```python
"""Backend for rendering PDF documents using Ghostscript.

This is an internal API and subject to change at any time.
"""

import os
import sys
import subprocess
import io

# Used for conversion of Postscript to PDF
import tempfile

# Downscaling support requires PIL
try:
    import PIL
except (ImportError):
    PIL = None

from .shared import Backend, BackendError, bytes_to_str, check_output
from .util import find_executable

# ---------------------------------------------------------------------------

# Path to the Ghostscript executable
if sys.platform.startswith("win"):
    from glob import glob

    # Possible names for the Ghostscript executable and Program Files
    # Only the console version of Ghostscript (gswin??c.exe) will work
    # because we need to check its output on stdout.
    if sys.maxsize > 2**32 or os.getenv("ProgramW6432"):
        # Favor 64-bit Ghostscript where supported
        gs_names = "gswin64c.exe", "gswin32c.exe"
        pf_vars = "ProgramFiles", "ProgramW6432", "ProgramFiles(x86)"
    else:
        gs_names = "gswin32c.exe",
        pf_vars = "ProgramFiles",

    # Possible locations to look for Ghostscript...
    # (This is a list, not a set, to ensure we preserve our search order)
    gs_dirs = []

    # 1. Your application directory
    #    This lets your application distribute its own Ghostscript binary,
    #    which may be preferable to ensure you have a known good version.
    app_dir = os.path.dirname(os.path.realpath(sys.argv[0]))
    for dirpath, dirnames, filenames in os.walk(app_dir):
        for dirname in dirnames:
            gs_dir = os.path.join(dirpath, dirname)
            if glob(os.path.join(gs_dir, "gswin??c.exe")):
                # Directory appears to contain a Ghostscript executable
                gs_dirs.append(gs_dir)

    # 2. A dedicated Ghostscript installation
    #    EXEs are usually under something like %PROGRAMFILES%\gs\gs9.27\bin.
    for program_files in map(os.getenv, pf_vars):
        if program_files:
            gs_dir = os.path.join(program_files, "gs")
            for dirpath, dirnames, filenames in os.walk(gs_dir):
                for dirname in dirnames:
                    gs_dir = os.path.join(dirpath, dirname)
                    # Because some of the variables in pf_vars may refer
                    # to the same location, we have to check that we didn't
                    # already include this directory in our search path
                    if dirname.lower() == "bin" and not gs_dir in gs_dirs:
                        gs_dirs.append(gs_dir)

    ## 3. Other locations in %PATH%
    ##    Deliberately omitted because this is potentially dangerous,
    ##    and we can't be sure that whatever we find will be any good.
    #gs_dirs += os.getenv("PATH").split(os.pathsep)

    # Now look for a Ghostscript executable
    gs_exe = find_executable(gs_names, gs_dirs)

else:
    # Assume anything else is some kind of Unix system
    # Most Unix systems have Ghostscript available somewhere in $PATH
    gs_names = "gs",
    gs_dirs = os.getenv("PATH").split(os.pathsep)
    gs_exe = find_executable(gs_names, gs_dirs)

gs_version = None
if gs_exe:
    # Retrieve the Ghostscript version number
    # This doubles as a test that our Ghostscript executable is usable.
    # The [:-1] is to remove the trailing newline from Ghostscript's output.
    try:
        gs_version = bytes_to_str(check_output([gs_exe, "--version"])[:-1])

    except (OSError):
        # There's something wrong with our Ghostscript executable!
        gs_exe = None

# -----------------------------------------------------------------------

# Resolution for Ghostscript rendering (in dots per inch)
# TODO: This should not be hard-coded, but queried at runtime.
gs_dpi = 96

# Resolution used internally when downscaling is enabled
hr_dpi = 2 * gs_dpi


__all__ = ["GhostscriptBackend", "GhostscriptNotAvailable", "gs_dpi"]


class GhostscriptBackend(Backend):
    """Backend to render a document using Ghostscript.

    This backend supports the following formats:
    PDF -- rendered directly.
    Postscript -- must be converted to PDF first because rendering
      individual pages from Postscript files is not supported. This
      happens automatically when input_file has a .ps extension.

    Internal format conversion creates temporary files, which are
    cleaned up when the backend object is destroyed.

    Supported keyword arguments:
    enable_downscaling -- whether to enable downscaling using PIL.

    This backend requires an external Ghostscript binary. Most Unix
    systems should already have this installed as 'gs'. Windows users
    can download a suitable installer from the Ghostscript website.

    To enable downscaling, this backend also requires the PIL module.
    If downscaling is enabled and the PIL module is not available, the
    enable_downscaling argument will be silently ignored.
    """

    __slots__ = ["enable_downscaling"]

    def __init__(self, input_path, **kw):
        """Return a new Ghostscript rendering backend."""

        Backend.__init__(self, input_path, **kw)
        # Make sure we have a Ghostscript binary available
        if not gs_exe:
            # Identify possible executable names and search dirs
            search_names = ", ".join(gs_names)
            search_dirs = "\n".join(gs_dirs)
            if not search_dirs:
                search_dirs = "<no Ghostscript installations found>"

            raise BackendError(
                "Could not render {0}.\n"
                "Please make sure you have Ghostscript installed.\n"
                "\n"
                "Searched for {1} in these locations:\n"
                "{2}"
                .format(self.input_path, search_names, search_dirs)
            )

        # Whether to enable downscaling
        # This has no effect if PIL is not available on your system.
        if "enable_downscaling" in kw:
            self.enable_downscaling = kw["enable_downscaling"]
        else:
            self.enable_downscaling = False

        # Determine whether we can render this file based on its extension
        base, ext = os.path.splitext(input_path)

        if ext.lower() == ".pdf":
            # Render PDF files directly
            self.input_path = input_path

        elif ext.lower() == ".ps":
            # Convert Postscript files to PDF
            fd, pdf_path = tempfile.mkstemp(suffix=".pdf")
            os.close(fd)
            self.render_to_pdf(pdf_path)

            # Render the converted PDF file
            self.input_path = pdf_path
            self.temp_files.append(pdf_path)

        else:
            raise BackendError(
                "Could not render {0} because the file type is not "
                "supported by the Ghostscript backend."
                .format(input_path)
            )

    def page_count(self):
        """Return the number of pages in the input file."""

        base, ext = os.path.splitext(self.input_path)
        if ext.lower() != ".pdf":
            raise BackendError("Only PDF files are supported.")

        # The Ghostscript interpreter expects forward slashes in file paths
        gs_input_path = self.input_path.replace(os.sep, "/")

        # Ghostscript command to return the page count of a PDF file
        gs_pc_command = "({0}) (r) file runpdfbegin pdfpagecount = quit"

        # Ghostscript command line
        gs_args = [gs_exe,
                   "-q",
                   "-dNODISPLAY",
                   "-c",
                   gs_pc_command.format(gs_input_path)]
        # Return the page count if it's a valid PDF, or None otherwise
        return int(self._check_output(gs_args))

    def render_page(self, page_num):
        """Render the specified page of the input file."""

        base, ext = os.path.splitext(self.input_path)
        if ext.lower() != ".pdf":
            raise BackendError("Only PDF files are supported.")

        # Resolution for Ghostscript rendering
        if PIL and self.enable_downscaling:
            gs_res = hr_dpi
        else:
            gs_res = gs_dpi

        # Ghostscript command line
        gs_args = [gs_exe,
                   "-q",
                   "-r{0}".format(gs_res),
                   "-dBATCH",
                   "-dNOPAUSE",
                   "-dSAFER",
                   "-dPDFSettings=/SCREEN",
                   "-dPrinted=false",
                   "-dTextAlphaBits=4",
                   "-dGraphicsAlphaBits=4",
                   "-dCOLORSCREEN",
                   "-dDOINTERPOLATE",

                   # Newer versions of Ghostscript support the -sPageList
                   # option, but our user's version might not have it.
                   # This approach is clumsier, but backwards-compatible.
                   "-dFirstPage={0}".format(page_num),
                   "-dLastPage={0}".format(page_num),

                   # Raw PPM is the only full-color image format that all
                   # versions of Tk are guaranteed to support.
                   "-sDEVICE=ppmraw",
                   "-sOutputFile=-",
                   self.input_path]

        # Call Ghostscript to render the PDF
        image_data = self._check_output(gs_args)

        if PIL and self.enable_downscaling:
            page_bytes = io.BytesIO(image_data)
            page_image = PIL.Image.open(page_bytes)

            # Scale down the output from Ghostscript
            w, h = page_image.size
            return page_image.resize((w * gs_dpi // hr_dpi,
                                      h * gs_dpi // hr_dpi),
                                     resample=PIL.Image.BICUBIC)

        else:
            # Return the image data from Ghostscript directly
            return image_data

    def render_to(self, device, output_path, *args):
        """Render the input file to the specified Ghostscript output device.

        Positional arguments are appended to the Ghostscript command line.
        """

        # Sanity checks
        if output_path == self.input_path:
            raise BackendError("Input and output must be separate files.")
        elif output_path == "-":
            raise BackendError("Rendering to stdout is not supported.")
        elif not output_path:
            raise BackendError("You must provide an output path.")

        # Ghostscript command line
        gs_args = [gs_exe,
                   "-q",
                   "-dBATCH",
                   "-dNOPAUSE",
                   "-dSAFER",
                   "-sDEVICE={0}".format(device),
                   "-sOutputFile={0}".format(output_path)]
        if args:
            gs_args += list(args)
        gs_args.append(self.input_path)

        # Call Ghostscript to convert the file
        self._check_output(gs_args)

    def render_to_pdf(self, output_path):
        """Render the input file to PDF."""

        return self.render_to("pdfwrite", output_path)

    # -------------------------------------------------------------------------

    def _check_output(self, args):
        """Wrapper for check_output() to handle error conditions."""

        try:
            return check_output(args)

        except (subprocess.CalledProcessError) as err:
            # Something went wrong with the call to Ghostscript
            if err.output:
                # Save the output from Ghostscript
                gs_output = bytes_to_str(err.output)[:-1]

                # Quote command line arguments containing spaces
                args = []
                for arg in err.cmd:
                    if " " in arg:
                        args.append('"{0}"'.format(arg))
                    else:
                        args.append(arg)

                # Raise a more informative exception
                raise BackendError(
                    "{0}\n"
                    "\n"
                    "Ghostscript command line (return code = {1}):\n"
                    "{2}"
                    .format(gs_output, err.returncode, " ".join(args))
                )

            else:
                # Raise the exception as-is
                raise

    # -------------------------------------------------------------------------

    @staticmethod
    def executable():
        """Return the path to the Ghostscript executable."""

        return gs_exe

    @staticmethod
    def search_path():
        """Return the search path for the Ghostscript executable."""

        return gs_dirs

    @staticmethod
    def version():
        """Return the version of the Ghostscript executable."""

        return gs_version
```