

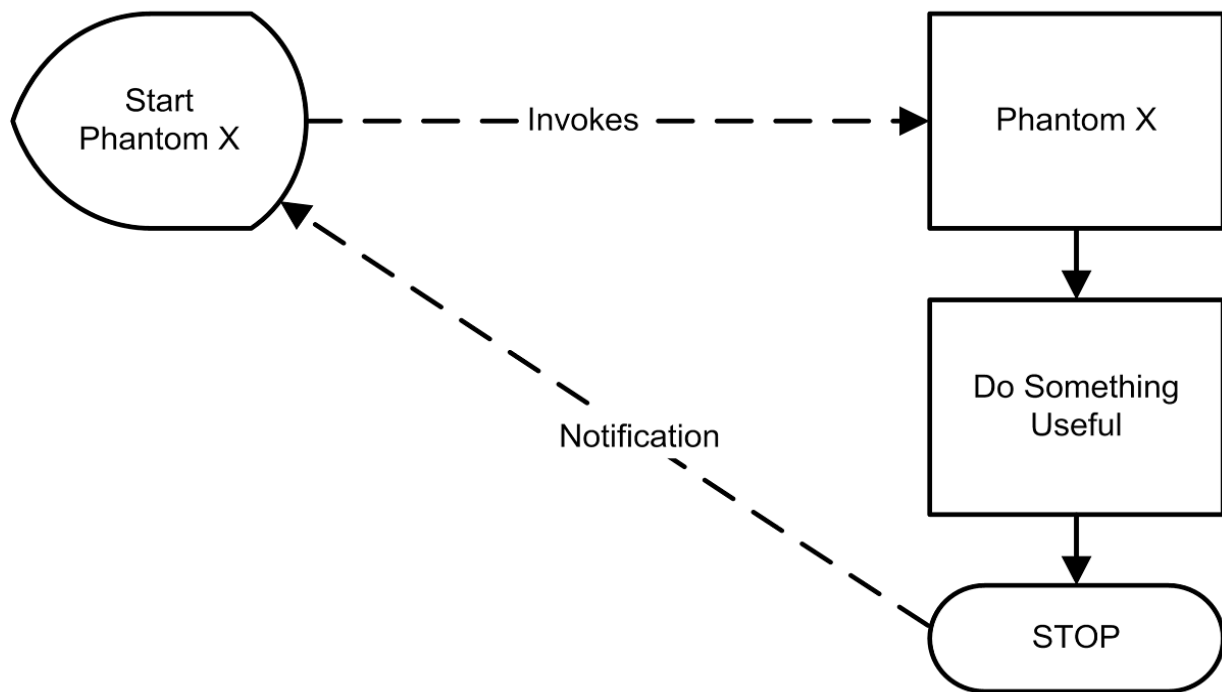
A UniVerse Web tool for Phantom Control

Ross Morrissey
OSDA Spring 2002 Newsletter

In this article, I'll present a cool web interface to monitor, stop, and restart phantoms. I'll start by going through a simple mechanism for starting and stopping phantoms on a green screen, then show how this was adapted for use on an intranet in UniVerse running in a Unix environment with a local Apache web server. Many of the techniques used could be extended to different problems in other environments. We'll use a single html form and introduce the concept of state.

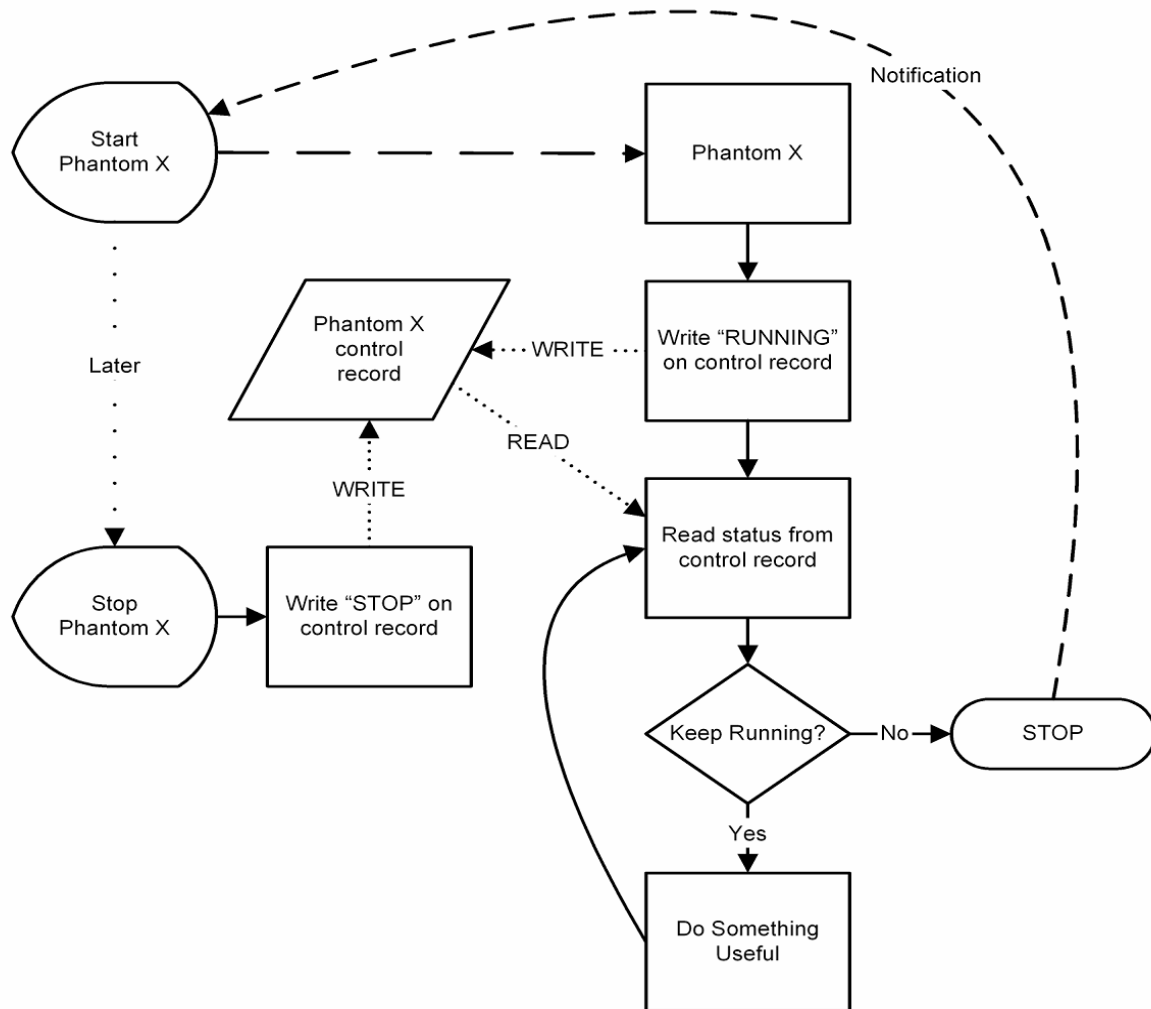
The PHANTOM Command

The UniVerse documentation succinctly states: "Use PHANTOM to start a process that executes in the background." This is very convenient for tasks like running reports where you don't want to tie up your green screen while they run. You get a message on your screen when the report finishes - if you're still logged on:



Phantom Services

We can run programs indefinitely as a phantom - posting programs, data transfer programs, and in this example, process control. Since the Phantom runs without terminal input, we need a way to tell the phantom to stop running. Here is a simple method where the phantom checks an item in a file to see if it's time to stop. Note that when the phantom terminates, it attempts to notify the user who started the phantom. If that user has logged off, we won't get a message anywhere.



This is a simple example of what is commonly known as a service. A web server usually runs as a service. Most services handle data requests, either to move the data from one point to another, transform or post some data, or perform a lookup.

Problems

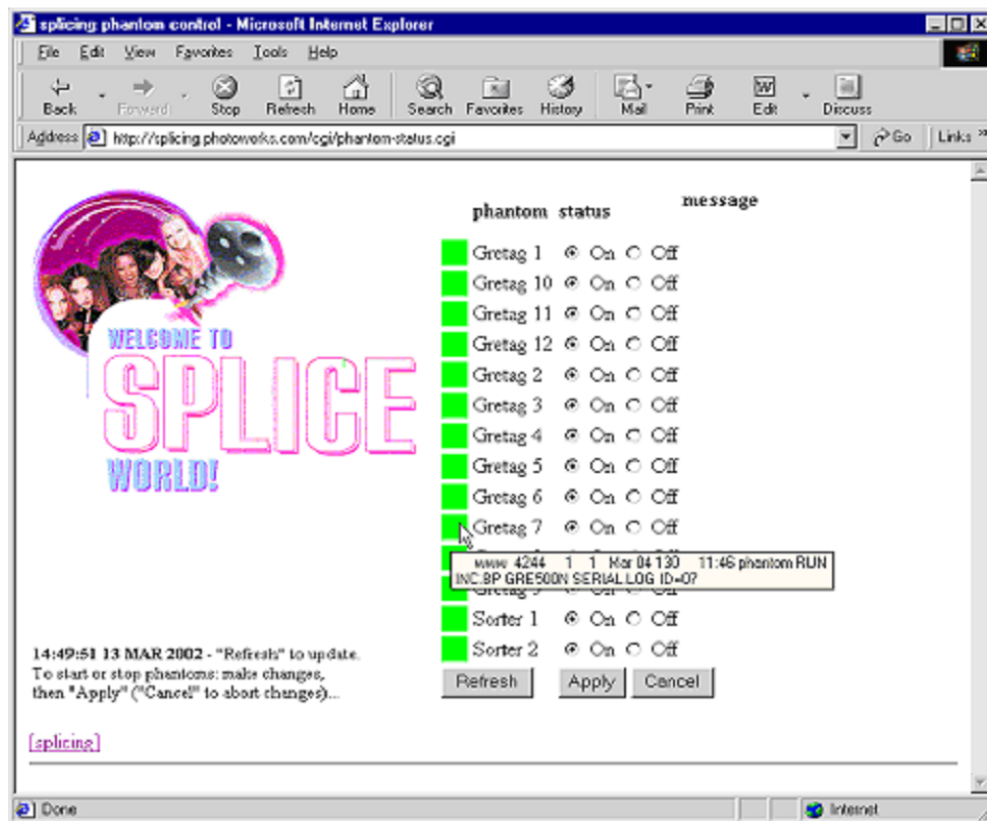
If phantoms are a good way to manage a service, how do we manage the phantoms? If we have ten versions of the phantom running for different partners or pieces of equipment, how do we know which phantom goes with which? How do we keep track of their current status? If they run 24x7 for weeks, who remembers the exact command to restart them? If they require restarting randomly at all hours of the day, who will take care of this, the end users or your DP staff?

One Web Solution

This phantom control page has been in production use for about four years now, and elicits a full range of responses, from the yawns of the production staff who interact with it daily, to the grins of visiting UniVerse programmers. It was far easier to build than an equivalent green screen (or even VB) application might have been. The interface is completely intuitive and pretty close to self-documenting; Let's step through the code with commentary and related exhibits interspersed.

For each of a series of phantoms (in this case a bi-directional interface talking to film splicing and sorting equipment) we display a simple visual cue (a green box) indicating the phantom status and give the user an intuitive method to "switch off" a phantom. To make this screen useful to our technical staff, the relevant output of a Unix "pf" command is used as the alternate text for each green box; this is displayed when the user pauses their mouse pointer over one of the green boxes. I can't take credit for the SPLICE WORLD! graphic, but it does serve the purpose of identifying this page and its purpose quickly. A quick glance is all it takes to see that all the splicing phantoms are running normally.

Splicer? Bidirectional? A splicer splices barcoded rolls of 35mm film into reels of 60 to 80 barcode delimited rolls. Some splicers use a "unidirectional interface" – we listen on the printer report for a report of original and replacement barcodes for rolls spliced. A "bidirectional interface" like the Gretag splicers use has the splicer check the film type on the host computer before splicing. Unidirectional and Bidirectional interfaces exist in many environments; a simple piece of medical lab equipment may report results for a standard set of tests for each barcoded sample while more sophisticated equipment may query the host computer for a specimen-specific battery of tests to perform before reporting results.



In real life, these green boxes aren't always green. When a phantom is "switched off" the box is shown as red until it terminates, at which point it is greyed out. Phantoms that end-users shouldn't shut down are displayed, but without the off switch. We'll step through the code showing how the web page is built.

Shell script

The UniVerse program is invoked by a simple shell script **phantom-status.cgi** in the cgi directory on the splicing department computer. For more details about this, see [Using Web Forms to access UniVerse](#) from the Winter 2002 OSDA Newsletter.

```
bash-2.01$ more phantom-status.cgi
#!/bin/sh
echo Content-type: text/html
echo
cd /splice/INC
/usr/opt/uv/bin/uv PHANTOM.STATUS
```

UniVerse program PHANTOM.STATUS

We start by opening our HTML template file and our phantom control and parameter files. We then shell out to Unix and perform a Unix **ps -ef** "process status" which we'll cache locally. The HTML template is read in, and we determine what arguments, if any, the user passed in. No arguments will be set when they first arrive at the page.

```
* PHANTOM.STATUS
*
* Ross Morrissey July 17th 1998
*
* Invoked from phantom-status.cgi
*
$INCLUDE INC.EQU DCF
$INCLUDE INC.EQU PHC
OPEN 'HTML' TO FILE.HTML ELSE PRINT "PHANTOM.STATUS ERROR 1 -
contact MIS" ; STOP
OPEN 'DCF' TO FILE.DCF ELSE PRINT "PHANTOM.STATUS ERROR 2 -
contact MIS" ; STOP
OPEN 'PHC' TO FILE.PHC ELSE PRINT "PHANTOM.STATUS ERROR 3 -
contact MIS" ; STOP

EXECUTE 'SH -c "ps -ef"' CAPTURING PS.LISTING

READ ITEM.HTML FROM FILE.HTML, 'PHANTOM.STATUS' ELSE CALL PRINT
"PHANTOM.STATUS ERROR 5 - contact MIS" ; STOP

CALL PARSE (LABELS, VALUES)
LOCATE "Button" IN LABELS SETTING POS THEN
FORM.ACTION = VALUES<POS>
END ELSE
FORM.ACTION = "Refresh"
END

BEGIN CASE
CASE FORM.ACTION = "Refresh"
LABELS = ""
VALUES = ""
CASE FORM.ACTION = "Cancel"
LABELS = ""
VALUES = ""
CASE FORM.ACTION = "Apply"
NULL
END CASE
```

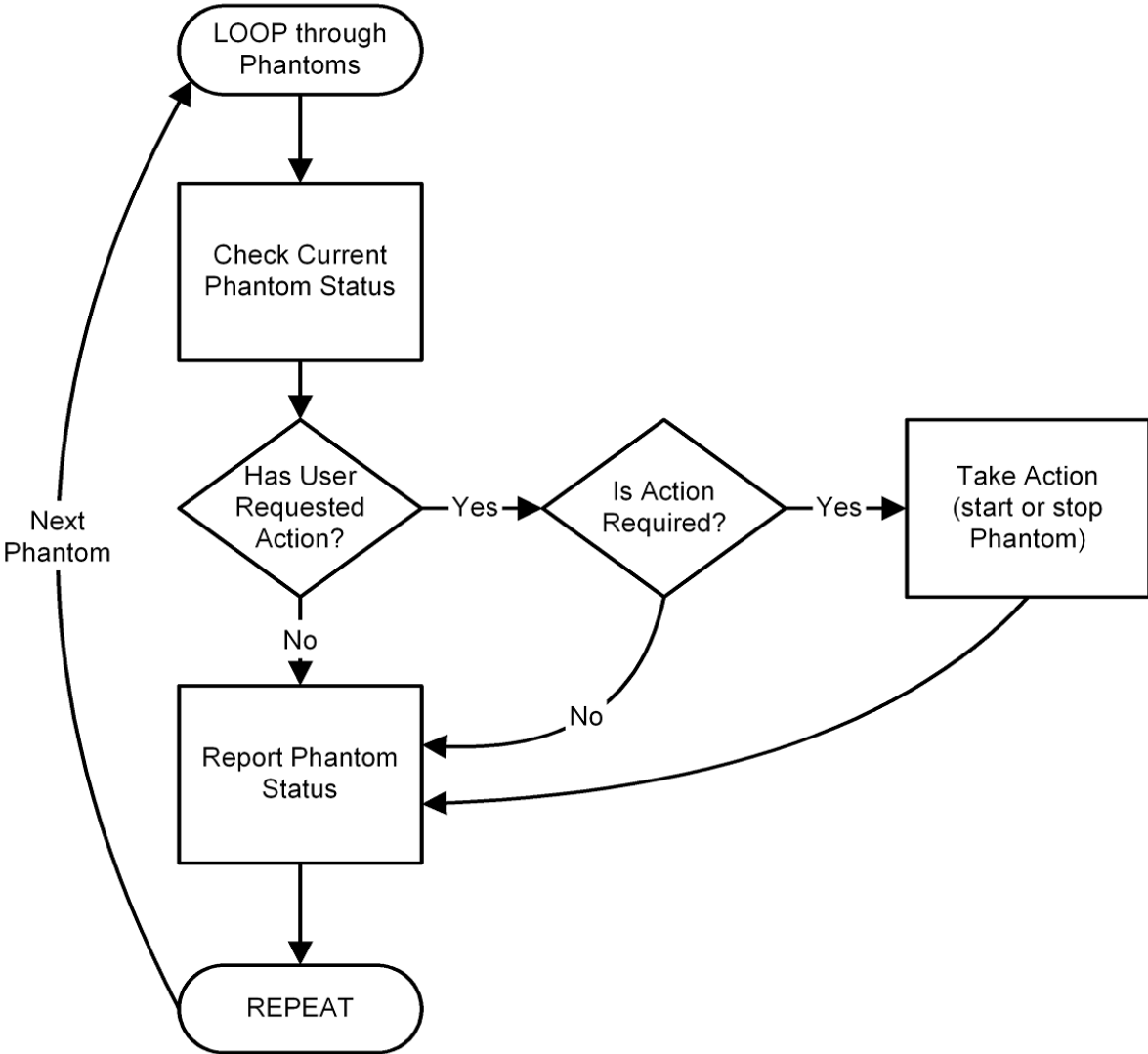
Refresh Apply Cancel

If the user selects Refresh, or they hadn't selected any button because this is the first time the page is loaded, any other arguments are ignored. If they select Apply, we may be coming from a session where they may have identified phantoms to start or stop. A cancel button is included as a "safe" option to avoid confusion about whether refresh will ignore "on" or "off" mistakes.

```
DEBUGINFO=' '
HTML.TEXT=""
```

We cycle through a parameter file (DCF) containing a list of phantoms we expect to see running. Some of these will be monitored by this web page. A phantom control file (PHC) handles the signaling. By searching the cached ps output for the startup command we can locate each phantom to determine if it is running. It is important to ensure that the command line is unique for each phantom, even by adding a command line argument that is actually ignored by the phantomed routine.

```
DCF.LIST=1
EXECUTE "SSELECT DCF BY DESC" CAPTURING OUTPUT RTNLIST DCF.LIST
LOOP
```



```

READNEXT DCF.ID FROM DCF.LIST ELSE EXIT

READ ITEM.DCF FROM FILE.DCF, DCF.ID ELSE
    DEBUGINFO<-1>='DCF "' :DCF.ID:'"' not on file.'
    CONTINUE
END

```

DCF @ID	-1
DCF\$PORT	/dev/tty104
DCF\$DESC	Sorter 2
DCF\$MAXTIME	60
DCF\$errorLOG	SORTER.2
DCF\$STARTUP.COMMAND	RUN INC.BP SORTER SORTER.2 AUTO (E
DCF\$PHC.ID	SORTER.2
DCF\$CONTROL.VIA.WEB	1
DCF\$ALLOW.WEB.ON	1
DCF\$ALLOW.WEB.OFF	1

```

STARTUP.COMMAND =
TRIMB(FIELD(ITEM.DCF<DCF$STARTUP.COMMAND>,"(",1))
RUNNING.AT.UNIX = INDEX(PS.LISTING, STARTUP.COMMAND,1)
IF RUNNING.AT.UNIX THEN
    PS.LINE = DCOUNT(PS.LISTING[1,RUNNING.AT.UNIX],@AM)
    PS.TEXT = PS.LISTING<PS.LINE>
END ELSE
    PS.TEXT = ""
END
PHC.ID = ITEM.DCF<DCF$PHC.ID>
READV PHC.STATUS FROM FILE.PHC, PHC.ID,PHC$COMMAND ELSE
PHC.STATUS = ""
ABOUT.TO.STOP = INDEX(PHC.STATUS,"STOP",1)

```

We search the arguments for any phantom that is being halted or launched.

```

PHANTOM.ID=TRIM(DCF.ID)
PHANTOM.DESC=ITEM.DCF<DCF$DESC>

LOCATE PHANTOM.ID IN LABELS SETTING POS THEN
    LINE.ACTION = VALUES<POS>
END ELSE
    LINE.ACTION = "ignore"
END

STARTING = 0
STARTED = 0
STOPPING = 0
BEGIN CASE
    CASE LINE.ACTION = "ignore"
        NULL
    CASE LINE.ACTION = "on"
        IF NOT(RUNNING.AT.UNIX) THEN
            STARTING = 1
            GOSUB START.PHANTOM
        END
    CASE LINE.ACTION = "off"
        STOPPING = 1
        IF RUNNING.AT.UNIX AND NOT(ABOUT.TO.STOP) THEN
            GOSUB STOP.PHANTOM

```

```

        END
      CASE 1
        NULL
    END CASE

```

In addition to the "phantom state" variables RUNNING.AT.UNIX, ABOUT.TO.STOP, STARTING, and STOPPING, we set some parameters that determine the buttons enabled for each phantom. In this way we control what photos an intranet user can monitor, start, or stop. This is our access control system.

```

WEB.DISPLAYABLE = ITEM.DCF<DCF$CONTROL.VIA.WEB>
WEB.STARTABLE = ITEM.DCF<DCF$ALLOW.WEB.ON>
WEB.STOPPABLE = ITEM.DCF<DCF$ALLOW.WEB.OFF>
BUTTON.TEXT = ""

```

There are twelve cases to worry about: The phantom can be running or not running; a start or stop may have just been requested, or a stop may be pending; the user may or may not have permission to start or stop the phantom. In practice, a couple of these cases overlap, and ten unique outcomes are described below.

```

BEGIN CASE
  CASE NOT(WEB.DISPLAYABLE)
    CONTINUE

```

We don't show this phantom on the web page.

```

CASE RUNNING.AT.UNIX AND STOPPING
  ICON = '<td>'
  BUTTON.TEXT = '<td colspan=4><td><li>Stopping'

```

 Sorter 2 • Stopping

This is displayed when the user applies an "off" action. Note that no buttons are shown. The user cannot take any further action until the phantom has actually terminated.

```

CASE RUNNING.AT.UNIX AND ABOUT.TO.STOP
  ICON = '<td>'
  BUTTON.TEXT = '<td colspan=4><td><li>Stopping (prior
request)'

```

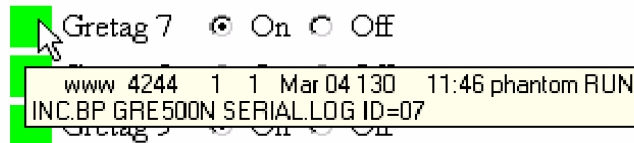
 Sorter 2 • Stopping (prior request)

Although the phantom is running, a request to stop it was placed earlier. If this status is displayed for longer than any processing or wait cycle you expect from your phantom, you will want to investigate and possibly kill the phantom at Unix.

```

CASE RUNNING.AT.UNIX AND WEB.STOPPABLE
  ICON = '<td>'
  BUTTON.TEXT = '<td><input checked type="radio"
name="":PHANTOM.ID:"" value="ignore"><td>On'
  BUTTON.TEXT:= '<td><input type="radio"
name="":PHANTOM.ID:"" value="off"><td>off<td>'

```



This is the most common scenario. The phantom is running and the user can stop it. The alt="":PS.TEXT:" portion of the image tag "hides" the ps information for each process. In this way, DP staff can check this detail without grepping through a unix ps listing for an exact run command - without typos. This technical detail is elegantly hidden from the user, but they could find it if they were asked (perhaps over the phone).

The "radio" button is used to both display and control the phantom status. The button name identifies each phantom and helps hold "state" information in the web page. The web form value of the initially checked box is set to "ignore." We skip these when "applying" changes. In this way if the user changes the value from "on" to "off" and back to "on," we don't perform any processing.

State Web clients are "stateless." They are not connected to the host. They have no open files; they are not running programs. We can remember which "state" the user's session was in by saving the "state" or "session" information in one of three places: the host computer, the client computer, or the web page itself. To store the information on the host or client, we use cookies – a hard disk "sandbox" on the client reserved for this web server. If it doesn't contain the session information, it contains a pointer to the saved session information on the host. A far simpler, but much less secure method, is hiding the session information within a web form to be resubmitted to the host during the next transaction. This is the method we use here. The button name identifies each phantom, the initial value identifies phantom status, and the returned value identifies action to take.

```

CASE NOT(RUNNING.AT.UNIX) AND STARTED AND WEB.STOPPABLE
  ICON = '<td>'
  BUTTON.TEXT = '<td><input checked type="radio"
name="" :PHANTOM.ID:'' value="ignore"><td>On'
  BUTTON.TEXT:= '<td><input type="radio"
name="" :PHANTOM.ID:'' value="off"><td>off<td><li>' : START.MESSAGE

```

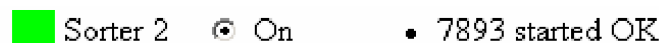


Displayed when the user applies an "on" action. We return the PID from the phantom command – this is usually returned when a phantom is started at TCL. To see the entire return string, the user mouses over the green box.

```

CASE NOT(RUNNING.AT.UNIX) AND STARTED
  ICON = '<td>'
  BUTTON.TEXT = '<td><input checked type="radio"
name="" :PHANTOM.ID:'' value="ignore"><td>On'
  BUTTON.TEXT:= '<td><td><td><li>' : START.MESSAGE

```



This process was just started, but the user lacks permission to stop it. This would be the uncommon situation where the user only had permission to start a phantom, but could not stop it on their own.

```

CASE NOT(RUNNING.AT.UNIX) AND STARTING
  ICON = '<td>'

```

```

        BUTTON.TEXT = '<td><input checked type="radio"
name="" : PHANTOM.ID: "" value="ignore"><td>On'
        BUTTON.TEXT:= '<td><td>off<td><li>' : START.REPORT

```

 Sorter 2 On Off

This is a serious error – the PHANTOM verb failed at TCL. Even if you were trying to PHANTOM a program that didn't exist, PHANTOM would normally return a PID, then terminate. In this case a PID was not returned.

```

        CASE NOT(RUNNING.AT.UNIX) AND WEB.STARTABLE
            ICON = '<td>'
            BUTTON.TEXT = '<td><input type="radio"
name="" : PHANTOM.ID: "" value="on"><td>On'
            BUTTON.TEXT:= '<td><input checked type="radio"
name="" : PHANTOM.ID: "" value="ignore"><td>off<td>'

```


 Sorter 2 On Off

We show a grey box when the phantom is not running and the control file is consistent with this state. Many phantoms are stopped routinely, and this shouldn't be considered a serious error. This is why a red box is not used.

```

        CASE RUNNING.AT.UNIX
            ICON = '<td>'
            BUTTON.TEXT = '<td><input checked type="radio"
name="" : PHANTOM.ID: "" value="ignore"><td>On'
            BUTTON.TEXT:= '<td><td><td>'

```

 Sorter 2 On

The phantom is running, but the user does not have permission to stop it from the web page.

```

        CASE 1
            ICON = '<td>'
            BUTTON.TEXT = '<td><td>'
            BUTTON.TEXT:= '<td><input checked type="radio"
name="" : PHANTOM.ID: "" value="ignore"><td>off<td>'

```

 Sorter 2 Off

The phantom is stopped and the user does not have permission to restart it from the web page.

END CASE

HTML.TEXT<-1>= '<tr>' : ICON: '<td>' : PHANTOM.DESC: BUTTON.TEXT

This line builds the next line of the html table containing the statuses.

REPEAT

HTML PHANTOM.STATUS

```
<html><head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<title>splicing phantom control</title>
</head>
<body bgcolor="#FFFFFF">

<table><tr><td valign=top><td rowspan=2>

  <table><tr><td>
    <th align=left>phantom
    <th colspan=4 align=left>status<th align=left>message

    <form action=/cgi/phantom-status.cgi method="GET" name="apply">

      <u><%StatusList%></u>

      <tr>
        <td colspan=2><input type="submit" name="Button" value="Refresh">
        <td colspan=5><input type="submit" name="Button" value="Apply">
        <input type="submit" name="Button" value="Cancel">

      </tr>
    </table>

    <tr>
      <td valign=bottom>
        <font size=-1>
          <b><%Time%></b> - &quot;Refresh&quot; to update.<br>
          To start or stop phantoms: make changes,<br>
          then &quot;Apply&quot; (&quot;Cancel&quot; to abort changes)...
        </font>

      </td>
    </tr>
  </table>

  <a href="/">[splicing]</a>
</body>
</html>
```

```
ITEM.HTML=CHANGE(ITEM.HTML, '<%Time%>', TIMEDATE())
ITEM.HTML=CHANGE(ITEM.HTML, '<%StatusList%>', HTML.TEXT)
```

We swap the status block we just built into the web page, and display it...

```
PRINT ITEM.HTML
```

```
STOP
```

```
START.PHANTOM:
```

```
EXECUTE "PHANTOM " : STARTUP.COMMAND CAPTURING START.REPORT
IF (START.REPORT[1,15] EQ "Phantom process") THEN
  STARTED = 1
  START.MESSAGE = OCONV(START.REPORT,"MCN") : " started OK"
  START.REPORT = START.REPORT
END ELSE
  STARTED = 0
```

```
END  
RETURN
```

We have a bit of legacy code left over from a green screen phantom interface. If the item is locked, a red box will be displayed initially and the green box will reappear when the page is refreshed.

```
STOP.PHANTOM:  
  FOR CTR=1 TO 10  
    READU ITEM.PHC FROM FILE.PHC, PHC.ID LOCKED  
      RQM ; RQM  
      CONTINUE  
    END ELSE  
      ITEM.PHC=' '  
    END  
  
    STOP.COMMAND = ITEM.DCF<DCF$STOP.COMMAND>  
    IF (STOP.COMMAND NE "") THEN  
      ITEM.PHC<PHC$COMMAND> = STOP.COMMAND  
    END ELSE  
      ITEM.PHC<PHC$COMMAND>='REQUEST STOP'  
    END  
    WRITE ITEM.PHC ON FILE.PHC, PHC.ID  
    EXIT  
  NEXT CTR  
  RETURN  
END
```

Summary

Lightweight intranet web tools like this are easy and fun to build, and they look cool too; but their true value lies in how well they respond to problems with phantom management: *If we have ten versions of the phantom running for different partners or pieces of equipment, how do we know which phantom goes with which?* The machine names are displayed... *How do we keep track of their current status?* Green means on... *If they run 24x7 for weeks, who remembers the exact command to restart them?* Click "on"... *If they require restarting randomly at all hours of the day, who will take care of this, the end users or your DP staff?* More power to the end users!