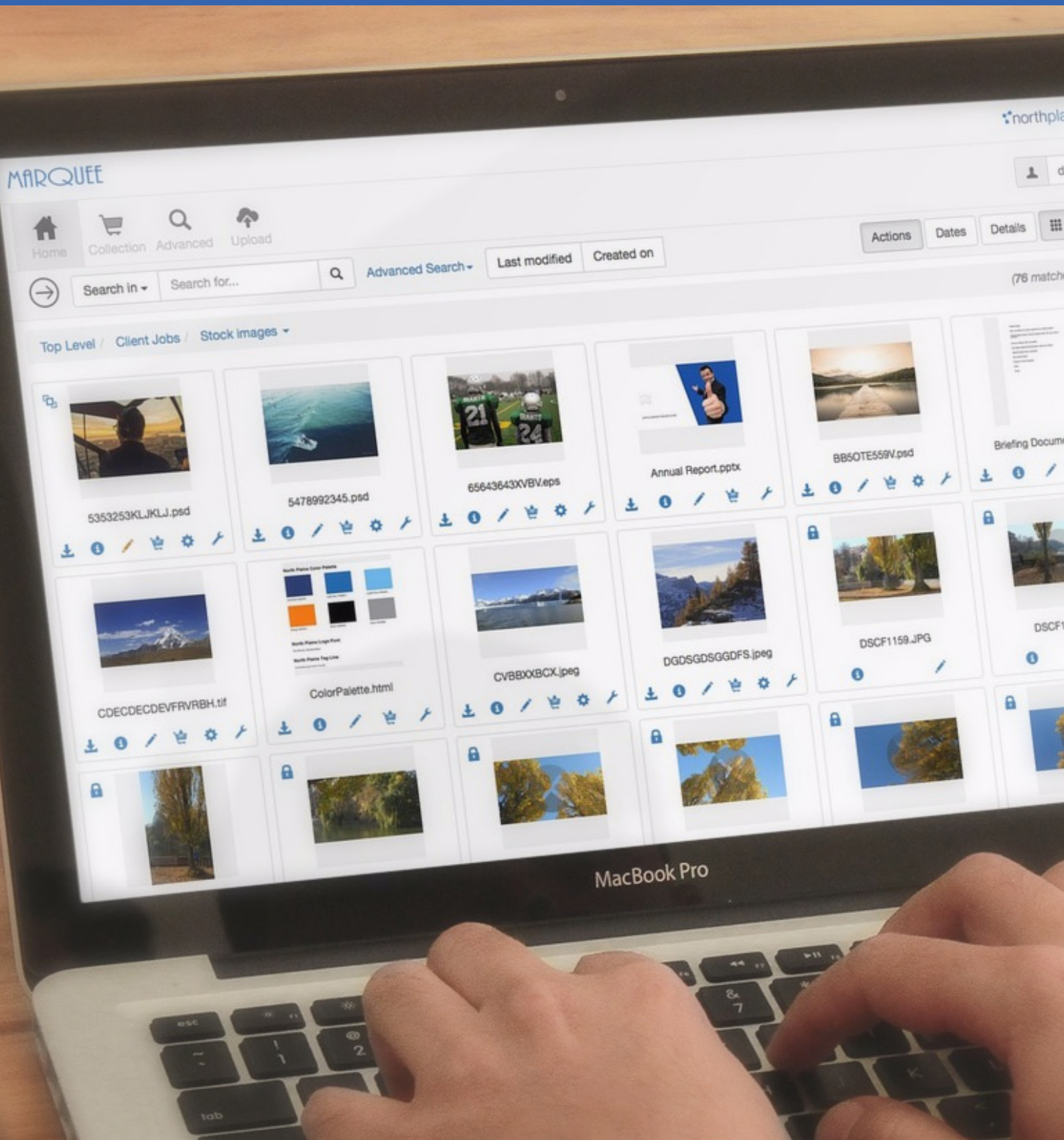
northplains

xinet

Updates to the Xinet Basket API



Dear Authorized Xinet Reseller,

On behalf of Northplains, I would like to inform you about an extension to our product that allows you as a Xinet Basket Plug-In developer to run any basket (now labeled 'collection') via our recently introduced business rule filter feature.

With Xinet version 18.1, we have introduced a new feature to hide assets from being shown to a particular user group based on business rule filters. For example, a particular group of users might not see any file that has a specific metadata field set to "Rejected", if the asset lives under a certain path.

Any asset that matches the filter condition will not be visible in Xinet Portal when browsing assets or viewing any search results. However if a logged-in user adds a folder or a selection of folders to a collection, it is important to run any asset, including folders and their subfolders and assets contained in subfolders through the business rule filters, to avoid downloads or any other unexpected access to non-permitted assets.

This document contains the full chapter of our Basket API chapter from our Xinet Admin manual. If you feel comfortable with how Xinet basket plugIns work in general, you can also jump straight to the new section at the end of this document, **"Applying Filters to basketfile (NEW)"**.

We have updated all of our plugins to use this new method as well.

Regards,

Timo Faber
Xinet Product Manager
Northplains
Office: +49 (0) 89 44 47 810

Xinet Basket API

Introduction

The Xinet basket API (now labeled 'collection') has been designed to allow Xinet integrators to provide unique services while still preserving the easy-to-use interface of Xinet Portal. The basket API allows additional functionality to be added to Xinet Portal in the form of simple "plugins." The API is designed to keep the plugins easy to write while still providing enough flexibility to allow most user demands to be met.

Overview of basic programming requirements

In order to generate any Xinet Basket plugin, you need three things:

- An icon which represents the operation to users, stored normally as a GIF image no greater than 30 pixels
- A name for the operation
- A CGI program that accepts a basket as an argument.

After you review the examples in this document, you should be able to follow the steps in the next section to develop your own interfaces.

Now, here are the steps to follow:

1. First, name the operation, develop your CGI, and design the icon you'll need.

It is easy to generate the icon and come up with a name, but generating the actual CGI is a little bit more work. The CGI can be any program executable on the server, including compiled C programs, shell-scripts, PERL scripts, etc. If the program wishes to invoke a user-interface, it must act as a CGI-form and arrange to have itself called again. The section below, Examples of custom shopping basket operations, shows examples of actual CGIs.

2. Place a copy of your program and icon in the WebNative plugins directory, set appropriate permissions, and put a copy of the new program in each user's configuration options.

```
# cp myplugin /usr/etc/webnative/plugins
# chmod 4555 /usr/etc/webnative/plugins/myplugin
# chown username /usr/etc/webnative/plugins/myplugin
# cp myicon /var/adm/webnative/images/myicon.gif
# chmod 444 /var/adm/webnative/images/myicon.gif
```

</> Connect to the server on a command line as root user and run the following commands:

3. Then cd to the `/var/adm/webnative` directory.
Edit the `basket.config` file there, so that it will contain information about the components of the API you've written and how you'd like users to interact with the API. The format of each entry in the `basket.config` file (tab separated) is shown in the following table in the next page

CGI PROGRAM	Represents the name of the CGI. The program should always be located in `/usr/etc/webnative/plugins` and it should be executable by the world.
NAME	Represents the name of the CGI. The program should always be located in `/usr/etc/webnative/plugins` and it should be executable by the world.
ICON	Represents the name of the icon file, which must be stored in `/var/adm/webnative/images/`
USERNAME	Not used at this time.
Flags	A decimal number representing the options set for this plugin. The flags themselves are the bits set in the number's binary representation: 1 in this field means that new users will see this plugin by default. 2 says the plugin handles archive files. 4 means the plugin needs the database enabled.

1. Test the plugin to make sure it functions correctly. The user nativeadmin can delete plugins. By default, plugins will run as the user "nobody" and have that user's permissions. This is standard Apache behavior. If the plugin needs to be run as a different user with more permissions (say, as root), you have the option to make the plugin a `setuid(2)` program, using the following command: `# chmod 4755 plugin_name`. This allows the plugin to run as the current owner of the program. If the owner is root, then root permissions will apply when the program is run.

However, this approach will not work if the plugin is a script, because UNIX, as a security measure, does not allow scripts to be run with `setuid(2)`. If you have written a script and "nobody" permissions are insufficient, you have three options:

- Change permissions on the server so all affected areas are writeable by world.
- Rewrite your script as a program and make it run as `setuid(2)`.
- Write a one-line C program that simply calls the script. Then make that program be `setuid(2)`. The permissions will be inherited by the script.

Examples of custom shopping basket operations

This section describes some sample plugin operations in order to demonstrate the power and simplicity of the Xinet plugin interface. After describing what they do, we will show how they were implemented as examples of what can be done easily and quickly to customize Xinet.

Ordering images

For a variety of reasons, you may not want customers to download images over the Web. For example, the available bandwidth may not make it reasonable, or there may be legal considerations, or the shop may want to charge for the service of providing the images. A reasonable scenario might be for these shops to allow customers to view their images via Xinet but not be able to download them. The customers could select which images they wanted, and using the basket interface, request that the service provider, for example, place the images on removable media and send them to the customer. Using the Request shipment interface, the customer would provide the mailing address, and select the type of media. In our example, the plugin

emails instructions to an operator on what to generate; but many other mechanisms are possible.

In this case, the program provides a way for the user to select the type of media that will be shipped and to supply an address. (A complex user interface isn't necessary.) We implemented the program as a C-Shell script; it could, however, be easily implemented in any language.

</> The source code, shown below, is also available online in `/usr/etc/webnative/plugins/ship`.

```
#!/bin/csh -f
#
# This shell script requests some data from the user, then when all
# the information is collected it sends mail to another user with the
# information collected and the names of the file in the users' collection.
# This sample is mainly here to show how to get input from the user and
# combine it with the collection information
# This is the header. It tells the browser that we are producing html.
# Note the 2 carriage returns, which are essential to end the header
echo "Content-type: text/html"
echo ""
echo "<HTML>"

# Set the title bar of the window.
echo "<TITLE>Request shipment</TITLE>"

# Initialize some variables
set NAME=""
set COMPANY=""
set ADDR=""
set MAILUSER="root"
set MEDIA=""
set nonomatch
if (-f "/usr/adm/webnative/$REMOTE_USER/mailto") then
 set MAILUSER=`cat "/usr/adm/webnative/$REMOTE_USER/mailto"`
else if (-f /usr/adm/webnative/mailto) then
 set MAILUSER=`cat /usr/adm/webnative/mailto`
endif

# Extract our input variables (come through stdin) and place them
# in their equivalent csh variables. You are not expected to understand this
# Made extra complicated by Solaris sed bugs.

cat > /tmp/tmp.$$
echo " " >> /tmp/tmp.$$
foreach arg ( `cat /tmp/tmp.$$ | sed -e "s/&/ /g" -e "s:%2F:/:g"` )
       set var=`echo $arg | sed 's/=.*//'`
       set value=`echo $arg | sed -e 's/^.*=//' -e 's/+/ /g' -e
       's/%[0-9ABCD][0-9ABCD]/ /g' -e 's/\\r//'`
       eval "set $var=\"$value\""
end
rm -f /tmp/tmp.$$

# Get rid of odd argument passing artifacts
set basketfile=`echo $1 | sed -e 's/\\//g'`
set MAILER=""
foreach file (/usr/ucb/Mail /usr/bsd/Mail /usr/sbin/Mail /usr/bin/mail)
 if (-x $file) then
 set MAILER=$file
 break
 endif
end

if ($MAILER == "") then
 echo "Sorry, I can't send mail"
endif
```

```
# Here we check and see if the information is filled out. If so, it
# means we are being called from ourselves and the submit button
if ("$NAME" != "" && "$COMPANY" != "" && "$ADDR" != "") then

# generate a mail message in a temporary file and place the user
# information in it. The $$ makes the file unique.
 touch /tmp/mailreq.$$
 echo "$NAME" >> /tmp/mailreq.$$
 echo "$COMPANY" >> /tmp/mailreq.$$
 echo "$ADDR" >> /tmp/mailreq.$$
 echo " " >> /tmp/mailreq.$$
 echo "Requested a $MEDIA be shipped containing: " >> /tmp/mailreq.$$
 echo " " >> /tmp/mailreq.$$

 # The basketfile variable contains the name of the collection file
 # In this case we just list out the contents of the file to
 # our temp file.

 # OLD--OLD--OLD--OLD

 cat /tmp/mailreq.$$ "$basketfile" | $MAILER -s "Web Order" $MAILUSER

 # Clean up the temporary file
 rm /tmp/mailreq.$$

 # Give the user feedback. You can't have buttons
 # unless you have a FORM
 echo "<FORM>Thank you for your order $NAME<BR>"
 echo "Your $MEDIA will be sent as soon as possible."
 echo "<P>"
 echo "<INPUT TYPE=BUTTON VALUE='Close Window'
 onClick='window.close()'>"
 echo "</FORM></HTML>"
 exit 0
endif

# Here the user filled out some of the information but not all
# A more robust program would complain about the specific
# entries that were not filled out.
if ("$NAME" != "" || "$COMPANY" != "" || "$ADDR" != "") then
 echo "Please fill out all of the requested fields"
endif

# Request the information if we do not have it yet
echo "<FORM METHOD=POST>"
echo "Your Name: <INPUT TYPE=TEXT NAME=NAME VALUE='$NAME'><BR>"
echo "Company: <INPUT TYPE=TEXT NAME=COMPANY VALUE='$COMPANY'><BR>"
echo "Address: <INPUT TYPE=TEXT NAME=ADDR VALUE='$ADDR' SIZE=40><BR>"

# Use a fancy pull-down for the media selection
echo "Media: <SELECT NAME=MEDIA>"
echo "<OPTION value=CDROM selected>CDROM"
echo "<OPTION value='ZIP Disk'>ZIP Disk"
echo "<OPTION value='Syquest Cartridge'>Syquest Cartridge"
echo "</SELECT>"
echo "<BR>"
# We have a button to submit the form (calls the same program)
# and one to cancel the whole operation.
echo "<INPUT Type=submit Value='Request Shipment'>"
echo "<INPUT TYPE=BUTTON VALUE=Cancel onClick='window.close()'>"
echo "</FORM>"
```

Basket files

Xinet maintains a user's collection state as a text file on the server. The file exists in the user's Xinet home directory (/var/adm/webnative/user_name) with the name of the file being controlled entirely by the BASKETNAME cookie.

Typically, Xinet maintains the BASKETNAME cookie, which is composed of the user's IP address, followed by eight random hexadecimal digits, followed by the extension basket. For example, 192.168.0.23.ca84e231.basket.

This format ensures that users sharing a login and IP address, like those behind a NAT router, have unique baskets.

However, a style author can take control of the BASKETNAME cookie and use any naming scheme he or she prefers.

The basket file contains a new-line-delimited list of file descriptions. Those descriptions follow two formats:

- The first, for regular files that are not archived, is simply the path to the file on the server.
- The second, for files that have been archived, is as follows:

  voltime:volname:archiveno:path

  where voltime represents the time the file was archived, volname indicates the name of the volume from which the file was archived, archiveno indicates how many times the file has been archived, previous to the current archive, and path is where the file lived in the file system at the time the archive was created.

Applying Filters to basketfile (*NEW*)

With Xinet version 18.1, we have introduced a new feature to hide assets from being shown to a particular user group based on business rule filters. For example, when a particular group of users might not see any file that has a specific metadata field set to "Rejected", if the asset lives under a certain path.

Any asset that matches the filter condition will not be visible in Xinet Portal when browsing assets or viewing any search results. However if a logged-in user adds a folder or a selection of folders to a collection, it is important to run any asset, including folders and its subfolders and assets contained in subfolders through the business rule filters, to avoid downloads or any other access to filtered assets.

This is why we have introduced a new way to run all collected files through any possible configured business rule filters.

When processing a basket, the plugin should use the portalDI2 program to "expand" and filter the contents of a basket. The basket file may contain paths to files and/or folders, and folders may contain descendant files, some of which may need to be excluded because of business rule filters.

`portalDI2` performs both the expansion and filtering of the basket contents (entries that refer to archives are simply passed through). Call `/usr/etc/webnative/portalDI2` with the following changes in the environment:

```
REQUEST_METHOD="GET"

QUERY_STRING="resource=basket&gatherbasket&output=stdout&basketfile=/path/to/basket/file"
```

Then read the basket contents from stdout. Folders will still be present in the output, but their descendants will also be included. It is possible for the entire basket contents to be filtered out due to changes in business rules after the basket contents were set, so be prepared for that possibility.

The source code from the example above would change accordingly to the following:

```
#!/bin/csh -f
#
# A very simple example of a webnative plug-in
# This shell script requests some data from the user, then when all
# the information is collected it sends mail to another user with the
# information collected and the names of the file in the users' collection.
# This sample is mainly here to show how to get input from the user and
# combine it with the collection information
#
# Two optional configuration files may accompany this plugin:
#      ship.mailfrom:      The "From:" address for the shipping request email.
#      ship.mailto: The "To:" address (overrides "root", or user/system "mailto")

# This is the header.  It tells the browser that we are producing html.
# Note the 2 carriage returns, which are essential to end the header
echo "Content-type: text/html"
echo ""
echo "<HTML>"

# Set the title bar of the window.
echo "<TITLE>Request shipment</TITLE>"


# Initialize some variables
set NAME=""
set COMPANY=""
set ADDR=""
set MAILUSER="root"
set MEDIA=""
set nonomatch

#prevent mail on osx from complaining to apache's error log
if (`uname` == Darwin && ! $?home) set home=/tmp

if (-f /usr/etc/webnative/plugins/ship.mailto) then
        set MAILUSER="`cat /usr/etc/webnative/plugins/ship.mailto`"
else if (-f "/usr/adm/webnative/$REMOTE_USER/mailto") then
        set MAILUSER=`cat "/usr/adm/webnative/$REMOTE_USER/mailto"`
else if  (-f /usr/adm/webnative/mailto)  then
        set MAILUSER=`cat /usr/adm/webnative/mailto`
endif

# Extract our input variables (come through stdin) and place them
# in their equivalent csh variables.  You are not expected to understand this
# Made extra complicated by Solaris sed bugs.
cat > /tmp/tmp.$$
echo " " >> /tmp/tmp.$$
foreach arg ( `cat /tmp/tmp.$$ | sed -e "s/&/ /g" -e "s:%2F:/:g"` )
       set var=`echo $arg | sed 's/=.*//'`
       set value=`echo $arg | sed -e 's/^.*=//' -e 's/+/ /g' -e 's/%[0-9ABCD][0-9ABCD]/ /g'
-e 's/\\r//'`
       eval "set $var="\"$value\""
end
rm -f /tmp/tmp.$$
```

```
# Get rid of odd argument passing artifacts
set basketfile=`echo $1 | sed -e 's/\\//g'`

if (-x /usr/sbin/sendmail) then
       set MAILER=/usr/sbin/sendmail
else if (-x /usr/lib/sendmail) then
       set MAILER=/usr/lib/sendmail
else
       echo "Sorry, I can't send mail"
endif

# Here we check and see if the information is filled out.  If so, it
# means we are being called from ourselves and the submit button

if ($?MAILER && "$NAME" != "" && "$COMPANY" != "" && "$ADDR" != "") then

# generate a mail message in a temporary file and place the user
       # information in it.  The $$ makes the file unique.
       set tmpfile=/tmp/shipreqtmp$$
       if (-f /usr/etc/webnative/plugins/ship.mailfrom) then
              set FROMADDR="`cat /usr/etc/webnative/plugins/ship.mailfrom`"
       else
              set FROMADDR='"Shipping Request" <noreply@noreply.net>'
       endif
       rm -f $tmpfile
       echo "From: $FROMADDR" > $tmpfile
       echo "Subject: Web Order" >> $tmpfile
       echo "To: $MAILUSER" >> $tmpfile
       echo "" >> $tmpfile
       echo "$NAME" >> $tmpfile
       echo "$COMPANY" >> $tmpfile
       echo "$ADDR" >> $tmpfile
       echo " " >> $tmpfile
       echo "Requested a $MEDIA be shipped containing: " >> $tmpfile
       echo " " >> $tmpfile

# The basketfile variable contains the name of the collection file  In this case
# we just list out the contents of the file to our temp file.

# NEW--NEW--NEW--NEW

if (-x /usr/etc/webnative/portalDI2) then
              (setenv REQUEST_METHOD GET; setenv QUERY_STRING "resource=basket&gatherbas-
ket&output=stdout&basketfile=$basketfile"; /usr/etc/webnative/portalDI2) >> $tmpfile
       else
              cat $basketfile >> $tmpfile
       endif
       $MAILER "$MAILUSER" < $tmpfile

       # Clean up the temporary file
       rm $tmpfile

       # Give the user feedback.  You can't have buttons unless you have a FORM
       echo "<FORM>Thank you for your order $NAME<BR>"
       echo "Your $MEDIA will be sent as soon as possible."
       echo "<P>"
       echo "<INPUT TYPE=BUTTON VALUE='Close Window' onClick='window.close()'>"
       echo "</FORM></HTML>"
       exit 0
endif
```

```
# Here the user filled out some of the information but not all
# A more robust program would complain about the specific
# entries that were not filled out.
if ("$NAME" != "" || "$COMPANY" != "" || "$ADDR" != "") then
       echo "Please fill out all of the requested fields"
endif

# Request the information if we do not have it yet
echo "<FORM METHOD=POST>"
echo "Your Name: <INPUT TYPE=TEXT NAME=NAME VALUE='$NAME'><BR>"
echo "Company: <INPUT TYPE=TEXT NAME=COMPANY VALUE='$COMPANY'><BR>"
echo "Address: <INPUT TYPE=TEXT NAME=ADDR VALUE='$ADDR' SIZE=40><BR>"
# Use a fancy pull-down for the media selection
echo "Media: <SELECT NAME=MEDIA>"
echo "<OPTION value=CDROM selected>CDROM"
echo "<OPTION value='ZIP Disk'>ZIP Disk"
echo "<OPTION value='Syquest Cartridge'>Syquest Cartridge"
echo "</SELECT>"
echo "<BR>"

# We have a button to submit the form (calls the same program)
# and one to cancel the whole operation.
echo "<INPUT Type=submit Value='Request Shipment'>"
echo "<INPUT TYPE=BUTTON VALUE=Cancel onClick='window.close()'>"
echo "</FORM>"
```

About Northplains

Northplains is recognised as the world leader in helping companies effectively leverage their visual media content through images, photos, videos, 3D designs, and others to maximise market success.

Providing a complete array of products, services and solutions, we are uniquely positioned to help corporate marketers, advertisers, content creators and publishers to be more agile in the creative development process and management and distribution of marketing and brand communications.

Customers can maintain control of their visual assets and support processes globally, while improving collaboration and efficiency.

Since 1994, we have helped our customers stay ahead of their rapidly changing marketing needs by expanding and evolving our suite of offerings to include Digital Asset Management (DAM), Marketing Resource Management (MRM), Creative Workflow and more.

With offices in Canada, USA and Europe we count many of the world's leading companies and brands among our 1,400+ clients and 1,000,000 users.