Automatic Scalar Script Builder

Peter Mason, January 2019

Introduction

TSG's batch-script-scalar functionality is powerful in the right hands. If you know your way around TSG's scalar script language then you can build scalars with complex rules and built-in thresholds.[1]

Many TSG users do not and *will* not write scalar scripts, but they will explore ideas by creating interactive TSG scalars. Although non-trivial spectral processing 'algorithms' can be developed like this, there are impediments to applying these algorithms in a production workflow[2].

TSG can now generate scalar scripts for several kinds of interactive scalars. The tool understands complex scalars that are calculated from other scalars, not just standalone scalars. My hope is that TSG users will be encouraged to explore more complex methods with chained interactive scalars, knowing that their final result scalar[3] can be scripted automatically – if they follow some guidelines. I hold that *this is the best way to design a script anyway – first prototype the method with interactive TSG scalars*[4]. That way you can *see* what each component is doing and get the thresholds and other settings right.

This document

TSG's new tool for generating scalar scripts will be presented first.

Then we'll go through a medium-complexity MFEM[5] scalar script. We'll replicate its functionality by creating a little squad of interactive TSG scalars. (Each of these scalars can be viewed and tweaked in TSG. As noted above this is an important step when you design your own scalar methods.) Then we'll have TSG build a script for the method's final scalar.

[1] For example you could build a 'mineral presence' mask scalar that seeks 4 absorption features of which 3 must be present and one must not, and has acceptance thresholds for each feature's depth and wavelength. Given a spectrum, this mask scalar would deliver a result 'yes' or 'no'. The script, complete in a small text file, could easily be given to colleagues, and easily be used by them.

[2] TSG's 'copy processing' can do it, but it leaves dataset recipients with a confusing and fragile assortment of scalars. In contrast a scripted algorithm has its workings in a tidy capsule (a small text file) and yields just one scalar.

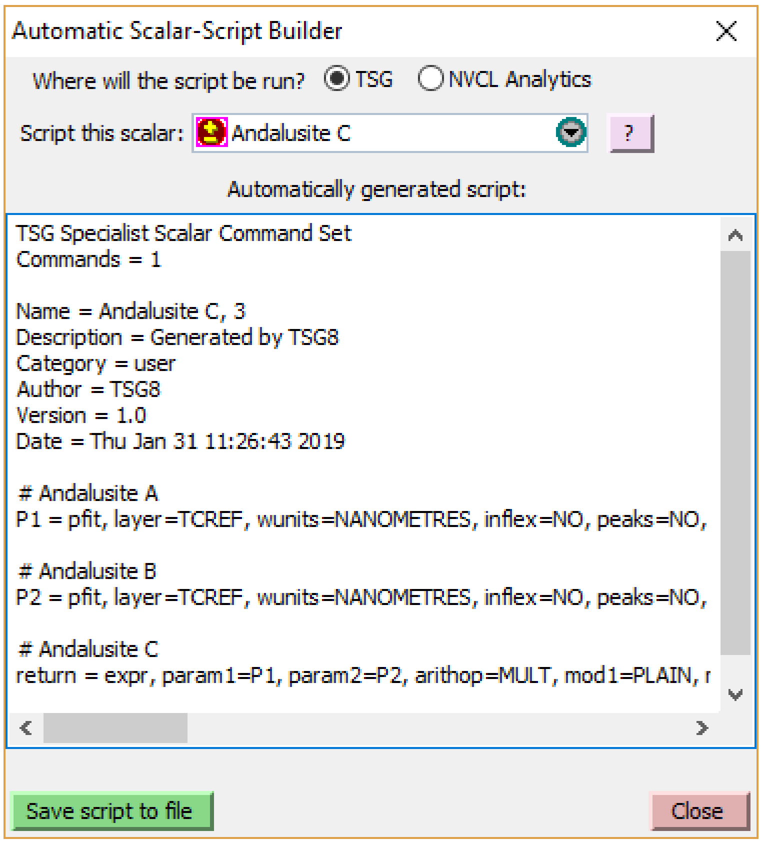
[3] Along with all the others that it is built from

[4] It can be very hard to design a script 'cold' (starting with the script text file) when all you have to judge is the script's final result.

[5] CSIRO researcher Carsten Laukamp is the custodian of the 'MFEM script library'. MFEM stands for 'Multiple Feature Extraction Method'.

TSG8's Scalar Script builder

From the menu: **Edit -> Scalar script builder…**



Automatic Scalar-Script Builder

Where will the script be run? (● TSG) ○ NVCL Analytics

Script this scalar: Andalusite C

Automatically generated script:

```
TSG Specialist Scalar Command Set
Commands = 1

Name = Andalusite C, 3
Description = Generated by TSG8
Category = user
Author = TSG8
Version = 1.0
Date = Thu Jan 31 11:26:43 2019

# Andalusite A
P1 = pfit, layer=TCREF, wunits=NANOMETRES, inflex=NO, peaks=NO,

# Andalusite B
P2 = pfit, layer=TCREF, wunits=NANOMETRES, inflex=NO, peaks=NO,

# Andalusite C
return = expr, param1=P1, param2=P2, arithop=MULT, mod1=PLAIN, r
```

Select a scalar from the list and TSG will give you a script. Edit the script if you like (e.g., the 'Description' field) and save it to disk. That's basically it. Here comes the long version…

TSG or NVCL Analytics?

What's all this? NVCL analytics? Several TSG customers work with HyLogger data and the AuScope NVCL (National Virtual Core Library). 'Analytics' is an NVCL activity that's in development. It allows a TSG scalar script to be run against drillhole spectra in the NVCL database. The main point here is that the NVCL script engine does not recognise the full set of methods that the TSG engine does.
If you are preparing a script for NVCL Analytics then select its radio button so that TSG only shows you compatible scalars in the list.

Supported script methods[6]

Method	Script file	Interactive TSG scalar	NVCL Analytics
Profile	Yes	Yes	Yes
PFit	Yes	Yes	Yes
FeatEx	Yes	Yes	Yes
Arith. Expression	Yes	Yes	Yes
AuxMatch	Yes	Yes	No (but coming)
Import	Yes	Yes	No
Ratio	Yes	No	Yes
Pseudocolour	Yes	No	Yes

So in short:
Do not use Import or Aux-match scalar types when preparing a method for NVCL analytics.

[6] I include a "script file" column here for interest. It shows the methods that are available when you create scripts the old fashioned way, purely with a text editor. As you can see there are no interactive-TSG-scalar counterparts (at this time) for the Ratio and Pseudocolour methods. In other words the automatic scalar script builder will not use these methods in its scripts.

Scalar selection list

It's a normal scalar selection list, like the ones in TSG's scatterplot screen. If a primary + associated dataset pair is open (normally VNIRSWIR + TIR) then it allows access to both datasets.
Importantly, the list only shows scalars that are judged to be scriptable. Therefore it can be affected by the **TSG** / **NVCL** radio-button selection.
Note that even when the **TSG** radio button is selected, the list will not show any Import-type scalars. Who would want to 'calculate' one? TSG won't clutter the list with the things. However TSG will, for example, allow you to build a script for an arithmetic-expression scalar that uses an import-type scalar as an input.

Which scalar should you select?

The system is designed to deal with scalar dependencies. If you have worked up an idea that involves more than one interactive TSG scalar then you should select your final scalar. (This would normally be an arithmetic-expression scalar that 'brings it all together'.) TSG will track down all the scalars that it depends on, directly or indirectly, and incorporate them in the script.

Un-scriptable TSG scalars (why isn't 'my scalar' on the list?)

Not all interactive TSG scalars can be scripted. These kinds are unsupported:

- Smooth
- Batch
- Core
- Class-extraction
- Stats
- PLS

In addition, import and aux-match scalars can't be scripted if **NVCL** is selected.
If 'your scalar' depends on any unsupported scalars, directly or indirectly, then TSG won't show it on the list.

The text window (and things you might want to edit there)

As soon as you select a scalar, TSG will generate a script for it and display the script in the main text window. You can edit in this window.
Now you wouldn't want to be changing any of the calculation-related fields here[7]. TSG lets you edit the script so that you can personalise it and / or make it more accessible to others by changing 'metadata' fields and adding comments. If you look at the example screengrab on page 2 you will see that TSG doesn't give you a bare-bones script (unless you select NVCL Analytics). It includes metadata fields **Description**, **Author** and **Version** with placeholder values, and it inserts a comment line before each script method. (A comment line starts with the **#** character.)

- **Description**: It behoves you describe your method here. You have a single line of up to 255 characters to accomplish it. If you need more then use comment lines.
- **Author**: It's your method – put your name here.

[7] Rather do that at the source – modify the scalars themselves in TSG, then come back and generate the script again. Keeping things coherent like this can prevent confusion at some later date.

- **Version**: Version numbers can be handy when a method evolves. Adjust TSG's default **1.0** as necessary.
- **Name**: TSG names the script after the selected scalar. You can change the name here but be careful – don't change the comma or the number after it.
- **Comment lines**: As noted these start with the **#** character. TSG comments each method with the name of the interactive scalar that it represents. You might want to change these comments or add some of your own. You can add as many comment lines as you like, anywhere in the script.

Save script to file

When you click this button, TSG will parse the script to check that you didn't break anything with your edits, then prompt you for a filename and save the script file to disk.

A worked example

We'll take a look at the '**kaolin composition**' script from Carsten Laukamp's MFEM script collection. We'll set about reproducing its functionality with interactive TSG scalars. (Hopefully, seeing this process laid out will encourage TSG users to develop more complex methods themselves, using interactive scalars.) Finally we'll ask TSG's script builder to generate a script from our 'final' scalar.

About the script

'Kaolin composition' is a medium-complexity script with 23 methods. Now 23 methods sounds like quite a lot but I call it 'medium complexity' because it really has just three components: two masks and the actual composition result. Let's take a look at the script and then I'll discuss its implementation some more.

```
name = Kaolin composition, 23

#'is it kaolin' mask
P1 = profile, stat=mean, wcentre=2138, wradius=0, layer=ref
P2 = profile, stat=mean, wcentre=2190, wradius=0, layer=ref
P3 = profile, stat=mean, wcentre=2156, wradius=0, layer=ref
P4 = profile, stat=mean, wcentre=2179, wradius=0, layer=ref
P5 = expr, param1=P1,param2=p2,arithop=add
P6 = expr, param1=P3,param2=p4,arithop=add
p7 = expr, param1=p5,param2=p6,arithop=div
p8 = expr, param1=p7, const2=1.005, arithop= lgt, nullhandling=out

#minimum threshold on 2200D (Al-clay mask)
p9 = profile, layer=ref, stat=depth, bkrem=div, fit=3, wcentre=2183,
wradius=63
p10 = expr, param1=p9, const2=0.1, arithop=lgt, nullhandling=out

#combine the two masks (both must be on)
p11= expr, param1=p8, param2=p10, arithop=mult

# kaolin composition index
P12 = profile, stat=mean, wcentre=2138, wradius=0, layer=ref
P13 = profile, stat=mean, wcentre=2173, wradius=0, layer=ref
p14 = expr, param1=p12, param2=p13, arithop=add
P15 = profile, stat=mean, wcentre=2156, wradius=0, layer=ref
p16 = expr, param1=p14, param2=p15, arithop=div
P17 = profile, stat=mean, wcentre=2155, wradius=0, layer=ref
P18 = profile, stat=mean, wcentre=2190, wradius=0, layer=ref
P19 = expr, param1=p17, param2=p18, arithop=add
P20 = profile, stat=mean, wcentre=2173, wradius=0, layer=ref
p21 = expr, param1=p19, param2=p20, arithop=div
p22 = expr, param1=p16, param2=p21, arithop=div

#return the masked composition index
return= expr, param1=p11, param2=p22, arithop=mult
```

'Is it kaolin' mask

The first part of the script builds a mask that checks if the spectrum shows kaolin. This check is based on the MFEM ratio:

```
[ref(2138) + ref(2190)] / [ref(2156) + ref(2179)]
```

(where, for example, **`ref(2138)`** means 'the reflectance at 2138nm').
This ratio is put together by script methods **`P1`** to **`P7`**.

This ratio must be 1.005 or more for the spectrum to be deemed to show kaolin. This check is done by method `P8`, which finishes the yes / no kaolin mask.

'Al-clay mask' mask

The SWIR spectrum of an aluminium clay is expected to have a noticeable 2200nm absorption. Method `P9` calculates the relative absorption depth of the 2200nm absorption, and `P10` checks if it is greater than the acceptance threshold of 0.1 (no if not).

Combined mask

Method `P11` combines our two masks. It returns yes (1) if the spectrum shows kaolin presence *and* a reasonable 2200nm absorption, otherwise no (NULL).

Kaolin composition index

Methods P12 to 22 calculate the MFEM kaolin composition index, which is a ratio of two ratios:

- A = `[ref(2138) + ref(2173)] / ref(2156)` (methods `P12` to `P16`)
- B = `[ref(2155) + ref(2190)] / ref(2173)` (methods `P17` to `P21`)
- Composition index = `A / B` (method `P22`)

Final result

Method `P23` masks the kaolin composition in `P22` with the combined mask in `P11`, returning the composition index only if the spectrum shows kaolin and a reasonable 2200nm absorption (otherwise NULL).

Simulating the script with TSG scalars

Now we'll set about making TSG scalars for each part of the script. We'll take a few short-cuts as a few of the `Ref(x)` methods are repeatedly extracted in the script. There are a couple of other short-cuts we could take, but we won't.

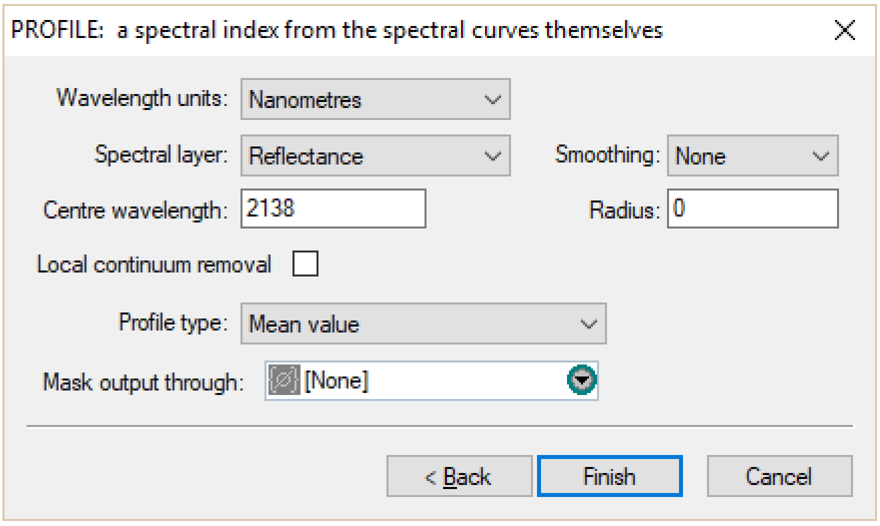
Ideally we'd create these scalars in a dataset that has a variety of kaolin spectra along with other spectra that have some similarities to kaolins (but aren't kaolins). That way we'd be able to evaluate our work.

Once again the spruik: A big motive here is to encourage you to develop non-trivial spectral techniques with interactive TSG scalars. Don't be afraid to create a whole *raft* of experimental scalars. It is best to be able to *see* how each and every bit of your method is performing if you are to judge it effectively. Ultimately you will be using a scalar script when the time comes for processing, and it will generate just one tidy scalar.

‘Is it kaolin’ mask

We’ll need to create four profile scalars and four arithmetic-expression scalars. Let’s go.

Profile scalar for Ref(2138)



PROFILE: a spectral index from the spectral curves themselves

Wavelength units: Nanometres

Spectral layer: Reflectance

Smoothing: None

Centre wavelength: 2138

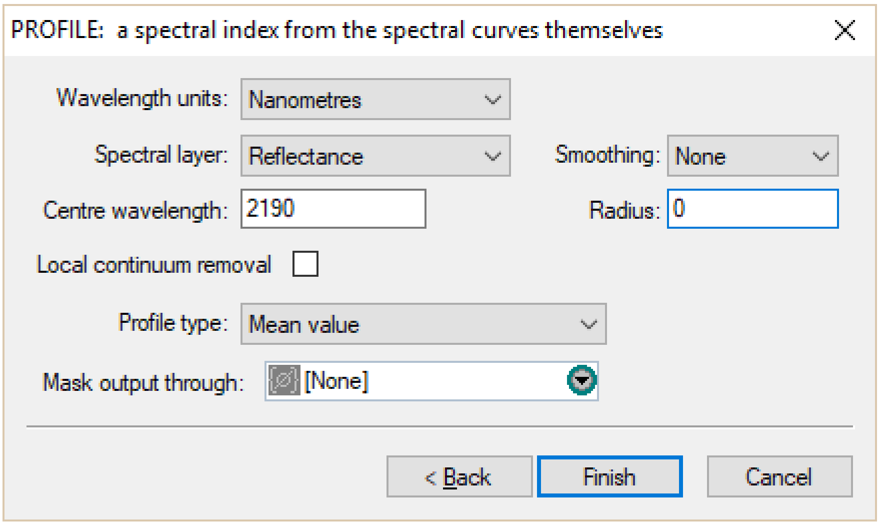
Radius: 0

Local continuum removal ☐

Profile type: Mean value

Mask output through: [None]

Profile scalar for Ref(2190)



PROFILE: a spectral index from the spectral curves themselves

Wavelength units: Nanometres

Spectral layer: Reflectance

Smoothing: None

Centre wavelength: 2190

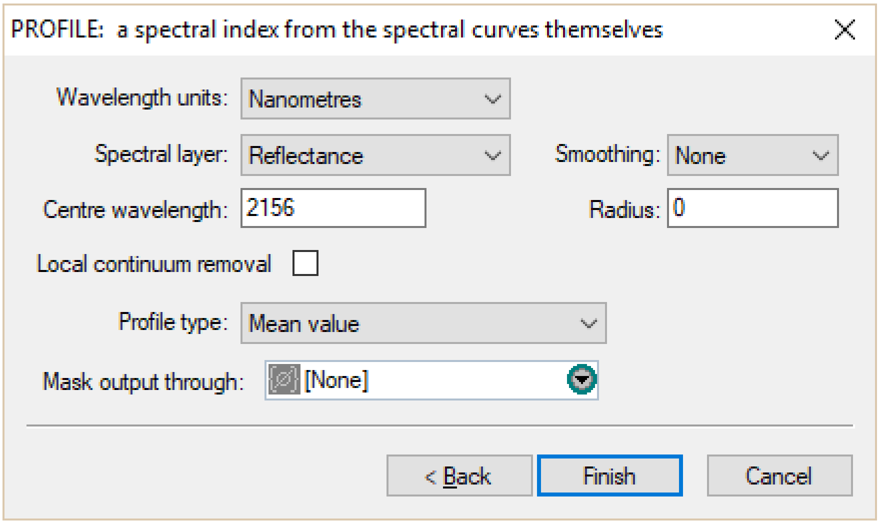
Radius: 0

Local continuum removal ☐

Profile type: Mean value

Mask output through: [None]

Profile scalar for Ref(2156)



PROFILE: a spectral index from the spectral curves themselves

Wavelength units: Nanometres

Spectral layer: Reflectance

Smoothing: None

Centre wavelength: 2156

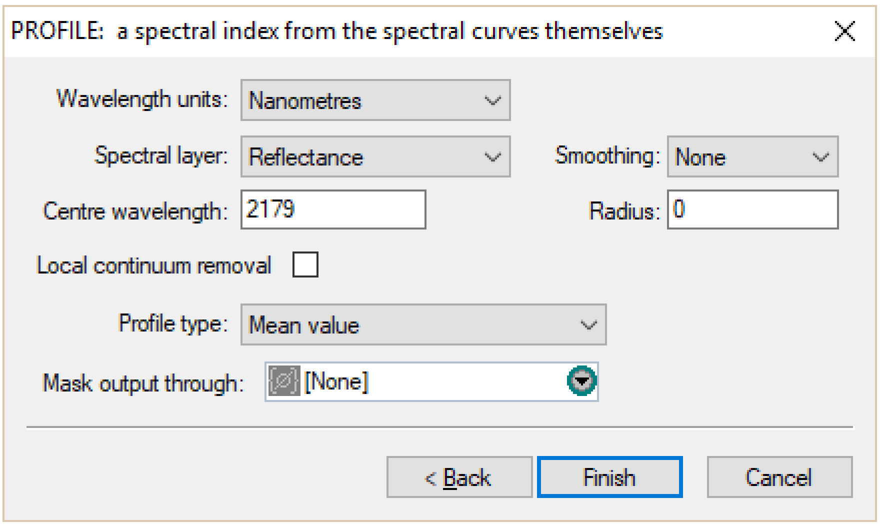
Radius: 0

Local continuum removal ☐

Profile type: Mean value

Mask output through: [None]

Profile scalar for Ref(2179)


PROFILE: a spectral index from the spectral curves themselves

Wavelength units: Nanometres

Spectral layer: Reflectance   Smoothing: None

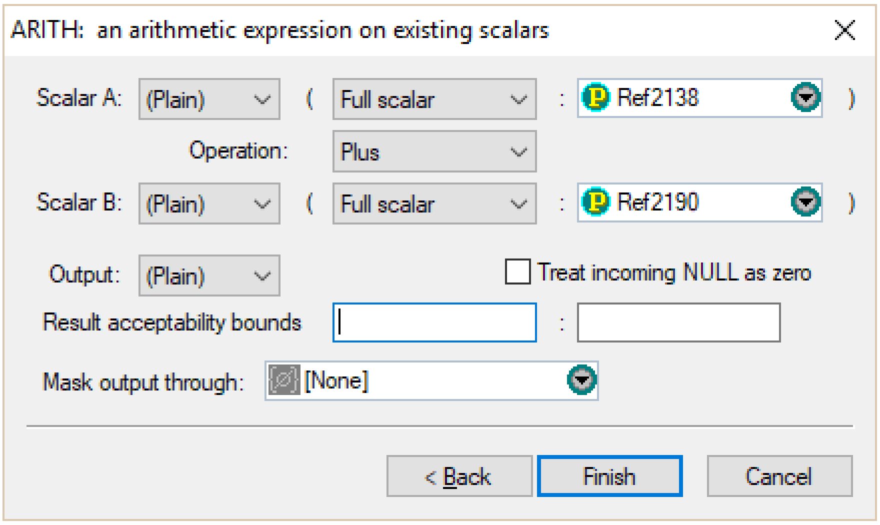
Centre wavelength: 2179   Radius: 0

Local continuum removal ☐

Profile type: Mean value

Mask output through: [None]

Arithmetic scalar 'num1', which is Ref2138 + Ref2190


ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : Ref2138)

Operation: Plus

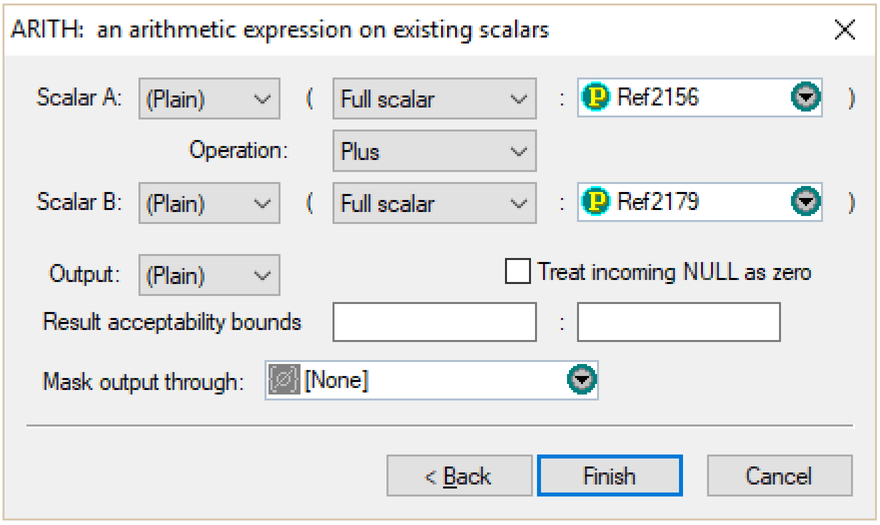
Scalar B: (Plain) (Full scalar : Ref2190)

Output: (Plain)   ☐ Treat incoming NULL as zero

Result acceptability bounds :

Mask output through: [None]

Arithmetic scalar 'denom1', which is Ref2156 + Ref2179


ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : Ref2156)

Operation: Plus

Scalar B: (Plain) (Full scalar : Ref2179)

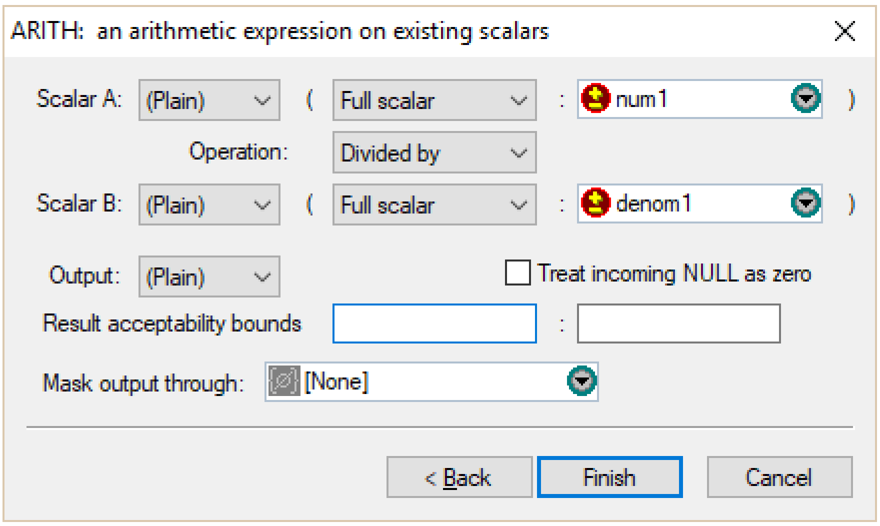
Output: (Plain)   ☐ Treat incoming NULL as zero

Result acceptability bounds :

Mask output through: [None]

Arithmetic scalar 'kaolin_ratio' which is num1 / denom1.

It is tempting to do the masking right here by putting 1.005 in the higher result-acceptability bound but that would inhibit interactive exploration.


ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : num1)

Operation: Divided by

Scalar B: (Plain) (Full scalar : denom1)

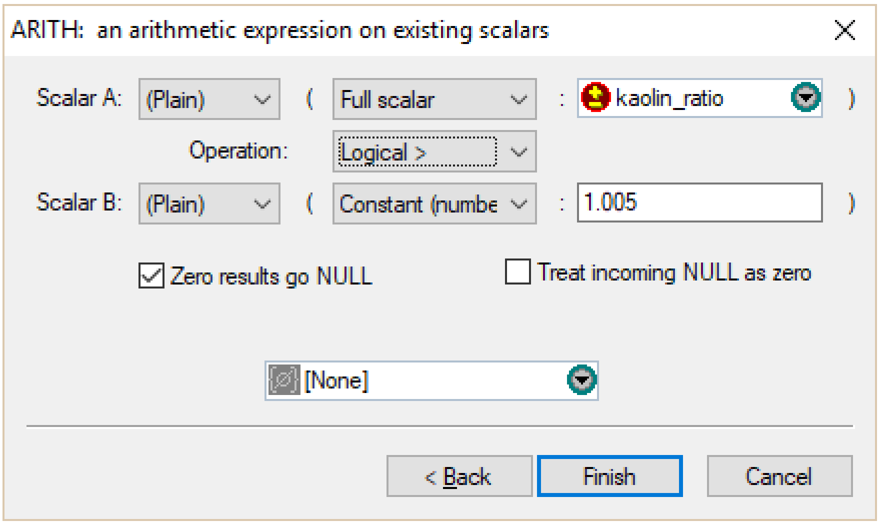
Output: (Plain)

☐ Treat incoming NULL as zero

Result acceptability bounds :

Mask output through: [None]

Arithmetic scalar 'kaolin_mask' which is or first mask. It returns 1 if the spectrum shows kaolin and NULL if not.


ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : kaolin_ratio)

Operation: Logical >

Scalar B: (Plain) (Constant (numbe : 1.005)

☑ Zero results go NULL

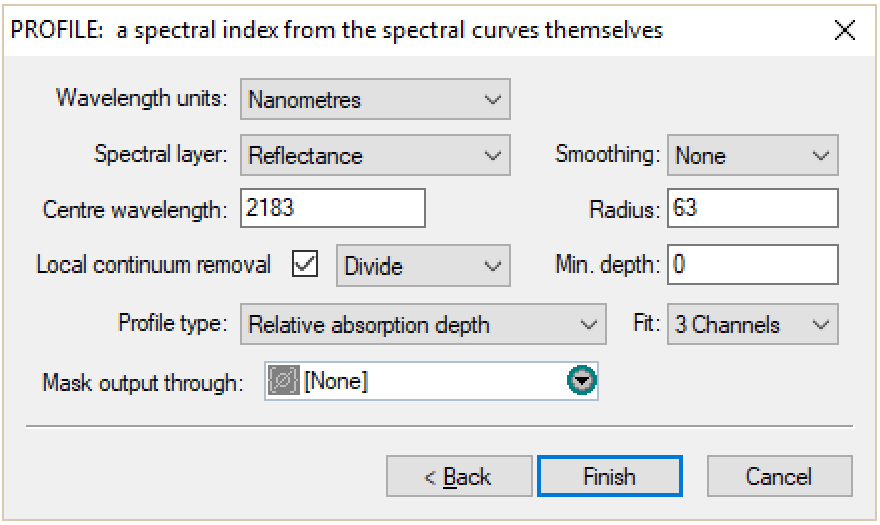
☐ Treat incoming NULL as zero

[None]

'Al-clay' mask

For this we'll need a profile scalar and an arithmetic-expression scalar.

Profile scalar 'D2200', which gives the relative absorption depth of the dominant feature around 2183nm +- 63nm.


PROFILE: a spectral index from the spectral curves themselves

Wavelength units: Nanometres

Spectral layer: Reflectance

Smoothing: None

Centre wavelength: 2183

Radius: 63

Local continuum removal ☑ Divide

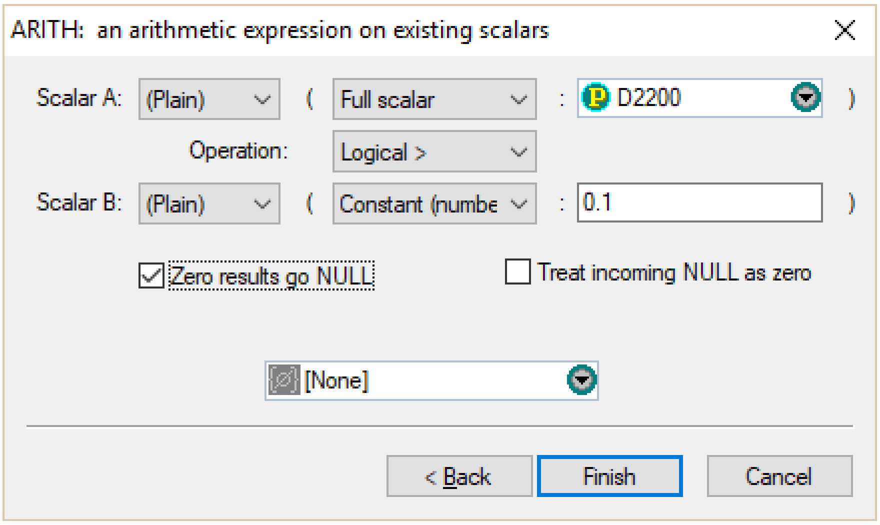
Min. depth: 0

Profile type: Relative absorption depth

Fit: 3 Channels

Mask output through: [None]

Arithmetic scalar 'Al_Clay_Mask', which is our second mask. It returns 1 if D2200 is greater than 0.1, otherwise 0.



ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : D2200)

Operation: Logical >

Scalar B: (Plain) (Constant (numbe : 0.1)

[✓] Zero results go NULL

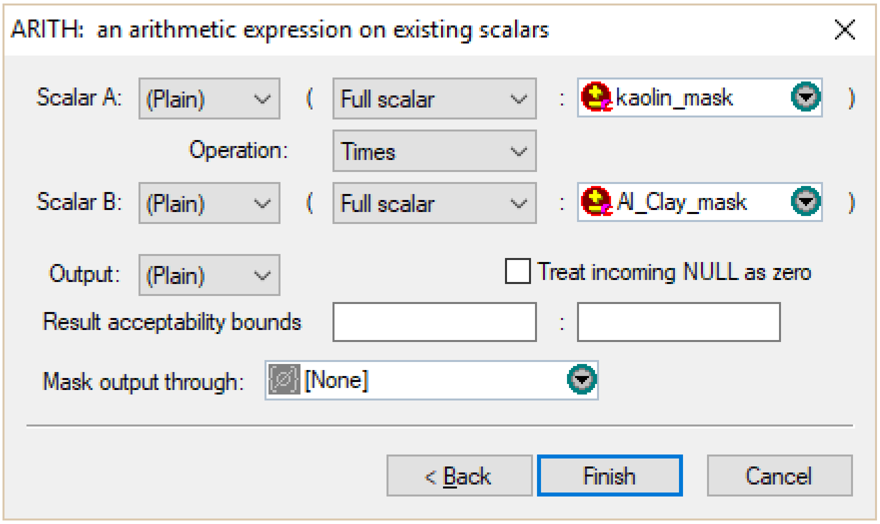
[ ] Treat incoming NULL as zero

[None]

Combined mask

Now we'll combine the two masks that we've made. Both must be 1 for the combined mask to be 1.

Arithmetic scalar 'Combined_mask', which is our final mask in the method.
Note we could have used the operation 'Boolean AND' and received the same result.



ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : kaolin_mask)

Operation: Times

Scalar B: (Plain) (Full scalar : Al_Clay_mask)

Output: (Plain)

[ ] Treat incoming NULL as zero

Result acceptability bounds :

Mask output through: [None]

Interlude

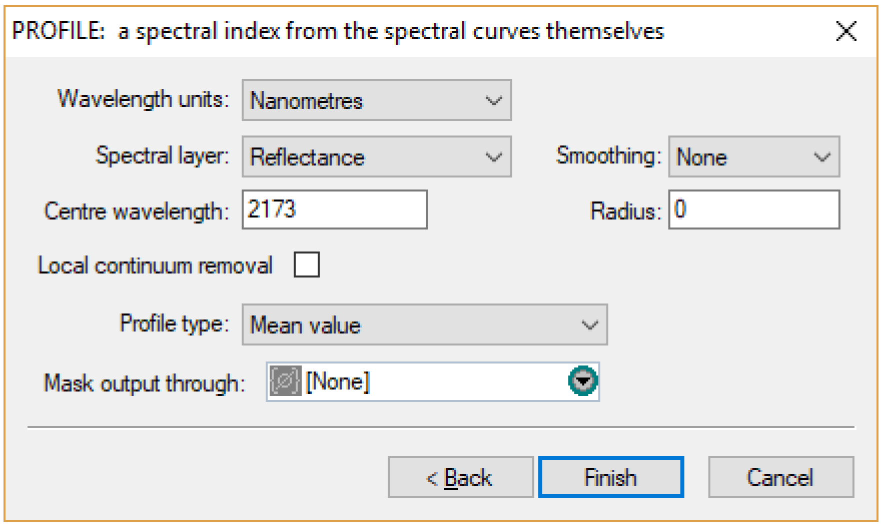
At this point we could do some useful interactive checking, given that we have created these scalars in a suitable dataset. Is our masking strategy working? Are any kaolin spectra getting masked off? Are any non-kaolin spectra staying masked on?
We could plot all the scalars we created (e.g., in the Log screen) but the most useful ones are kaolin_ratio, kaolin_mask, D2200, al_clay_mask and combo_mask. The masking thresholds in kaolin_mask and al_clay_mask might need adjustment.
We'd judge using other TSG plots (e.g., spectral floater plots) and could drag in some other scalars to help, e.g., TSA scalars and imported assays that might be available.

Kaolin composition index

A = [Ref(2138) + Ref(2173)] / R2156

We have Ref(2138) and Ref(2156) already.

Profile scalar for Ref(2173)



PROFILE: a spectral index from the spectral curves themselves

Wavelength units: Nanometres

Spectral layer: Reflectance

Smoothing: None

Centre wavelength: 2173

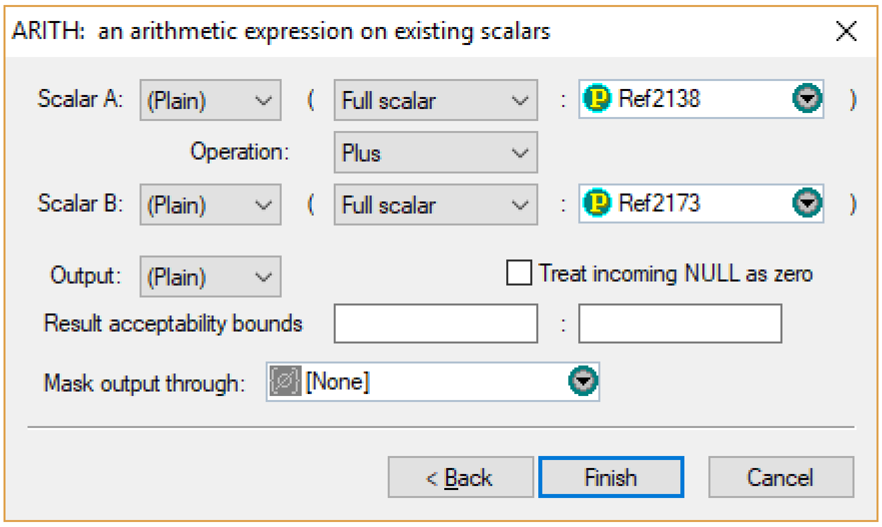
Radius: 0

Local continuum removal ☐

Profile type: Mean value

Mask output through: [None]

Arithmetic scalar 'num2', which is Ref2138 + Ref2173



ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : Ref2138)

Operation: Plus

Scalar B: (Plain) (Full scalar : Ref2173)

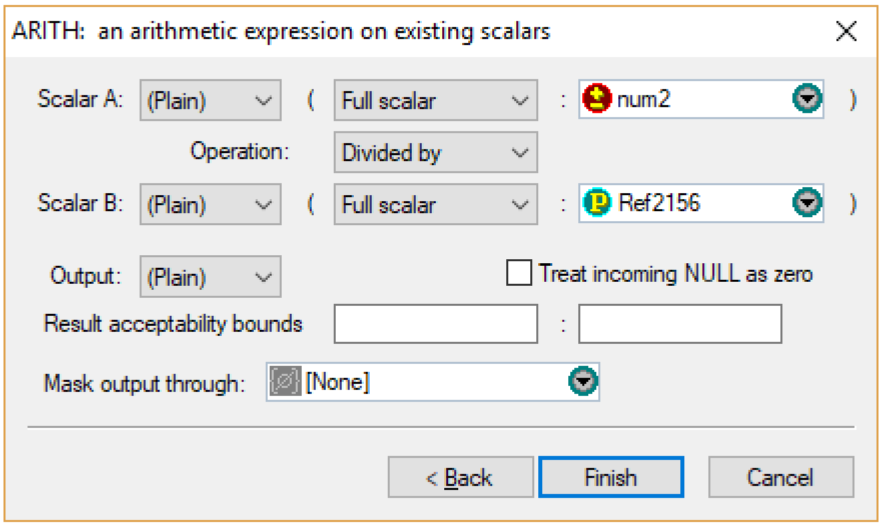
Output: (Plain)

☐ Treat incoming NULL as zero

Result acceptability bounds :

Mask output through: [None]

Arithmetic scalar 'ratio_A', which is num2 / Ref(2156)



ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : num2)

Operation: Divided by

Scalar B: (Plain) (Full scalar : Ref2156)

Output: (Plain)

☐ Treat incoming NULL as zero

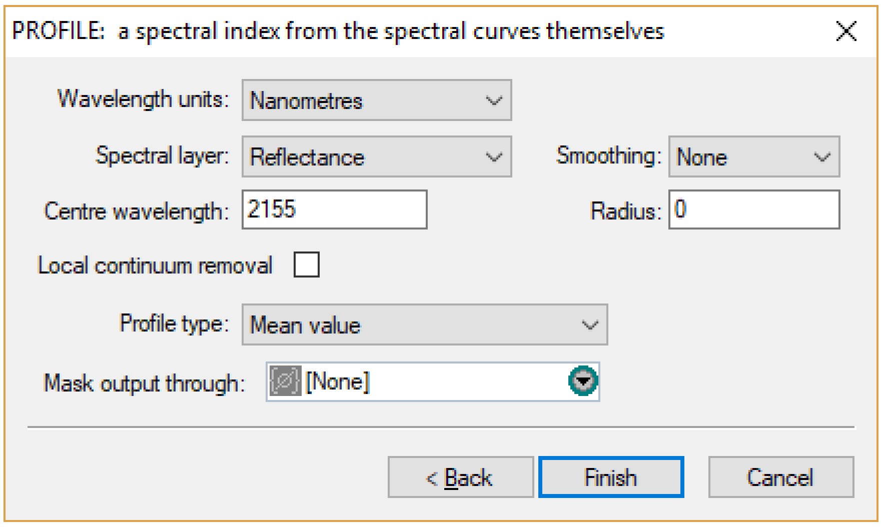
Result acceptability bounds :

Mask output through: [None]

B = [Ref(2155) + Ref(2190)] / R2173

We have Ref(2190) and Ref(2172) already. Ref(2155) is going to be the same as Ref(2156) when given 4-nm-spacing spectra (e.g., HyLogger) but we'll create it anyway.

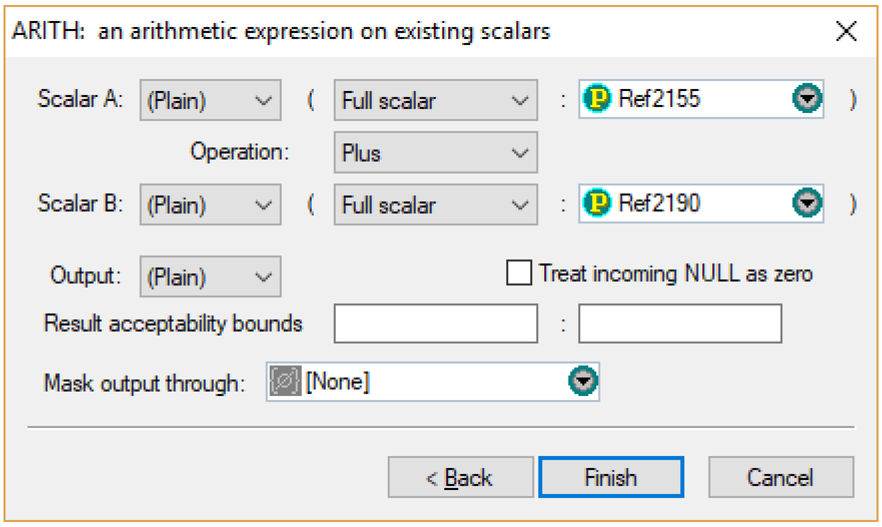
Profile scalar for Ref(2155)



PROFILE: a spectral index from the spectral curves themselves

Wavelength units:	Nanometres		
Spectral layer:	Reflectance	Smoothing:	None
Centre wavelength:	2155	Radius:	0
Local continuum removal	☐		
Profile type:	Mean value		
Mask output through:	[None]		

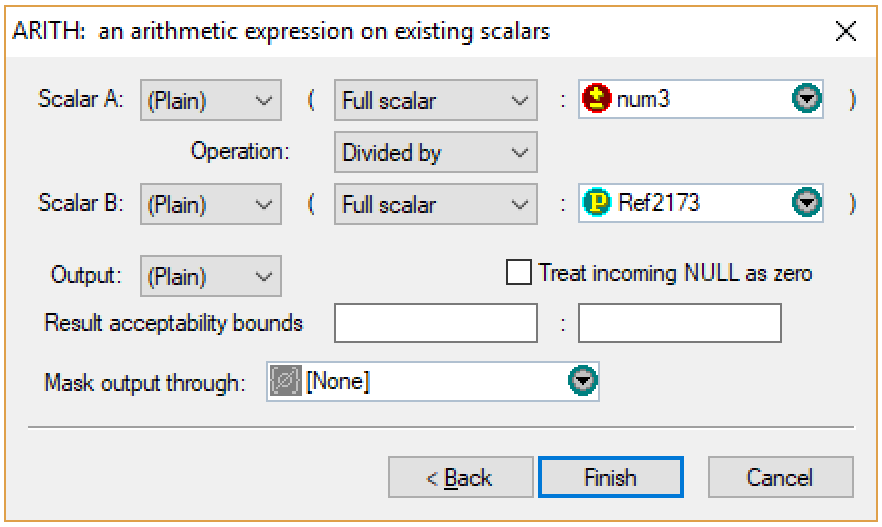
Arithmetic scalar 'num3', which is Ref2155 + Ref2190



ARITH: an arithmetic expression on existing scalars

Scalar A:	(Plain)	(Full scalar	: Ref2155)
Operation:		Plus	
Scalar B:	(Plain)	(Full scalar	: Ref2190)
Output:	(Plain)	☐ Treat incoming NULL as zero	
Result acceptability bounds		:	
Mask output through:	[None]		

Arithmetic scalar 'ratio_B', which is num3 / Ref(2173)

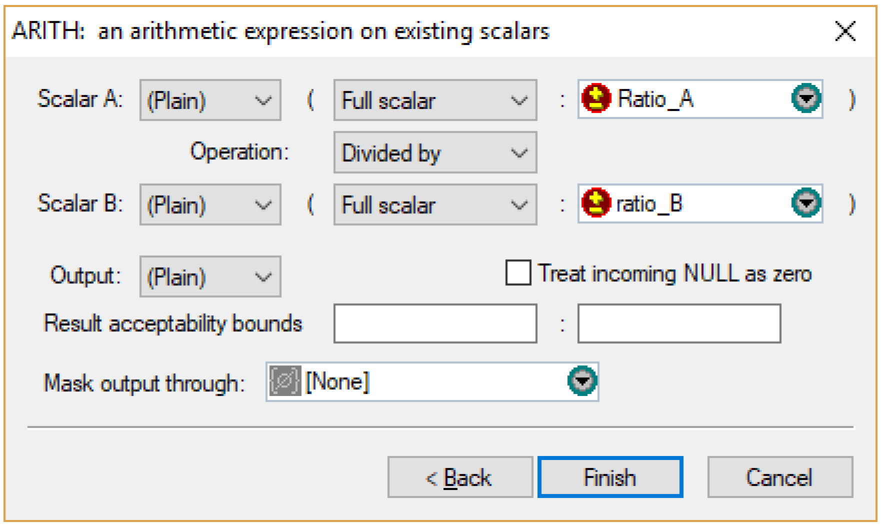


ARITH: an arithmetic expression on existing scalars

Scalar A:	(Plain)	(Full scalar	: num3)
Operation:		Divided by	
Scalar B:	(Plain)	(Full scalar	: Ref2173)
Output:	(Plain)	☐ Treat incoming NULL as zero	
Result acceptability bounds		:	
Mask output through:	[None]		

Finishing off

Arithmetic scalar 'comp_unmasked', which is ratio_A / ratio_B. This is our result but it is unmasked. It returns an index for any spectrum.



ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : Ratio_A)

Operation: Divided by

Scalar B: (Plain) (Full scalar : ratio_B)

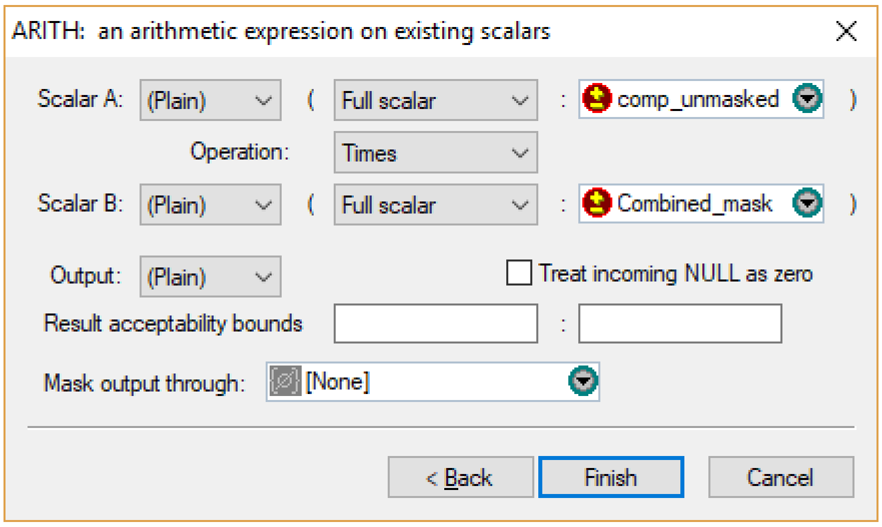
Output: (Plain)

☐ Treat incoming NULL as zero

Result acceptability bounds :

Mask output through: [None]

Arithmetic scalar 'kaolin_comp', which is comp_unmasked times combined_mask. It is our final masked result.



ARITH: an arithmetic expression on existing scalars

Scalar A: (Plain) (Full scalar : comp_unmasked)

Operation: Times

Scalar B: (Plain) (Full scalar : Combined_mask)

Output: (Plain)

☐ Treat incoming NULL as zero

Result acceptability bounds :

Mask output through: [None]

Redux

Now we have our method exposed in the form of individual TSG scalars. Exploring the host dataset, we can see when the method is returning false positives and when it's missing its target, and we should have a good indication *why*. We can tweak it by adjusting any of the scalars. Like a spreadsheet, TSG will automatically recalculate all relevant scalars if we modify an 'earlier' one.

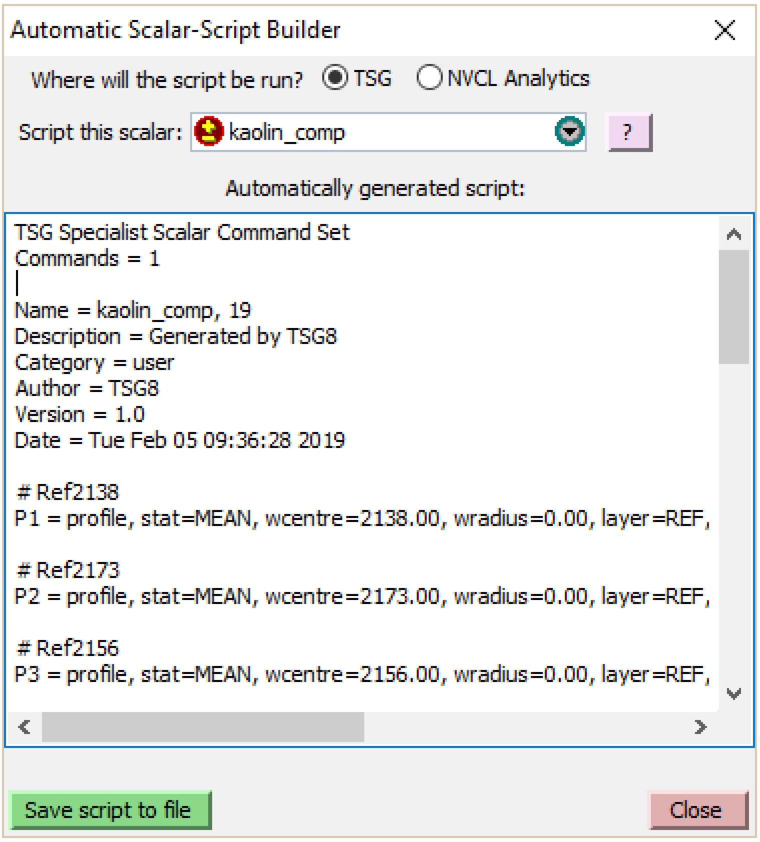
Generating the script

Bring up the script builder using the **Edit->Scalar script builder** menu.

Select the final scalar **kaolin_comp** from the list.

TSG displays an automatically-generated script (see opposite, and below).

Given just our final scalar, TSG has generated a script with 19 methods. The script calculates everything it needs – all the Ref(x) extractions, ratios and masks. It's all right here.
...But why 19 methods and not 23, like in the original script? This is because four of the Ref(x) profile extractions were repeated in the original script, but we didn't make duplicate profile scalars.


Automatic Scalar-Script Builder
Where will the script be run? [✓ TSG] NVCL Analytics
Script this scalar: kaolin_comp
?
Automatically generated script:
Save script to file
Close

```
TSG Specialist Scalar Command Set
Commands = 1
|
Name = kaolin_comp, 19
Description = Generated by TSG8
Category = user
Author = TSG8
Version = 1.0
Date = Tue Feb 05 09:36:28 2019

# Ref2138
P1 = profile, stat=MEAN, wcentre=2138.00, wradius=0.00, layer=REF,

# Ref2173
P2 = profile, stat=MEAN, wcentre=2173.00, wradius=0.00, layer=REF,

# Ref2156
P3 = profile, stat=MEAN, wcentre=2156.00, wradius=0.00, layer=REF,
```

The automatically-generated script

The method order isn't the same as the script builder has its own peculiar opinion on how to order its methods[8]. However the comment (source TSG scalar name) before each method helps one to follow what's going on. You should find that it has the same logic as the original. If you calculate a batch-script scalar from this script then you should find that it looks identical to our final **kaolin_comp** scalar.

```
TSG Specialist Scalar Command Set
Commands = 1

Name = kaolin_comp, 19
Description = Generated by TSG8
Category = user
Author = TSG8
Version = 1.0
Date = Tue Feb 05 09:36:28 2019

# Ref2138
P1 = profile, stat=MEAN, wcentre=2138.00, wradius=0.00, layer=REF,
smooth=NONE, fit=SEVEN, bkrem=NONE, minrad=0.000000

# Ref2173
```

[8] It works backwards from the final scalar and writes out methods as they become ready. A method is 'ready' if it does not depend on any others, or if the other(s) it depends on have been written out.

```
P2 = profile, stat=MEAN, wcentre=2173.00, wradius=0.00, layer=REF,
smooth=NONE, fit=SEVEN, bkrem=NONE, minrad=0.000000

# Ref2156
P3 = profile, stat=MEAN, wcentre=2156.00, wradius=0.00, layer=REF,
smooth=NONE, fit=SEVEN, bkrem=NONE, minrad=0.000000

# Ref2155
P4 = profile, stat=MEAN, wcentre=2155.00, wradius=0.00, layer=REF,
smooth=NONE, fit=SEVEN, bkrem=NONE, minrad=0.000000

# Ref2190
P5 = profile, stat=MEAN, wcentre=2190.00, wradius=0.00, layer=REF,
smooth=NONE, fit=SEVEN, bkrem=NONE, minrad=0.000000

# num1
P6 = expr, param1=P1, param2=P5, arithop=ADD, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# Ref2179
P7 = profile, stat=MEAN, wcentre=2179.00, wradius=0.00, layer=REF,
smooth=NONE, fit=SEVEN, bkrem=NONE, minrad=0.000000

# D2200
P8 = profile, stat=DEPTH, wcentre=2183.00, wradius=63.00, layer=REF,
smooth=NONE, fit=THREE, bkrem=DIV, minrad=0.000000

# num2
P9 = expr, param1=P1, param2=P2, arithop=ADD, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# num3
P10 = expr, param1=P4, param2=P5, arithop=ADD, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# denom1
P11 = expr, param1=P3, param2=P7, arithop=ADD, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# Al_Clay_mask
P12 = expr, param1=P8, const2=0.10000000149011612000, arithop=LGT,
mod1=PLAIN, mod2=PLAIN, mainmod=PLAIN, nullhandling=OUT

# Ratio_A
P13 = expr, param1=P9, param2=P3, arithop=DIV, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# ratio_B
P14 = expr, param1=P10, param2=P2, arithop=DIV, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# kaolin_ratio
P15 = expr, param1=P6, param2=P11, arithop=DIV, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# comp_unmasked
P16 = expr, param1=P13, param2=P14, arithop=DIV, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# kaolin_mask
```

```
P17 = expr, param1=P15, const2=1.00499999523162840000, arithop=LGT,
mod1=PLAIN, mod2=PLAIN, mainmod=PLAIN, nullhandling=OUT

# Combined_mask
P18 = expr, param1=P17, param2=P12, arithop=MULT, mod1=PLAIN, mod2=PLAIN,
mainmod=PLAIN, nullhandling=NONE

# kaolin_comp
return = expr, param1=P16, param2=P18, arithop=MULT, mod1=PLAIN,
mod2=PLAIN, mainmod=PLAIN, nullhandling=NONE
```