```
#!/usr/bin/env python

import ConfigParserRT
import cortex
import dcs
import filterwheel
import linear
import logging
import logging.config
import mechanism
import rotator
import SimpleXMLRPCServer
import tc08
import gui
import detmech

import ptest

class instrument:
    """
    Class representing the instrument as a whole.  Provides methods
    for the highest level operations and has as members objects
    representing the sub-systems of the instruments.
    """
    def __init__(self):
        """
        Parse the config file dics.ini and create the required instances
        of subsystem objects.
        """
        # Set up logging from the details in the config file.
        logging.config.fileConfig('dics.ini')
        self.logger = logging.getLogger('dazle')

        # Read the config file
        self.logger.debug('Opening config file dics.ini...')
        try:
            configFile = file('dics.ini')
        except IOError:
            # If dics.ini isn't there then we're screwed.
            self.logger.critical('Failed to open config file dics.ini!')
            return
        self.config = ConfigParserRT.SafeConfigParser()
        self.config.readfp(configFile)
        configFile.close()

        # Create any specified tc08 temperature logger instances.
        #self.__create('tc08.tc08')
        # Create any specified cortex-based ICU daisy chain instances.
        self.__create('cortex.cortex')
        # Create any specified linear mechanism instances
        self.__create('linear.linear')
        # Create any specified detmech instances
        self.__create('detmech.detmech')
        # Create any specified filterwheel instances.
        self.__create('filterwheel.filterwheel')
        # Create any specified target field instances
        self.__create('rotator.target')
        # Create any specified telescope instances
        self.__create('rotator.telescope')
        # Create any specified rotator mechanism instances
```

```
        self.__create('rotator.rotator')
        # Create any specified dcs interface instances.
        self.__create('dcs.dcs')

    def __del__(self):
        """
        Several types of subsystems need to store status/config information
        in the dics.ini file when the program is shut down.
        """
        self.logger.debug('__del__ called')
        # Look for objects that want to write to the config, and
        # for those that do update their entries in the config object
        for ref in self.__dict__.values():
            try:
                ref.updateConfig(self.config)
            except AttributeError:
                # This object doesn't want to write to config
                pass

        # Open the config file for writing.
        self.logger.info('Opening config file dics.ini...')
        try:
            configFile = file('dics.ini', 'w')
        except IOError:
            self.logger.critical('Failed to open config file dics.ini!')
            self.logger.critical('Unable to write status/config data.')
            return

        # Now write the up to date config to the config file before exit
        self.config.write(configFile)
        self.logger.info('Config saved.')

    def __create(self, klass):
        """
        Creates all instances of the class passed as an argument that are
        specified in the ConfigParser.SafeConfigParser object self.config.
        """
        # Parse the klass string.
        parts = klass.split('.')
        class_name = parts[-1]
        class_ref = reduce(getattr, parts[1:], globals()[parts[0]])

        # First check there is a config, and it has specified instances of
        # the given class.
        if isinstance(self.config, ConfigParserRT.SafeConfigParser):
            if self.config.has_option(class_name, 'instances'):
                # Get a list of the names to use for the instances
                # and loop over the names creating the instances
                for name in \
                        self.config.get(class_name, 'instances').split(','):
                    if self.config.has_section(name):
                        self.logger.info('Creating %s object %s' % \
                                             (class_name, name))
                        # Extract the settings for the instance and turn
                        # the rather useless list of tuples format that
                        # the config module gives into a far more
                        # useful dictionary.
                        args_dict = dict(self.config.items(name))
                        args_dict['name'] = name
                        # Now look through the values in the dictionary,
```

```
                        # and if one is a string equal to the name of
                        # an attribute of this object replace the string
                        # with a reference to the attribute.
                        for (key, value) in args_dict.iteritems():
                            args_dict[key] = getattr(self, value, value)
                        # Now create the required instance.
                        setattr(self, name, class_ref(**args_dict))

                    else:
                        self.logger.warning(\
                            '%s instance %s listed but has no section.' \
                            % (class_name, name))
            else:
                # No instances listed in file for this class.
                self.logger.warning(\
                    'Trying to create %s instances but none in config file.'\
                    % class_name)
        else:
            # For reson there's no config object.  Something is screwed up bad
            self.logger.critical(\
                'Trying to create %s instances but no config object found!' \
                % class_name)

    def gui(self):
        self.g = gui.gui(self)
        self.g.main()


#if __name__ == '__main__':
    # If executed create dazle, an instance of the instrument class,
    # then start up an XML-RPC server to expose its methods and
    # attributes.
#    dazle = instrument()
#    server = SimpleXMLRPCServer.SimpleXMLRPCServer(('optics24.ast.cam.ac.uk',\
#                                                     1042))
#    server.register_instance(dazle)
#    server.register_introspection_functions()
#    server.serve_forever()
```