ROBOT USAGE
OF THE
XSPACE PROGRAM

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

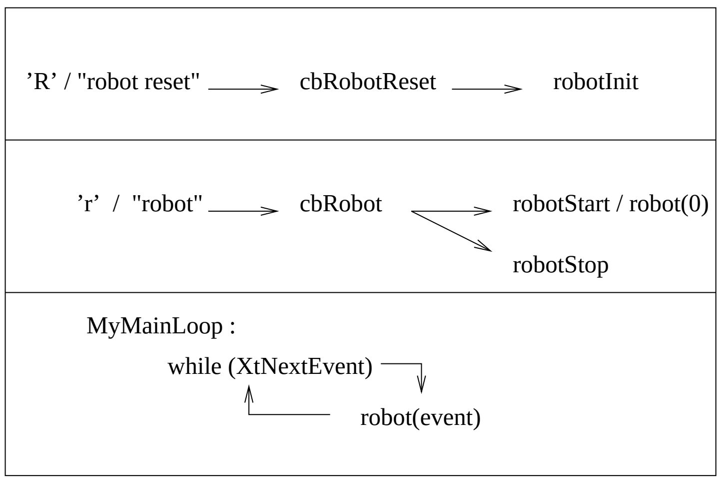
Report EWI-ENS 05-01
April 20, 2005

Last revision: May 10, 2005

1. INTRODUCTION

This document describes how the robot of the *Xspace* program works.
The robot is the demonstration facility of the *Xspace* program, whereby automatically extractions can be executed and displayed.

The following robot interface calls are used:



The *Xspace* program starts by calling robotInit to init the robot facility. As you can see in appendix B, function robotInit opens default a robot program file with the name "program". To use the robot facility, you must start the robot program. This can be done in the Extract menu or just with the hot-key 'r'. This hot-key is a toggle and can also be used to stop the robot program. To restart a robot program from the beginning you can use keys 'R' and 'r'. Note that after the end of the program a robot reset is done automatically.

A handshake event is used to keep the robot program going. This handshake event is only produced for the "window" statements. See in appendix A function makeEvent. The other statements do not return from makeEvent. Also makeEvent must look for events, else they are not processed. I added a event loop to makeEvent to dispatch events. Also a return from the makeEvent function is done, when the pointer position is changed. Thus, you can stop the running robot program, by moving the pointer with the mouse. This is a build-in feature.

2. THE PROGRAM FILE

The robot program file uses its own ASCII-text file format. One robot statement can be given on each line of the file.

The following program file statements can be used:

program file statement	comment
\n	empty line (skipped)
#	comment line (skipped)
click window	do generate event
corner window	do generate event
move window	do generate event
press window	do generate event
release window	do generate event
goto label_name	rewind file, search label_name
call proc_name	rewind file, search proc_name
return	end of procedure (does fseek)
text message	display message in window
system command	does system(command)
wait seconds	sleep(seconds)
delay number	delay between each statement
label name	place of goto statement
proc name	start of procedure
xyz	rewind file, search proc_name ’xyz’

The argument ’window’ can be a main-menu name or a sub-menu name. The message of the text-statement is written to stderr, when the message window is not used. To use the message window, set the X-window resource:

```
Xspace*messageWindow    : True
```

3. PROGRAM FILE EXAMPLE

The following example shows a program file which is using the message window. After the 'start' label, it displays a "Welcome ..." text. The text is each time (after the delay) a word longer. The delay is needed, else it goes too fast. Two procedures are called: 'click_capacitance' and 'click_run_free'. The program is looping. To forcome executing the procedures without calling them, on second line a jump to the 'start' label is made.

```
delay 20000
goto start

proc click_capacitance
move Options
click capacitance
wait 1
return

proc click_run_free
move Extract
click display clear
click extract again
return

label start
text Welcome
text Welcome to
text Welcome to this
text Welcome to this demo!
text Welcome to this demo!  Welcome
text Welcome to this demo!  Welcome to
text Welcome to this demo!  Welcome to this
text Welcome to this demo!  Welcome to this demo!
text Welcome to this demo!  Welcome to this demo!  Welcome
text Welcome to this demo!  Welcome to this demo!  Welcome to
text Welcome to this demo!  Welcome to this demo!  Welcome to this
text Welcome to this demo!  Welcome to this demo!  Welcome to this demo!
wait 1
text We are running a loop with two displays
wait 2
text Setting 'capacitance' "on"/"off" and doing 'extract again'.
wait 2
click_capacitance
click_run_free
goto start
```

4. APPENDICES

APPENDIX A -- Source file part: makeEvent

```
Private void makeEvent ()
{
    static int delay = 1;
    static char buf[1024];
    XEvent event;
    char *command;
    int d;

    if (pointerChanged ()) {
        cbRobot ((Widget)0, (caddr_t)0, (caddr_t)0); // robotStop
        return;
    }

    while (fgets (buf, sizeof (buf), program)) {

        command = word (buf);

        if (command[0] == '\0')    ;    /* empty line */
        else if (command[0] == '#') ;    /* comment */
        else if (strsame (command, "click")
             ||  strsame (command, "corner")
             ||  strsame (command, "move")
             ||  strsame (command, "press")
             ||  strsame (command, "release")) {
            /* Use 'line', because window names can contain spaces.
             */
            if (generate (command, line (NULL))) return;
        }
        else if (strsame (command, "goto"  )) jump ("label", word (NULL));
        else if (strsame (command, "call"  )) call (word (NULL));
        else if (strsame (command, "return")) doreturn ();
        else if (strsame (command, "text"  )) domessage (line (NULL));
        else if (strsame (command, "system")) dosystem (line (NULL));
        else if (strsame (command, "wait"  )) dowait (word (NULL));
        else if (strsame (command, "delay" )) delay = atoi (word (NULL));
        else if (strsame (command, "label" )) ;
        else if (strsame (command, "proc"  )) ;
        else call (command); /* assume subroutine call */

        for (d = delay; d > 0; d--) {
            if (XtPending ()) {
                XtNextEvent (&event);
                dispatchEvent (&event);
                if (!running_state) return;
            }
                       }

        if (pointerChanged ()) {
            cbRobot ((Widget)0, (caddr_t)0, (caddr_t)0); // robotStop
            return;
        }
    }

    cbRobotReset ((Widget)0, (caddr_t)0, (caddr_t)0); // robotInit
}
```

```
/* Jump to procedure or label
 */
Private void jump (char *type, char *name)
{
    static char buf[1024];

    rewind (program);

    for (;;) {
        if (!fgets (buf, sizeof (buf), program)) {
            say ("%s '%s' not found", type, name);
            if (*type == 'p') doreturn ();
            return;
        }
        if (strsame (word (buf),  type)
        &&  strsame (word (NULL), name))
            return;
    }
}

/* Perform subroutine call,
 * by saving return addres on stack
 * and jumping to "proc <callee>" label.
 */
Private void call (char *callee)
{
    if (sp >= STACKSIZE) { say ("Stack overflow"); return; }
    stack[sp++] = ftell (program);
    jump ("proc", callee);
}

/* Return from subroutine
 */
Private void doreturn ()
{
    if (sp <= 0) { say ("Stack underflow"); return; }
    fseek (program, stack[--sp], 0);
}

Private void domessage (char *text)
{
    xMessage (text);
    XSync (display, 0);
}

Private void dosystem (char *cmd)
{
    Debug (fprintf (stderr, "system: %s\n", cmd));
    (void) system (cmd);
}

Private void dowait (char *seconds)
{
    XFlush (display);
    sleep (atoi (seconds));
}
```

APPENDIX B -- Source file part: interface functions

```
void robotInit ()
{
    char *programFile = paramLookupS ("disp.program_file", "program");

    if (program) (void) fclose (program);
    program = fopen (programFile, "r");
    pointerChanged (); /* to set the Root window */
    sp = 0; /* reset stack pointer */
    running_state = 0;
}

void robotStop ()
{
    running_state = 0;
}

int robotStart ()
{
    if (!running_state && program) running_state = 1;
    return running_state;
}

void robot (XEvent *event)
{
    static int shake = 1;

    if (!running_state) return;

    if (!event) {               /* boot the robot */
        XSync (display, 0);
        (void) pointerChanged ();
        makeEvent ();
        shake = 0;
        return;
    }

    /* An handshake event for synchronization */
    if (event -> type == ClientMessage) shake = 1;

    if (shake == 1 && QLength (display) == 0) {
        XSync (display, 0);
        if (QLength (display) > 0) return;
        makeEvent ();
        shake = 0;
    }
}
```