112.

Defer Action

Check **Defer Action** to defer the action until it is safe to execute the trigger action. For example, suppose you wanted to make sure your action function did not run within ISR or system context. In this case, check **Defer Action**, if the function contains calls to any function that is not permitted in ISR context.

NOTE: Triggering automatically spawns a task called **tActDef** that executes functions on the trigger work queue when the system is out of any critical region. If your application runs at a higher priority than **tActDef**, you may have to adjust the priority if you want the trigger action to be executed.

Step 5: Determine if the Trigger will be linked to Another Trigger

The trigger can be programmed to enable another trigger when it fires. The trigger that should be enabled can be specified in this text box. For more information, see the example in *13.4.2 Chaining Simple Conditional Triggers Example*, p.102.

Step 6: Click OK to return to the Triggering Utility

13.3.4 Saving Triggers

To save a trigger to a trigger file, choose **File > Save** or the corresponding toolbar button, and save the file with a **.trig** extension. Saving triggers allows you to store many triggers so they can be reloaded for re-use.

13.3.5 Defining Variables to Validate Triggers

Wind River System Viewer indicates valid triggers in black text. Before using a trigger, variables on which it depends, must be defined. Invalid triggers require that you define variables before they can be used.

In the trigger list, a red text item relies on a target requirement that does not exist. This is typically a variable that needs to be defined.

When you point to the invalid item and pause briefly for the error information, Wind River System Viewer indicates the reason why a red text item is invalid, through a tooltip. After correcting the error, you can update the trigger by clicking the refresh toolbar button.

After variables are defined and refreshed in the trigger list, the green **GO** toolbar button indicates the triggers are validated and ready to download.

13.3.6 Downloading and Running Triggers

You must download a trigger to the target before you use it by clicking **GO**. Once you download a trigger, the **Target** column indicates the status of that trigger, changing from **Not Set** to **ARMED**, **FIRED**, **COUNTING**, and so on, as the trigger runs.

Reading Target Icons

The **Target** column indicates whether a trigger is successfully downloaded to the target, resident on the host computer or in a changing state using icons as follows:

Host-only Triggers
This icon indicates the trigger is not downloaded to the target, and only stored in triggering. Host-only triggers are lost if the Wind River System Viewer launcher is closed on UNIX, or Wind River Workbench is closed on Windows, and you have not saved your trigger to a trigger file. Host-only triggers will remain only if triggering is closed and then re-opened.


Target-Resident Triggers
This icon indicates the trigger is resident on the target. Target-resident triggers still exist after you close and re-open triggering, that is if the same target is connected and you have not rebooted the target.


Trigger Changed on Target
This icon indicates the trigger changed on the host since it was downloaded to the target. Wind River System Viewer will update the target trigger the next time the trigger is started.

Reading Trigger Icons

Various icons in the **Trigger** column indicates the state of a trigger as follows:


Yellow
This icon indicates the trigger is on the host computer, valid and ready for download. The **GO** button is enabled.


Red
This icon indicates the trigger is on the host computer, but not valid and cannot be downloaded. The **GO** button is disabled.


Green
This icon indicates the trigger is downloaded to the target and ARMED. If you are running triggering, the trigger fires when the conditions for firing are met. If triggering stops before the trigger fires, this icon indicates that the last state of the trigger is ARMED.


Gray
This icon indicates the trigger is downloaded to the target, but not FIRED because it was initially DISABLED. If you are running triggering, the trigger does not fire until it is enabled. If triggering stops before the trigger is ARMED or FIRED, this icon indicates the last state of the trigger is DISABLED.


Gray with Check mark
This icon indicates the trigger is FIRED and now DISABLED. If you are running triggering, the trigger is ARMED, has FIRED and has been disabled because the **Disable after fire** option of the trigger was selected. The trigger remains in this state until you click **GO**.


Yellow with 123
This icon indicates a COUNTING trigger. The trigger is FIRED, but has not disabled because **Disable after fire** was not selected when the trigger was defined. If you are running triggering, the trigger is ARMED, has FIRED and

continues to fire each time its firing condition is met. To update the value in the **Hit Count** column, click **View > Refresh**.

13.4 Creating and Running the Sample Triggers

Wind River System Viewer provides three sample triggering files as follows:

helloGoodbye.trig
Trigger file that demonstrates the process of chaining in triggering which uses a combination of trigger files, each invoked from the other.

simpleCond.trig
Trigger file that demonstrates the use of a simple conditional trigger.

startStopwv.trig
Trigger file to start and stop Wind River System Viewer logging.

13.4.1 Simple Conditional Trigger Example

This example describes how to create a simple conditional trigger using **simpleCond.trig** in *installDir*/**workbench-***N.N*/**wrsv/***N.N*/**samples/vxworks***N*/**triggering** . This trigger file fires when variable **foo** takes the value 1. When that condition is met, **printf( )** is called with the value **helloString**.

You can choose to load the example trigger and begin using it, or recreate the trigger and save it under a different name. If you have not loaded **simpleCond.trig**, create and define the trigger as follows.

Step 1: Create and Define the Conditional Trigger

1. Open the **Trigger Maintenance** Utility.
2. Select **Edit > New trigger**.
3. Enter **hello** in **Trigger Name**.

4. From **Specification**, define the trigger as follows:
   - Check **Trigger is initially enabled**
   - Select **Default (VxKernel)** from **Domain**
   - Select **Any Task** from **Context**
   - Select **Any Event** from **Event**. The **Event Id** will be disabled
   - Select **Any Object** from **Object**
   - Check **Disable trigger after firing**
5. From **Condition**, define the trigger as follows:
   - Select **Conditional on Variable** from the drop-down listbox
   - Enter **foo** as the variable name
   - Choose **==** from the drop-down listbox
   - Enter **1** as the constant value
6. From **Actions**, define the trigger as follows:
   - Select **Call Function** in the drop-down listbox
   - Enter **printf** in the text box
   - Enter **helloString** in the argument text box
   - Check **Defer action**
7. As this simple trigger is not chained, from **Chaining**, select **None** from **Enable trigger**.
8. Save your trigger using a unique name, other than those used in the **samples** folder.

Step 2: Define the Variables and Validate the Trigger

1. Start the Host Shell and create the variable **foo**, as follows:

```
-> foo=0
new symbol "foo" added to "vxKernel" symbol table.
```

2. Create **helloString** as follows:

```
-> helloString="hello\n"
new symbol "helloString" added to "vxKernel" symbol table.
```

WARNING: The type entered into the argument text box, in this case **helloString**, must be a variable not an integer.

3. Select **View > Refresh**, or click the **Refresh** toolbar button to validate the trigger.

Step 1: Download and Run the Trigger

1. Click **GO** to download the trigger.
2. To fire the trigger, the condition for **foo** must be met. In the Host Shell, set **foo** equal to the value 1, as follows:

   ```
   -> foo=1
   ```

   The string "hello" prints out. The trigger status is updated to show that it has fired.
3. Click **STOP**.

13.4.2 Chaining Simple Conditional Triggers Example

The process of chaining in Wind River System Viewer ensures triggers are fired in a certain order.

Using **helloGoodbye.trig** in *installDir*/**workbench-***N.N*/**wrsv/***N.N*/**samples/vxworks***N*/**triggering**, this example shows you how to create a simple set of chained triggers that print out "hello" and "goodbye" based on the system variable **vxTicks**.

You can choose to load the example trigger and begin using it, or recreate the trigger and save it under a different name. If you have not loaded **helloGoodbye.trig**, create and define the trigger as follows.

Step 1: Create and Define the Hello Trigger

1. Open the **Trigger Maintenance** Utility.
2. Select **Edit > trigger**.
3. Enter hello in **Trigger Name**.
4. From **Specification**, define the trigger as follows:
   - Check **Trigger is initially enabled**
   - Select **Default (VxKernel)** from **Domain**
   - Select **Any Task** from **Context**
   - Select **Any Event** from **Event**. The **Event Id** will be disabled
   - Select **Any Object** from **Object**
   - Check **Disable trigger after firing**

5. From **Condition**, define the trigger as follows:
   - Select **Conditional on Variable** from the drop-down listbox
   - Enter **vxTicks** as the variable name
   - Choose **>** from the drop-down listbox
   - Enter **6144** as the constant value
6. From **Actions**, define the trigger as follows:
   - Select **Call Function** in the drop-down listbox
   - Enter **printf** in the text box
   - Enter **helloString** in the argument text box
   - Check **Defer action**
7. From **Chaining**, type **goodbye** in **Enable Trigger**. The text turns red because you have not as yet created the **goodbye** trigger. After you click **OK** to complete your trigger, notice that it appears in red text in the table.
8. Save your trigger using a unique name, other than those used in the **samples** folder.

Step 2: Define the Goodbye Trigger

1. Open the **Trigger Maintenance** Utility.
2. Select **Edit > New trigger**.
3. Enter **goodbye** in **Trigger Name**.
4. From **Specification**, define the trigger as follows:
   - Un-check **Trigger is initially enabled**
   - Select **Default (VxKernel)** from **Domain**
   - Select **Any Task** from **Context**
   - Select **Any Event** from **Event**. The **Event Id** will be disabled
   - Select **Any Object** from **Object**
   - Check **Disable trigger after firing** as this trigger is not initially enabled because it is chained to the **hello** trigger, therefore it becomes enabled once that trigger fires.
5. From **Condition**, define the trigger as follows:
   - Select **Conditional on Variable** from the drop-down listbox
   - Enter **vxTicks** as the variable name
   - Choose **>** from the drop-down listbox
   - Enter **6744** as the constant value

6. From **Actions**, define the trigger as follows:
   - Select **Call Function** in the drop-down listbox
   - Enter **printf** in the text box
   - Enter **goodbyeString** in the argument text box
   - Check **Defer action**
7. From **Chaining**, select **None** from **Enable trigger**.
8. Save your trigger using a unique name, other than those used in the **samples** folder.

Now that you have created the goodbye trigger, notice the hello trigger no longer appears in red. Triggering looks for matching trigger names or IDs, and if it finds them, it uses them. If not, it assumes the trigger will be created later, and it keeps the invalid trigger until the object that validates it is created.

This works for many boxes. If you have a semaphore **mySem**, you don't have to search the object yourself; just enter **mySem** in the **Object** listbox and it is matched if it actually exists.

Step 3: Define the Variables and Validate the Triggers

1. Start the Host Shell and create the variable **helloString**. as follows:

```
-> helloString="hello\n"
new symbol "helloString" added to symbol table.
```

2. Create the variable **goodbyeString**. as follows:

```
-> goodbyeString="goodbye\n"
new symbol "goodbyeString" added to symbol table.
```

3. Select **View > Refresh**, or simply click the **Refresh** button to validate the trigger.

Step 4: Download and Run the Hello and Goodbye Triggers

1. To download the trigger, click **GO**. **vxTicks** is a global variable that counts ticks. Trigger **hello** prints "hello" when **vxTicks** is greater than 0x6144.
2. Click **View** to refresh items or toolbar buttons until the **Triggering** utility shows that the hello trigger has fired. Once the **hello** trigger fires, the **goodbye** trigger is activated, and it prints "goodbye" when **vxTicks** is greater than 0x6744.
3. Click **Stop** to stop the triggering. Triggering stops automatically when there are no triggers left armed or counting.

NOTE: You can use the shell to find the current value of **vxTicks** and reset the values in the **Trigger Maintenance** Utility to be greater than the current value. Otherwise, one or both triggers fire immediately and the log you collect is almost empty.

13.4.3 Chaining Triggers for System Viewer Logging Example

To use triggering to collect a Wind River System Viewer log, you must define two triggers, one having the action to start log collection and one with the action to stop log collection. In most cases, triggers used to collect a Wind River System Viewer log are chained, since the order of their firing is critical.

Using **startStopwv.trig** in *installDir*/**workbench-***N.N*/**wrsv/***N.N*/**samples/vxworks***N*/**triggering**, you can create the sample triggers used to start and stop Wind River System Viewer logging.

You can choose to load the example trigger and begin using it, or recreate the trigger and save it under a different name. If you have not loaded **startStopwv.trig**, create and define the trigger as follows:

Step 1: Define the Start System Viewer Trigger

1. Open the **Trigger Maintenance** Utility.
2. Select **Edit > New trigger**.
3. Enter **startWV** in **Trigger Name**.
4. From **Specification**, define the trigger as follows:
   - Check **Trigger is initially enabled**
   - Select **Default (VxKernel)** from **Domain**
   - Select **Any Task** from **Context**
   - Select **Any Event** from **Event**. The **Event Id** will be disabled
   - Select **Any Object** from **Object**
   - Check **Disable trigger after firing**. Calling **startWV** more than once results in an error, so it is important, for a trigger that starts logging to disable it after it fires.

5. From **Condition**, define the trigger as follows:
   - Select **Conditional on Variable** from the drop-down listbox
   - Enter **foo** as the variable name
   - Choose **==** from the drop-down listbox
   - Enter **1** as the constant value
6. From **Actions**, once you have defined how the trigger will fire, select **Start Wind River System Viewer Logging** in the drop-down listbox. Although not required, it is often desirable to define a trigger to stop log collection. Doing so focuses the log on the sequence of interest and saves both resources and analysis effort.
7. From **Chaining**, type **stopWV** in **Enable Trigger**, if you decide to create another trigger to stop log collection.
8. Click **OK** to save this trigger. The **System Viewer Configuration** utility opens automatically, allowing you to review the current configuration, and change it if necessary.
9. If you are not creating a trigger that stops Wind River System Viewer logging, leave the **System Viewer Configuration** utility open so you can stop logging manually. Otherwise, the window my be closed.

Step 2: Define the Stop System Viewer Trigger

To create this trigger, use the same trigger file as for the **startWV** trigger.

1. Open the **Trigger Maintenance** Utility.
2. Select **Edit > New trigger**.
3. Enter **stopWV** in **Trigger Name**.
4. From **Specification**, define the trigger as follows:
   - Un-check **Trigger is initially enabled**
   - Select **Default (VxKernel)** from **Domain**
   - Select **Any Task** from **Context**
   - Select **Any Event** from **Event**. The **Event Id** will be disabled
   - Select **Any Object** from **Object**
   - Check **Disable trigger after firing**. This trigger is not initially enabled, but is chained to the **startWV** trigger, so that it cannot fire until **startWV** has fired.

5. From **Condition**, define the trigger as follows:
   - Select **Conditional on Variable** from the drop-down listbox
   - Enter **foo** as the variable name
   - Choose **==** from the drop-down listbox
   - Enter **2** as the constant value
6. From **Actions**, once you have defined how the trigger will fire, select **Stop Wind River System Viewer Logging** in the drop-down listbox.
7. From **Chaining**, select **None** in **Enable Trigger**, as the **stopWV** trigger does not, itself, chain to another trigger.
8. Click **OK** to save this trigger. Save both triggers to the same file.

Step 3: Define the Variables and Validate the Trigger

1. Start the Host Shell and create the variable **foo** as follows:

```
-> foo=0
new symbol "foo" added to symbol table.
```

2. Select **View > Refresh**, or click the **Refresh** button to validate the trigger.

Step 4: Download and Run the Triggers

1. Click **GO** to download the trigger.
2. Wind River System Viewer logging starts when the **startWV** trigger fires. To fire this trigger, the condition for **foo** for the **startWV** trigger must be met. When you are ready to begin Wind River System Viewer logging, enter the following in the Host Shell:

```
-> foo=1
```

3. Click the **Update** button.

   This starts Wind River System Viewer logging.

   Click refresh on the triggering utility toolbar or **View > refresh** from the menu. The trigger is now shown as disabled and the status is updated to show it has fired.

   Wind River System Viewer logging stops when the **stopWV** trigger fires. This trigger has now become enabled because it was chained to the **startWV** trigger, which has fired.

4. To fire the **stopWV** trigger, the condition for **foo** for that trigger must be met. When you are ready to stop Wind River System Viewer logging, enter the following in the Host Shell:

   ```
   -> foo=2
   ```

   This stops Wind River System Viewer logging.

NOTE: When a triggering event fires, it is logged as an event in the Wind River System Viewer log. A trigger can start or stop logging part way into a resulting log (for example, 25% into the resulting log). This facility is known as *pre-triggering* and results in a record of the events on either side of a trigger firing event.

13.5 Using Functions with Triggering

Triggering allows you to use a function as a condition or as a trigger action.

13.5.1 Using a Function as a Condition

You can write and use a function specifically to test a condition. The function used as a condition is executed during trigger evaluation if the criteria for event, context, and object conditions are satisfied.

One technique is to write a function that fires after a specified time period, for instance, by comparing the value of the system tick count against its value when the trigger to start the sequence fired. A conditional function cannot be deferred.

You must make the function available on the target and enter its function name as the condition item. The function is executed if all of the criteria defined for the trigger under **Specification** are met. Once the function executes, the trigger fires only when the function returns the value specified by the constant value.

When calling a function as a condition, it is very important that the condition is only to be used in conjunction with a specification that limits the number of times the condition is tested. For example, if a condition function to test a semaphore is written and is tested without a limiting specification, then the execution of the conditional function will cause more events which will each cause the condition function to be called. This nesting of calls will cause a race condition on the target and will eventually lead to the target crashing.

NOTE: If you use a function as a condition, the trigger **Specification** *must* define the **Event** as **user#**, and the **Event ID** as **10**. See also *13.5.3 Starting and Stopping System Viewer with User Events*, p.111.

Defining and Loading Condition Functions

The format and content of the condition function must follow the syntax below:

```
int  conditionFunction (void)
    {
    int returnValue;
     */ Function body */
    ...
    return (returnValue);
    }
```

1. Provide a valid conditional function as per the above example
2. Compile the function and load the resulting object module onto the target. For example, if a function **conditionFunction( )** is contained in a C module called **conditionFunction.c** and compiled into an object module **conditionFunction.o**, the object module can be loaded on to the target from the shell using the following command:

   ```
   -> ld < conditionFunction.o
   ```

3. Create a trigger that uses this function as a condition
4. Define the trigger with the function name and the value returned under the conditions you want your trigger to fire.

Writing Condition Function Code

The body of a function used as a condition cannot contain any function calls that are not permitted in an ISR context. This is because, depending on the criteria specified for event, context, and object conditions, the function provided could be executed within an ISR. Therefore, unless the combination of trigger specifications set in the event, context, and object fields guarantees that the function can only be satisfied in task context, (for example, by setting **Context** to **Any Task** or to a specific task) the body of the function can include only code (or function calls) that can be safely executed within an ISR or system context. For example, the following conditional function is invalid because **semTake( )** must not be used in an ISR.

```
int conditionFunction (void)
    {
    if (vxTicks >= (sysClkRateGet() * 60 * 60))
        {
        /* semTake NOT ALLOWED in ISR context*/
        semTake (mySem);

        /* printf NOT ALLOWED in ISR context*/
        printf ("The system has been up for more than an hour\n");

        return 1;
        }
    return 0;
    }
```

As trigger evaluation points are present throughout the VxWorks kernel, you can not use **printf( )** or **logMsg( )** in condition functions. The above example could be corrected by making the **printf( )** into a trigger action function and checking **Defer**, described in *13.5.2 Writing a Call Function as an Action*, p.110. For information about which routines can safely be called from ISRs, see the *VxWorks Application Programmer's Guide*.

You can also call a user-defined function that collects appropriate diagnostic information at the time the trigger fires. One way to do this would be to use **wvEvent( )** in user-defined function. For more information, see *The wvEvent( ) Routine*, p.147.

13.5.2 Writing a Call Function as an Action

A call function takes an integer argument and returns an integer value. Unlike a condition function, an action function can be deferred because the function is not executed to evaluate the trigger; instead it is added to a trigger work queue. For more information, see *Defer Action*, p.97.

An example of an action function is:

```
int myActionFunc (int arg)
    {
    printf("Trigger fired.\n");
    /* other code */
    return 1; (returnValue);        /* return value is ignored */
    }
```

To create a trigger that uses the function

1. Open the **Trigger Maintenance** Utility.
2. From **Action**, select **Call Function** from the drop-down list.

3. Enter the function name in the first text box and the integer argument to the function in the second text box.
4. Check **Defer Action** as **myActionFunc** calls **printf( )** for this function.

13.5.3 Starting and Stopping System Viewer with User Events

This example demonstrates how to start and stop System Viewer with a VxWorks family User Event and triggering. User Event IDs must be in the range of 40000-65535.

Step 1: Create a New Trigger with these Definitions

1. Open the **Trigger Maintenance** Utility.
2. Select **Edit > New trigger**.
3. Enter **startWV** in **Trigger Name**.
4. From **Specification**, define the trigger as follows:
   - Check **Trigger is initially enabled**.
   - Select **user#** from **Event**.
   - Enter an **Event ID** of 10 which is the equivalent of the user event number **40010**.
5. From Action, select **Start System Viewer Logging**.
6. Select **stopWv** from **Enable Trigger**.
7. Click **OK**. If you have not already opened the **System Viewer Configuration** utility, System Viewer opens this window, so you can setup configuration.

Step 2: Create a Second Trigger with these Definitions

1. Open the **Trigger Maintenance** utility.
2. Select **Edit > New trigger**.
3. Enter **stopWV** in **Trigger Name**. Note the upper-case V.
4. From **Specification**, define the trigger as follows:
   - Uncheck **Trigger is initially enabled**.
   - Select **user#** from **Event**.
   - Enter an **Event Id** of 20 which is the equivalent of the user event number **40020**.

5. From Action, select **Stop System Viewer Logging**.
6. Click **GO** to download the triggers.

Step 3: Create the User Events

1. Start the shell
2. Create a user event using the **trgEvent( )** routine and the **Event Id** for the **startWv** trigger:

   ```
   -> trgEvent(40010)
   ```

   Wind River System Viewer logging starts as soon as this event is created.
3. Click **Refresh** on **Triggering** to see that the trigger has fired.
4. Create another user event, in the same manner, but this time using the **Event Id** for the **stopWv** trigger as the routine argument:

   ```
   -> trgEvent(40020)
   ```

   Wind River System Viewer logging stops as soon as this event is created.

   Triggering stops automatically. Your triggers still remain on the target, but the triggering function is turned off. If you are using deferred upload mode, upload your Wind River System Viewer log using the upload button in the **System Viewer Configuration** utility.

   When you click **GO** to restart triggering, whatever configuration is currently in place on the host is downloaded and started. Even if it is the same configuration that was previously loaded, all triggers are reset as they were initially. Execution does not continue from where it was when you stopped triggering.

13.5.4 VxWorks 653 Only: The Action Library Manager

The Action Library Manager utility lets you easily maintain Triggering's target based action library list. See *Action Library (VxWorks 653 only)*, p.97 for more information about the Action Library.

The Action Library Manager utility displays the target's action library list in a table on the utility. The table shows each of the action functions and argument and also the index of the action in the list.

Actions can be added or removed from the list by using the buttons. On adding an action, a dialog prompting for the function and argument will be presented. In

order to be able to close the dialog with the **OK** button both the function and argument names *must* exist on the target.

NOTE: The argument provided for the function must be a variable or an address that contains the information for the function call. For example. the action library entry **doMyFunc 0xbadadd** will attempt to call **doMyFunc** with the value at address **0xbadadd**.

13.6 Importing Previous Version Trigger Files

Trigger files saved by WindView 2x and WindView 3x have different formats to the current trigger files. Although trigger files from previous versions of Tornado have a different format to the trigger files currently used, Wind River System Viewer treats a trigger file generated from Tornado in the same manner as a Wind River Workbench generated trigger file, and all of the triggering facilities apply.

However, you must *import* previous version trigger files, and save them in the current version of System Viewer. To import trigger files from previous versions of WindView, use the **File > Import** menu item.

To import previous version trigger files, use the **File** menu. If you use **File > Open**, the parser fails and an **Error parsing file** message appears.The **Import** facility allows you to convert and load old format trigger files. If you import a non Tornado.x trigger file using the import option or if you use the **File > Import** to open a Wind River Workbench generated file, the **Trigger Import Error** message appears.

When you import trigger files, the new triggers are created using a name and number based on the old file. For example, suppose you import from an old trigger file, **tor2.trg** which has two triggers. The two triggers are imported and named **tor2.0** and **tor2.1**.

An imported trigger with an unsupported action or event can be successfully imported, but it will be marked as invalid and appears in red text. It will remain invalid until all of the unsupported parameters are corrected.

14

User Events (VxWorks Family)

14.1 Introduction

System Viewer can display user-defined events, generally referred to as *User Events*. These are inserted into an event log as the thread of execution on a target passes over the corresponding customer-inserted instrumentation points.

Depending on the version and variant of the VxWorks Target Operating System (referred to as *TOS* in following) you use, details on how to add these instrumentation points will vary as described in documentation for that VxWorks version and variant. This chapter details how the presentation of these events is controlled in the System Viewer Log Viewer on a VxWorks family TOS.

NOTE: *User Events* as used by the VxWorks family of operating systems are not to be confused with Wind River Linux *Custom Events*. Although the purpose and output are essentially the same, input and internal handling differ.

14.2 User Event Display

All aspects of the definition of the structure of System Viewer events are encapsulated in a set of XML files which are specific to each supported TOS. User Events are no exception to this rule.

Changing the way in which User Events are displayed and decoded will involve editing these XML files, a process which must be performed with the utmost care. Injecting errors into an XML file will most likely render System Viewer inoperative.

⚠ **CAUTION:** Before changing *any* XML file shipped with System Viewer, you must ensure that you create a backup copy of that file and put it in a safe place, outside the Workbench installation tree. You can use these backup files to restore your installation to its original, functioning state should the need arise.

There are two levels of customization which can be performed on the display of User Events simply by editing the appropriate section of the supplied XML for your chosen TOS:

1. change the displayed icon
2. change the displayed text

In addition, the way in which any encapsulated data is formatted for display may be changed by writing custom formatters in Java and then making these formatters available to System Viewer at runtime. This is, however, an advanced technique which should be used only by experienced Java programmers, see *14.5 Advanced Techniques: Custom Parameter Formatting*, p.129. The default formatter provided in System Viewer should be sufficient for most needs.

14.3 The User Events Description File

Firstly, you will need to know the name and version of the TOS you are running on your target. In the example below, it is assumed you are using VxWorks 6.4.

14.3.1 Location of the User Events Description File

In your Workbench installation, locate the directory at

*installDir***/workbench-***N.N***/wrsv/***N.N***/host/resource/windview/**

Below this directory there will be subdirectories for all installed TOS versions. Locate the VxWorks variant and version which corresponds to your TOS or its nearest historical predecessor.

For example, for VxWorks 6.4, the XML dictionary definitions may be found at

*installDir***/workbench-***N.N***/wrsv/***N.N***/host/resource/windview/VxWorks/6.0**

Within that directory, the XML file which defines the structure and display characteristics of User Events is called **user.xml**. It is this file which must be edited (after keeping a backup of the original) in order to modify the way in which User Events will be displayed.

14.3.2 Structure of the User Events Description File

The outline structure of a System Viewer XML file is fixed and that format must be adhered to if the file is to be loaded correctly by the System Viewer runtime.

CAUTION: All System Viewer XML files must conform to a strict structure, or they will fail to parse correctly at runtime, and may render System Viewer inoperative. Before editing any System Viewer XML file, be sure to put a backup copy of the original file in a safe place (outside the Workbench installation tree) so that you can restore it should the need arise.

The structure of the file is as follows:

```
XML Header
Doctype header
<EventDictionary>             // start of top level document element
  <EventClass>                // start of EventClass element
    <EventRangeDecription>    // 0 or more EventRangeDescription elements
    <EventDescription>        // 0 or more EventDescription elements
  </EventClass>               // end of EventClass element
</EventDictionary>            // end of top level document element
```

The formal structure of a System Viewer event dictionary XML file is defined in its DTD (Document Type Description), which may be inspected at:

installDir/**workbench-***N.N*/**wrsv/***N.N*/**host/resource/windview/DTDs/EventDictionary.dtd**

⚠ **CAUTION:** Do not edit the above file!

By default, User Events occupy a single block of contiguous Event IDs. The content of **user.xml** for any TOS will reflect this in that all the corresponding XML descriptions of these events are grouped into a single **EventRangeDescription** element.

For example, here is the default **user.xml** file for VxWorks 6.0 (and higher) which demonstrates the above XML file structure and shows a single **EventRangeDescription** element which describes the entire User Event range:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EventDictionary SYSTEM "../../DTDs/EventDictionary.dtd">
<EventDictionary>
  <EventClass
    key="user"
    displayName="User Events"
    helpTopicId="VXWORKS6_user_CLASS_HELP">
    <EventRangeDescription
      eventIdStart="40000"
      eventIdCount="25536"
      nameRoot="EVENT_USER"
      displayNameRoot="user"
      nameRootSuffixStart="0"
      icon="images/defaultUser.gif"
      trigger="true"
      helpTopicId="VXWORKS6_EVENT_USER_EVENT_HELP"
     handler="com.windriver.windview.plugins.wv.vxworks.UserEventDescription">
      <EventParam
        type="UINT32"
        name="pc"
        formatStr="0x%08x"/>
      <EventParam
        type="BLOB"
        name="data"
       formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
    </EventRangeDescription>
  </EventClass>
</EventDictionary>
```

⚠ **CAUTION:** The **EventRangeDescription** element encapsulates information about the entire event range and how it should be displayed, as well as describing the structure of each event and its parameters. It is vital that the structure of the event (and the **EventParam** child elements) remains intact, since they reflect the data structure that will be created by the instrumentation embedded within the TOS.

CAUTION: The original element defining the User Event range defines the entire permitted range for User Events (40000-65535 in the example above). Do not, when editing User Events, stray outside this range.

Description of EventRangeDescription Attributes

Within the **EventRangeDescription**, the meaning of each attribute is as follows:

eventIdStart
the starting EventID for the user event range

eventIdCount
the number of user events in the range. Note that in the example above, eventIdStart = "40000" and eventIdCount = "25536", meaning that the range of Event IDs covered is 40000 to 65535 inclusive.

nameRoot
can be any alphanumeric name, but without spaces.

displayNameRoot
used for display of the event

nameRootSuffixStart
can be set to any non-negative integer. It defines the User Event number that will correspond to the first event in the range. Since, in the default example above, **eventIdStart = "40000"**, **displayNameRoot = "user"** and **nameRootSuffixStart = "0"**, the event with event ID = 40000 will be displayed as **user0**; the event with event ID = 40001 will be displayed as **user1**, and so on.

icon
the path to the icon to be used for the event in the event graph, relative to the **user.xml** file.

NOTE: If you design your own icon, please abide by the following guidelines:

- Your icon should be no more than 16 pixels high.
- Use a maximum of 256 colours.
- Use a transparent background.
- Remember that System Viewer can display your icon on either a black or a white background. Ensure that your icon will show up against both.

It is also possible to get System Viewer to generate a default, textual icon on the fly, in which case no icons need to be designed and placed in your installation. See *Using Textual Icons*, p.125.

trigger
"true" or **"false"** depending on whether System Viewer Triggering can trigger on events in this range. Normally you would leave this at **"true"**.

helpTopicId
used to link with the Workbench online help system. Can be omitted.

handler
this is the class name of a Java handler which will add funtionality to the graphical display of the events in this range. The default handler shown will composite the user event number with the given icon. If this functionality is not required, the handler attribute may be removed completely.

In the example above, for VxWorks 6.*N*, there follow two **EventParam** elements. The number and type **EventParam** elements may vary for different VxWorks versions or variants.

For any **EventParam** element:

type
do *not* change the original value!

name
may be changed to any display name you require

formatStr
this is a C-compatible **printf** format string which will be used to format the integer value of the parameter. May be changed to any C **printf** format string which is valid for an integer.

formatter
whilst this may be changed to name any Java class which implements the required interface (and is available at runtime), this is an advanced technique and is best left alone. The default formatter will print the value of any User Event's payload as ASCII if it contains *only* ASCII, or as a combined HEX/ASCII dump if the payload contains any non-ASCII characters.

14.3.3 Editing the User Event EventRangeDescription

If you want to change the display characteristics of *all* User Events, all you have to to do is to change the attributes of their collective **EventRangeDescription**.

As described in the previous section, the attributes that may be changed in the **EventRangeDescription** are:

- **nameRoot**
- **nameRootSuffixStart**
- **icon**
- **trigger**
- **handler**

And, within the **EventParam** child elements:

- **name**
- **formatter**

14.3.4 Editing a Single User Event, or a Block of User Events

The first step in editing a User Event is to remove the required UserEvent ID (or block of UserEvent IDs) from the **EventRangeDescription**. This is done by splitting the range into two and leaving a "hole" to hold the extracted IDs.

For example, if we want to remove a block of 100 User Events from the range, starting at EventID=40100, we would replace the original one **EventRangeDescription** element with *two* **EventRangeDescription**s, the first covering the EventID Range 40000-40099 and the second covering the EventID range 40200-65535. This leaves a hole in the defined Event IDs from 40100 to 40199 (i.e. 100 events).

The XML to do this is shown below:

```
<!-- eventId range 40000-40099 inclusive -->
<EventRangeDescription
  eventIdStart="40000"
  eventIdCount="100"
  nameRoot="EVENT_USER"
  displayNameRoot="user"
  nameRootSuffixStart="0"
  icon="images/defaultUser.gif"
  trigger="true"
  helpTopicId="VXWORKS6_EVENT_USER_EVENT_HELP"
handler="com.windriver.windview.plugins.wv.vxworks.UserEventDescription">
  <EventParam
    type="UINT32"
    name="pc"
    formatStr="0x%08x"/>
  <EventParam
    type="BLOB"
    name="data"
  formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
</EventRangeDescription>


<!-- there is an event ID gap from 40100-40199 inclusive -->

<!-- eventId range 40200-65535 inclusive -->
<EventRangeDescription
  eventIdStart="40200"
  eventIdCount="25336"
  nameRoot="EVENT_USER"
  displayNameRoot="user"
  nameRootSuffixStart="200"
  icon="images/defaultUser.gif"
  trigger="true"
  helpTopicId="VXWORKS6_EVENT_USER_EVENT_HELP"
handler="com.windriver.windview.plugins.wv.vxworks.UserEventDescription">
  <EventParam
    type="UINT32"
    name="pc"
    formatStr="0x%08x"/>
  <EventParam
    type="BLOB"
    name="data"
  formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
</EventRangeDescription>
```

In the first block, the **EventIdStart** remains at 40000 as before, but only covers 100 events, as shown by the new **eventIdCount**. Display numbering in this range will start from 0 as shown in the **nameRootSuffixStart** attribute. Thus, for an event with event ID 40000, the displayed name will be **user0**.

In the second block, the **EventIdStart** now contains 40200 which is the starting point of the block defined by this new range. The **eventIdCount** is 25336. The

defined range is therefore 40200-63335 (inclusive). The **nameRootSuffixStart** attribute has been set to 200, meaning that this range will have a display number starting at 200. Thus, for an event with event ID 40200, the displayed name will be **user200**.

The hole (event IDs 40100-40199 inclusive) may be filled in by producing:

- another **EventRangeDescription** to cover the entire range
- 100 **EventDescription** elements to describe each event in the range individually
- any combination of **EventRangeDescription**s and **EventDescription**s to fill the hole exactly

CAUTION: If any **EventDescription** elements are used, they must appear *after* all the **EventRangeDescription** elements in the XML file. Failure to observe this restriction will result in the XML file failing to parse and load.

Inserting a New EventRangeDescription

To fill in the gap using a single **EventRangeDescription**, you could, for example, append the following XML:

```
<!-- eventId range 40100-40199 inclusive -->
    <EventRangeDescription
      eventIdStart="40100"
      eventIdCount="100"
      nameRoot="EVENT_USER"
      displayNameRoot="myUserEvent"
      nameRootSuffixStart="0"
      icon="images/myUserIcon.gif"
      trigger="true"
    handler="com.windriver.windview.plugins.wv.vxworks.UserEventDescription">
      <EventParam
        type="UINT32"
        name="pc"
        formatStr="0x%08x"/>
      <EventParam
        type="BLOB"
        name="data"
      formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
    </EventRangeDescription>
```

This would fill the hole exactly, from Event IDs 40100-40199. The display of the first event in this range would be **myUserEvent0** since we have changed the **displayNameRoot** and the **nameRootSuffixStart**. Also, the icon for the events would change to that provided in **images/myUserIcon.gif**, but still suitably

decorated with the user event number, because we are still using the default value in the handler attribute. The **helpTopicId** attribute has been removed since there will be no corresponding entry in the Workbench documentation for this new range of event descriptions.

Inserting New EventDescriptions

Gaps in the [re]defined event range may also be filled using discrete **EventDescription** elements. Each of these covers just one event ID and offers the greatest scope for customizaton of individual events.

As an example, we could fill in the very first event ID in the gap created above with the following fragment of XML which defines the event for ID 40100:

```
<!--   This discrete EventDescription is provided for Event ID # 40100 which
        has the data structure of a User Event.
        The number, order and type of each of the parameters
        must be EXACTLY as shown (i.e. EXACTLY as for all other user events),
        since this format is dictated by the VxWorks runtime.
        Here, the "icon" attribute in the EventDescription has been
        modified to point to a user-defined icon for this event.
-->
    <EventDescription id="40100"
      name="MY_USER_EVENT_0"
      trigger="true"
      displayName="My User Event #0"
      icon="images/myUserEvent0Icon.gif">
      <EventParam
        type="UINT32"
        name="pc"
        formatStr="0x%08x"/>
      <EventParam
        type="BLOB"
        name="data"
      formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
    </EventDescription>
```

Here, event ID 40100 has been defined so that it will display as **My User Event #0** and have the custom-supplied icon **images/myUserEvent0Icon.gif** located relative to the **user.xml** file. Nothing else has changed.

The remaining bits of the hole created in the user event ID range may be filled with any combination of **EventRangeDescription** and **EventDescription** elements.

NOTE: It is not necessary to fill in the entire range; holes may be left. However, if a hole *is* left, you must be sure that your TOS runtime will *not* emit any events which correspond to the omitted event IDs, or your log will fail to parse completely due to missing event descriptions.

Using Textual Icons

In System Viewer 4.9 and higher, it is possible to use a different form of the **icon** attribute of the **EventDescription** element to have an icon, optionally coloured, complete with text, inserted on the fly, without having to manually design images and save them in your installation. This is a real time-saver.

As seen before, the default form of an **EventDescription**'s **icon** attribute is **icon="images/myIcon.gif",** where the value of the **icon** attribute is simply the path to an image file relative to the containing XML file.

NOTE: Path separators in the **icon** attribute *must* be a forward slash, regardless of which host operating system you are using; that is, including Windows.

The Extended Form of the `icon` Attribute

The extended form of the icon attribute allows for quick creation of new EventDescriptions, removing the need for manually creating image files.

If you want to use the extended form of the icon attribute, you should *not* use your own **handler** attribute. Rather, you should not insert any **handler** attribute at all. This is because the extended functionality is provided by the default Event Description handler.

Syntax

The extended form of the icon attribute has a formal syntax as follows:

icon="specifier [,specifier]*"

specifier: text=TEXT | color=#RRGGBB | icon=ICON_PATH

where:

TEXT
is the text to be composited into the icon (must not contain an equals or comma character). If the text specifier is omitted, no text will be composited on to the icon.

RRGGBB
is the hex Red, Green, and Blue values of the desired color of an auto-generated icon and any composited text. The format is per HTML color definitions, with each color component being between 00 and FF Hex. If the color specifier is omitted, the default color will be used (#00C0C0, a dark cyan).

ICON_PATH
is the path of a desired icon for compositing, relative to the containing XML file. If the icon specifier is omitted, a default icon will be generated on the fly.

The table below summarizes what happens depending on the combination of specified icon attribute specifiers.

Table 14-1 **Combinations of icon Attribute Specifiers**

text=	color=	icon=	Meaning
No	No	No	No icon, no text
Yes	No	No	Text and default icon, both in default color
No	Yes	No	No text, default icon in specified color
Yes	Yes	No	Text and default icon, both in specified color
No	No	Yes	Specified icon only
Yes	No	Yes	Text in default color, specified icon
No	Yes	Yes	Specified icon only
Yes	Yes	Yes	Text in specified color, specified icon

Examples

1. **icon="text=myUserEvent"**

   Simply creates an icon with the default shape (a small downward facing triangle), with the given text composited above, both in the default color (dark cyan).

2. **icon="color=#FF0000"**

   Creates an icon with the default shape in bright red, with no text.

3. **icon="text=myUserEvent,color=#00FF00"**

   Creates an icon with the default shape with text "myUserEvent" composited above, both in bright green.

4. **icon="icon=images/myIcon.gif"**

   Uses the supplied icon, with no text. This is equivalent to the non-extended form of the icon attribute: **icon="images/myIcon.gif"**

5. **icon="text=myUserEvent,icon=images/myIcon.gif"**

   Composites the given icon and the given text, with the text in the default color (dark cyan). The icon will be rendered exactly as in the supplied image file.

6. **icon="text=myUserEvent,color=#0000FF,icon=images/myIcon.gif"**

   Composites the given icon and the given text, with the text in the given color (bright blue). The icon will be rendered exactly as in the supplied image file.

So, to redesign our new **EventDescription** for our User Event with event ID 40100 (as shown above), but using the extended form of the icon attribute, our new **EventDescription** element could be as follows:

```
<EventDescription id="40100"
      name="MY_TRACE_EVENT_0"
      trigger="true"
      displayName="Trace#0"
      icon="text=Trace0">
      <EventParam
        type="UINT32"
        name="pc"
        formatStr="0x%08x"/>
      <EventParam
        type="BLOB"
        name="data"
      formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
    </EventDescription>
```

By doing this, instances of the event with event ID 40100 would be shown as **Trace#0** as the displayed event name, and have an icon which looks like a small, downward pointing, dark cyan triangle with the text **Trace0** above it, also in dark cyan.

14.3.5 Example of a Complete VxWorks 6.*N* user.xml File

An example of a complete new **user.xml** file, written specifically for VxWorks 6.*N*, is provided as an appendix (see *C. VxWorks 6.N user.xml Example File*). The example demonstrates many of the techniques described in this section.

If your TOS is not VxWorks 6.*N*, you should use the default **user.xml** file (see *14.3.1 Location of the User Events Description File*, p.117) for your TOS as a prototype, and be sure not to alter the range of user event IDs permitted for that TOS.

14.4 Validating XML Modifications

All System Viewer installations come complete with a command line utility that checks the dictionary XML files for consistency and correctness.

It should be run from a shell in which the correct environment has been set up.

Step 1: Set up the environment

1. Open a shell (or Windows Command Prompt).
2. Change directory to the root of your Wind River Workbench installation
3. At the prompt, type:

   `wrenv -p workbench-`*N.N*

   where *N.N* is the version of Wind River Workbench you have installed.

You are now ready to run the **wrsv-xmldictcc** utility to validate your XML.

Step 2: Validate your XML files

The syntax of the command for validating your XML files is:

wrsv-xmldictcc [-v] [-c] *path-to-XML-directory*

where

-v = verbose
prints OK after the file name of each XML file tested

-c = check images
also tests for the presence of referenced image files

The **wrsv-xmldictcc** utility is located in the directory
*installDir***/workbench-***N.N***/wrsv/***N.N***/host/***HostType***/bin**
where *HostType* is one of:

- **sun4-solaris2** for Solaris hosts
- **x86-linux2** for Linux hosts
- **x86-win32** for Windows hosts

Examples

These examples illustrate how you would use the **xmldictcc** command line utility on different hosts and with given Workbench, System Viewer, and TOS versions.

Example 14-1 **Solaris host, Workbench 2.6, System Viewer 4.9**

Workbench installation directory:
/user/fred/wb

XML directory to validate:

```
/user/fred/wb/workbench-2.6/wrsv/4.9/host/resource/windview/VxWorks/6.0
```

The required command is (all in one line):

```
/user/fred/wb/workbench-2.6/wrsv/4.9/host/sun4-solaris2/bin/wrsv-xmldictcc -v
-c /user/fred/wb/workbench-2.6/wrsv/4.9/host/resource/windview/VxWorks/6.0
```

Example 14-2 **Windows host, Workbench 2.6, System Viewer 4.9**

Workbench installation directory:
C:\wb

XML directory to validate:

```
C:\wb\workbench-2.6\wrsv\4.9\host\resource\windview\VxWorks\6.0
```

The required command is (all in one line):

```
C:\wb\workbench-2.6\wrsv\4.9\host\sun4-solaris2\bin\wrsv-xmldictcc -v -c
C:\wb\workbench-2.6\wrsv\4.9\host\resource\windview\VxWorks\6.0
```


14.5 Advanced Techniques: Custom Parameter Formatting

CAUTION: Custom formatting of parameters is an advanced technique which requires knowledge of Java programming and access to a Java 1.4 SDK.

User Events contain a **BLOB** parameter which can contain a binary payload in the event generated by the TOS instrumentation. The Java formatter used in the default **user.xml** is:
com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter
which will display the content of the **BLOB** payload as ASCII if the content consists entirely of ASCII characters, or as a mixed HEX/ASCII decode if there are

any non-ASCII characters in the payload. However, in your own TOS runtime code, it is possible to construct a binary payload for your User Event which contains a predefined data structure.

For example, your runtime code may define the following data structure

```
struct
    {
    unsigned int field1;
    unsigned int field2;
    unsigned int field3;
    } myUserEventPayload;
```

and have this included as the **BLOB** payload for your User Event.

It is then possible to write a custom **EventParameterFormatter** implementation in Java which understands and appropriately formats the data structure of this payload, rather than using the standard Hex/ASCII formatter.

For full details on how to write a custom Event Parameter Formatter, please see the online Wind River System Viewer API documentation provided for the **com.windriver.windview.agnostic.wv.eventlog.EventParameterFormatter** interface. The easiest way to find the relevant documentation is to open the Wind River Workbench online help and search for:
"**eventlog.EventParameterFormatter**"

⚠ **CAUTION:** As part of this process, you will need to compile your new Java class and put it into a Jar file. This must be done with a Java SDK which produces code that is compatible with Java 1.4. Extensions to the Java language introduced in Java 1.5 (or later) must *not* be used.

Your Java compilation command line must include the **wv.jar** Jar file in the compilation classpath. The **wv.jar** file is located in

installDir/**workbench-***N.N*/**wrsv/***N.N*/**host/java/lib/**

15

System Viewer for Wind River Linux

15.1 Configuring Wind River Linux for System Viewer

System Viewer configuration for Linux is, on the whole, the same as for VxWorks. Configuration features are fully supported and generic tasks work the same way as on VxWorks. See *3. Configuring a Logging Session* and following chapters for general information in this respect.

System Viewer for Wind River Linux leverages the LTTng project. The Wind River Linux distribution includes the LTTng patches.

15.2 Using System Viewer Configuration in Workbench

The **System Viewer Configuration** editor's **Configuration** tab is used to create commands that are executed on the target to start and stop tracing.

To create the command-line parameters, configuration is subdivided into separate categories, each of which provide a known parameter as described below.

15.2.1 Configuration Summary

This provides a full summary of the current System Viewer configuration (not the configuration of your target system), as well as a copy of the command line, which you can copy for direct use on the target shell.

15.2.2 Flight Recorder Options

Events collected in *flight recorder* mode are stored in buffers located in target memory. The trace data remains in the buffers and is not transferred to a file until requested. The buffers are re-used when full; as a result only the most recent events will be in the log.

This mode is particularly useful for debugging a target process. If a process ends unexpectedly for any reason, you can upload the **Flight recorder** log to see the sequence of events leading up to the failure.

15.2.3 Target File System Options

The output directory for storing LTTng target trace files, and the system used when creating new trace files.

Specify a new name for each trace collected
After every log collection you have to enter a new directory name. The directory name must not exist on the target. The trace files will remain on the target until deleted manually.

Add incrementing suffix to target trace name
A unique, sequential number is suffixed to the directory name each time a log is collected. The trace files will remain on the target until deleted manually.

Overwrite target trace files when collecting a new log
The output directory is re-used, and existing trace files in the directory are overwritten every time a log is collected.

Append to existing trace
Collected data is appended to the end of the specified trace files. If the trace files do not exist, new ones will be created.

15.2.4 Buffer Configuration

The number and size of sub-buffers in the target memory used during tracing. If this option is not selected, default values are used.

NOTE: Choosing 2 buffers of a small size is likely to lead to lost events. To check for lost event run **dmesg(1)** on the target. To reduce the risk of losing events either increase the number of buffers, increase the size of the buffer, or both.

15.2.5 Output Filename

The host directory and filename to save the collected event log. The LTTng daemon collects and saves trace files in the directory you specified in the *15.2.3 Target File System Options*, p.132. These files are then copied from the target to the host.

The trace files are copied from the target and saved in a subdirectory, named in **Output Filename** field, of the directory specified in the **Directory** field. These files are then converted to single file, in a format which is understood by System Viewer, and saved to the filename you specified in t he **Output Filename** field. If conversion fails, the LTTng files are left in the specified directory.

Output Filename options:

Automatically download when logging stops
After log collection stops, copy the LTTng files from the target to the host and convert to a Wind River System Viewer raw file (**.wvr**).

Automatically view a downloaded log
Once successful download and conversion is complete, System Viewer automatically opens the log in the **Log Viewer**.

Add incrementing suffix to filename
This avoids existing logs being overwritten.

15.2.6 Log Conversion Options

Save the LTTng trace files after conversion to System Viewer format (**.wvr**). The default is to delete trace files after conversion.

15.3 Custom Events

LTTng can record custom user-defined events that can be displayed in System Viewer. These are generally referred to as *Custom Events*.

NOTE: Wind River Linux *Custom Events* are not to be confused with *User Events* as used by the VxWorks family of operating systems. Although the purpose and output are essentially the same, input and internal handling differ.

Custom Events are not included in the Linux kernel by default, so the kernel must be patched and configured in order to use them. See the Wind River Linux documentation to ensure you are able to locate your project build directory and kernel sources.

Once the kernel is rebuilt, kernel functions can make calls to the logging routines to write event-data into the LTTng trace files.

Example files that define a number of ready-to-use Custom Event logging routines are supplied in:

installDir/**workbench-***N.N*/**wrsv/***N.N*/**Linux/ltt.tar.gz**

The **ltt.tar.gz** archive contains:

- **genevent/genevent-***N.N*/**example/custom.xml**
- **genevent/genevent-***N.N*/**example/keyboard.c**
- **genevent/genevent-***N.N*/**example/wrlinux-***N.N*-**ltt-custom-events.patch**

The supplied routines are added to the kernel by the patch (see *15.3.1 Preparing a Kernel for System Viewer Custom Events*, p.135), and are listed in the table below for your convenience.

Table 15-1 **Custom Event Logging Routines**

Event Name	Logging Routine Prototype
custom0	`trace_custom_custom0 (char * file, unsigned int line, char * message);`
custom1	`trace_custom_custom1 (unsigned int p0, unsigned int p1, unsigned int p2, unsigned int p3x);`
custom2	`trace_custom_custom2 (char * message);`
custom3	`trace_custom_custom3 (unsigned int );`
custom4	`trace_custom_custom4 (char * p0, unsigned int p1, unsigned int p2, unsigned int p3);`

15.3.1 Preparing a Kernel for System Viewer Custom Events

If you add Custom Events to a kernel, you first have to apply a patch and then rebuild the kernel. Please refer to the Wind River Linux documentation for instructions on how to rebuild. Before you rebuild, the following preparations are necessary to add Custom Events:

1. The kernel sources must first be patched by adding the header and source files required for Custom Events.
   a. Extract the patch: *installDir*/**workbench-***N.N*/**wrsv/***N.N*/**Linux/ltt.tar.gz**
   b. In a shell, navigate to the Linux kernel directory under the project build directory in your Wind River Linux installation tree. Typically, this directory will be **.../build/ linux-***N.N.N*-*KernelType* (where *KernelType* is either **cgl** or **small**).
   c. Apply the patch by entering:

      `patch -p1 <` *path*/**`wrlinux-`***`N.N`***`-ltt-custom-events.patch`**

      (The *path* referred to above is the path to the extracted **ltt.tar.gz** plus the extracted subpath; that is, **.../ltt/genevent/genevent-***N.N*/**example**.)

2. Now add instrumentation to the kernel files you are interested in by adding:

   a. **#include <linux/ltt/ltt-facility-custom.h>** in the includes section

   b. calls to the appropriate instrumentation functions as required (see Table 15-1, or the header file included above)

   The example file **keyboard.c** (in **ltt.tar.gz**) shows the result of adding instrumentation to the file **drivers/char/keyboard.c** in the Linux kernel.

3. You then need to reconfigure your kernel to include the new instrumentation (see *Reconfiguring and Rebuilding the Kernel* in the Wind River Linux documentation).

   a. Change directory to the project build directory (see step 1.b., above, and enter:
   `make linux.menuconfig`
   This command invokes the configuration needed in the following step.

   b. In the **Instrumentation Support** configuration option, enable the **Linux Trace Toolkit Custom Events** entry (disabled by default).

   c. Save the configuration.

4. Rebuild the kernel as described in the Wind River Linux documentation:
   `make linux.rebuild`

5. Before booting with the new kernel, copy the file **custom.xml** (in **ltt.tar.gz**) to the target file system's **/usr/share/ltt-control/facilities/** directory.

6. Boot the target with the new image.

   You can now collect logs using System Viewer, and they will record the Custom Events as they occur.

A Programming Data Collection

A.1 Introduction

A critical aspect of data collection is managing and controlling the volume of information you generate in a log as Wind River System Viewer allows you to collect a great deal of information.

At the same time, instrumentation classes allow you to limit data collection either by limiting the detail you collect, or by limiting detailed data collection to specific objects. Multiprocessor support and the high-resolution timestamp drivers contribute to the volume of detailed information that must be stored and uploaded. The dynamic ring buffer provides a flexible method of balancing the need to hold large amounts of data for upload to the host and the need to leave the target applications enough resources to work normally.

The routines in **wvLib** and **wvNetDLib** provide a much finer level of control over how events are logged than the GUI tools or the basic **wvOn( )** and **wvOff( )** routines described in *7. Logging and Uploading Data*.

Detailed information about the routines and examples of their usage are available in the *VxWorks OS Libraries API Reference*; you can also examine the source code in *installDir*/**vxworks-***N.N*/**target/config/comps/src/usrWindview.c**. These routines are more complex to use than **wvOn( )** and **wvOff( )**, since the order in which they are invoked is critical.

A.2 Instrumenting Objects Programmatically

Instrumentation of events is implemented in Wind River System Viewer by classes, which are designated by the logging levels discussed in *4. The Event Logging Level*. When you select AIL level logging in the **Event Logging Level Configuration** pane, the libraries **taskLib**, **semLib**, **msgQLib**, and **wdLib** are selected by default in the **Additional Instrumentation Library Selection** checklist. You can modify the amount of data collected by selecting all libraries, only one library, or any combination of the listed libraries.

A.2.1 Kernel Libraries

When you select a library in the GUI by checking the library name in the **Event Logging Level Configuration** pane, all objects in that library are instrumented. When you start data collection, events are logged for these objects.

You might be interested in how specific tasks, semaphores, message queues, and so on interact so target routines are available to instrument individual objects. This allows you greater precision in collecting only the data you are interested in seeing.

Table A-1 lists the Wind River System Viewer target routines that you can call to control the instrumentation of particular objects.

Table A-1 **System Viewer Instrumentation Routines**

Routine	Description
wvObjInst( )	instrument objects
wvSigInst( )	instrument signals
wvObjInstModeSet( )	set object instrumentation level
wvEvtClassSet( )	set the class of events to log
wvEventInst( )	instrument VxWorks events
wvNetEnable( )	begin reporting network events
wvNetEventEnable( )	instrument specific network events

To instrument individual objects and groups of objects programmatically, call **wvObjInst( )** instead of selecting the library in the **Additional Instrumentation Library Selection** checklist. Events that affect these objects are logged if and when AIL level logging begins.

The **wvObjInst( )** routine is declared as follows:

```
STATUS wvObjInst
    (
    int     objType      /* object type: windSemClass, windMsgQClass, */
    void *  objId        /* object ID: specific ID or NULL for all */
                         /* objects of objType */
    int     mode         /* turn instrumentation on/off:
                         /* INSTRUMENT_ON, INSTRUMENT_OFF */
    )
```

The **wvObjInst( )** routine sets up instrumentation for objects of the specified type whether or not they already exist on the target system. To instrument only those objects that are created after a certain point, use the **wvObjInstModeSet( )** routine.

You can instrument specific objects in **taskLib**, **semLib**, **msgQLib**, and **wdLib**. You cannot specify particular signals to instrument within **sigLib**, either all signals are instrumented, or none are.

To instrument signals programmatically, call **wvSigInst( )** instead of selecting the library in the **Additional Instrumentation Library Selection** checklist. Either method results in all signals being instrumented.

The **wvSigInst( )** routine is declared as follows:

```
STATUS wvSigInst
    (
    int     mode         /* turn instrumentation on/off:
                         /* INSTRUMENT_ON, INSTRUMENT_OFF */
    )
```

To instrument VxWorks events programmatically, call **wvEventInst( )** instead of selecting the library in the dialog box checklist. Either method results in all signals being instrumented.

The **wvEventInst( )** routine is declared as follows:

```
STATUS wvEventInst
    (
    int     mode         /* turn instrumentation on/off:
                         /* INSTRUMENT_ON, INSTRUMENT_OFF */
    )
```

You can start event collection using the GUI, or you can start it programmatically. The following example code shows how to create a log file on the host, enable instrumentation of all possible classes, collect VxWorks events, signals, and network instrumentation data, and start and stop logging.

```
/*********************************************************************
* wvLogFileUploadStart - create a logfile on the host, and start logging
*
* This routine uses various System Viewer functions to create a log on a
* host, to select the required instrumentation, and to start log
* collection. It is invoked with a logging level, and the name of the
* file to use on the host.
*
* e.g. wvLogFileUploadStart 7, "/tgtsvr/logfile.wvr"
*
* RETURNS: OK, or ERROR
*
* SEE ALSO: wvLogStop()
*
*/
```

```
STATUS wvLogFileUploadStart
    (
    int         instClass,      /* 1, 3, or 7, for CLASS1, CLASS2,
                                   CLASS3 instrumentation */
    char *      filename        /* logfile name */
    )
    {
    STATUS      result;
    int         fd;
    int         n;

    if (filename == NULL || instClass == 0)
        {
            puts ("Usage is:\n  'wvLogStart <instLevel> <filename>'\n" \
              "where instLevel can be 1, 3 or 7\n");
            return ERROR;
        }

    remove (filename);

    /* wvOn uses open() which cannot create a file. So we ensure
        the file exists */

    fd = open (filename, O_WRONLY, 0);
    if (fd == ERROR)
        {
            fd = creat (filename, O_WRONLY);

            if (fd == ERROR)
                {
                logMsg ("Error creating logfile\n",0,0,0,0,0,0);
                return ERROR;
                }
            }
    close (fd);

    /* Enable instrumentation for all possible classes */

    for (n=0; n < sizeof (instClasses) / sizeof (instClasses [0]); n++)
        {
        wvObjInst (instClasses [n], NULL, INSTRUMENT_ON);
        }

    /* Enable events, signals and network instrumentation */

    wvEventInst (INSTRUMENT_ON);
    wvSigInst (INSTRUMENT_ON);
    wvNetEnable (8);

    result = wvOn (instClass, (int)filename, O_WRONLY, 0);

    return result;
    }
```

```
/*
 * Stop collection and close the file.
 */

void wvLogStop (void)
    {
    wvOff ();
    }
```

The **wvObjInstModeSet( )** routine lets you instrument a group of objects beginning from the time of their creation. This routine is declared as follows:

```
STATUS wvObjInstModeSet
    (
    int    mode         /* turn instrumentation on/off:
                        /* INSTRUMENT_ON, INSTRUMENT_OFF */
    )
```

For example, suppose you are adding a new module to an already existing piece of code. You might be only interested in the new objects affect the existing application and you do not want to instrument any other objects. Within the object initialization code of the new module, you can surround the creation calls with **wvObjInstModeSet( )**, as in the following example:

```
void newCode ()
    {
    ...

    /* enable instrumentation for all objects created here */
    status = wvObjInstModeSet(INSTRUMENT_ON);
    ...
    /* create message queue 1 */
    mq1Id = msgQCreate (5, 20, MSG_Q_FIFO);
    /* create message queue 2 */
    mq2Id = msgQCreate (5, 20, MSG_Q_FIFO);
    /* create semaphore 1 */
    sem1Id = semMCreate (SEM_Q_FIFO);
    /* create task 1 */
    t1Id = taskSpawn ("task1", TASK_1_PRI, TASK_1_OPTS,
                  TASK_1_SIZE, task1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    ...
    /* disable object instrumentation */
    status = wvObjInstModeSet (INSTRUMENT_OFF)
    ...
    }
```

All objects created between the two calls to **wvObjInstModeSet( )** are instrumented. Events that affect those objects are logged if and when AIL level logging is started, for example, with the **wvEvtLogStart( )** routine, as in the following example (which occurs after the **newCode( )** fragment shown above):

```
void existingCode ()
    {
    ...
    /* area of existing code where objs from newCode module used */
    wvEvtLogStart;
    ...

    /* send a message to message queue 1 */
    status = msgQSend (mq1Id, &buff, MSG_4, MSG_4_TIMEOUT,
                       MSG_PRI_NORMAL);
    /* receive a message from message queue 2 */
    numbytes = msgQReceive (mq2Id, &buff, BUFF_SIZE, TICKS);
    /* take semaphore 1 */
    status = semTake (sem1Id, SEM_1_TIMEOUT);
    /* suspend task 1 */
    status = taskSuspend (t1Id);
    ...

    /* interesting section finished; stop event logging */
    wvEvtLogStop();
    ...

    }
```

To instrument all objects in the system programmatically, perform the following at the beginning of your application:

- Call **wvObjInst( )** once for each type of instrumented object, with *objID* set to **NULL** to instrument all objects of the specified type, and *mode* set to **INSTRUMENT_ON**. This instruments all objects already created by the operating system.
- Call **wvObjInstModeSet( )** with the argument **INSTRUMENT_ON**. This instruments all objects you may create thereafter.
- Call **wvSigInst( )** with the argument **INSTRUMENT_ON**.
- Call **wvEventInst( )** with the argument **INSTRUMENT_ON**.

These programmatic steps have the same effect as starting Wind River System Viewer with all libraries checked in the GUI, starting Wind River System Viewer logging, and starting your application.

For additional insight into the configuration start, stop and upload process, see *installDir***/vxworks-***N.N***/target/config/comps/src/usrWindview.c**.

A.2.2 Additional Libraries

The data collected by these instrumented libraries is displayed in the usual way: with icons in the main **Event Graph**.

Each instrumented library is enabled or disabled as a whole. To control which libraries are logged, use the **Event Logging Level Configuration** pane as described in *4. The Event Logging Level*. To control what is displayed in the view graph, the **Filter Events** dialog contains an entry for each additional library as described in *12.2.2 Filter Events*, p.81.

memLib

The information displayed from this library is slightly different from the information generated and displayed for events in other libraries. While the event data generated by **memlib** can be useful, the critical information is usually the scalar information generated by the event. For example, the most interesting information about memory activities is usually how much memory is allocated at a particular time. For more information, see *9.5 Reading the Memory Usage Analysis Pack*, p.58.

On the host, you can view cumulative information depicting what happens as allocated memory grows and shrinks in each target partition. You can see the addresses of memory blocks that get allocated and freed. This simplifies detecting memory leaks. For details about the information collected for events, the routines associated with them, and so on, see the *Wind River System Viewer User's Reference: Event Dictionary*.

wvNetDLib

This library is available at the AIL level of logging, and includes events related to activity in the dual IPv4/IPv6 stack. As with other instrumented objects, network events may be instrumented individually or in groups. **wvNetDLib** events are grouped according to class and priority.

Networking vents are also divided into five priority levels, with 1 being the highest. (**VERBOSE**, **INFORMATION**, **WARNING**, **CRITICAL**, and **FATAL**). The network stack is instrumented for the following event classes:

WV_NETD_IP4_DATAPATH_CLASS
This class is for events that are directly related to IPv4 data transfer.

WV_NETD_IP6_DATAPATH_CLASS
This class is for events that are directly related to IPv6 data transfer.

WV_NETD_IP4_CTRLPATH_CLASS
This class is for events related to IPv4 network stack operations, such as updating the routing table and handling socket operations.

WV_NETD_IP6_CTRLPATH_CLASS
This class is for events related to IPv6 network stack operations, such as updating the routing table and handling socket operations.

To instrument network event logging programmatically, use the **wvNetEnable( )** and **wvNetDisable( )** routines, which start and stop network event logging, respectively.

To enable Wind River System Viewer instrumentation for network events, you must include network instrumentation in your VxWorks configuration (**INCLUDE_WVNETD**). The start routine takes a single parameter specifying the minimum priority level you wish to log. Events in both core and auxiliary classes are logged and if no priority is specified, events for all priority levels are logged.

In general, an instrumented target reports all events having priority greater than or equal to the selected priority. In addition, the **wvNetDLib** library contains routines that give much finer control over networking event logging. For example, you can explicitly include or exclude individual events whose priorities are lower than the selected level, explicitly enable or disable other events, and apply filters to events based on addresses and port numbers.

For more information about logging network events and fine tuning these settings, see the entry for **wvNetDLib** in the *VxWorks OS Libraries API Reference*.

A.3 Adding Eventpoints

You can specify which level of logging to implement, but data is collected only when events are generated by the instrumented VxWorks kernel and libraries. Your application will make many calls to VxWorks routines, but sometimes you will want information on some of your application routines as well. There are two ways to do this. You can set eventpoints dynamically from the Wind River Workbench shell with the **e( )** routine, or insert event-generating function calls into application source code.

The e( ) Routine

The simplest type of user-generated event is the *eventpoint*, which can be set from the shell with the **e( )** routine. Eventpoints are analogous to breakpoints; they are program locations that display the **default User** event icon when the instruction at that location executes.

The **e( )** routine has the following syntax:

```
STATUS e
    (
    INSTR *  addr,          /* address to set eventpoint; */
                            /* NULL = all eventpoints and */
                            /* breakpoints are raised */
    event_t  eventNo,       /* event number */
    int      taskNameOrId,  /* task in which to raise event- */
                            /* point; 0 = all tasks */
    FUNCPTR  evtRtn,        /* function to be called when */
                            /* eventpoint encountered; NULL */
                            /* = no function, so eventpoint */
                            /* always raised */
    int      arg            /* argument to evtRtn function */
    )
```

NOTE: Eventpoints follow the same rules as breakpoints, including the following: (1) Eventpoints in unbreakable tasks are not executed and thus do not appear in the **Event Graph**. (2) Eventpoints cannot be set at interrupt level.

To see how a log looks when you set an eventpoint with **e( )**, follow these steps:

1. To set an eventpoint with an ID of 10 at the address of the **sin( )** routine, click the **Launch Shell** button and enter the following:

   ```
   -> e sin, 10, 0, 0, 0
   value - 1 = 0x1
   ```

2. Click **GO** to start logging using any event logging level.

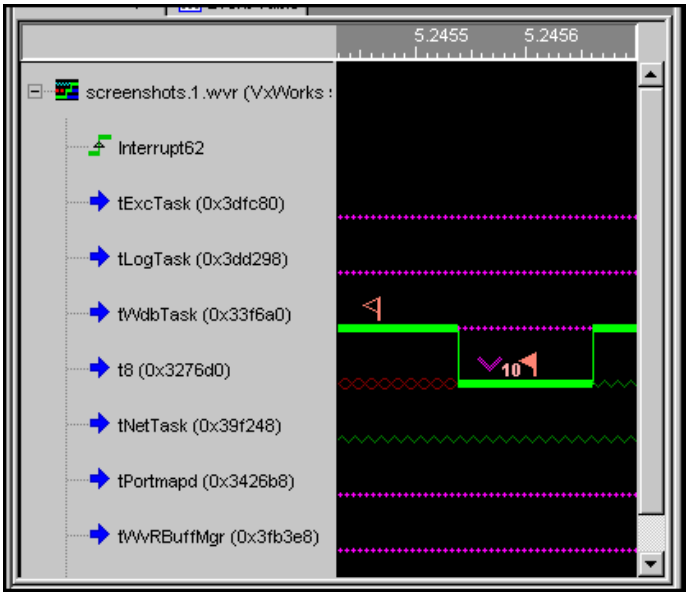
3. In the shell window, enter the following:

```
-> sin 1
value = 1 = 0x1
```

4. Click **STOP** to stop logging and upload the data.

5. Locate the numeral **10**, click, drag, and zoom in over this area of the **Event Graph** repeatedly until you see your user event icon with the numeral 10 beside itm which may appear as in Figure A-1.

Figure A-1 **Event Graph with User Event**



The wvEvent( ) Routine

The **wvEvent( )** routine provides a generic event function which is always available and which has more flexibility than **e( )**. The **wvEvent( )** routine is declared as follows:

```
STATUS wvEvent
    (
    event_t    eventNo,        /* event ID */
    char *     buffer,         /* user-supplied buffer */
    size_t     bufSize         /* buffer size */
    )
```

The **wvEvent( )** routine can be called from unbreakable tasks or from interrupt level, and is therefore more flexible than the **e( )** routine.

Unbreakable tasks are those that are created with the **VX_UNBREAKABLE** bit set in the options argument of the taskInit( ) or taskSpawn( ) routine. For example, you can use user-generated events for error checking, as shown in the following example:

```
if (something)
    {
    ... /* run this code */
    }

else
    {
    ... /* we should never get here, but if we do, load any values */
        /* of interest into event1Buff, then log using wvEvent     */

        return = wvEvent (EVENT_1_ID, &event1Buff, EVENT_1_BUFSIZE);
    }
```

If this event is ever encountered, the **defaultUser** event icon is displayed in the **Event Graph**, with the event number just to the right of the icon. You can double-click this icon to bring up the **Event Properties/Search (***filename***)** dialog, which shows the timestamp at which the user event occurred, the context in which it occurred, and the user event number, For more information, see *Using the Event Properties/Search (filename) Dialog*, p.73.

If you customize the **Event Properties/Search (***filename***)** dialog, you can also display information held in the user event buffer, the *buffer* argument to **wvEvent( )**.

To see how a log looks when an eventpoint is raised by **wvEvent( )**, perform the following:

1. Click **GO** to start logging using any event logging level.
2. In the shell window, enter the following:

```
-> wvEvent (99, 0, 0)
value = 0= 0x0
```

3. Click **STOP** to stop logging and upload the data.

A.4 Timestamping

High-Resolution Timestamp Driver

When you develop your application for a target with a supported timestamp driver, the instrumented kernel tags certain events with a high-resolution *timestamp*. Those events are displayed in the Wind River System Viewer **Event Graph** along a timeline that shows when each event occurred, based on the timestamps. You can see the exact timestamp for any particular event in the status bar, for example, when you click on the corresponding event icon.

The timestamp driver's resolution is hardware dependent, but is typically between 100 KHz (10 microseconds per tick) and 1 MHz (1 microsecond per tick). For information on the system timestamp driver for any supported board, see the entry for the board in the online *VxWorks BSP Reference*.

In addition to the WRS-supplied system timestamp, you may choose to write your own timestamp driver or use the timestamp provided by an emulator. For more information, see *VxWorks Device Driver Developer's Guide:Timestamp Drivers*.

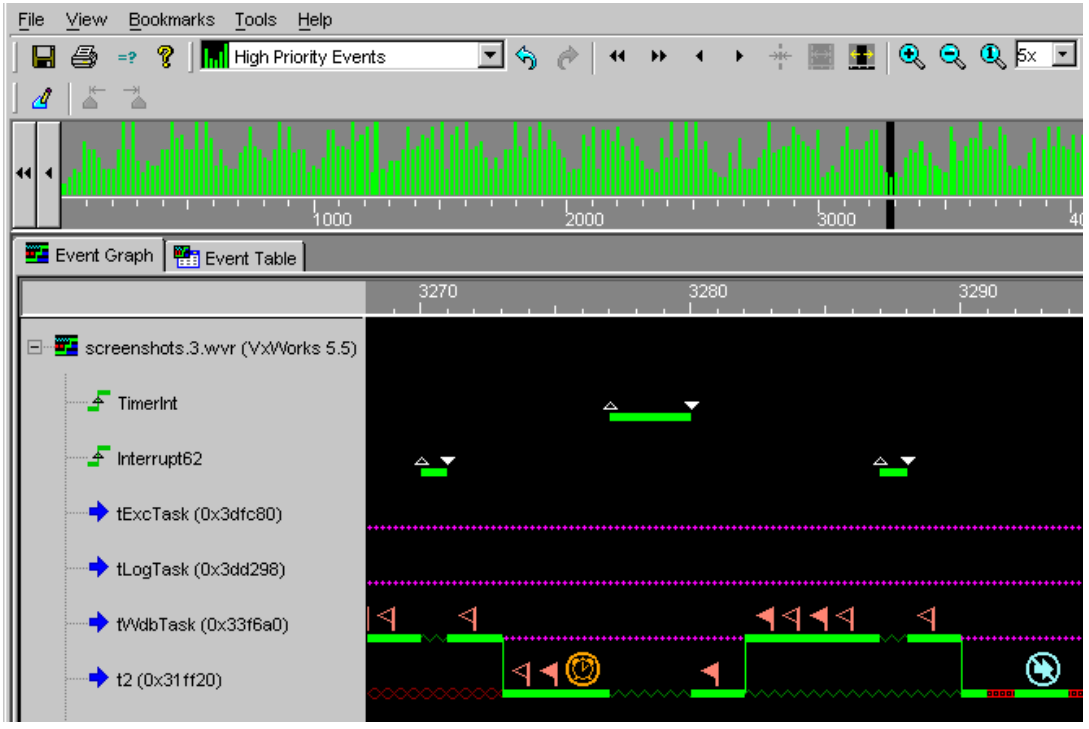
Sequential Timestamp Driver

When the real-time system runs on an unsupported board, or on a supported board without the timestamp driver, the instrumented kernel automatically uses its *sequential timestamp driver.*

The sequential timestamp driver tags events with sequence numbers that simply represent the order in which events occur on the target. The distance between events does not reflect any measurement of time.

Figure A-2 shows data captured with the sequential timestamp driver with events all equidistant from each other. If the same events were captured with a high-resolution timestamp driver, the distance between events would reflect an actual passage of time.

Figure A-2 **Sequential Event Display**



A.5 Dynamic Buffer Allocation

Wind River System Viewer uses a dynamic ring buffer (*rBuff*) structure to support data collection. The **rBuffLib** library provides a ring buffer structure which grows and shrinks dynamically during operation. The **rBuff** library introduces a new class of object, allowing **rBuffs** to be browsed and manipulated like other objects. For more information about **rBuffLib**, see the *VxWorks OS Libraries API Reference*.

The size of each individual buffer and the minimum and maximum number of buffers are configured when the buffer ring is created. The library initially allocates the minimum number plus one additional buffer for immediate addition to the ring if necessary. If these buffers are filled, more buffers are added until the ring reaches the maximum size. If data is read from the **rBuff**, the unused buffer memory is freed so that it may be used by other processes.

A.5.1 Configuring the Event Log Buffer

Wind River System Viewer allows you to configure the **rBuff** which holds the event log as it is captured and stored on the target. How the **rBuff** is configured affects how logging interacts with the target application and thus the quality of the resulting event log.

The optimum **rBuff** configuration is dependent on two factors:

- the upload mode
- any constraints on target memory

The configuration of the **rBuff** has a significant impact on the frequency with which it is dynamically extended and, consequently, how much of this activity is represented in the log. Optimum values depend on whether you are using deferred or continuous upload mode.

In both deferred and continuous modes, the **rBuff**'s memory requirements are sourced from the system memory partition, the general partition from which all malloc requests are sourced. Therefore it is important to consider the size of the **rBuff** carefully to ensure that it does not conflict with target application requirements.

Upload Modes

Deferred Upload

The most effective way to use Wind River System Viewer is in deferred upload mode. In this mode the event data remains on the target during collection and can be uploaded to the host once logging is completed. The entire log must be stored on the target, which potentially requires a large ring buffer. If the system partition is exhausted to the granularity of one buffer block or if the specified maximum number of buffers is full, data collection stops. Triggering can be used to focus the sequence captured, avoiding an unnecessarily large buffer requirement.

In deferred upload mode, you may want all available memory to be used to hold event data. The extendable ring of buffers can share available memory between the customer application and the buffers. To collect as much event data as possible within the constraints of the system and application, point the **rBuff** at the system memory partition, set the maximum size to infinite by specifying 0xffffffff, and recreate the problem. The **rBuff** starts at the specified minimum size, allowing the application to allocate memory during initialization, and only begins extending the buffer when its pre-allocated buffer is full.

The limitation of this approach is that the activity necessary to extend the **rBuff** appears in the log. This makes it more difficult to determine exactly how the system behaves when logging is not active. To avoid the intrusion of buffer allocation on the log, you can pre-allocate the buffer space by setting the minimum number of buffers equal to the maximum number of buffers. If you choose this approach, configure the **rBuff** with a smaller number of large individual buffers.

Continuous Upload

In continuous upload mode, the data in the log is uploaded to the host periodically as it is captured. Network activity required to upload the data itself generates events that appear in the log. In addition, any activity required to resize the **rBuff** is reflected in the log. This results in logs which are larger and more complex than would be required to represent the application activity alone.

In continuous upload mode, you select a minimum and maximum number of buffers. When an individual buffer is full, data is written to the next buffer. If there are no more buffers and the current number of buffers is less than the maximum, a new buffer is added. If a buffer is emptied and the current number is greater than the minimum, the empty buffer is removed. This implementation is flexible and dynamic. The collected data can fill all the memory available if necessary and if the maximum number of buffers specified is that large, but the memory is not reserved for logging if it is not needed.

The size of the individual buffers is used to determine when to start transferring the event data to the host. A smaller individual buffer size initiates upload sooner and more often. As there are variations in the rate at which events are generated on the target and at which the network and host can achieve upload, the **rBuff** may dynamically resize to accommodate the events stored on the target. Using a small individual size maximizes the possibility of a suitable area of memory being available to extend the **rBuff**.

Occasionally, the **rBuff** is still growing, although data is being uploaded to the host. This happens when the number of events generated by the upload process is greater than the number of events being uploaded. This is most likely to occur when you have specified AIL level logging and instrumented **semLib** because the upload implementation generates many semaphore events.

The ability to specify the minimum number of buffers to be retained avoids buffer thrashing, where the ring is repeatedly extended and shortened as the amount of data in the buffer crosses a particular threshold. There is nothing special about the set of buffers that is originally allocated. As the ring of buffers is filled and then emptied, the initial set is just as likely to be freed as those that are subsequently allocated to meet demand. In a balanced system, the **rBuff** is not constantly resized; the ring of buffers remain at the original size and allows steady upload of event data.

For these reasons, when using continuous upload mode, configure the **rBuff** with a larger number of small sized individual buffers and adjust the minimum number of buffers if necessary to prevent thrashing.

Post-mortem Mode

In post-mortem mode, Wind River System Viewer automatically configures the number and size of buffers in the ring. The only configuration step you must take is to place your buffer in an area of memory that is not overwritten when VxWorks reboots. For more information, see *5.3 Post-Mortem Upload Modes*, p.22.

rBuff Task Priority

The **rBuff** is managed by the task **tWvRBuffMgr**. The default priority of this task is 100. This priority is usually appropriate, particularly when you use either deferred upload mode or post-mortem mode. In some environments when you use continuous upload mode, **tWvRBuffMgr** is not high enough priority to be able to expand the number of buffers before they fill. If you find that the **rBuff** is overflowing in continuous mode, use **wvRBuffMgrPrioritySet( )** to give **tWvRBuffMgr** a higher priority.

Target Memory Constraints

If you are using a target configuration with sufficient memory resource to dedicate a proportion to Wind River System Viewer, the ring of buffers may be configured such so its individual buffers are pre-allocated. This is achieved by configuring the minimum number of buffers in the ring equal to the maximum number of buffers.

If you are using a target with limited memory resource, use the ring buffer dynamic resizing facility. This is done by allocating a low minimum number of buffers with a greater maximum number of buffers. In deferred mode, only those buffers required above the minimum, and for which space is available at the time of request, are allocated. In continuous mode, the buffer only extends above the minimum when necessary to hold the upload backlog, and those extra buffers are freed when not in use. When using a dynamically resizing buffer configuration, the ring buffer will not give up buffers it is holding to satisfy a request for memory made by another part of the system. Therefore, care should be taken when specifying the maximum ring buffer size.

When using post-mortem mode, the ring buffer is automatically configured to make appropriate use of the available memory dedicated to this purpose.

A.5.2 Configuration Tuning

The precise ring buffer configuration used is dependent upon many factors such as relative host and target performance, network bandwidth, the rate at which the target application generates events, and the upload method used. Therefore it is advisable to experiment with the ring buffer and upload configuration if any of these factors is altered to find a configuration which best suits your development system.

B

Triggering API

B.1 Introduction

Wind River System Viewer provides a triggering API and triggering that works in conjunction with logging. You can use the triggering interface to enter specific information and commands, which are then downloaded to the target. Although this data is usually entered in the triggering interface window, you can enter data at command line. After this data is received, the target handles the following tasks:

- Organizes the data received from the host in a trigger list.
- Manages the trigger list and sets the flag to activate triggering when necessary.
- Activates and deactivates event triggering.
- Performs the actions requested by the host.
- Interacts with Wind River System Viewer and the event points.
- Saves the information coming from the host in a trigger structure.

When a flag is set, the event can be detected by the regular Wind River System Viewer instrumentation, **e( )**, or **trgEvent( )**. Once the event occurs, the **ACTION_IS_SET** variable is checked to see whether System Viewer (instrumentation) logging or triggering is active. If triggering is active, the list of triggers is checked to see if any is related to that event. If one is found, the specified action is performed. Otherwise, execution continues, as shown in Figure B-1.

Macros

The **ACTION_IS_SET** macro is shared by both logging and triggering, and regulates the activation and interactions of both logging and triggering. The process flow is shown in the following pseudocode:

```
if ACTION_IS_SET
    {
    if WV_ACTION_IS_SET
        do logging
    if TRG_ACTION_IS_SET
        do triggering
    }
```

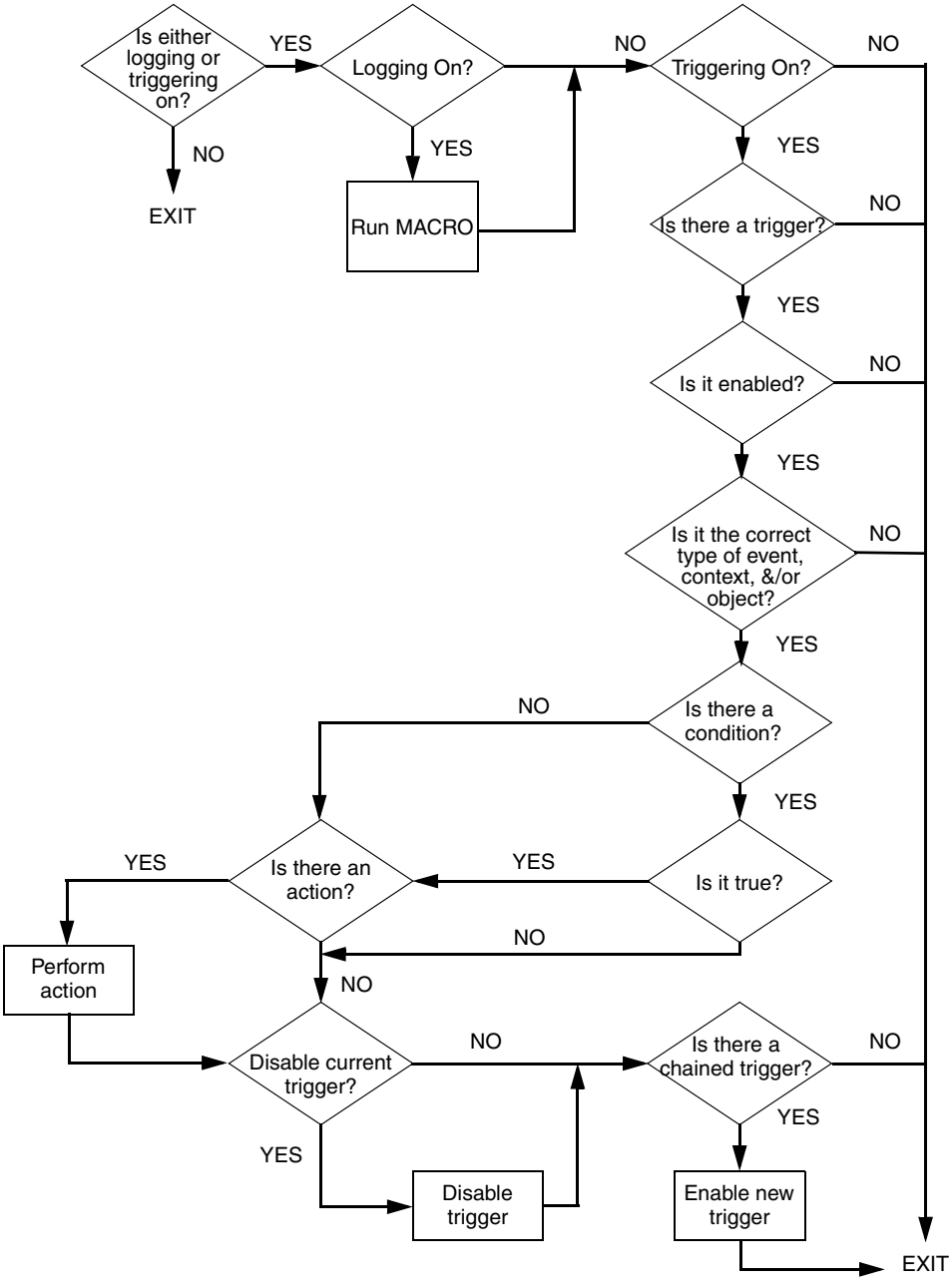
Macros that defined the status of a trigger are:

```
#define TRG_ENABLE 1
#define TRG_DISABLE 0
```

Macros that define whether the condition is a variable or a function are:

```
#define TRIGGER_COND_FUNC       0
#define TRIGGER_COND_VAR        1
#define TRIGGER_COND_LIB        2
```

Figure B-1 **Process Followed if Triggering is Activated**



Triggering Structure

The triggering structure is defined as follows:

```
typedef struct trigger
    {
    OBJ_CORE          objCore;       /* trigger object core */
    event_t           eventId;       /* event type */
    UINT16            status;        /* status of the trigger, */
    BOOL              disable;       /* check if disable after use */
    int               contextType;   /* type of context where event occurs */
    UINT32            contextId;     /* id of context where event occurs */
    OBJ_ID            objId;         /* object type */
    struct trigger    *chain;        /* pointer to chained trigger */
    struct trigger    *next;         /* ptr to next trigger in list */
    int               conditional;   /* check if a condition is set */
    int               condType;      /* check the expression type (var/fn) */
    void *            condEx1;       /* ptr to conditional expression */
    int               condOp;        /* conditional operator */
    int               condEx2;       /* second operand (constant) */
    int               actionType;    /* type of action (none, fn, lib) */
    FUNCPTR           actionFunc;    /* pointer to the action */
    int               actionArg;     /* argument passed to the action */
    BOOL              actionDef;     /* defer the action */
    } TRIGGER;
```

B.2 Using the Triggering API Functions

All the operations allowed on triggers, such as create, destroy, enable, and disable, are handled by the triggering API functions. Because these functions involve many parameters, using the GUI tools is preferable because it ensures the proper environment for the construction or modification of a trigger.

The following functions are used to manipulate the triggering structure, to detect the presence of a trigger, and to perform the action specified in the trigger definition:

Adding a Trigger to the Trigger List

trgAdd( ) adds a new trigger to the trigger list.

```
TRIGGER_ID trgAdd
    (
    event_t    event,        /* System Viewer event type as defined in
eventP.h */
    int        status,       /* initial status (enabled/disabled) */
    int        contextType,  /* type of context where event occurs */
    UINT32     contextId,    /* ID (if any) of context where event occurs */
    OBJ_ID     objId,        /* object type, if given */
    int        conditional,  /* flag specifying if there is a condition */
    int        condType,     /* flag specifying variable or a function */
    int        * condEx1,    /* pointer to conditional expression */
    int        condOp,       /* operator (==, !=, <, <=, >, >=, |, &) */
    int        condEx2,      /* second operand (constant) */
    BOOL       disable,      /* flag specifying whether to disable a trigger
                              /* after it is fired */
    TRIGGER    *chain,      /* flag specifying if trigger is chained */
    int        actionType,  /* action type associated with trigger */
    FUNCPTR    actionFunc,  /* pointer to the function */
    BOOL       actionDef,   /* defer the action */
    int        actionArg    /* argument passed to function if any */
    )
    {
    /* fill in trigger struct and add it to the trigger list; */
     /* return the trigger ID */
    }
```

Deleting a Trigger from the Trigger List

Triggers are deleted when they are removed from the trigger list. The function **trgDelete( )** also checks to see if any other triggers are still active; if none are, but triggering is still active, it turns triggering off. Triggering introduces some overhead and should be disabled if no function is present.

```
STATUS trgDelete
    (
    TRIGGER_ID trgId
    )
    {
    /* delete trigger from table; if last trigger, turn triggering off */
    }
```

Activating and Deactivating Triggering

When an event point is hit with triggering active, a check for existing triggers is performed. Because of this overhead, it is important to activate triggering only when necessary. These functions activate and deactivate triggering.

```
STATUS trgOn()
    {
    /* set evtAction to TRG_ACTION_IS_SET if not already set */
    }

void trgOff()
    {
    /* set evtAction to TRG_ACTION_IS_UNSET if it is on */
    }
```

Showing Information on Triggers

The following information is given: trigger ID, event ID, status, condition, disable trigger, and chained trigger. If *options* is 1 and *trgId* is specified, all parameters are shown for the specified trigger. If *trgId* is **NULL** all the triggers are shown.

```
STATUS trgShow
    (
    TRIGGER_ID trgId,
    int        options
    )
    {
    /* display information on specified trigger or on all triggers */
    }
```

Changing Trigger Status

trgEnable( ) sets the status of an existing trigger. This is the mechanism used to activate a chained trigger. A counter is also incremented to keep track of the total number of currently enabled triggers. This information is used by **trgDisable( )**.

```
STATUS trgEnable
    (
    TRIGGER_ID trgId
    )
    {
    /* enable trigger unless max number of triggers is already enabled */
    }
```

trgDisable( ) is used to disable a trigger. This function also checks to see if there are any other triggers still active. This is done through the counter **trgCnt**. If **trgCnt** is 0 and triggering is still on, it calls **trgOff( )**.

```
STATUS trgDisable
    (
    TRIGGER_ID trgId
    )
    {
    /* turn off trigger; if last active trigger, turn triggering off */
    }
```

Creating a User Event to Fire a Trigger

trgEvent( ) triggers a user event. A trigger must exist and triggering must have been started with **trgOn( )** or from the triggering GUI to use this routine. The *evtId* must be in the range from 40000 to 65535 on VxWorks-family Target Operating Systems, or from 4096 to 4351 on Wind River Linux.

.

```
void trgEvent
    (
    event_t evtId     /* event*/
    )
    {
    /* set a user event with the specified ID */
    }
```

C

VxWorks 6.N user.xml *Example File*

C.1 Introduction

This appendix shows an example of a complete new **user.xml** file, written specifically for VxWorks 6.*N*. The example demonstrates many of the techniques described under *14.3 The User Events Description File*, p.117. Explanations, where required, are provided in embedded comments.

If your Target Operating System (TOS) is not VxWorks 6.*N*, you should use the default **user.xml** file (see *14.3.1 Location of the User Events Description File*, p.117) for your TOS as a prototype, and be sure not to alter the range of user event IDs permitted for that TOS.

C.2 VxWorks 6.*N* user.xml Example File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EventDictionary SYSTEM "../../DTDs/EventDictionary.dtd">
<EventDictionary>
  <EventClass
    key="user"
    displayName="User Events"
    helpTopicId="VXWORKS6_user_CLASS_HELP">

    <!-- eventId range 40000-40099 inclusive,
         This is just as per the default user.xml, but with a reduced range
    -->
    <EventRangeDescription
      eventIdStart="40000"
      eventIdCount="100"
      nameRoot="EVENT_USER"
      displayNameRoot="user"
      nameRootSuffixStart="0"
      icon="images/defaultUser.gif"
      trigger="true"
      helpTopicId="VXWORKS6_EVENT_USER_EVENT_HELP"
    handler="com.windriver.windview.plugins.wv.vxworks.UserEventDescription">
      <EventParam
        type="UINT32"
        name="pc"
        formatStr="0x%08x"/>
      <EventParam
        type="BLOB"
        name="data"
      formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
    </EventRangeDescription>

    <!-- eventId range 40100-40199 inclusive
         This range has been extracted in order to change the display
         characteristics. The range will be displayed starting with
         "myUserEvent0" and use a custom icon which must be located in
         "images/myUserIcon.gif" relative to the location of this XML file.
         Since the "handler" attribute is included as shown, the user event
         number (starting at the the number defined in "nameRootSuffixStart")
         will be composited with the given icon.
    -->
    <EventRangeDescription
      eventIdStart="40100"
      eventIdCount="100"
      nameRoot="EVENT_USER"
      displayNameRoot="myUserEvent"
      nameRootSuffixStart="0"
      icon="images/myUserIcon.gif"
      trigger="true"
    handler="com.windriver.windview.plugins.wv.vxworks.UserEventDescription">
```

```
  <EventParam
    type="UINT32"
    name="pc"
    formatStr="0x%08x"/>
  <EventParam
    type="BLOB"
    name="data"
  formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
</EventRangeDescription>


<!-- leaving a gap here for eventIDs 40200-40201 inclusive. These will
     be completed using discrete UserEventDescriptions which must appear
    after all the defined EventRangeDescriptions.
-->

<!-- eventId range 40202-65535 inclusive
    This range is the remainder of the original user event definition.
    It starts at EventID 40202 and continues for the remaining 25334
    events taking us up to EventID 65535. The range will have a display
    name starting with "user202".
-->
<EventRangeDescription
  eventIdStart="40202"
  eventIdCount="25334"
  nameRoot="EVENT_USER"
  displayNameRoot="user"
  nameRootSuffixStart="202"
  icon="images/defaultUser.gif"
  trigger="true"
  helpTopicId="VXWORKS6_EVENT_USER_EVENT_HELP"
handler="com.windriver.windview.plugins.wv.vxworks.UserEventDescription">
  <EventParam
    type="UINT32"
    name="pc"
    formatStr="0x%08x"/>
  <EventParam
    type="BLOB"
    name="data"
  formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
</EventRangeDescription>

<!-- This is the end of the EventRangeDescriptions. There now follow
     any discrete EventDescriptions to fill in holes.
-->
```

```
    <!-- eventID 40200. Uses the extended icon attribute format.
        Displayed event name will be "Trace#0".
        Will have the default, auto-generated icon in the default color.
        The text "Trace0" will be composited above the icon, also in the
        default color.
    -->
    <EventDescription id="40200"
      name="MY_TRACE_EVENT_0"
      trigger="true"
      displayName="Trace#0"
      icon="text=Trace0">
      <EventParam
        type="UINT32"
        name="pc"
        formatStr="0x%08x"/>
      <EventParam
        type="BLOB"
        name="data"
      formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
    </EventDescription>

    <!-- eventID 40201. Uses the extended icon attribute format.
         Displayed event name will be "Trace#1".
        Will have the default, auto-generated icon in bright red.
        The text "Trace1" will be composited above the icon, also in
        bright red.
    -->
    <EventDescription id="40201"
      name="MY_TRACE_EVENT_1"
      trigger="true"
      displayName="Trace#1"
      icon="text=Trace1,color=#FF0000">
      <EventParam
        type="UINT32"
        name="pc"
        formatStr="0x%08x"/>
      <EventParam
        type="BLOB"
        name="data"
      formatter="com.windriver.windview.agnostic.wv.SmartHexAsciiFormatter"/>
    </EventDescription>


  </EventClass>
</EventDictionary>
```

D

Configuring VxWorks for System Viewer

D.1 Introduction

This chapter tells you where and how to configure a VxWorks Image Project to appropriately support System Viewer functionality.

For more information on VxWorks Image Projects and how to configure, customize, scale, and build a VxWorks kernel image, see *Wind River Workbench User's Guide*.

D.2 Configuring the Kernel

You add the various System Viewer components using the **Kernel Editor**.

1. In the **Project Navigator**, expand your VxWorks Image Project and double-click the **Kernel Configuration** node.
2. In the **Kernel Editor** that appears, select the **Components** tab.
3. Expand **development tool components > System Viewer components** and include, as needed (components that are not already included are shown in pale blue):
   - *D.3.1 Basic System Viewer Components*, p.169.
   - *D.3.2 Upload Method Components*, p.169
   - *D.3.3 Upload Mode Buffer Components*, p.170
   - *D.3.4 Timestamping Components*, p.170
   - *D.3.5 Triggering Components*, p.170
   - *D.3.6 Network Components*, p.170

NOTE: The easiest way to find components in the **Kernel Editor** is to click into the window and type **`CTRL+f`**. In the **Find** dialog's **Pattern** field, type an asterisk (**`*`**) followed by the first few letters of the component.

4. When you are done, save the new kernel configuration.
5. Build your project.

D.3 System Viewer Components

The following lists System Viewer kernel components grouped by functionality.

D.3.1 Basic System Viewer Components

WINDVIEW
Required to initialize and control logging.

WINDVIEW_CLASS
Required for kernel instrumentation.

D.3.2 Upload Method Components

- Upload method: **Socket via TSFS**
  Required components:
  - **WVUPLOAD_TSFSSOCK**
- Upload method: **Socket via TCP/IP**
  Required components:
  - **WVUPLOAD_SOCK**
- Upload method: **File via TSFS**
  Required components:
  - **WVUPLOAD_FILE**
- Upload method: **File via NFS**
  Required components:
  - **WVUPLOAD_FILE**
  - **NFS**
  - **NFS_MOUNT_ALL**
- Upload method: **File via netDrv**
  Required components:
  - **WVUPLOAD_FILE**

D.3.3 Upload Mode Buffer Components

RBUFF
Supports the System Viewer ring buffer library (recommended as sufficient if you are new to System Viewer).

WV_BUFF_USER
Allows you to provide a custom implementation of the **rBuff** library.

RBUFF_SHOW
Displays **rBuff** information and statistics about ring buffer performance. Optional and only available if you include **RBUFF**.

D.3.4 Timestamping Components

SEQ_TIMESTAMP
Supports sequential timestamping (recommended if you are new to System Viewer)

USER_TIMESTAMP
Supports user-defined timestamping.

SYS_TIMESTAMP
Supports BSP-specific timestamping (not supported by the simulator).

D.3.5 Triggering Components

NOTE: If yyou are stripping System Viewer components, do not remove this. It is a VxWorks component.

TRIGGERING
Adds support for triggering.

D.3.6 Network Components

WVNETD
Adds support for dual stack Network Instrumentation (wvNetDLib).

Index

A

B

C

D

E

F

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X