# How to build a WMS from Free Parts

This example examines a WMS built using Python. This WMS works by keeping a 3600x1800 JPEG image which contains a full map of the world from -180,-90 to 180,90 and sending chunks of it out in response to map requests by clients.

The list of what's needed to build it is:

- A Linux box (RedHat 7)[1]
- Python 2.1.1c1 - get it at Python.org[2]
- Apache 1.3.20 - get it at Apache.org[3]
- mod_python 2.7.5 - get it at modpython.org[4]
- Netpbm - get it at SourceForge[5]
- A decent JPEG image of the world (note: that's a 1.7 MByte file). One can be obtained from CubeWerx[6] by running their CubeWerx Web Demo[7]. Depending on your hardware/software setup, you may have to pull in things like Tcl/Tk, JPEG libraries, etc. in order to build some of this stuff.

Here's the result:

basic-wms2.py --> see Appendix B. Also see the license page[8]. This could probably also be done in PERL or Tcl or the programming environment of your choice.

The general flow from a client's perspective is to first ask the WMS to return a list of what map layers it can draw, then the client can ask for maps by requesting specific layers or combinations of layers over specific geographic regions. In practice, the hard parts of this are done by people who set up web pages that provide user interface controls. Two examples of this are the CubeWerx demo and the NASA Digital Earth viewer[9].

These are known as "viewer clients." A viewer client can be as simple as a web browser that you paste a fully formed WMS request into or as complex as a commercial GIS system that can make WMS requests based on the context of what a user is doing.

In this example, the viewer client is a Web browser. All examples will be shown as URLs that you can click on.

```
http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-
wms2.py?
WMTVER=1.0.0&REQUEST=map&LAYERS=RELIEF&STYLES=default&SRS=EPSG:43
26&BBOX=-2.197265625,39.55078125,20.302734375,50.80078125
&WIDTH=256&HEIGHT=128&FORMAT=PNG
```

Note: All examples of URLs or other code that are too wide are broken up into separate lines. In the case of a URL such as this one, if you were to copy it and paste it into a browser, you'd need

---

*[1] Linux Box from Reddhat☹: http://www.redhat.com/*
*[2] Python.org: http://www.python.org/*
*[3] Apache.org: http://www.apache.org/*
*[4] Modpython.org http://modpython.org/*
*[5] SourceForge: http://sourceforge.net/projects/netpbm/*
*[6] Cubewerx: http://www.cubewerx.com/*
*[7] Cubewerx WMS demo: http://www.cubewerx.com/demo/cubeview.cgi*
*[8] International-Interfaces license:See the full text of the License for WMS Cookbook and basic-wms2.py in Appendix 2.*
*[9] NASA Digital Earth viewer: http://viewer.digitalearth.gov/*

to copy each of the lines and tack them together with no white space. The examples will be linked to the same URL so you can just click on them instead of having to cut and paste.

WMS has a base URI prefix (or URL prefix if you prefer)

```
http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-
wms2.py?
```

In order to find out what layers it supplies and what projections it supports, a client makes a "Capabilities Request." Here's the prefix with a Capabilities request:

```
http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-
wms2.py?request=GetCapabilities&wmtver=1.1.1
```

The response is formatted according to the WMS Appendix A. Comments in that DTD are considered normative and must be followed by WMS providers. The response has to be valid XML, meaning that it should pass a validation test[10]. The response has a MIME type of text/xml. Let's take a quick look at the parts of the XML document. First there's some XML info:

```
<?xml version='1.0' encoding="UTF-8" standalone="no"?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
 "http://www.digitalearth.gov/wmt/xml/capabilities_1_0_0.dtd">
<WMT_MS_Capabilities version="1.0.0" [
  <!ELEMENT VendorSpecificCapabilities EMPTY>
```

The DOCTYPE entry states that this is a WMT_MS_Capabilities document and that you can find the DTD for this file at the DigitalEarth website[11]. The part about VendorSpecificCapabilities means that this WMS has none.

The next section describes the overall service. (There's a lot of work going on within OGC about service descriptions and service models[12]. The WMS 1.0.0 spec predates most of this work. Thus, if you look at the 1.0.7 spec or at other materials coming from OGC, you will see different points on an evolutionary path. The ultimate goal is to develop a service model that can be used to describe many spatial services.)

```
<!-- Service Metadata -->
<Service>
 <!-- The WMT-defined name for this type of service -->
 <Name>GetMap</Name>
```

A WMS must be named `GetMap`. It's stated so in the DTD.

```
<!-- Human-readable title for pick lists -->
 <Title>Basic Map Server</Title>
 <!-- Narrative description providing additional information -->
 <Abstract>Basic WMS Map Server built as an example for a WMS
cookbook
      Contact: adoyle@intl-interfaces.com.</Abstract>
 <Keywords>Demo WMS Cookbook</Keywords>
```

As a WMS implementer/provider, you pick the Title, Abstract, and Keywords. The Title is meant to be used in user interfaces software (i.e. in Viewer Clients) as part of a list of map servers that the Viewer Client can access. Keep it short. Abstract is meant to provide a longer description of the service. Keep it informative. The Keywords are meant to be useful if someone were searching for your service. These are hard to select.

[10] *XML validation test tool: http://www.stg.brown.edu/service/xmlvalid/*
[11] *Capabilities 1.0.0 DTD file: http://www.digitalearth.gov/wmt/xml/capabilities 1 0 0.dtd*
[12] *About OGC work on service descriptions and service models: http://www.intl-interfaces.net/servicemodel/*

```
<!-- Top-level address of service or service provider.
See also onlineResource attributes of <dcpType> children.-->
<OnlineResource>
   http://www.intl-interfaces.net/cookbook/WMS/
 </OnlineResource>
 <!-- Fees or access constraints imposed. -->
 <Fees>none</Fees>
 <AccessConstraints>none</AccessConstraints>
</Service>
```

The OnlinResource should contain a URI that leads to a description of the service. In this case, it points to this WMS Cookbook. The Fees and AccessConstraints elements are really not well defined except that the spec states that the string none indicates no constraint exists.

Note that all the elements of the Service section can be gathered into a searchable catalog of WMS implementations. In such a catalog, it would be possible to look for WMS implementations that have no associated Fees, or to find those whose Keywords do not include the term "Demo" and so on. If you searched a catalog and found an entry that interests you, you could use the OnlineResource to find out more about that implementation. In fact, the rest of the capabilities document, the Capability section is used in catalogs as well.

The Capability section for basic-wms has three subsections: Request, Exception, and Layer.

```
<Request>
  <Map>
   <Format>
    <PNG />
    <JPEG />
    <PPM />
    <TIFF />
   </Format>
   <DCPType>
    <HTTP>
     <Get onlineResource=
     "http://www.intl-interfaces.net/cookbook/WMS/basic-
wms2/basic-ms2.py?" />
    </HTTP>
   </DCPType>
  </Map>
  <Capabilities>
   <Format>
    <WMS_XML />
   </Format>
   <DCPType>
    <HTTP>
     <Get onlineResource=
        "http://www.intl-interfaces.net/cookbook/WMS/basic-
wms2/basic-wms2.py?" />
    </HTTP>
   </DCPType>
  </Capabilities>
 </Request>
```

As you can see, the Request section is split into two sections, Map and Capabilities. These describe the two operations that this WMS supports. For the Map request, it can handle PNG, JPEG,

PPM, and TIFF return formats, and is listening for Map requests at the URI specified by onlineResource. These requests must be made using the HTTP Get request (as opposed to HTTP Put or SOAP or anything else). For the Capabilities request, it can return WMS_XML (the tag "XML" was already reserved, hence the WMS_ prefix) and again, listens at the onlineResource URI.

It's worth noting that the WMS spec allows the onlineResource values for each request type to be different. Depending on how you set up your WMS implementation, it may be more convenient for the two to be the same or it may be easier for them to be different. A WMS client should always start with a Capabilities request to find the Map request URI. Never assume they are the same.

```
<Exception>
 <Format>
  <INIMAGE />
  <BLANK />
 </Format>
</Exception>
```

The Exception tag tells about the exception formats a client can ask for when making requests. The basic-wms advertises that it can return INIMAGE or BLANK style exceptions.

At last we get to the Layer section. This is how the map server tells the clients what kinds of maps they can request.

```
 <Layer>
  <Title>Demo Map Server</Title>
  <SRS>EPSG:4326</SRS>
  <LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90"
/>
  <Layer queryable="0">
   <Name>RELIEF</Name>
   <Title>Relief (ETOPO/GTOPO)</Title>
   <Abstract>Colored relief map with political boundaries and
coastlines</Abstract>
  </Layer>
  </Layer>
```

Our map server says that it has a top level Layer with a Title, SRS, and LatLonBoundingBox, and a single sub-Layer. Note that the top level Layer has no Name. That means that it's just being used to gather other layers into a logical group. Instead of that grouping, we could have used the form below.

```
 <Layer queryable="0">
   <SRS>EPSG:4326</SRS>
   <Name>RELIEF</Name>
   <Title>Relief (ETOPO/GTOPO)</Title>
   <Abstract>Colored relief map with political boundaries and
coastlines</Abstract>
  <LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90"
/>
 </Layer>
```

However, the WMS 1.0.0 DTD (Appendix A) shows that there can only be zero or one Layer elements inside a Capability. By using the top level grouping Layer, we ensure that later additions of new Layers will be easier to do. So here's the Layer info again:

```
    <Layer>
     <Title>Demo Map Server</Title>
     <SRS>EPSG:4326</SRS>
     <LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90"
   />
     <Layer queryable="0">
      <Name>RELIEF</Name>
      <Title>Relief (ETOPO/GTOPO)</Title>
      <Abstract>Colored relief map with political boundaries and
   coastlines</Abstract>
     </Layer>
     </Layer>
```

The top level Layer has an SRS and a LatLonBoundingBox. These are inherited by the sub Layer elements. The sub Layer elements can add SRS values to this list by including its own SRS element. It can replace the LatLonBoundingBox by including one of its own. These inheritance/replacement rules are given in a table in the DTD in Appendix A. Our single sub Layer must have a Name and a Title. We also include an Abstract. The queryable="0" attribute says that our WMS will not respond to `GetFeatureInfo` requests.

In order to actually test this WMS implementation, you can make use of a service offered by CubeWerx. If you go to their demo page, down at the bottom is a text entry box labeled "'URL of server: " If you enter the prefix of your WMS implementation in there and hit one of the "Go" buttons on the page, it will query your WMS for its capabilities, fill in the user-interface elements on the page, and let you exercise your WMS.

# Appendix 1. International Interfaces: Basic-wms2.py file

```
# basic-wms2.py : A very small WMS implementation
# V2 - removed PIL, trying PBM instead
#
#=================================================================
========
# LICENSE -- This is the "MIT License"
#
# Copyright (c) 2001 Allan Doyle
#
# Permission is hereby granted, free of charge, to any person
obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction,
including
# without limitation the rights to use, copy, modify, merge,
publish,
# distribute, sublicense, and/or sell copies of the Software, and
to
# permit persons to whom the Software is furnished to do so,
subject to
# the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE
# LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION
# WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#=================================================================
========
#
# Allan Doyle - adoyle@intl-interfaces.com
#
# This WMS works by keeping a 3600x1800 JPEG image which contains
a full
```

```
# map of the world from -180,-90 to 180,90 and sending chunks of
it out
# in response to map requests by clients.
#
# Things are deliberately hardcoded in this WMS to keep it
simple.
#
# The image was generated using the CubeWerx WMS demo which you
can find
# at http://www.cubewerx.com
#
# Debugging help was provided by Jeff de La Beaujardiere at NASA
#
#-----------------------------------------------------------------
--------
### The big chunks of functionality come from mod_python and PIL
#
# mod_python is available from www.modpython.org
#
#    It provides the connection from Apache CGI to python code
#
from mod_python import apache
#
# Python imports
#
import sys
import os
import string
# Open the map image once and load it (this may be causing a
memory leak)
map = "/home/apache/www.intl-
interfaces.net/htdocs/images/cubeserv-best.pnm"
# The WMS version that this WMS implements
version = '1.0.0'
# This is the capabilities XML
# Thanks to Jeff de La Beaujardiere of the NASA Digital Earth
program
# for a good one that I used as a template
#
capabilities = """<?xml version='1.0' encoding="UTF-8"
standalone="no" ?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
"http://www.digitalearth.gov/wmt/xml/capabilities_1_0_0.dtd"
 [
<!ELEMENT VendorSpecificCapabilities EMPTY>
 ]>
<WMT_MS_Capabilities version="1.0.0" updateSequence="0">
<!-- Service Metadata -->
<Service>
```

```
<!-- The WMT-defined name for this type of service -->
<Name>GetMap</Name>
<!-- Human-readable title for pick lists -->
<Title>Basic Map Server</Title>
<!-- Narrative description providing additional information -->
<Abstract>Basic WMS Map Server built as an example for a WMS
cookbook Contact: adoyle@intl-interfaces.com.</Abstract>
<Keywords>Demo WMS Cookbook</Keywords>
<!-- Top-level address of service or service provider. See also
onlineResource attributes of <dcpType> children. -->
<OnlineResource>http://www.intl-
interfaces.net/cookbook/WMS/</OnlineResource>
<!-- Fees or access constraints imposed. -->
<Fees>none</Fees>
<AccessConstraints>none</AccessConstraints>
</Service>
<Capability>
<Request>
<Map>
<Format>
<PNG />
<JPEG />
<PPM />
<TIFF />
</Format>
<DCPType>
<HTTP>
<Get onlineResource="http://www.intl-
interfaces.net/cookbook/WMS/basic-wms2/basic-wms2.py?" />
</HTTP>
</DCPType>
</Map>
<Capabilities>
<Format>
<WMS_XML />
</Format>
<DCPType>
<HTTP>
<Get onlineResource="http://www.intl-
interfaces.net/cookbook/WMS/basic-wms2/basic-wms2.py?" />
</HTTP>
</DCPType>
</Capabilities>
</Request>
<Exception>
<Format>
<INIMAGE />
<BLANK />
</Format>
```

```
</Exception>
<Layer>
<Title>Demo Map Server</Title>
<SRS>EPSG:4326</SRS>
<LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90" />
<Layer queryable="0">
<Name>RELIEF</Name>
<Title>Relief (ETOPO/GTOPO)</Title>
<Abstract>Colored relief map with political boundaries and
coastlines</Abstract>
</Layer>
</Layer>
</Capability>
</WMT_MS_Capabilities>
"""
##
# Name/value utilities
#
## split_args(args)
#
# Takes a CGI string. Turns it into a list of name/value pairs.
Names with
# no value are given a None value (a python special value). All
names
# are converted to upper case since WMS arguments are case
insensitive
#
def split_args(args):
"split_args : take a CGI string and return a list with name/value
pairs"
  canon_args = {}           # Start an empty list
  if args == None:          # Return the empty list if no arg
return canon_args
  arglist = args.split('&')      # Split into list of name=value
string
  for arg in arglist:         # Now split each name=value and
    tmp = arg.split('=')      # turn them into sub-lists
    if len(tmp) == 1:        # with name in the first part
      canon_args[tmp[0]] = None  # and value in the second part
else:
canon_args[tmp[0].upper()] = tmp[1]
return canon_args
## send_html_error(req, s, status)
#
# Returns a text/html response to the client with an error
message
# packaged inside. The status is raised as an exception which
neatly
# bumps us all the way back to the apache server.
```

```
#
def send_html_error(req, s, status):
  req.content_type = 'text/html'   # Set the return Content-Type
  req.send_http_header()       # Send the HTTP return header
  req.write('<p>' + s + '</p>')    # Wrap the message in <p></p>
raise apache.SERVER_RETURN, status # return to apache
## handler(req)
#
# The name of this function is dictated by mod_python. This is
the entry
# point to the WMS. It is called by apache with the map request.
# A file in the local directory called .htaccess defines some of
this.
# (Or it can be configured into the main apache httpd.conf file)
#
def handler(req):
"handler : called when apache gets a map request URI"
# mod_python stuff
  #
request = req.args         # Provides a string with the CGI
# arguments in it
req.content_type = 'text/plain'   # Useful if we have to send
messages
# back to the client. Later we'll
# override it with image/jpeg
# WMS argument processing
# starts here...

canon_args = split_args(req.args)  # This turns the args into a
python
# list
# If there are no arguments in the request, exit. The WMS spec
does not
# specify what to return here since technically, unless there's
at least
# a 'REQUEST' parameter, it's not a WMS request. For now, let's
# use HTTP_BAD_REQUEST and return a message
  #
if len(canon_args) == 0:
send_html_error(req, 'No parameters found',
apache.HTTP_BAD_REQUEST)
# Next look at the REQUEST argument.
# If it's not there, this is also not a WMS request...
request = canon_args.get('REQUEST', None);
if request == None:
send_html_error(req, 'No REQUEST parameter found',
apache.HTTP_BAD_REQUEST)
# Here are the 3 choices. In the Capabilities XML we say that the
# layer is not queryable, so we should not be getting a
feature_info
# request. If we do, we can say HTTP_BAD_REQUEST... this is
consistent
# with the WMS 1.0.0 spec 6.2.9.4 that says an error response
must be
# MIME typed.
if request == 'capabilities':
send_capabilities(req, canon_args)
elif request == 'map':
send_map(req, canon_args)
elif request == 'feature_info':
send_html_error(req, 'REQUEST=%s is not implemented:' %
canon_args['REQUEST'], apache.HTTP_BAD_REQUEST)
else:
send_html_error(req,'REQUEST=%s is not a valid WMS request' %
canon_args['REQUEST'], apache.HTTP_BAD_REQUEST)
## version_cmp(v1, v2)
#
# Compare version strings. Works like strcmp.
# Since versions are dotted strings with 1, 2, or 3 components,
we first
# check if the strings are actually equal. If not, then we make
sure we have
# three components to compare by adding trailing '0' elements.
Then we
# compare the high-order part, the next part, and the next part.
#
# For this WMS we only need to know if they are equal or not
equal. This
# WMS does not do version negotiation.
#
def version_cmp(v1, v2):
# If they are already equal, great
if v1 == v2: return 0
# turn them into lists
L1 = v1.split('.')
L2 = v2.split('.')
# build up things like '1.0' and '1' into '1.0.0'
if len(L1) == 1: L1.append('0')
if len(L1) == 2: L1.append('0')
if len(L2) == 1: L2.append('0')
if len(L2) == 2: L2.append('0')
# now if they are equal, great
if L1 == L2: return 0
if string.atoi(L1[0]) < string.atoi(L2[0]): return -1
if string.atoi(L1[0]) > string.atoi(L2[0]): return 1
if string.atoi(L1[1]) < string.atoi(L2[1]): return -1
if string.atoi(L1[1]) > string.atoi(L2[1]): return 1
if string.atoi(L1[2]) < string.atoi(L2[2]): return -1
if string.atoi(L1[2]) > string.atoi(L2[2]): return 1
## send_capabilities(req, args)
#
# Simply sets the Content-Type to 'text/xml' and returns the
Capabilities
# string that's included at the top of this file.
#
def send_capabilities(req, args):
# This is where version negotiation would go. We'll ignore it for
now
# since we only support one version. If the client tries to
version
# negotiate, we'll just send our 1.0.0 capabilities back each
time.
# Eventually the client will accept this or go away
req.content_type = 'text/xml'
req.send_http_header()
req.write(capabilities)
raise apache.SERVER_RETURN, apache.OK
## Projections
#
# Currently assume a base world image of 3600x1800 with the whole
world
# from -180,-90 to 180,90
#
# No inverse is needed since we don't handle feature_info
requests.
#
def LonToPix(lon):
return int ((lon * 10) + 1800 + .5)
def LatToPix(lat):
return int ((-lat * 10) + 900 + .5)
## send_map(req, args)
#
# Checks to see if all the args that it knows about are present
and correct
# If so, send a map.
# Note: 2001.05.07 adoyle - added .upper() to all references to
#    found['FORMAT'] to improve leniency for people who use
lowercase 'png'
#    'jpeg' etc by mistake.
#
def send_map(req, args):
formats = {'JPEG' : '| ppmtojpeg',
'PNG' : '| pnmtopng',
'TIFF' : '| pnmtotiff',
'PPM' : ' '}
# These are the required parameters (WMS 1.0.0 Table 6.3)
```

```
required = ['LAYERS', 'STYLES', 'SRS', 'BBOX', 'WIDTH', 'HEIGHT',
'FORMAT']
# These are the optional parameters (WMS 1.0.0 Table 6.3)
optional = ['TRANSPARENT', 'BGCOLOR', 'EXCEPTIONS']
# Loop through the list of required args. If any are missing,
return
# an error.
# The ones that we start with are the optional ones, set to the
default
found = {'BGCOLOR' : '0xFFFFFF',
'TRANSPARENT' : 'FALSE',
'EXCEPTIONS' : 'INIMAGE'}
for param in required:
found[param] = args.get(param, None);
if found[param] == None:
send_html_error(req, 'No ' + param + ' parameter found',
apache.HTTP_BAD_REQUEST)
for param in optional:
found[param] = args.get(param, found[param]);
# Find the 4 values in the BBOX
bbox = found['BBOX'].split(',')
# Turn the BBOX values into pixel values
for i in (0, 1, 2, 3):
bbox[i] = string.atof(bbox[i]
x0 = LonToPix(bbox[0])
y0 = LatToPix(bbox[1])
x1 = LonToPix(bbox[2])
y1 = LatToPix(bbox[3])
# get the width/height values
width = string.atoi(found['WIDTH'])
height = string.atoi(found['HEIGHT'])
error = 0
# Let's do a little checking
if bbox[0] < -180.0 or bbox[0] > 180.0 \
or bbox[1] < -90.0 or bbox[1] > 90.0 \
or bbox[2] < -180.0 or bbox[2] > 180.0 \
or bbox[3] < -90.0 or bbox[3] > 90.0:
error = 1
message = "BBOX out of range"
# If there's an error, then we have to decide whether to return
# an INIMAGE error (i.e. write an error message on an image) or
# whether to make a blank image. In both cases, inimage or blank,
we

# then need to decide whether we're supposed to do transparency
and
# whether the image format supports it (only PNG does). Then we
# have to make sure the result has transparency.
if error and found['EXCEPTIONS'] == 'INIMAGE':
```

```
# Build the image with the text
cmd = 'ppmmake \#%s %s %s' % (found['BGCOLOR'][2:], width,
height) \
       + ' | ' \
•  'ppmlabel -background \#888888 -colour \#000000' \
•  ' -x 5 -y 20 -text \"%s\"' \
% message + formats[found['FORMAT'].upper()]
# decide whether to make it transparent
if found['FORMAT'].upper() == 'PNG' and found['TRANSPARENT'] ==
'TRUE':
cmd = cmd + ' -force -transparent \#%s' % (found['BGCOLOR'][2:])
# If we're supposed to return a blank image, just make an image
# with BGCOLOR as the entire image.
elif error and found['EXCEPTIONS'] == 'BLANK':
# Build the image
cmd = 'ppmmake \#%s %s %s' % (found['BGCOLOR'][2:], width,
height) \
•  formats[found['FORMAT'].upper()]
# decide whether to make it transparent
if found['FORMAT'].upper() == 'PNG' and found['TRANSPARENT'] ==
'TRUE':
cmd = cmd + ' -force -transparent \#%s' % (found['BGCOLOR'][2:])
# If there was no error, then build the command that will return
# a new map that is a rectangle cut from the old map and the
scaled
# into the new dimensions.
else:
cmd = "pnmcut -left %s -bottom %s -right %s -top %s < %s" \
% (x0,y0,x1,y1, map) \
       + ' | ' \
•  "pnmscale -xysize %s %s" % (width, height) \
•  "| pnmsmooth" + formats[found['FORMAT'].upper()]
# If you added a debug=1 (or any debug) parameter to the request
# this bit will return some debugging info as text instead of the
# image. This is not advertised in the capabilities because this
# is not meant to be used by WMS clients.
if args.get('DEBUG', None):
req.send_http_header()
req.write('bbox %s ' % bbox)
req.write('WxH=%sx%s ' % (width, height))
req.write('x0,y0=%s,%s ' % (x0,y0))
req.write('x1,y1=%s,%s\n' % (x1,y1))
req.write(' params %s\n' % found)
req.write(cmd + '\n')
req.write(message)
raise apache.SERVER_RETURN, apache.OK
# This executes the command we built above and gathers the output
# of the command for reading as a file
pipe = os.popen(cmd, 'r')
```

```
# Set the return Content-Type to 'image/<FORMAT>'
req.content_type = "image/%s" % found['FORMAT'].lower()
  req.send_http_header()       # Send the header
  req.write(pipe.read())       # Send the image
raise apache.SERVER_RETURN, apache.OK # exit to apache
```