The Morning Update

Creating an automated daily report to viewers using internet-based data

JMP Live is a powerful new collaboration tool, but it is only as useful as the quality of the content that you provide to it. This paper discusses the development of a JMP JSL script to acquire data through the internet via a REST API. It then will then show how to publish an initial report to JMP Live, and then automatically update the data within that same report on a daily basis. In this fashion you can provide automated reporting to your viewers who just want to see the latest data when they start work in the morning.

REST API's are essentially stateless interfaces to information on web servers. They define standard operations on, and methods for passing information between, the caller and a web service. REST API's are increasingly popular way for providing a public mechanism to expose data. The World Bank, U.S. Census Bureau, Gov U.K., Google and Twitter are examples of major services using REST.

HTTP Request in JMP was developed to aid in the use of REST interfaces. It became full featured in JMP 15. Typical operations with a REST interface include a "GET" operation to ask for data, and a "POST" or "PUT" to send data. While REST defines the standard operations and protocols, the actual information that can be obtained, how to ask for it, and how it is returned are unique to the provider. Because of this, you will typically need a different script for each data provider that you access. There are certain patterns that can be commonly used, and this paper will describe one of those.

The REST API that I will use for this demonstration is the COVID-19 database from Johns Hopkins University in the U.S. It is the most often cited source of pandemic data in the U.S. and is an aggregator for world-wide statistics. It also happens to have a handy REST API.

To make a request to a site like Johns Hopkins, the basic requirement is a URL for the type of request that you are making. In some cases, like this one, the URL is customized to include the information we wish to obtain. In other cases, you can provide JMP a series of name/value pairs in an associative array and JMP will format that request for you. In this case, the basic form of the URL is:

```
https://api.covid19api.com/total/country/<COUNTRY>/status/confirmed?;
```

Where you substitute in a valid country name for <COUNTRY>. You then append the beginning and ending dates to form a URL that might look like:

```
https://api.covid19api.com/total/country/germany/status/confirmed?from=2020-08-
31T00:00:00Z&to=2020-12-21T16:14:49Z
```

The formatting of the date is important, otherwise the call will fail. It's easy to create that date string with the JMP format function:

```
todayFormatted = Format(Today(), "yyyy-mm-ddThh:mm:ss") || "Z";
```

Once you get the URL endpoint formatting the way you need to, you can call **HTTP Request** to do the work:

```
request = New HTTP Request(Url(url),Method("GET"),Secure(0));
data = request << Send();
```

In this case, as specified by the documentation for the COVID API, we are saying ignore security because this is a public API. If you are curious, you can find a link to the documentation at covid19api.com.

So, if you call the REST endpoint, what do you actually get in return? Most data providers will return data in JSON format, which typically means name value pairs that are formatting in a particular way. An example might be "Cases":"123432". You can read more about JSON at **json.org** or the W3 schools site https://www.w3schools.com/js/js_json_syntax.asp.

JMP JSL has some nice facilities to handle JSON data. You start by breaking up (parsing) the JSON data, and then you can reference the name value pairs directly. An example from the script we are developing is:

```
json_data = Parse JSON(data);
rowsjson = N Items(json_data);
firstCaseData = json_data[2]["Cases"];
```

In this case, there is header data at the first position in the JSON, so we start at the 2nd. We can ask for the number of Cases for our first true observation by directly referencing "Cases".

Now we can put this all together to create a repeatable data fetch. The COVID API has a free tier and a paid tier for access. The free tier, as you might expect, does not provide the detailed capabilities of the paid tier. For instance, you cannot get the change in cases between today and yesterday. So, we do some simple math and keep a JMP data table around with the previous values. We can use a pattern where we create a new data table with all the data up until today if we can't find a previous data table, otherwise we just get the data for today and append that to an existing table. This ends up looking like:

```
New Namespace("covid19");

covid_path = "$DOCUMENTS\";
covid_table_name = "covid19_de.jmp";
covid_report_name = "Daily Covid Case Changes - Germany";
covid_table = Convert File Path(covid_path || covid_table_name);


// If we have data already, just add a row for the latest.  If we don't, construct
the table from data for Sept. 1, 2020 on.

if (File Exists(covid_table) ,
	dt = Open(covid_table);
	bHaveData = 1;
 ,
	dt = New Table("Daily Incidents of Covid-19 - Germany");
	dt << New Column("Date", Numeric);

	dt:Date << Format("m/d/y");

	dt << New Column("Cases", Numeric);
	dt << New Column("Daily Change", Numeric);
 );
```

The **If** statement says if the table already exists in the Documents folder, then just open it and set a flag to indicate it already existed. If it does not already exist, construct it and add columns for Date, Cases and Daily Change. So, someone getting this script to use as a daily scheduled task will create the table that they need the first time the script is run, and then every day after the latest data will just be appended to the table. That is done through this block of JSL:

```
//If we have data, then just add the newest observation.  Otherwise create a new
//table and fetch all the values up to today.
if (bHaveData,
	if (rowsjson > 1,
		dt << Add Rows(1);
		wait(0);
		rowsdata = N Rows(dt);
		dt:Cases[rowsdata] = json_data[rowsjson]["Cases"];
		dateval = json_data[rowsjson]["Date"];
		dt:Date[rowsdata] = covid19:MakeDate(dateval);
		tableval = dt:Cases[rowsdata-1];
		//Subtract the new value from the previous value to get the daily amount
		dt:Daily Change[rowsdata] = json_data[rowsjson]["Cases"]-tableval;
		dt << Save();
		,
	);
	,
	dt << Add Rows(rowsjson-1);
	for (i = 2, i <= rowsjson, i++,
		dt:Cases[i-1] = json_data[i]["Cases"];
		dateval = json_data[i]["Date"];
		dt:Date[i-1] = covid19:MakeDate(dateval);
		dt:Daily Change[i-1] = json_data[i]["Cases"] - json_data[i-1]["Cases"];
	);
	dt << Save As(covid_table);

);
```

Here if there is already a table, the JSL adds a row to the table, and fills in that new row with the new total case count. Then, farther down, the previous day's case count is subtracted from this new total to create the daily amount. The MakeDate function is just a convenience to strip off the "Z" on the end of the data and format the date for JMP's usage. It looks like:

```
covid19:MakeDate = Function({date_string},
	{Default Local},

	if(Ends With(date_string, "Z"),
		date_string = Substr(date_string, 1, length(date_string) - 1);
	);
	date = Parse Date(date_string, "yyyy-mm-ddThh:mm:ss");

);
```

So now every day when we run the script, we will get an update on the data and can generate a report with the latest information. Now we just need to settle on a report and create the JSL to publish to the JMP Live server.

An overview of the JMP Live scripting API itself might be useful prior to actually showing how to put up a graphic. The JSL interface to JMP Live was redesigned in JMP 16 to make it more powerful and hopefully easier to use.

JMP Live scripting revolves around the idea of using a connection whose information has been supplied and stored previously. You do this under **File->Publish->Manage Connections…** There you supply the URL and credentials for your JMP Live site and give that connection a name. In sites that require an API Key to script the JMP Live site, you will need to supply that as well.

Once you do that one time, you can use the credentials for interactively publishing, or refer to that named connection in JSL. This looks like:

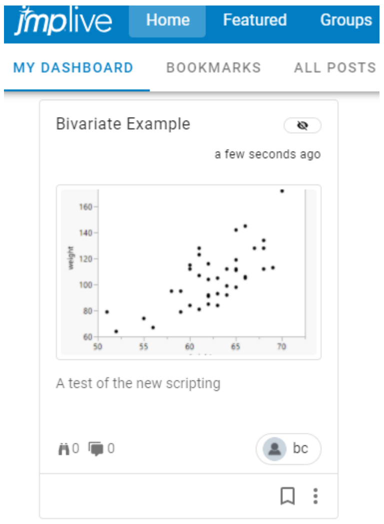
```
//Create a connection, log in and publish a report
lc = New JMP Live(Connection(“My JMP Live"), Prompt(If Needed));
```

Once you have a connection, much of the operation revolves around manipulating reports and folders, and publishing those. Reports can be created using **New Web Report**. Folders can be added using **Add Folder**, and then **Add Reports to Folder** can be used after that to put reports into the new folder. The simplest way to create a report is show below, using the common Big Class sample data and a Bivariate report.

```
dt = open("c:\program files\sas\jmp\16\samples\data\Big Class.jmp");
biv = Bivariate(:X(Height), :Y(Weight));

webreport = New Web Report();
webreport << Add Report(biv, Title("Bivariate Example"), Description("A test of the
new scripting"));

liveresult = lc << Publish(webreport);
```



The creation of the Bivariate report returns a report instance into the **biv** variable. This can then be added to the empty Web Report object using **Add Report** . The title and description are actually optional, but JMP will choose the title from content within the report if you don't supply one.

In the example above, a **JMP Live Result** is returned into the variable "liveresult". This in itself has properties that can be examined, for instance:

```
liveResult << Succeeded;  // 1 or 0
liveResult << Get HTTP Status;  // 200 if a success, 400+ for an error
liveResult << Get Error Message;
```

JMP Live is using the internet protocol HTTPS to publish your report and this returns results that are saved into the "liveresult" variable. If there is an error, the **Get Error Message** will return the HTTP error message from the attempted publication of the report.

Depending on the type of operation that was performed, the JMP Live Result may contain information on a report, folder or some other identification to what the action produced. If a report or folder was produced as a result of the operation, you can obtain a reference to this by using **As Scriptable** as in the following:

```
livereport = liveresult << As Scriptable;
```

Now, you can reference "livereport" as a JMP Live Report, with all of the capabilities that are shown in the JMP Scripting Index. The Scripting Index will also tell you what each operation on a JMP Live scripted object, like a Live Connection or Live Report, will return.

In the case above with "livereport", we could now change the title in JSL if we wanted to by doing:

```
livereport << Set Title("My New Report Title");
```

One of the most important operations that a Live Report or Live Folder provides is **Get ID()**. This returns the unique identifier, which is a long string of letters and numbers, for that report or folder. This ID is required for things like deleting the report. It is impractical to know or type this ID into your JSL, but having the report provide the ID allows you to do something like:

```
rc = lc << Delete Report(report << Get ID());
```

Where this becomes very important is when you start using search operations that obtain reports where you have no knowledge of the ID. If I wanted to delete all my Bivariate posts that had "Biv" in the title I could use **Find Reports** to return a list of those, and then delete them.

```
liveresult = lc << Find Reports(Search("Biv"));
livereports = liveresult << As Scriptable;

for (i = 1, i < livereports << Get Number of Items(), i++,
	reporti = livereports[i];
	lc << Delete Report(reporti << Get ID());
)
```

Find Reports returns a JMP Live Result List, which has a function called **Get Number of Items** that gives us the size of the list. We can then use that to iterate through the reports.

The final part of JMP Live scripting that we will need is to update the data for a report that has already been published. When you have a report that you are satisfied with, you may just want to supply updated data and have it recalculated. **Update Data** provides the means to do this. A data table variable, and the package ID of the report that you are updating are required. There is an option to change the name of the data table when uploaded to the JMP Live website, if you need to match the name of a table within the report and that name differs from the table you are supplying. The general syntax looks like:

```
updateresult = lc << Update Data(livereport << Get ID(), Data(dt));
```

Now we have the tools necessary to create an easily updated report.

Something like a control chart is not really an appropriate graph for this type of data, but I'm going to use it anyway. Why? Because it allows me to also show the new alarm feature in JMP Live. Let's pretend for the sake of the demo that I am a public official wanting to keep an eye on the data to see when there are concerning increases. I want to get a warning when the number of new cases goes "out of control" in the control chart.

I can create the chart in JMP and set that I warnings enabled for "Test Beyond Limits". Then I can save the script for the graph to a script window and use that JSL in my greater script. That section of JSL looks like:

```
ccb = Control Chart Builder(
	Size( 534, 456 ),
	Show Control Panel( 0 ),
	Show Capability( 0 ),
	Variables( Subgroup( :Date ), Y( :Daily Change ) ),
	Chart( Position( 1 ), Warnings( Test Beyond Limits( 1 ) ) ),
	Chart( Position( 2 ), Warnings( Test Beyond Limits( 1 ) ) )
);
```

Now it is possible to use a similar pattern from before to publish to a JMP Live website. We will standardize the report to always use the same name. In this way, we can use the JMP Live search capabilities to see if there is already a report published with the standardized name. If there is, we will just replace the data for the report and JMP Live will recalculate the graph. If no graph is found, we will publish the control chart for the first time.

First, we need to provide our standard name for the report, and then we can publish either just the data, or the full report. **Find Reports** in the JMP Live JSL support will return a list. We can look at the number of items in the list to see if the report already exists. If the list size is 0, then we know we have not previously published, and we can use the **Publish** message on the JMP Live Connection object to put the report on the website. If the report already exists, then we can take the data table that we just updated with the latest data and send that to the existing report. The JMP Live web server will replace the data and generate a new report within the existing JMP Live package to show the updated results. This will allow us to keep the report that was set up with the appearance that we liked, changing it just to show the new value. The script below writes some debugging information to the log window to show us if everything went well with the publication, but that is not a necessary part of the process.

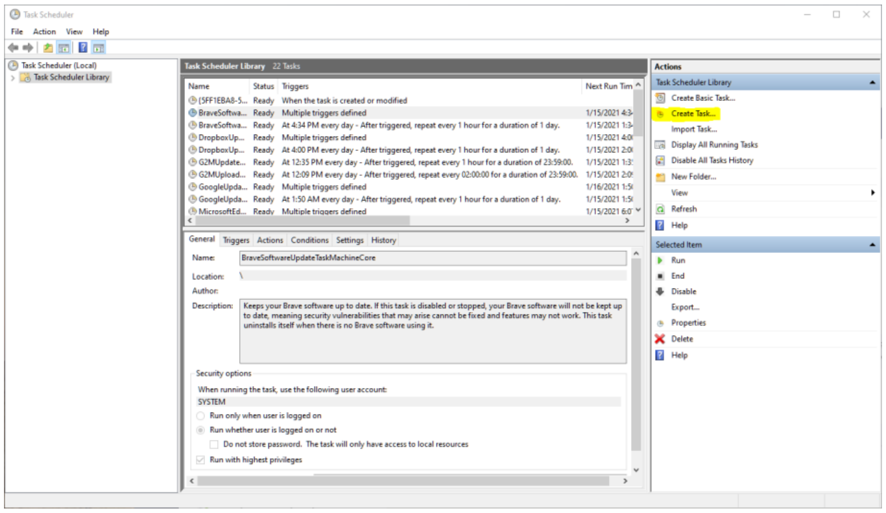
```
//See if there was already a report published.  If so, just update the underlying
//data. If there was not, create a report and publish it to JMP Live.

covid_report_name = "Daily Covid Case Changes - Germany";
//Search on report name and also only those published by me, “bc”
searchresult = lc << Find Reports(Search(covid_report_name), Publisher("bc"));
reportlist = searchresult << As Scriptable();
nreports = reportlist << Get Number Of Items();
show(nreports);

if (nreports == 0,
	ccbreport = New Web Report();
	ccbreport << Add Report(ccb, Title(covid_report_name), Description("Individual
and Moving Range Chart of Daily Covid Cases in Germany"));
	lc << Publish(ccbreport, Public(1));
	,
	ccbreport = reportlist[1];
	updateresult = lc << Update Data((ccbreport << Get ID()), Data(dt,
covid_table_name));
	show(updateresult << Succeeded);
	show(updateresult << Get HTTP Status);
	show(updateresult << Get Error Message);
	show(updateresult << Get Response);
	show(updateresult << Get Response Type);
);
```

This script is designed to be run once a day, and only once a day. It does not look to fill in days that are missing or do avoid trying to add a day if the same day already exists. In short, it is designed to be run as a scheduled task once per day. There a variety of products to help with this, or you can run the Task Scheduler on Windows, or Automator or cron on the Mac. I’ve shown how to do this in previous Discovery papers, but I’ll show how to run this particular job on the Windows Task Scheduler. You can bring up Task Scheduler by just typing “Task Scheduler” in the Windows search bar.

In the **Actions** pane of the Task Scheduler, select Create Task…



Task Scheduler Library 22 Tasks

Name	Status	Triggers	Next Run Tim
{5FF1EBA8-5...	Ready	When the task is created or modified	
BraveSoftwa...	Ready	Multiple triggers defined	1/15/2021 4:3
BraveSoftwa...	Ready	At 4:34 PM every day - After triggered, repeat every 1 hour for a duration of 1 day.	1/15/2021 1:3
DropboxUp...	Ready	Multiple triggers defined	1/15/2021 4:0
DropboxUp...	Ready	At 4:00 PM every day - After triggered, repeat every 1 hour for a duration of 1 day.	1/15/2021 2:0
G2MUpdate...	Ready	At 12:35 PM every day - After triggered, repeat every 1 hour for a duration of 23:59:00.	1/15/2021 1:3
G2MUpload...	Ready	At 12:09 PM every day - After triggered, repeat every 02:00:00 for a duration of 23:59:00.	1/15/2021 2:0
GoogleUpda...	Ready	Multiple triggers defined	1/16/2021 1:5
GoogleUpda...	Ready	At 1:50 AM every day - After triggered, repeat every 1 hour for a duration of 1 day.	1/15/2021 1:5
MicrosoftEd...	Ready	Multiple triggers defined	1/15/2021 6:0

Name: BraveSoftwareUpdateTaskMachineCore

Location: \

Author:

Description: Keeps your Brave software up to date. If this task is disabled or stopped, your Brave software will not be kept up to date, meaning security vulnerabilities that may arise cannot be fixed and features may not work. This task uninstalls itself when there is no Brave software using it.

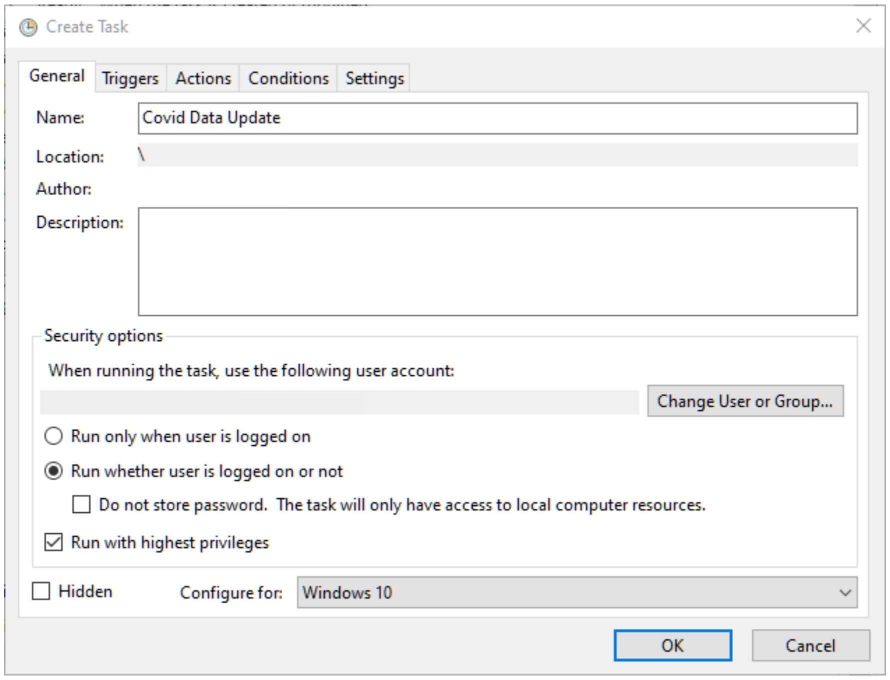
Security options

When running the task, use the following user account:

SYSTEM

- ○ Run only when user is logged on
- ● Run whether user is logged on or not
  - ☐ Do not store password. The task will only have access to local resources
- ☑ Run with highest privileges

Assign a name that you will remember, like Covid Data Update. If you want the task to run even if you are not logged in, which I usually do, then select "Run whether the user is logged on or not". Also select "Run with highest privileges". When you finish the other information for the task, you will be prompted for your Windows credentials for the cases when you are not logged in. You will need to remember to change these credentials whenever you change your password.



Name: Covid Data Update

Location: \

Author:

Description:

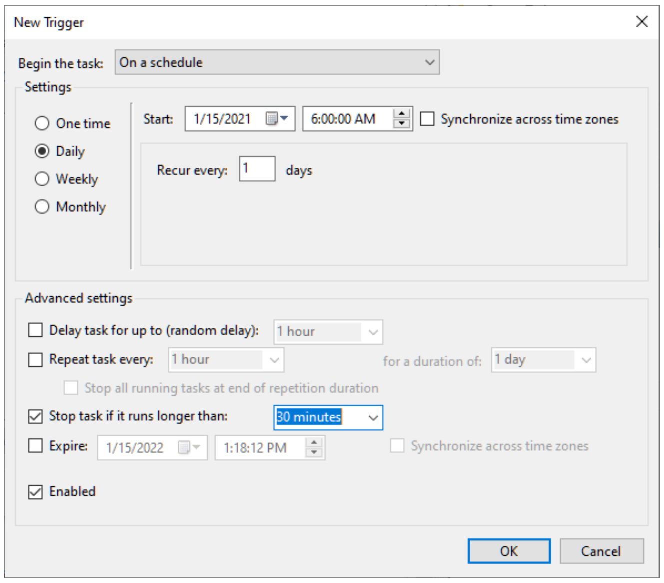
Security options

When running the task, use the following user account:

- ○ Run only when user is logged on
- ● Run whether user is logged on or not
  - ☐ Do not store password. The task will only have access to local computer resources.
- ☑ Run with highest privileges

☐ Hidden Configure for: Windows 10

Next select the "Triggers" pane. This is where you specify when and how often you want the task to run. I'll select the date when I want to start the task, that I don't want it to end, and that I want it to run Daily. I also like selecting the box "Stop task if it runs longer than:" and selecting 30 minutes. In this case, I've told the scheduler to run at 6:00 in the morning every day.


New Trigger

Begin the task: On a schedule

Settings

( ) One time
(●) Daily
( ) Weekly
( ) Monthly

Start: 1/15/2021 6:00:00 AM ☐ Synchronize across time zones

Recur every: 1 days

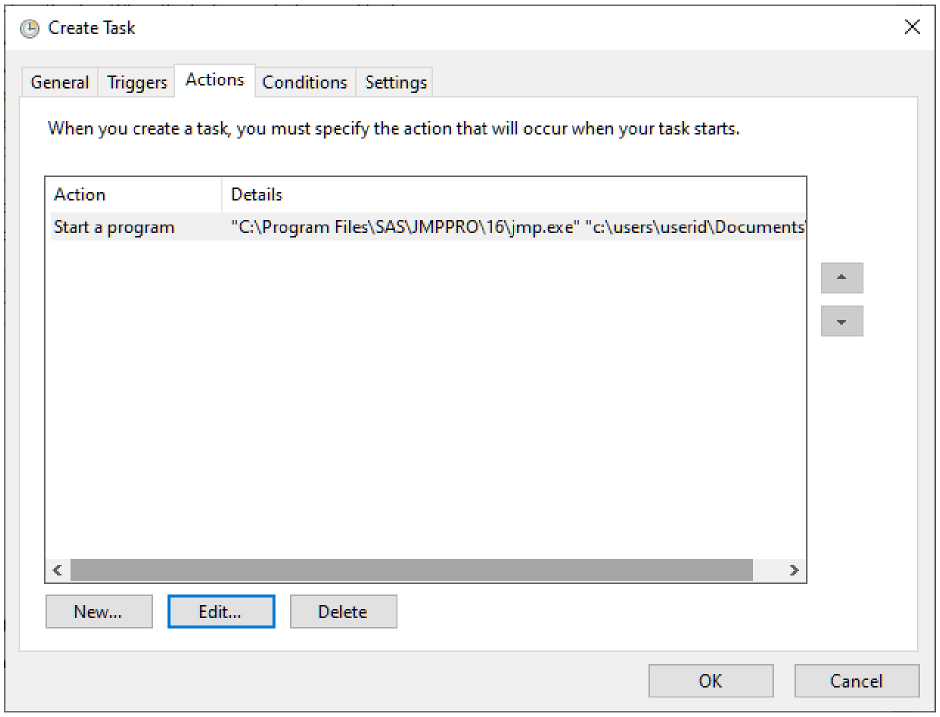
Advanced settings

☐ Delay task for up to (random delay): 1 hour
☐ Repeat task every: 1 hour for a duration of: 1 day
☐ Stop all running tasks at end of repetition duration
☑ Stop task if it runs longer than: 30 minutes
☐ Expire: 1/15/2022 1:18:12 PM ☐ Synchronize across time zones

☑ Enabled

OK Cancel

Next select the "Actions" pane. This is where you specify the application (JMP) and task that you want to perform (run the JSL script). Select the "New…" button. For Action, select "Start a program". For the Program/script, you will need to use the "Browse…" button to navigate to the location of **jmp.exe**, which for JMP 16 would be in "c:\program files\sas\jmp\16\jmp.exe" . For "Add Arguments (optional):" you will need to manually type in the location of the JSL file that you want to launch. Now hit OK. The result will look something like this:


Create Task

General | Triggers | Actions | Conditions | Settings

When you create a task, you must specify the action that will occur when your task starts.

Action	Details
Start a program	"C:\Program Files\SAS\JMPPRO\16\jmp.exe" "c:\users\userid\Documents

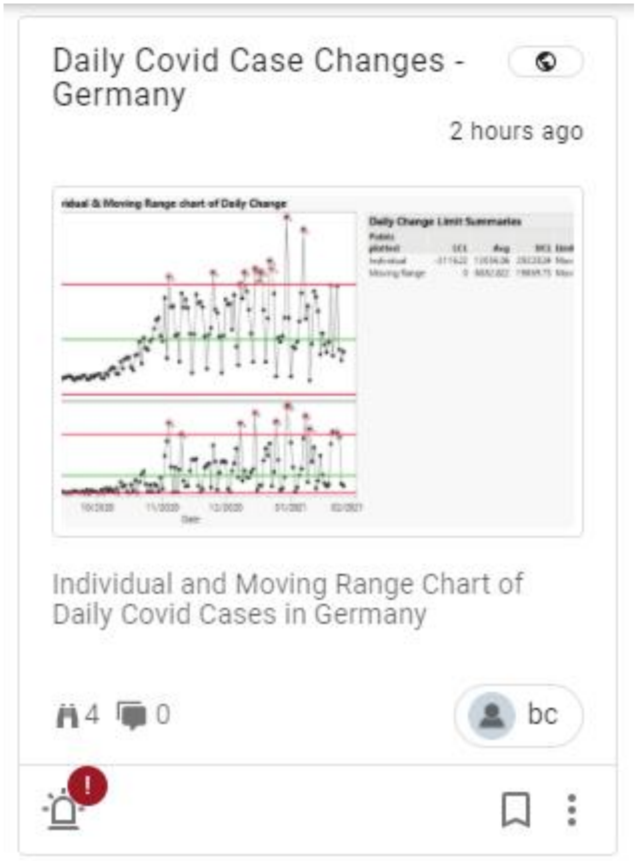
New… Edit… Delete

OK Cancel

Panel after adding the Action

The only other settings that I typically change are under the Settings tab. I usually make sure "Allow task to be run on demand is set". This allows me to test out the task at any time by using the context (right mouse) menu in the Task Scheduler Library listing. I also select “Stop the task if it runs longer than: 1 hour”.

Now the report should be generated the first day and updated every day thereafter.


Daily Covid Case Changes - Germany

2 hours ago

idual & Moving Range chart of Daily Change
Daily Change Limit Summaries
Date

Individual and Moving Range Chart of Daily Covid Cases in Germany

The types of patterns used in the report can now be used for any type of REST based data, and for any type of JMP analysis.