**[midea\_ac\_lan](https://github.com/georgezhao2010/midea_ac_lan)/**[**custom\_components**](https://github.com/georgezhao2010/midea_ac_lan/tree/master/custom_components)**/**[**midea\_ac\_lan**](https://github.com/georgezhao2010/midea_ac_lan/tree/master/custom_components/midea_ac_lan)**/climate.py**

from homeassistant.components.climate import \*

from homeassistant.components.climate.const import \*

from homeassistant.const import (

|  |
| --- |
|  |
|  |  |
|  |  |
|  | TEMP\_CELSIUS, |
|  | PRECISION\_WHOLE, |
|  | PRECISION\_HALVES, |
|  | ATTR\_TEMPERATURE, |
|  | CONF\_DEVICE\_ID, |
|  | CONF\_SWITCHES |
|  | ) |
|  |  |
|  | from .const import ( |
|  | DOMAIN, |
|  | DEVICES, |
|  | ) |
|  | from .midea.devices.ac.device import DeviceAttributes as ACAttributes |
|  | from .midea.devices.c3.device import DeviceAttributes as C3Attributes |
|  | from .midea.devices.cc.device import DeviceAttributes as CCAttributes |
|  | from .midea.devices.cf.device import DeviceAttributes as CFAttributes |
|  | from .midea.devices.fb.device import DeviceAttributes as FBAttributes |
|  | from .midea\_devices import MIDEA\_DEVICES |
|  | from .midea\_entity import MideaEntity |
|  |  |
|  |  |
|  | TEMPERATURE\_MAX = 30 |
|  | TEMPERATURE\_MIN = 17 |
|  |  |
|  | FAN\_SILENT = "Silent" |
|  | FAN\_FULL\_SPEED = "Super high" |
|  |  |
|  | PRESET\_SILENT = "Silent" |
|  |  |
|  |  |
|  | async def async\_setup\_entry(hass, config\_entry, async\_add\_entities): |
|  | device\_id = config\_entry.data.get(CONF\_DEVICE\_ID) |
|  | device = hass.data[DOMAIN][DEVICES].get(device\_id) |
|  | extra\_switches = config\_entry.options.get( |
|  | CONF\_SWITCHES, [] |
|  | ) |
|  | devs = [] |
|  | for entity\_key, config in MIDEA\_DEVICES[device.device\_type]["entities"].items(): |
|  | if config["type"] == "climate" and (config.get("default") or entity\_key in extra\_switches): |
|  | if device.device\_type == 0xAC: |
|  | devs.append(MideaACClimate(device, entity\_key)) |
|  | elif device.device\_type == 0xCC: |
|  | devs.append(MideaCCClimate(device, entity\_key)) |
|  | elif device.device\_type == 0xCF: |
|  | devs.append(MideaCFClimate(device, entity\_key)) |
|  | elif device.device\_type == 0xC3: |
|  | devs.append(MideaC3Climate(device, entity\_key, config["zone"])) |
|  | elif device.device\_type == 0xFB: |
|  | devs.append(MideaFBClimate(device, entity\_key)) |
|  | async\_add\_entities(devs) |
|  |  |
|  |  |
|  | class MideaClimate(MideaEntity, ClimateEntity): |
|  | def \_\_init\_\_(self, device, entity\_key): |
|  | super().\_\_init\_\_(device, entity\_key) |
|  |  |
|  | @property |
|  | def state(self): |
|  | return self.hvac\_mode |
|  |  |
|  | @property |
|  | def supported\_features(self): |
|  | return ClimateEntityFeature.TARGET\_TEMPERATURE | \ |
|  | ClimateEntityFeature.FAN\_MODE | \ |
|  | ClimateEntityFeature.PRESET\_MODE | \ |
|  | ClimateEntityFeature.SWING\_MODE | \ |
|  | ClimateEntityFeature.AUX\_HEAT |
|  |  |
|  | @property |
|  | def min\_temp(self): |
|  | return TEMPERATURE\_MIN |
|  |  |
|  | @property |
|  | def max\_temp(self): |
|  | return TEMPERATURE\_MAX |
|  |  |
|  | @property |
|  | def temperature\_unit(self): |
|  | return TEMP\_CELSIUS |
|  |  |
|  | @property |
|  | def target\_temperature\_low(self): |
|  | return TEMPERATURE\_MIN |
|  |  |
|  | @property |
|  | def target\_temperature\_high(self): |
|  | return TEMPERATURE\_MAX |
|  |  |
|  | @property |
|  | def hvac\_modes(self): |
|  | return self.\_modes |
|  |  |
|  | @property |
|  | def swing\_modes(self): |
|  | return self.\_swing\_modes |
|  |  |
|  | @property |
|  | def is\_on(self) -> bool: |
|  | return self.hvac\_mode != HVACMode.OFF |
|  |  |
|  | @property |
|  | def hvac\_mode(self) -> str: |
|  | if self.\_device.get\_attribute("power"): |
|  | return self.\_modes[self.\_device.get\_attribute("mode")] |
|  | else: |
|  | return HVACMode.OFF |
|  |  |
|  | @property |
|  | def target\_temperature(self): |
|  | return self.\_device.get\_attribute("target\_temperature") |
|  |  |
|  | @property |
|  | def current\_temperature(self): |
|  | return self.\_device.get\_attribute("indoor\_temperature") |
|  |  |
|  | @property |
|  | def is\_aux\_heat(self): |
|  | return self.\_device.get\_attribute("aux\_heat") |
|  |  |
|  | @property |
|  | def preset\_modes(self): |
|  | return self.\_preset\_modes |
|  |  |
|  | @property |
|  | def preset\_mode(self): |
|  | if self.\_device.get\_attribute("comfort\_mode"): |
|  | mode = PRESET\_COMFORT |
|  | elif self.\_device.get\_attribute("eco\_mode"): |
|  | mode = PRESET\_ECO |
|  | elif self.\_device.get\_attribute("boost\_mode"): |
|  | mode = PRESET\_BOOST |
|  | elif self.\_device.get\_attribute("sleep\_mode"): |
|  | mode = PRESET\_SLEEP |
|  | elif self.\_device.get\_attribute("frost\_protect"): |
|  | mode = PRESET\_AWAY |
|  | else: |
|  | mode = PRESET\_NONE |
|  | return mode |
|  |  |
|  | @property |
|  | def extra\_state\_attributes(self) -> dict: |
|  | return self.\_device.attributes |
|  |  |
|  | def turn\_on(self): |
|  | self.\_device.set\_attribute(attr="power", value=True) |
|  |  |
|  | def turn\_off(self): |
|  | self.\_device.set\_attribute(attr="power", value=False) |
|  |  |
|  | def set\_temperature(self, \*\*kwargs) -> None: |
|  | if ATTR\_TEMPERATURE not in kwargs: |
|  | return |
|  | temperature = float(int((float(kwargs.get(ATTR\_TEMPERATURE)) \* 2) + 0.5)) / 2 |
|  | hvac\_mode = kwargs.get(ATTR\_HVAC\_MODE) |
|  | if hvac\_mode == HVACMode.OFF: |
|  | self.turn\_off() |
|  | else: |
|  | try: |
|  | mode = self.\_modes.index(hvac\_mode.lower()) if hvac\_mode else None |
|  | self.\_device.set\_target\_temperature( |
|  | target\_temperature=temperature, mode=mode) |
|  | except ValueError as e: |
|  | pass |
|  |  |
|  | def set\_hvac\_mode(self, hvac\_mode: str) -> None: |
|  | hvac\_mode = hvac\_mode.lower() |
|  | if hvac\_mode == HVACMode.OFF: |
|  | self.turn\_off() |
|  | else: |
|  | self.\_device.set\_attribute(attr="mode", value=self.\_modes.index(hvac\_mode)) |
|  |  |
|  | def set\_preset\_mode(self, preset\_mode: str) -> None: |
|  | old\_mode = self.preset\_mode |
|  | preset\_mode = preset\_mode.lower() |
|  | if preset\_mode == PRESET\_AWAY: |
|  | self.\_device.set\_attribute(attr="frost\_protect", value=True) |
|  | elif preset\_mode == PRESET\_COMFORT: |
|  | self.\_device.set\_attribute(attr="comfort\_mode", value=True) |
|  | elif preset\_mode == PRESET\_SLEEP: |
|  | self.\_device.set\_attribute(attr="sleep\_mode", value=True) |
|  | elif preset\_mode == PRESET\_ECO: |
|  | self.\_device.set\_attribute(attr="eco\_mode", value=True) |
|  | elif preset\_mode == PRESET\_BOOST: |
|  | self.\_device.set\_attribute(attr="boost\_mode", value=True) |
|  | elif old\_mode == PRESET\_AWAY: |
|  | self.\_device.set\_attribute(attr="frost\_protect", value=False) |
|  | elif old\_mode == PRESET\_COMFORT: |
|  | self.\_device.set\_attribute(attr="comfort\_mode", value=False) |
|  | elif old\_mode == PRESET\_SLEEP: |
|  | self.\_device.set\_attribute(attr="sleep\_mode", value=False) |
|  | elif old\_mode == PRESET\_ECO: |
|  | self.\_device.set\_attribute(attr="eco\_mode", value=False) |
|  | elif old\_mode == PRESET\_BOOST: |
|  | self.\_device.set\_attribute(attr="boost\_mode", value=False) |
|  |  |
|  |  |
|  | def update\_state(self, status): |
|  | try: |
|  | self.schedule\_update\_ha\_state() |
|  | except Exception: |
|  | pass |
|  |  |
|  | def turn\_aux\_heat\_on(self) -> None: |
|  | self.\_device.set\_attribute(attr="aux\_heat", value=True) |
|  |  |
|  | def turn\_aux\_heat\_off(self) -> None: |
|  | self.\_device.set\_attribute(attr="aux\_heat", value=False) |
|  |  |
|  |  |
|  | class MideaACClimate(MideaClimate): |
|  | def \_\_init\_\_(self, device, entity\_key): |
|  | super().\_\_init\_\_(device, entity\_key) |
|  | self.\_modes = [HVACMode.OFF, HVACMode.AUTO, HVACMode.COOL, HVACMode.DRY, HVACMode.HEAT, HVACMode.FAN\_ONLY] |
|  | self.\_fan\_speeds = { |
|  | FAN\_SILENT.capitalize(): 20, |
|  | FAN\_LOW.capitalize(): 40, |
|  | FAN\_MEDIUM.capitalize(): 60, |
|  | FAN\_HIGH.capitalize(): 80, |
|  | FAN\_FULL\_SPEED.capitalize(): 100, |
|  | FAN\_AUTO.capitalize(): 102 |
|  | } |
|  | self.\_swing\_modes = [ |
|  | SWING\_OFF.capitalize(), |
|  | SWING\_VERTICAL.capitalize(), |
|  | SWING\_HORIZONTAL.capitalize(), |
|  | SWING\_BOTH.capitalize() |
|  | ] |
|  | self.\_preset\_modes = [PRESET\_NONE, PRESET\_COMFORT, PRESET\_ECO, PRESET\_BOOST, PRESET\_AWAY] |
|  |  |
|  | @property |
|  | def fan\_modes(self): |
|  | return list(self.\_fan\_speeds.keys()) |
|  |  |
|  | @property |
|  | def fan\_mode(self) -> str: |
|  | fan\_speed = self.\_device.get\_attribute(ACAttributes.fan\_speed) |
|  | if fan\_speed > 100: |
|  | return FAN\_AUTO.capitalize() |
|  | elif fan\_speed > 80: |
|  | return FAN\_FULL\_SPEED.capitalize() |
|  | elif fan\_speed > 60: |
|  | return FAN\_HIGH.capitalize() |
|  | elif fan\_speed > 40: |
|  | return FAN\_MEDIUM.capitalize() |
|  | elif fan\_speed > 20: |
|  | return FAN\_LOW.capitalize() |
|  | else: |
|  | return FAN\_SILENT.capitalize() |
|  |  |
|  | @property |
|  | def target\_temperature\_step(self): |
|  | return PRECISION\_WHOLE if self.\_device.temperature\_step == 1 else PRECISION\_HALVES |
|  |  |
|  | @property |
|  | def swing\_mode(self): |
|  | swing\_mode = (1 if self.\_device.get\_attribute(ACAttributes.swing\_vertical) else 0) + \ |
|  | (2 if self.\_device.get\_attribute(ACAttributes.swing\_horizontal) else 0) |
|  | return self.\_swing\_modes[swing\_mode] |
|  |  |
|  | @property |
|  | def outdoor\_temperature(self): |
|  | return self.\_device.get\_attribute(ACAttributes.outdoor\_temperature) |
|  |  |
|  | def set\_fan\_mode(self, fan\_mode: str) -> None: |
|  | fan\_speed = self.\_fan\_speeds.get(fan\_mode.capitalize()) |
|  | if fan\_speed: |
|  | self.\_device.set\_attribute(attr=ACAttributes.fan\_speed, value=fan\_speed) |
|  |  |
|  | def set\_swing\_mode(self, swing\_mode: str) -> None: |
|  | swing = self.\_swing\_modes.index(swing\_mode.capitalize()) |
|  | swing\_vertical = swing & 1 > 0 |
|  | swing\_horizontal = swing & 2 > 0 |
|  | self.\_device.set\_swing(swing\_vertical=swing\_vertical, swing\_horizontal=swing\_horizontal) |
|  |  |
|  |  |
|  | class MideaCCClimate(MideaClimate): |
|  | def \_\_init\_\_(self, device, entity\_key): |
|  | super().\_\_init\_\_(device, entity\_key) |
|  | self.\_modes = [HVACMode.OFF, HVACMode.FAN\_ONLY, HVACMode.DRY, HHVACMode.HEAT, HVACMode.COOL, HVACMode.AUTO] |
|  | self.\_swing\_modes = [ |
|  | SWING\_OFF.capitalize(), |
|  | SWING\_ON.capitalize() |
|  | ] |
|  | self.\_preset\_modes = [PRESET\_NONE, PRESET\_SLEEP, PRESET\_ECO] |
|  |  |
|  | @property |
|  | def fan\_modes(self): |
|  | return self.\_device.fan\_modes |
|  |  |
|  | @property |
|  | def fan\_mode(self) -> str: |
|  | return self.\_device.get\_attribute(CCAttributes.fan\_speed) |
|  |  |
|  | @property |
|  | def target\_temperature\_step(self): |
|  | return self.\_device.get\_attribute(CCAttributes.temperature\_precision) |
|  |  |
|  | @property |
|  | def swing\_mode(self): |
|  | return SWING\_ON.capitalize() if self.\_device.get\_attribute(CCAttributes.swing) else SWING\_OFF.capitalize() |
|  |  |
|  | def set\_fan\_mode(self, fan\_mode: str) -> None: |
|  | self.\_device.set\_attribute(attr=CCAttributes.fan\_speed, value=fan\_mode) |
|  |  |
|  | def set\_swing\_mode(self, swing\_mode: str) -> None: |
|  | self.\_device.set\_attribute( |
|  | attr=CCAttributes.swing, |
|  | value=swing\_mode.capitalize() == SWING\_ON.capitalize() |
|  | ) |
|  |  |
|  |  |
|  | class MideaCFClimate(MideaClimate): |
|  | def \_\_init\_\_(self, device, entity\_key): |
|  | super().\_\_init\_\_(device, entity\_key) |
|  | self.\_modes = [HVACMode.OFF, HVACMode.AUTO, HVACMode.COOL, HVACMode.HEAT] |
|  |  |
|  | @property |
|  | def supported\_features(self): |
|  | return ClimateEntityFeature.TARGET\_TEMPERATURE | ClimateEntityFeature.AUX\_HEAT |
|  |  |
|  | @property |
|  | def target\_temperature\_step(self): |
|  | return PRECISION\_WHOLE |
|  |  |
|  | @property |
|  | def min\_temp(self): |
|  | return self.\_device.get\_attribute(CFAttributes.min\_temperature) |
|  |  |
|  | @property |
|  | def max\_temp(self): |
|  | return self.\_device.get\_attribute(CFAttributes.max\_temperature) |
|  |  |
|  | @property |
|  | def target\_temperature\_low(self): |
|  | return self.\_device.get\_attribute(CFAttributes.min\_temperature) |
|  |  |
|  | @property |
|  | def target\_temperature\_high(self): |
|  | return self.\_device.get\_attribute(CFAttributes.max\_temperature) |
|  |  |
|  | @property |
|  | def current\_temperature(self): |
|  | return self.\_device.get\_attribute(CFAttributes.current\_temperature) |
|  |  |
|  |  |
|  | class MideaC3Climate(MideaClimate): \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \_powers = [ |
|  | C3Attributes.zone1\_power, |
|  | C3Attributes.zone2\_power, |
|  | ] |
|  |  |
|  | def \_\_init\_\_(self, device, entity\_key, zone): |
|  | super().\_\_init\_\_(device, entity\_key) |
|  | self.\_zone = zone |
|  | self.\_modes = [HVACMode.OFF, HVACMode.AUTO, HVACMode.COOL, HVACMode.HEAT] |
|  | self.\_power\_attr = MideaC3Climate.\_powers[self.\_zone] |
|  |  |
|  | @property |
|  | def supported\_features(self): |
|  | return ClimateEntityFeature.TARGET\_TEMPERATURE |
|  |  |
|  | @property |
|  | def target\_temperature\_step(self): |
|  | return PRECISION\_WHOLE if \ |
|  | self.\_device.get\_attribute(C3Attributes.zone\_temp\_type)[self.\_zone] else PRECISION\_HALVES |
|  |  |
|  | @property |
|  | def min\_temp(self): |
|  | return self.\_device.get\_attribute(C3Attributes.temperature\_min)[self.\_zone] |
|  |  |
|  | @property |
|  | def max\_temp(self): |
|  | return self.\_device.get\_attribute(C3Attributes.temperature\_max)[self.\_zone] |
|  |  |
|  | @property |
|  | def target\_temperature\_low(self): |
|  | return self.\_device.get\_attribute(C3Attributes.temperature\_min)[self.\_zone] |
|  |  |
|  | @property |
|  | def target\_temperature\_high(self): |
|  | return self.\_device.get\_attribute(C3Attributes.temperature\_max)[self.\_zone] |
|  |  |
|  | def turn\_on(self): |
|  | self.\_device.set\_attribute(attr=self.\_power\_attr, value=True) |
|  |  |
|  | def turn\_off(self): |
|  | self.\_device.set\_attribute(attr=self.\_power\_attr, value=False) |
|  |  |
|  | @property |
|  | def hvac\_mode(self) -> str: |
|  | if self.\_device.get\_attribute(self.\_power\_attr): |
|  | return self.\_modes[self.\_device.get\_attribute(C3Attributes.mode)] |
|  | else: |
|  | return HVACMode.OFF |
|  |  |
|  | @property |
|  | def target\_temperature(self): |
|  | return self.\_device.get\_attribute(C3Attributes.target\_temperature)[self.\_zone] |
|  |  |
|  | @property |
|  | def current\_temperature(self): |
|  | return None |
|  |  |
|  | def set\_temperature(self, \*\*kwargs) -> None: |
|  | if ATTR\_TEMPERATURE not in kwargs: |
|  | return |
|  | temperature = float(int((float(kwargs.get(ATTR\_TEMPERATURE)) \* 2) + 0.5)) / 2 |
|  | hvac\_mode = kwargs.get(ATTR\_HVAC\_MODE) |
|  | if hvac\_mode == HVACMode.OFF: |
|  | self.turn\_off() |
|  | else: |
|  | try: |
|  | mode = self.\_modes.index(hvac\_mode.lower()) if hvac\_mode else None |
|  | self.\_device.set\_target\_temperature( |
|  | zone=self.\_zone, target\_temperature=temperature, mode=mode) |
|  | except ValueError as e: |
|  | pass |
|  |  |
|  | def set\_hvac\_mode(self, hvac\_mode: str) -> None: |
|  | hvac\_mode = hvac\_mode.lower() |
|  | if hvac\_mode == HVACMode.OFF: |
|  | self.turn\_off() |
|  | else: |
|  | self.\_device.set\_mode(self.\_zone, self.\_modes.index(hvac\_mode)) |
|  |  |
|  |  |
|  | class MideaFBClimate(MideaClimate): |
|  | def \_\_init\_\_(self, device, entity\_key): |
|  | super().\_\_init\_\_(device, entity\_key) |
|  | self.\_modes = [HVACMode.OFF, HVACMode.HEAT] |
|  | self.\_preset\_modes = self.\_device.modes |
|  |  |
|  | @property |
|  | def supported\_features(self): |
|  | return ClimateEntityFeature.TARGET\_TEMPERATURE | ClimateEntityFeature.PRESET\_MODE |
|  |  |
|  | @property |
|  | def target\_temperature\_step(self): |
|  | return PRECISION\_WHOLE |
|  |  |
|  | @property |
|  | def preset\_modes(self): |
|  | return self.\_preset\_modes |
|  |  |
|  | @property |
|  | def preset\_mode(self): |
|  | return self.\_device.get\_attribute(attr=FBAttributes.mode) |
|  |  |
|  | @property |
|  | def min\_temp(self): |
|  | return 5 |
|  |  |
|  | @property |
|  | def max\_temp(self): |
|  | return 35 |
|  |  |
|  | @property |
|  | def target\_temperature\_low(self): |
|  | return 5 |
|  |  |
|  | @property |
|  | def target\_temperature\_high(self): |
|  | return 35 |
|  |  |
|  | @property |
|  | def hvac\_mode(self) -> str: |
|  | return HVACMode.HEAT if self.\_device.get\_attribute(attr=FBAttributes.power) else HVACMode.OFF |
|  |  |
|  | @property |
|  | def current\_temperature(self): |
|  | return self.\_device.get\_attribute(FBAttributes.current\_temperature) |
|  |  |
|  | def set\_temperature(self, \*\*kwargs) -> None: |
|  | if ATTR\_TEMPERATURE not in kwargs: |
|  | return |
|  | temperature = float(int((float(kwargs.get(ATTR\_TEMPERATURE)) \* 2) + 0.5)) / 2 |
|  | hvac\_mode = kwargs.get(ATTR\_HVAC\_MODE) |
|  | if hvac\_mode == HVACMode.OFF: |
|  | self.turn\_off() |
|  | else: |
|  | self.\_device.set\_attribute(attr=FBAttributes.target\_temperature, value=temperature) |
|  |  |
|  | def set\_hvac\_mode(self, hvac\_mode: str) -> None: |
|  | hvac\_mode = hvac\_mode.lower() |
|  | if hvac\_mode == HVACMode.OFF: |
|  | self.turn\_off() |
|  | else: |
|  | self.turn\_on() |
|  |  |
|  | def set\_preset\_mode(self, preset\_mode: str) -> None: |
|  | self.\_device.set\_attribute(attr=FBAttributes.mode,value=preset\_mode) |