$RCSfile: header.mm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:41:22 $ # $RCSfile: ch05.sm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:42:08 $

Chapter 5. Library Reference

Two libraries shipped with QATS, **libCommon.a** and **libMalloc.a** contain functionality which is useful both within and outside of automated tests. The new features of these libraries are discussed in this chapter.

5.1 The Malloc Library

This release includes a memory-management testing library, based on the Conor Cahill malloc package, with modifications made by OSF. The library is compiled into **libMalloc.a**. This malloc package checks for the following types of usage errors:

- Accessing memory which has already been freed.
- Accessing uninitialized memory.
- Failures which occur within string functions such as **strcpy**.

It also provides a block-tracking system to help locate memory leaks. Keep in mind when debugging tests that there is a fixed amount of memory allocated during class initialization which is never freed, even if all the widgets of that class are destroyed. This may look like a memory leak but is not. In addition, fragmentation can cause apparent memory leaks where none exist.

5.1.1 Building With libMalloc

Any test which calls **CommonTestInit** can be compiled with the malloc package in two steps. First, simply remake the Makefile in that test directory with the command:

make IMAKE_DEFINES="-DDebugLib" Makefile

One can set up a directory for both Automated and Malloc testing by rebuilding the Makefile with this command:

```
make IMAKE_DEFINES="-DAUTOMATION -DDebugLib" Makefile
```

After this step, rebuild the test(s). All calls to **XtMalloc**, **XtRealloc** and **XtFree** will be traced.

To use this malloc library outside of QATS, your application must perform two steps. The first is to make a call to **MallocInit()** in the beginning of the program being tested and make a call to **MallocExit()** at the end of the program. The second is to link with the malloc library. Note: **MallocInit** requires two parameters: the application shell widget ID and the display ID. (See the source file **MallocInit.c**).

Note that a side effect of using **MallocInit()** is that several signals will be handled by the malloc library, thereby eliminating any signal previously defined by your program. The signals which get defined are: **SIGTERM**, **SIGSEGV**, **SIGINT**, **SIGHUP**, and **SIGQUIT**. Upon the occurrence of any of these signals, **MallocDump()** will be called and the program will terminate.

5.1.2 Default Behavior

A fill pattern is written into any memory allocated by **XtMalloc**, **XtCalloc**, **XtRealloc**. The fill pattern by default is **0x01**. Using this pattern makes it more likely that any use of uninitialized memory will produce an error.

When **XtFree** is called, several validity checks are made on the pointer being passed. The data in the malloc block beyond the size requested in the initial malloc is checked to verify that it is still filled with the original fill pattern. This is useful for catching things like:

```
ptr = XtMalloc(5);
ptr[5] = ' ';
```

The freed block is filled with a different fill pattern so that you can easily determine if you are still using freed space. The default fill pattern for the freed areas is **0x02**. For example:

```
ptr = XtMalloc(20);
bptr = ptr+10;

/* Do something useful with bptr */

XtFree(ptr);

/*
 * Now try to do something useful with bptr. It
 * should be trashed enough so that would cause
 * real problems and when debugging, you would see
```

```
 *  that it would be filled with 0x02's.
 */
```

Whenever a **bstring(3)**, **string(3)**, or **memory(3)** function is called and the environment variable **MALLOC_CHECK_STRING** is set to true, the validity of the operation is checked *if and only if* the data being referenced is in the malloc arena. For example,

```
ptr = XtMalloc(5);
strcpy(ptr,"123456");
/* oops, too much!  wrote two extra chars, '6' and '  ' */
```

Note that all output (including errors and warnings) from the malloc library goes to standard output (**stdout**) by default. Standard output can be redirected to an output file by setting the environment variable **MALLOC_OUT_FILE** to a file name.

A routine exists in the malloc library, **MallocDump()**, which will dump the malloc chain and status of any outstanding memory at the time it is called.

For tests in the QATS, a **MallocDump()** will occur automatically at the end of the test program. The "end" of a test program is defined as any of the following:

- When the following signals occur: **SIGTERM**, **SIGSEGV**, **SIGINT**, **SIGHUP**, **SIGQUIT**
- When the **Exit** button in the control dialog box is selected
- When **Close** is selected from the window menu

5.1.3 Controlling libMalloc Behavior

You may modify the functionality of the malloc package by using the following environment variables:

MALLOC_ALLOC_CHAR — The fill character written into allocated memory. Default: **0x01**.

MALLOC_FREE_CHAR — The fill character written into freed memory. Default: **0x02**.

MALLOC_OUT_FILE — The name of the file to which all malloc output is written. Default: **stdout**.

MALLOC_END_DUMP — When true, tells the library to dump out the malloc chain and leak information at the end of a program. Default: **true**.

MALLOC_LOG_INFO — When true, all calls to **XtMalloc**, **XtRealloc,** **XtCalloc**, and **XtFree**, are logged when they occur. Default: **false**.

MALLOC_CHECK_STRING — When true, check for problems when string/memory functions are referenced. Default: **false**. Note: this feature has a significant performance impact.

5.2 Test Convenience Routine Library (libCommon.a)

The library **libCommon.a** groups a number of functions which can be called from tests. Several of these, like **CommonTestInit** and **CommonPause** have been described elsewhere. In addition to test convenience functions, **libCommon.a** provides additional functionality that can be useful for manual as well as automated testing. These functions and features are described in the following sections.

5.2.1 Test Results Bookkeeping for Manual testing

A feature has been added to the **tests/lib/Manual/Common** library which can be useful to keep track of successes and failures during manual (non-automated) testing. When this library is compiled with the flag **-DREGRESSION**, the single **Continue** button in the Instruction Box is replaced with two buttons, **Pass** and **Fail**. The number of passes and fails in each test are tallied and printed out to a file called **RUN_output**. This file will be appended by each test run in the same directory.

5.2.2 Command Line Options

A large number of command lines are supported by convenience routines which parse the command line. A summary of the command line options follows.

-l Disable displaying of the MessageBox for user instructions (Ignore Pause())

-I *file* Read an alternative instruction file. The default is <test_name>.dat.

-h Print the usage message.

-w *number* Specify width of MessageBox for user instructions. Default is 80.

-u *string* Specify a string which will available to the test writer as the global variable **char * UserData**.

-f *font* Use *font* as the "fallback" font to be used when a font specified in the test code is not available. The default is **fixed**.

-p *font* Specify the font to be used in the MessageBox for user instructions. Default is **fixed**.

The following options are for use only for the **libCommon** library in **tests/Auto/lib/Common**.

-a	Generate full error reporting on visuals in output file.
-r	Record mode.
-b	Batch mode for playback (default).
-m	Interactive comparison of visuals.
-c	Visual comparisons ignored.
-C *color*	Display differences using *color* in the interactive window. For use only with the **-m** flag.
-D *seconds*	Execute a delay of *seconds*.
-W	Produce xwd-format window dumps files for visual failures.
-S *file*	Alternative name for script file.
-O *file*	Alternative name for output file.
-V *file*	Alternative name for visual image file.
-T	Trace mode; print each script command as executed.

5.2.3 New Callable Functions

void CommonGetOptions(int *argc*, **char *****argv* **)**

Input: *argc*, *argv*

Output: None

Purpose: This function will read the command line options and set the appropriate flags to represent those options. Called from **CommonTestInit**().

void CommonTestInit(int *argc*, **char *****argv* **)**

Input: *argc*, *argv*

Output: None

Purpose: Performs the following functions:

1) Set up signal handlers (see section 4.1.2 for specific signals). The signal handler simply prints a message and exits.

2) Call **GetOptions()** to parse the command line options.

3) Call **XtToolkitInitialize**() to initialize the Intrinsics.

4) Create an Application context.

5) Create a connection to the display by calling **XtOpenDisplay**(). A global variable **display** is set to the return value.

6) Create an ApplicationShell with XmNallowShellResize set to true. The global variable **Shell1** is set to the return value.

Globals: Global variables **display** and **Shell1** are initialized.

XtCallbackProc CommonGenericCB(Widget *w*, **XtPointer** *client_data*, **XtPointer** **call_data*)

Input: Normal callback parameters

Output: None

Purpose: This is a "generic" callback routine that will simply print to **stdout** the reason why the callback was triggered. It can be used when a simple verification of a callback is needed.

Pixel CommonGetColor(char **ColorString*)

Input: A character string corresponding to a valid color in the rgb.txt file on a system.

Output: A color cell entry corresponding to the passed color string. If the color does not exist, NULL will be returned.

Purpose: This routine will convert a string representation of a color into a Pixel value.

XmFontList CommonGetFontList(char **FontString*)

Input: A character string describing a font.

Output: The corresponding XmFontList.

Purpose: This routine will convert a string representation of a font into a XmFontList. If the specified font does not exist on that system, then **default_font** is used. The variable **default_font** can be set on the command line. A default value for **default_font** is defined in the file **testlib.h**.

char CommonCsToRs(XmString *cs*)

Input: An XmString.

Output: A character string representation of that XmString.

Purpose: This function will convert a simple one-segment compound string into a regular character string.

void CommonPause()

Input: None

Output: None

Purpose: This function will place on the screen at (500, 0) an InformationDialog which contains instructions to the user, read from a **.dat** file.

void CommonDumpHierarchy(Widget *widget*, **FILE** **fileptr***)**

Input:	A widget and a pointer to the output file.
Output:	None
Purpose:	This function will "dump" all the widget names (fetched with **XtName**) which are descendents of the input widget. The output will be written to the file *fileptr*.

Note: you may dump the hiearchy of any test at any time by pressing **<Ctrl> Btn1** on the **Exit** button of the test's InstructionBox.