USPest.org Applications Programming Interface (API) for degree-day and hourly weather model products

Vers. 1.0 7/3/2024

Dan Upper and Len Coop
Oregon IPM Center, Oregon State University

Overview:

USPest.org is agricultural pest and crop modeling service provided by Oregon State University at the Oregon Integrated Pest Management (IPM) Center. This website, running without charge to end-users since 1996, was funded primarily by competitive grants from USDA National Institute for Food and Agriculture (NIFA), mainly through programs including Crop Protection and Pest Management (CPPM) and Tactical Sciences for Agricultural Biosecurity (TSAB). See disclaimer at bottom of this document.

We are making our API available to allow third parties such as statewide IPM programs, county Extension offices, and pest management consulting firms, to configure and embed our services within their own web pages to best reach and support IPM and agricultural decision makers.

This document allows website builders and programmers to access many of the modeling services provided by USPest.org (many insect, plant disease, weeds, crop and related model outputs, currently totaling around 160+ models). These models fall into two main types: degree-day models (ca. 145 models, listed at https://uspest.org/cgi-bin/sdb/viewdb.cgi) and hourly weather-driven models (ca. 20 models, listing available at https://uspest.org/risk/models [expand header and sub-headers starting at “Plant Disease/Other Hourly Driven Models”]).

This API currently does not detail any of our spatial products, rather only site-based models which are driven by (currently over 33,000 weather stations, ingested primarily from MesoWest / Synoptic Data). However, our daily series of 32, 41, and 50°F degree-day maps (cumulative degree-days since Jan. 1) for CONUS are available at https://uspest.org/ddmaps/. For example, the daily maps for the date 6/26/2024 should be at https://uspest.org/ddmaps/us2024-06-26.zip within 36 hours after this date. Our spatial (map) outputs for invasive species are indexed at our webpage https://uspest.org/CAPS.

Our API products and services are available at no cost to third parties under these conditions: 1) Send us a request to use our API, with a description of how you intend to use it, at coopl@oregonstate.edu, 2) You will not charge your users for the use of our models, which are intended to be available for no cost to end-users. You may charge end-users for any added value services (such as pest management advisories, local conditions, advanced visualizations, etc.) that you combine with our products, 3) You will assume all responsibility and liability for using our modeling services in your products, and will not pass along any liability to Oregon State University or its personnel including the development team at USPest.org. See disclaimer at bottom of this document. In addition you will provide us with feedback if there are any issues or inaccuracies detected in our models or API.

Our API is built using traditional HTTP CGI (common gateway interface) scripts, using the GET protocol. This approach should be familiar to old-school web programmers. This API was primarily designed for use in our own web applications, but we realize that third parties may wish to add pest and crop models to their own web applications and interfaces. Outputs, depending on which script and settings used, may be HTML, JSON, CSV, or Javascript. As mentioned in the use conditions above, please send us a description of how you are making use of our services. With

your consent, we can then add links at our website to your apps to help promote adoption of these models.

There are three separate sets of API instructions: 1. for degree-day models, 2. for most of our hourly weather-driven models (but not yet for hourly weather data which are used as parameters to drive these models), and 3. for the “Stationpicker” that provides the back-end support for users to select a weather station. The current list of weather stations in our database is found at https://uspest.org/data/sta_list.txt Our older (but not obsolete) technical documents for embedding our model outputs into webpages are linked from our homepage at https://uspest.org/wea (under “Additional Resources”).

1. Degree-day models API

1a. Major scripts available for degree-day models API:

There are several CGI APIs that are used by Degree-day model apps. All are in https://uspest.org/dd/. The scripts are briefly introduced as:

ddbits-maintable.pl
Runs a model at a station as specified in the parameters. Returns the output as an HTML fragment.

ddbits-modelinfo.pl
Returns model parameters and JSON hash. The hash has keys "modelInputs", "pictable", and "events", and the values are HTML fragments.

date-compare3.pl
Runs a model at a station as specified in the parameters, but for this year and the preceding two years. Returns a table, which compares the dates of model events for different years, as an HTML fragment.

ddbits-CSV.pl
Runs a model at a station as specified in the parameters. Returns model output as a CSV file. The first line is model parameters; proper CSV starts with the column headers are in the second line.

ddbits-json.pl
Returns some combination of banner, model info, events, and model output, depending on the value of the "req" parameter described below. Runs the model if output is requested. Returns all output in a single JSON data structure. (Most outputs are typical JSON, but some things are sent as are HTML fragments.)

This should probably be the tool of choice for new projects, and definitely for projects which are not just a user interface (UI).

app_graph.pl
Runs a model at a station as specified in the parameters. Returns the output as a javascript fragment which builds a structure suitable for graphing with highcharts and then invokes highcharts on it.

Note that it returns Javascript (JS) *code*, NOT return a JSON data structure. A JS interpreter is required to create the data structure. Converting it to something that anything other than JS could work with would be non-trivial, because dates all appear as JS Date object constructor calls (e.g.,

"Date.UTC(2024, 2, 17)"). Highcharts needs the objects, but only JS can construct them. And making them into something equivalent in another language is language-specific.

1b. Major parameters used by the above scripts

All of these scripts use HTTP GET, and need many of the same parameters. Several other scripts, including model_app and most of its legacy forebearers, also use most of these same parameters. Only those used by modern code are listed here.

* sta
station code (standard for most all public weather networks).
  eg, sta=CRVO

Our current list of weather stations and codes is here: https://uspest.org/wea/sta_list.txt

* spp
model code (Originally species code, but model code is more accurate)
  eg, spp=clm

Note that a list of all uspest.org degree-day models including codes (abbreviations) can be viewed here: https://uspest.org/cgi-bin/sdb/viewdb.cgi

* cal,tlow,thi
degree-day calculation parameters. Optional (and ignored?) unless spp is "aaa" used to denote degree-day calculator mode.

cal stands for calculation method
tlow stands for lower temperature threshold
thi stands for upper temperature threshold
  eg, cal=S1&tlow=41&thi=95

* stm,std
The month and day of the start of the calculation interval, as numbers. Defaults to the start date given in the model itself. For biofix models, this default is normally useless. For calendar driven models, using the default is normally appropriate, and overriding it is for experts.
  eg, stm=5&std=1

* enm,end
The month and day of the end of the calculation interval, as numbers. enm and end default to 12 and 31, respectively.
  eg, enm=12&end=31

* styr
Year of the start of the calculation interval, as a 4-digit number. Default is this year. [Note that there is no analogous "enyr" parameter for the year of the end of the interval. In modern code, the end year is chosen so that the interval is always at least a day and never more than a year.]
  eg, styr=2024

* cel

Celsius flag. If 1, tlow and thi are interpreted as Celcius temperatures and output is in Celsius degrees and degree-days. If 0 or omitted (i.e., default), tlow and thi are interpreted as Fahrenheit temperatures and output is in Fahrenheit degrees and degree-days.
eg, cel=0

* fcast
Which forecast to use, for weather data more than 7 days into the future. Values are:
1 average of the last 10 years (updated every 2 months)
2 30 year normals (note these are from 1981-2010, not yet updated to 1991-2020)
3 data from last year
4 data from two years ago
5 NMME 7-month forecasts; see: https://uspest.org/wea/weaexp.html#NOAANMME
6 CFSv2 3-month forecasts (and NMME after they run out); see:
https://uspest.org/wea/weaexp.html#NOAACFSV2

Default (in dd/model_app) is 5 (as of April 2024; it may change).
eg, fcast=5

* ipp
include photoperiod flag. If 1, a photoperiod column is added to the table. Default is no column.
eg, ipp=1

* cr_pp
critical photoperiod in hours. If set, an event is created for when photoperiod drops below this number. Requires ipp=1
eg, ipp=1&cr_pp=11.2

* div_id
[app_graph.pl ONLY]
The name of the div in which the graph will be drawn.
eg, div_id=graph_content

* req
[ddbits-json.pl ONLY]
Which components to include in the returned structure. A valid value for this parameter is a list of components, joined with '-'. The components are:

banner
modelinfo
events
shorttable
fulltable
(datecomp -- maybe someday but not in April 2024)
(graph -- maybe someday but not in April 2024)

Default value is 'banner-modelinfo-events-shorttable'.
eg, req=banner-modelinfo-events-fulltable

* tab
[model_app ONLY]

The tab the app should start on. This is only useful if you also supply enough parameters that the model can run. Values are:

intro
station
model
table
graph

The default is "station".
eg, tab=graph

1c. Examples of each script, with a full URL and (condensed) output

===ddbits-maintable.pl===

Example URL:
https://uspest.org/dd/ddbits-maintable.pl?sta=UP496&spp=cbl&stm=5&std=14&styr=24&enm=12&end=30&fcast=6&cel=0&cr_pp=11.2&ipp=1

Snapshot (top portion only) of example output rendered in a web browser:

- Temperatures (and degree-days) are in F; rain in inches.
- Photoperiod in hours; '*' means < critical

date	max	min	rain	DDs today	DDs cumu	QA	photo per.	events
5-14	75	59	0.00	17.0	17		14.2	* START *
5-15	83	55	0.00	19.0	36		14.2	
5-16	75	59	0.00	17.0	53		14.2	
5-17	78	59	0.00	18.5	72		14.2	
5-18	85	63	0.00	24.0	96		14.3	
5-19	87	60	0.00	23.5	119		14.3	
5-20	86	63	0.00	24.5	144		14.3	First egg laying
5-21	83	64	0.00	23.5	167		14.3	
5-22	82	65	0.00	23.5	190		14.4	
5-23	79	64	0.00	21.5	212		14.4	
5-24	82	65	0.00	23.5	236		14.4	First egg hatch
5-25	85	66	0.00	25.5	261		14.4	
5-26	86	73	0.00	29.5	290		14.4	
5-27	91	66	0.00	28.4	319		14.5	
5-28	75	62	0.00	18.5	337		14.5	Small larvae
5-29	79	60	0.00	19.5	357		14.5	
5-30	79	63	0.00	21.0	378		14.5	
5-31	73	64	0.00	18.5	396		14.5	
date	**max**	**min**	**rain**	**DDs today**	**DDs cumu**	**QA**	**photo per.**	**events**
6-1	84	65	0.00	24.5	421		14.5	

Example (partial) HTML Output:

```
<div id="model-output-div">
<ul class="mt-metadata">
<li>Temperatures (and degree-days) are in F; rain in inches.</li>
<li>Photoperiod in hours; '*' means < critical </li>
</ul>

<table id="mt-table">
<tr>
<th class="mtc-date">date</th>
<th class="mtc-max">max</th>
<th class="mtc-min">min</th>
<th class="mtc-rain">rain</th>
<th class="mtc-dds-today">DDs<br>today</th>
<th class="mtc-dds-cumu">DDs<br>cumu</th>
<th class="mtc-qa">QA</th>
<th class="mtc-pp">photo<br>per.</th>
<th class="mtc-events">events</th>

</tr>
<tr class="event-even">
<td class="mtc-date">5-14</td>
<td class="mtc-max">73</td>
<td class="mtc-min">63</td>
<td class="mtc-rain">0.15</td>
<td class="mtc-dds-today">18.0</td>
<td class="mtc-dds-cumu">18</td>
<td class="mtc-qa"><a href="/wea/weaexp.html#NOAACFSV2">CF</a></td>
<td class="mtc-pp">14.2</td>
<td class="mtc-events">* START *</td>
</tr>
.....
</tr>
<tr class="noevent">
<td class="mtc-date">12-30</td>
<td class="mtc-max">53</td>
<td class="mtc-min">37</td>
<td class="mtc-rain">0.17</td>
<td class="mtc-dds-today"> 0.7</td>
<td class="mtc-dds-cumu">4917</td>
<td class="mtc-qa"><a href="/wea/weaexp.html#HxNoMax">Hx</a></td>
<td class="mtc-pp">10.2</td>
<td class="mtc-events"></td>
</tr>
</table>
</div>
</body>
</html>
```

===ddbits-modelinfo.pl===

Example URL:

https://uspest.org/dd/ddbits-modelinfo.pl?spp=eab&cel=1

The "events" html fragment looks approximately like this in a web browser:

DDs(C) after biofix:	Model Event
217	First adult emergence approx. beginning
258	10% adult emergence
359	50% adult emergence
461	First egg hatch
670	50% egg hatch and 95% adult emerg.
1139	Approx. end of adult activity

The "modelInputs" html fragment looks approximately like this in a web browser:

Model species/general links	emerald ash borer v2 [ash trees]
Type	invasive insect
Model source/other links	Duarte 2013 OSU OIPMC model revised analysis
Calculation method	single sine
Lower threshold	12.2°C
Upper threshold	36.1°C
Directions for starting/BIOFIX	calendar date
Starting date	standard date 1-1 2020
Ending date	default date 12-31 2020
Model validation status	Could benefit from data from additional years and regions
Region of known use	Developed from 2011-2012 data in Great Lakes region US
Extended forecast type	no forecast -- observed data from 2020

The "banner" html fragment consists mostly of images and is not readily approximated in ascii. It shows images of the species in various developmental stages, with links to reference material for the species and for the model.

Example HTML output:

{
"events": "<table id=\"events-table\"\n <tr>\n <th>DDs(C) after biofix:</th>\n <th>Model Event</th>\n </tr>\n <tr>\n <td>306</td>\n <td>First adult emergence approx. beginning</td>\n </tr>\n <tr>\n <td>363</td>\n <td>10% adult emergence</td>\n </tr>\n <tr>\n <td>475</td>\n <td>50% adult emergence</td>\n </tr>\n <tr>\n <td>828</td>\n <td>95% adult emergence</td>\n </tr>\n <tr>\n <td>1389</td>\n <td>Last adult emergence approx. end</td>\n </tr>\n</table>\n",
"modelInputs": "<table id=\"model-inputs\">\n <tr>\n <td>Model species/general links</td>\n <td>emerald ash borer [ash trees]</td>\n </tr>\n <tr>\n <td>Type</td>\n <td>invasive insect</td>\n </tr>\n <tr>\n <td>Model source/other links</td>\n <td>Duarte 2013 OSU OIPMC model analysis</td>\n </tr>\n <tr>\n <td>Calculation method</td>\n <td>single sine</td>\n </tr>\n <tr>\n <td>Lower threshold</td>\n <td>10.0°C</td>\n </tr>\n <tr>\n <td>Upper threshold</td>\n <td>37.8°C</td>\n </tr>\n <tr>\n <td>Directions for starting/BIOFIX</td>\n <td>calendar date</td>\n </tr>\n <tr>\n <td>Starting date</td>\n <td>standard date 1-1 2024</td>\n </tr>\n\n <tr>\n <td>Ending date</td>\n <td>default date

12-31 2024</td>\n </tr>\n <tr>\n   <td>Model validation status</td>\n   <td>Could benefit from data from additional years and regions</td>\n </tr>\n <tr>\n   <td>Region of known use</td>\n <td>Developed from 2011-2012 data in Great Lakes region US</td>\n </tr>\n <tr>\n <td>Extended forecast type</td>\n   <td>After 7 days, use 3.5-month CFSv2 based seasonal climate forecast</td>\n </tr>\n</table>\n",
  "pictable": "<table id=\"banner\"> <tr id=\"pic-row\"><td>\n\n </td></tr>\n <tr><td id=\"banner-model-name\">emerald ash borer [ash trees]
<FONT SIZE=+1>invasive insect model of Duarte 2013 OSU OIPMC model analysis</FONT></td></tr>\n</table> \n"
}

===date-compare3.pl===

Example URL:

https://uspest.org/dd/date-compare3.pl?sta=BIRM5&spp=aaa&cal=S1&tlow=41&thi=130&stm=1&std=1&styr=24&enm=12&end=31&fcast=5

Example HTML output as rendered in a web browser:

- Accumulation for BIRM5 from 1-1-2024 through 7-1: 1507 DDs(F)
- data quality is *ok*.

This year is about	versus	QA
6 days behind	2023	ok
4 days ahead	2022	ok
4 days ahead	30-yr normal	ok

Example HTML Output:

```
<ul class="dc-metadata">
  <li>Accumulation for BIRM5 from 1-1-2024 through 4-18: 184 DDs(F)</li>
  <li class="dc-qa-ok">data quality is <i>ok</i>.</li>
</ul>
<table id="date-compare">
<tr>
    <th class="dc-delta">This year is about</th>
    <th class="dc-year">versus</th>
    <th class="dc-qa">QA</th>
</tr>
<tr>
  <td class="dc-delta dc-ahead">15 days ahead</td>
  <td class="dc-year">2023</td>
  <td class="dc-qa dc-qa-ok">ok</td>
</tr>
<tr>
```

```
  <td class="dc-delta dc-ahead">20 days ahead</td>
  <td class="dc-year">2022</td>
  <td class="dc-qa dc-qa-ok">ok</td>
</tr>
<tr>
  <td class="dc-delta dc-ahead">13 days ahead</td>
  <td class="dc-year">30-yr normal</td>
  <td class="dc-qa dc-qa-ok">ok</td>
</tr>
</table>
```

===ddbits-CSV.pl===

Example URL:

https://uspest.org/dd/ddbits-CSV.pl?sta=C7563&spp=eab2&stm=1&std=1&styr=23&enm=12&end=31

Example CSV Output:

```
"Model: <a href=http://www.emeraldashborer.info>emerald ash borer v2</a> [ash trees] (eab2)","Station: C7563","Forecast type: no forecast -- observed data from 2023","Scale: Degrees F"
"month","day","maximum temperature F","minimum temperature F","precipitation in","degree-days today F","cumulative degree-days F","data quality reference","model events today"
1,1,47.0,34.0,0.01,0.0,0,,"* START *"
1,2,45.0,31.0,0.14,0.0,0,,
1,3,45.0,37.0,0.30,0.0,0,,
1,4,52.0,37.0,0.46,0.0,0,,
.....
6,7,82.0,55.0,0.00,14.5,457,,
6,8,70.0,53.0,0.14,7.6,464,,"10% adult emergence"
6,9,62.0,54.0,0.02,4.0,468,,
.....
12,29,55.0,48.0,0.25,0.2,2279,,
12,30,52.0,44.0,0.11,0.0,2279,,
12,31,45.8,34.1,"M",0.0,2279,"Hx",
```

===ddbits-json.pl===

Example URL:

https://uspest.org/dd/ddbits-json.pl?req=shorttable-datecomp&sta=AR824&spp=plr&stm=3&std=1&styr=24&enm=12&end=31&fcast=5

Example JSON Output:

[Note that as of 19Apr2024, the date comparison always fails because it hasn't been implemented yet.]

```
{
  "datecomp": {
    "success": null
  },
  "shorttable": {
    "rows": [
      {
        "min": "26.0",
        "max": "50.0",
        "ddcum": "2",
        "events": [
          "* START *"
        ],
        "rain": "0.65",
        "doy": 61,
        "qa_str": "",
        "date": "3-1",
        "yyyymmdd": "20240301",
        "dds": " 2.4"
      },
      .....
    ],
    "header_line": " 2024  AR824      {APRSWXNET Sturbridge K3GM MA Lat:42.1303 Long:-
72.0978 Elev:709 Rec:MnDayMaxMinPrecDD50}",
    "colkeys": [ "date", "max", "min", "rain", "dds",
                "ddcum", "qa_str", "events" ],
    "success": 1,
    "head": {
      "dds": "DDs<br>today",
      "date": "date",
      "rain": "rain",
      "qa": "QA",
      "events": "events",
      "ddcum": "DDs<br>cumu",
      "min": "min",
     "max": "max"
    },
    "messages": [
      "Temperatures (and degree-days) are in F; rain in inches."
    ],
    "columns": {
      "dds": "degree-days today F",
      "date": "date",
      "rain": "precipitation in",
      "qa_tag": "data quality notes",
      "events": "model events today",
      "ddcum": "cumulative degree-days F",
      "qa_ref": "data quality reference",
      "min": "minimum temperature F",
      "max": "maximum temperature F"
```

```
    }
  }
}
```

===app_graph.pl===

Example URL:

https://uspest.org/dd/app_graph.pl?
div_id=graph_content&sta=AV668&spp=cbl&cal=S1&tlow=41&thi=130&stm=3&std=15&styr=24&enm=12&end=30&fcast=6

Example Javascript Output:

```
/* Data to graph a degree days and model events.
 * Created by /usr/local/dds/bin/ddtable_json.pl by DU 2016-2022
 *
 * This data is in the public domain.
 */
// deep copy defaultsDDs or multiple graphs overwrite each other
var graph_content_opts = $.extend(true, {}, defaultsDDs);
graph_content_opts.title.text = 'cabbage looper U. Minn.  DDs(F) at AV668';
graph_content_opts.yAxis.plotLines = [{     color: '#dddddd',
    width: 1,
    value: 120,
    label: {text:'First egg laying (120)', style:{ fontSize:'90%'}}
},
.....
graph_content_opts.xAxis.plotLines = [{     color: '#dddddd',
    width: 1,
    value: Date.UTC(2024, 2, 22),
    label: {
        text:'22 Mar: First egg laying',
        rotation: -20,
        textAlign: 'left',
        verticalAlign: 'bottom',
        y: -7,
        style:{ fontSize:'90%'}
    }
},
.....
graph_content_opts.series = [{ name: 'current observed', data:          [
    [Date.UTC(2024, 2, 15), 15.8],
    [Date.UTC(2024, 2, 16), 29.8],
    [Date.UTC(2024, 2, 17), 44.3],
.....
$(document).ready(function() {
    var graph_content = new Highcharts.Chart('graph_content', graph_content_opts);
    graph_content.xAxis[0].setExtremes(Date.UTC(2024, 2, 15), Date.UTC(2024, 4, 23));
    graph_content.showResetZoom();
```

;
});

2. API for hourly-driven models

get_model_json.pl runs our hourly models (the ones used by https://uspest.org/risk/models or "MyPest Page") and returns the output to the user with metadata.

===PARAMETERS===

The common parameters are:

* outFMT
"json" or "csv"

* mdl [or model]
model name – here is a list of currently available models, documented at https://uspest.org/npdn/riskdoc.html:

- apple scab: a_scab
- botrytis: boti
- boxwood blight: bxwd_s
- cherry powdery mildew: ch_pm
- chilling units: chl
- cleistothecial powdery mildew: cl_pm
- dollar spot: dol_spot
- fire blight: fb
- fire blight new 2010: fb10
- GT powdery mildew: p_mild
- hop powdery mildew: h_pm
- mummy berry: mum_b
- muskmelon melcast: mm_mc
- pear scald: p_scld
- pear scab: p_scab
- pearson gadoury: pgpm
- strawberry powdery mildew: str_pm
- tomato potato late blight: tplb
- tomcast dsv: tdsv
- Utah chilling units: chl_u
- vapor drift: v_drift
- watermelon melcast: wm_mc

* sta [or station code]
Our station code list (over 33,000 stations) is online at: https://uspest.org/data/sta_list.txt

* span
How many days of output data to send. Default=14.

* stdt [or start_date]

The date of the first day of output date to send, in YYYYMMDD format. (Note that most models start running a few days or weeks earlier. Most of these models need some "spin-up time" before they start producing valid output.)

The default start date is chosen so that the last date is the last date for which we have a full day's NDFD forecast data, 6 days in the future.

Some models make use of other parameters:

* dollar_spot_risk
  [dollar spot model]
2 if fungicide has been used in the previous 14 days; 1 if it has not.

* lead_date
  [muskmelon and watermelon melcast]
The date from which the index starts accumulating. This is distinct from the start date, which is the date of the first line of output.

It should be noted that these models are much less homogenous than degree-day models. The risk apps, which front-ends for get_model_json.pl, all use a common codebase, but every app has to override some aspect of the default behavior. Most often this is some details of the graph, but other aspects may need special attention too.

get_model_json.pl is based on mypestpage, and as such it can take a large set of parameters. Most of them ignored.

===JSON Example===

Request:
https://uspest.org/risk/get_model_json.pl?
outFMT=json&mdl=powdery_mildew&sta=KRBG&span=30&stdt=20230322

Output:

```
{
  "data" : {
    "20230522" : {
      "powdery_mildew" : 50,
      "powdery_mildew6_consecutive_conducive_hours_seen" : 0,
      "powdery_mildewavg_daily_temp" : 58.175,
      "powdery_mildewconsecutive_conducive_days" : "0",
      "powdery_mildewconsecutive_conducive_hour_cnt" : 0,
      "powdery_mildewpoints" : 60,
      "powdery_mildewsecondary_conidial_phase" : 1,
      "powdery_mildewtemp_cum_PMI" : 50,
      "powdery_mildewvery_hot_today" : 0
    },
    "20230523" : {
      "powdery_mildew" : 40,
      "powdery_mildew6_consecutive_conducive_hours_seen" : 0,
      "powdery_mildewavg_daily_temp" : 58.475,
```

```
        "powdery_mildewconsecutive_conducive_days" : "0",
        "powdery_mildewconsecutive_conducive_hour_cnt" : 0,
        "powdery_mildewpoints" : 60,
        "powdery_mildewsecondary_conidial_phase" : 1,
        "powdery_mildewtemp_cum_PMI" : 40,
        "powdery_mildewvery_hot_today" : 0

      },
      ...
},
  "data_quality_percent" : 86,
  "model" : {
    "binary_units_of_precision" : {
      "6_consecutive_conducive_hours_seen" : 0,
      "conidial_phase" : 0,
      "consecutive_conducive_days" : 1,
      "consecutive_conducive_hour_cnt" : 0,
      "current_hour" : 0,
      "daily_rain_accum" : 3,
      "daily_temperature_cnt" : 0,
      "daily_temperature_sum" : 1,
      "index" : 1,
      "points" : 0,
      "prev_cum_index" : 1,
      "secondary_conidial_phase" : 0,
      "very_hot_today" : 0
    },
    "cum_units_label" : "% RISK",
    "current_day_max_index" : 100,
    "event_threshold_colors" : {
      "High Powdery Mildew Risk" : "FF0000",
      "Low Powdery Mildew Risk" : "008800",
      "Medium Powdery Mildew Risk" : "FFAA00"
    },
    "event_thresholds" : [
      0,
      "Low Powdery Mildew Risk",
      31,
      "Medium Powdery Mildew Risk",
      56,
      "High Powdery Mildew Risk"
    ],
    "html_table_cum_dh_label" : "GTPW<br>Cum.<br>Index",
    "html_table_risk_label" : "GT Powdery<br>Mildew Risk",
    "id" : "powdery_mildew",
    "info_html_link" : "http://uspest.org/npdn/riskdoc.html#GTPM",
    "label" : "Gubler Thomas Powdery Mildew",


    "max_cum_dh" : 100,
```

```
    "other_params_to_table" : {
      "6_consecutive_conducive_hours_seen" : "6 cons.<br>cond hrs<br>seen",
      "avg_daily_temp" : "Avg<br>Daily<br>Temp.",
      "consecutive_conducive_days" : "Conducive<br>Day Cnt",
      "consecutive_conducive_hour_cnt" : "Consecutive<br>Conducive<br>Hour<br>Cnt",
      "points" : "Points",
      "secondary_conidial_phase" : "Secondary<br>Conidial<br>Phase",
      "temp_cum_PMI" : "temp_cum<br>PMI",
      "very_hot_today" : "Very<br>Hot<br>Today"
    }
  },
  "station" : "KRBG",
  "stationValid" : 1,
  "success" : 1,
  "warnings" : []
}
```


There is a lot of extraneous stuff here. Note that the hash key for each day's risk level is the name of the model. The first instance here is:


```
      "powdery_mildew" : 50,
```


The rest of each day's hash is vestigial stuff, probably model state information, which is irrelevant to almost all users.

Similarly, much of what's under "model" is vestigial, but some parts are intended for use in a UI that presents model results. In particular: event thresholds, event_threshold_colors, info_html_link, max_cum_dh, and the keys whose names end in "label".

You should test "success" first thing. If it isn't 1, then there should be something of an explanation in "error", and the rest of the structure is probably absent or meaningless.

===CSV Example===

If outFMT is "csv", the output data starts on the 4th line with headers on the 3rd line. The first two are the input parameters --valid CSV, but with different columns. Example:

Request:
https://uspest.org/risk/get_model_json.pl?
outFMT=csv&mdl=powdery_mildew&sta=KRBG&span=30&stdt=20230322

Output:

```
"station","network","start_date","span","data quality percent"
KRBG, METAR, 20230322, 30, 82
"Date","is
forecast","powdery_mildew","powdery_mildew6_consecutive_conducive_hours_seen","powdery_
mildewavg_daily_temp","powdery_mildewconsecutive_conducive_days","powdery_mildewconsec
utive_conducive_hour_cnt","powdery_mildewpoints","powdery_mildewsecondary_conidial_phase
","powdery_mildewtemp_cum_PMI","powdery_mildewvery_hot_today"
20230322,,0,0,46.30875,0,0,0,,,0
```

20230323,,0,0,48.10625,0,0,0,,,0
20230324,,0,0,40.625,0,0,0,,,0
20230325,,0,0,38.3,0,0,0,,,0
...
20230419,,0,0,44.7145833333333,0,0,0,,,0
20230420,,0,0,44.975,0,0,0,,,0

3. Station picker

The stationpicker is meant to be analogous to a date picker. It's a couple pieces of code which, together, give you an easy(*) way to ask users to choose a station from our list. The user's choice is passed back as a stationcode, which is an identifier matching [A-Z0-9]{3,10}.

[* Easy to both the programmer and the user. That's the intent; your mileage my vary.]

The stationpicker consists of a JS file, stationpicker.js, which runs on the client and two CGI scripts, stationpicker.pl and get_stations_in_area_via_json, which the JS file queries. The JS file supplies the UI and manages interactions with the CGI APIs and with a 3rd-party mapping API (Google maps in and before April 2024) to display the maps.

stationpicker.pl receives the query text typed into the search box.
It checks to see if the search text is a stationcode, and if not it uses 3rd party "reverse geocode" lookup (OpenCage in and before April 2024) to find associated locations and stations near them.

get_stations_in_are_via_json provides station locations to populate the map. It receives a rectangle in latitude and longitude and returns a list of stations in that rectangle. If the rectangle is large, it will not return all of the stations, but only a (manageable) subset.

To use the stationpicker, your page needs:

A. To load the JS, which can be done by including one of these fragments in your page.

If your page is on uspest.org:
<script src="/js/stationpicker.js"></script>

If your page is not on uspest.org:
<script src="https://uspest.org/js/stationpicker.js"></script>

B. A div for use by the stationpicker, with an id. The div must be inside a form, also with an id and with onSubmit="return false".

C. A function for the stationpicker to use as a callback when the user chooses a station. It is called with one argument, a stationcode string. In this document, it is referred to as the "setStation" callback function.

An optional second callback function, referred to as the "finished callback function, is called when the user signals clicks the stationpicker's "OK" button.

D. Styles for a number of css classes. The styles used on uspest.org are in

/home/httpd/html/js/stationpicker.css. If your page is on uspest.org, you should include them with:

```
<link rel="stylesheet" href="/css/stationpicker.css">
```

If your page is not on uspest.org, you will probably want to use your own styles. However, you should use our styles file,

https://uspest.org/css/stationpicker.css

to see what classes the stationpicker uses as a starting point. At the very least, you will need to define styles for these classes:

warningMsg
span-expanded
span-collapsed
place_list
station-list
understated

E. A call to createStationPicker(), passing these arguments in this order:

- The id of the div described in B.

- the setStation callback function described in C.

- initial stationcode value, or ""

- the id of the form described in B.
  (optional; assumes "input-form" if omitted)

- The name of a qa_scheme. Valid choices are:
  yearToDate,sinceMar1,sinceMar1_T,120Day,60Day,30Day,14Day
  (optional; default "120Day". These are defined by the code in
  /home/wea/qa_two_week/qa_multi_avg.pl.)

- The minimum QA score a station must have to appear, as a percentage.
  (optional; default 75)

- the "finished" callback function described in C. (optional; default none)

The stationPicker object returned by createStationPicker() should be assigned to a variable. The variable name "sp" is used as a placeholder here to refer to this variable.

F. A credit, somewhere appropriate with content like this: Geo-coding (location search using place names) by OpenCage, using data © OpenStreetMap contributors.

G. Optionally, call(s) to sp.getStationName() which returns a readable a place name string if available, and/or to sp.getStationDetails(), which returns a data structure of station metadata. Both of these functions take one argument, which is a stationcode.

H. Optionally, call(s) to sp.stationFeedback(). This function takes one argument, a string, and it shows this string to the user. The intent is to supply feedback (e.g., a message that the chosen station is bad) before the user goes on to the next step.

I. Optionally, call(s) to sp.hideSearchResults() and sp.hideMap() to clean up the page when these features are not needed. These functions take no arguments. (The corresponding show* functions are available too, but mostly for internal use; I don't know why the app would ever want to call them.)

Here is an example of a complete web page which uses the stationpicker:
The comments refer to items on the list above.

```
<html>
 <head>
  <title>Stationpicker Demo</title>

  <script src="/js/stationpicker.js"></script>    <!-- A -->
  <link rel="stylesheet" href="/css/stationpicker.css"> <!-- D -->
 </head>
 <body>
  <!-- B -->
  <form id="stationPickerForm" onSubmit="return false">
   <div id="stationPicker"></div>
  </form>

  <script>
   var stationPicker;
   var station;

   function setStation( stationcode ) {         // C
    station=stationcode;
    var info = stationPicker.getStation
    var msgdiv = document.getElementById("message");
    msgdiv.innerHTML="<p>You chose station "+
      station+", "+
      stationPicker.getStationName(station)+  // G
      ", which is in zipcode "+
      stationPicker.getStationDetails(station).zipcode+
      ".</p>";
    stationPicker.stationFeedback("Picked station "+station); // H
   }
   function spFinished() {
    document.getElementById("message").innerHTML="<p>Done.</p>"
   }

  var stationPicker;
   var station;
```

```
    function setStation( stationcode ) {         // C
      station=stationcode;
      var info = stationPicker.getStation
      var msgdiv = document.getElementById("message");
      msgdiv.innerHTML="<p>You chose station "+
        station+", "+
        stationPicker.getStationName(station)+  // G
        ", which is in zipcode "+
        stationPicker.getStationDetails(station).zipcode+
        ".</p>";
      stationPicker.stationFeedback("Picked station "+station); // H
     }
    function spFinished() {
      document.getElementById("message").innerHTML="<p>Done.</p>"
    }

    stationPicker = createStationPicker(  // E
      "stationPicker",  // div id
      setStation,       // callback fn
      "",           // initial stationcode; may be empty
      "stationPickerForm",     // form id
      "30Day",          // QA scheme
      75,               // QA threshold
      spFinished
    );
   </script>
   <div id="message">
    <p>This is a placeholder.</p>
   </div>

   <!-- F -->
   <p style="font-size:80%">Geo-coding (location search using place names)
    by OpenCage, using data &copy; OpenStreetMap contributors.
   </p>
  </body>
</html>
```

Disclaimer

No claims are made as to the correctness or appropriateness of this information for your particular needs. No specific pest control products are intended for endorsement or use. All responsibility rests solely with the people who interpret and implement information from this and other sources. Use this information with caution and at your own risk - errors occur, and predictive models do not replace the need for proper monitoring in the field. If you observe conditions that differ substantially from model predictions, please contact us to determine if the model inputs were incorrect, if the model functioning or weather data are in error, or if the model is inappropriate for your conditions.


Oregon State University
Oregon IPM Center
PRISM
GROUP
Western
IPM
Center
ipmPIPE
NIFA