B: SURFACE REFLECTANCE IMAGES

Like *Frequently Asked Questions*, a question is posed, e.g., *B1. What is SRFI?* Then, an answer is given[1] with comments and opinions. For cross referencing, each item is labeled, e.g., *B1*.

This tutorial deals with SRFI.sml and REPAIR_IMAGE.sml, their uses, and their options.

SRFI.sml converts multispectral (MS) digital numbers (DNs) to Standardized Reflectance Factor Index (SRFI). Three Correction Level Options are available:

1. Make no corrections for atmospheric effects, i.e., produce top of atmosphere SRFI: SRFItoa.
2. Correct only for atmospheric reflectance effects, i.e., produce atmospheric-path-corrected SRFI: SRFIapc.
3. Correct for all atmospheric effects, i.e., estimate surface SRFI: SRFIsfc.

When appropriate, SRFI.sml also produces a pair of calibrated index rasters called the Perpendicular Vegetation Index (PVI) and Perpendicular Brightness Index (PBI). PVI and PBI rasters are required by DIAG.sml, a script designed for diagnostic analyses.

In Brief …

This tutorial discusses key SML functions and model concepts related to **SRFI.sml** and **REPAIR_IMAGE.sml**. The list below is divided into two groups: one for the key SML functions and the other for key model concepts.

If you are interested in a particular topic below, please go directly to it.

Sec.	Topic (Unique Topics are Bold)	Pages
	Quick Guide to SRFI.sml	**pp. B3-B4**
	Quick Guide to REPAIR_IMAGE.sml	**p. B5**
	KEY SML ITEMS	
B2.	Preparing to use SRFI.sml	**pp. B6-B7**
B3.	Opening the SRFI.sml Script	**p. B7**
B4.	Colors and **Fonts** in a the SML Editor Window	**pp. B7-B8**
B5.	Warning Level 3	**p. B8**

[1] Jack F. Paris, Ph.D., 2407 Maplewood Cir. E., Longmont, Colorado 80503 USA, *jparis37@msn.com* , 303-775-1195

B6. Why not just use Warning Level 1 or 2 p. B8
B7. string Declaration and **text** p. B8
B8. clear Function p. B8
B9. PopupMessage Function: Adjusting the Console Window pp. B8-B9
B10. DATETIME Class pp. B9-B10
B11. CONTRAST Class p. B10
B12. writeTitle Procedure (**proc**) p. B10
B13. printf Function pp. B10-B11
B14. Declarations p. B11
B15. Placement of Declarations pp. B11-B13
B16. How the User Inputs Information to a Running SML pp. B13-B14
B21. If () then Expressions p. B15
B27. Global Functions p. B20
B30. Using a Look-Up Table Array to Improve Efficiency p. B22
B33. Setting up Output Rasters p. B25
B34. Transferring Subobjects p. B25
B35. Filling SRFI Rasters with Values p. B25

KEY MODEL-CONCEPT ITEMS

Sec. Topic (Unique Topics are Bold) **Pages**
B1. **Standardized Reflectance Factor Index** (**SRFI**) Defined p. B6
B17. **Collection Date** (**cdate**) Parameter p. B14
B18. **Sun Elevation Angle** (**sunelevang**) Parameter p. B14
B19. **Imager Number** (**imager**) Parameter p. B14
B20. **Atmospheric Correction** (**atcor**) Parameter p. B15
B22. **Histogram Edge Parameter** (**delcf**) pp. B15-B17
B23. **MSFactor** (**msfac**) Parameter p. B17
B24. **Input cRL** (**icRL**) Parameter pp. B17-B18
B25. Sensor-Specific Parameter Loops pp. B18-B19
B26. **Model for c-Factors** pp. B19-B20
B28. **Method for Finding Dark Histogram Edges** p. B20
B29. **Method for Finding dnpath and SRFpath Values** pp. B21-B22
B31. Histogram Analyses Report pp. B22-B24
B32. Calculating **SRFI** Values p. B24
B36. **Perpendicular Vegetation Index** (**PVI**) and **Perpendicular Brightness Index** (**PBI**) Algorithm pp. B26-B28
B37. Diagnostic Analyses of Output Products: An Example pp. B28-B37
B38. To Re-Run (Modified Parameters) or Not to Re-Run pp. B37-B38
B39. Why is REPAIR_IMAGE.sml Necessary? p. B39

REFERENCES p. B40

Quick Guide to Using SRFI.sml …

If you are already familiar with SML functions and syntax … and you just want to Run SRFI.sml, this Quick Guide is designed to help you.

BEFORE you run SRFI.sml …

- Import the source MS imagery to a **TNTmips .RVC** (Project File).
- Use the suggested Band-Code Names (CB, BL, GL, YL, RL, RE, NA, NB, MA, MB, MC, MD, ME, MF, and/or MG as defined in *Table A7*.
- Be sure to set the Null value equal to **0** when you import these images.
- You *may* need to use REPAIR_IMAGE.sml (see Page 5 & B39).
- From the vendor's metadata, note the following information items: SITE NAME, COLLECTION DATE, SUN ELEVATION ANGLE, SUN AZIMUTH ANGLE, IMAGING SYSTEM, PROCESSING DATE, PRODUCT SOURCE, and BAND-SPECIFIC GAIN SETTINGS (not applicable for some imagers).

AFTER you run the script, the script will ask you to provide or to accept specific information items via a series of **Popup Windows**, in the following order:

- **CONSOLE-WINDOW ADJUSTMENT:** Use your mouse to adjust the size and placement of the Console Window. Under TNTmips Version. 7.1, you only need to do this the first time you use SML.
- **SITE-NAME ENTRY:** Replace NAME with the actual name, e.g., Stockton, CA
- **COLLECTION-DATE ENTRY:** Use the datecode format: YYYYMMDD.
- **SUN-ELEVATION-ANGLE ENTRY:** Use the floating-point format: NN.NN.
- **IMAGER-NUMBER SELECTION:** Read the list. Then, select the Imager Number. Type that number in the box, e.g., 4 for Landsat 7 ETM+.
- **CORRECTION-LEVEL SELECTION:** Type in an integer (one of the three choices). In most cases, you will accept the default Level 3. An exception might be for cases where you are trying to map features that are in the atmosphere itself, rather than on the surface.
- **delcf VALUE ENTRY:** The Histogram-Edge Finding Parameter (delcf) relates to the threshold for changes (del) in the cumulative-frequency (cf). This threshold is used by SRFI.sml to find the image DN value that relates to the reflectance of the atmospheric path. Initially, accept the default value, which is 0.05 (Units: Change in Percentage Points). If, after analysis of the SRFI rasters produced by this script, you believe that the SRFpath values are wrong, you can address this problem by using a different delcf value. Lowering delcf lowers all SRFpath values; raising it, raises all SRFpath values. **For imagery having bit depths greater than 8, you might need to set delcf = 0.01 (rather than to the default, which is 0.05).**

Quick Guide to Using SRFI.sml … Concluded

- **msfac VALUE ENTRY:** Accept the default value, 1.0000. The msfac factor is applied to all of the nominal SRFIsfc values in order to raise or lower them in the same way for all MS bands. Using a msfac value that differs from 1.0000 is advisable ONLY if you have sufficient cause. This usually requires that you have independent and reliable information about the true reflectance factor for one or more scene objects. For example, suppose that you know that the true reflectance factor (RF) for a particular vegetated field is 60% in the near infrared band. If the value of SRFI for that field were only 5000, then you can force SRFI.sml to produce a SRFI value of 6000 by changing msfac from 1.0000 to 1.2000 when you re-run the script. Bidirectional and hill shading effects can cause the observed RF (i.e., SRFI) to be different than the SRFI value.
- **icRL VALUE ENTRY:** Accept the default value, 1.34. icRL affects the c-factors for all MS bands. Raising icRL raises all c-factors in a **non-linear** way with cBL being raised the most and with cMG being raised the least. In contrast, msfac effectively affects all c-factors in the same linear way. The altitude of the site and the haziness of the atmosphere affect your choice concerning icRL. The lower the elevation and the hazier the atmosphere, the higher would be the icRL value. But, it is best to have sufficient cause to change icRL from its default value – based on an analysis of SRFI values.
- **PROCESSING-DATE ENTRY:** Get this date from the metadata**.**
- **PRODUCT-SOURCE ENTRY:** Get this from the metadata.
- **GAIN-CODE ENTRY:** Get this from the metadata. The GAIN CODE is a series of characters from the shortest wavelength band (e.g., BL) to the longest wavelength band (e.g., MG). The number of characters in this GAIN-CODE string varies imager to imager. If you change the default GAIN-CODE string, be sure that the replacement has the same spectral-band order and same number of spectral bands as in the default string. GAIN-CODE length and allowed characters differ from imager to imager. **Be sure** to use the **length** suggested by the **default string.** And, **read** the **legend in the Popup Window** to use the **correct characters** in this string, e.g., **H**, **L**, **M**, **1**, **or 2**. **In some cases (ASTER), there are 9 GAIN-CODE characters; so be careful!**
- **INPUT RASTERS:** Input rasters, e.g., BL, GL, RL, … are the ones you imported from source data before running this SML.
- **HISTOGRAM ANALYSES REPORT ASSESSMENT:** Take time to examine this report before accepting the resulting SRFI, PVI, and PBI rasters.
- **SRFI RASTERS** are output rasters. Put them in a new Project File. Each output raster (SRFIBL … PVI and PBI) are created as new Objects.
- **WHEN SCRIPT FINISHES RUNNING** you should read and/or save the contents of the Console Window. Use Right-Button and Save as… option to save the Console Window contents to a named txt file.
- Other post-processing suggestions are in *B38*.

Quick Guide to Using REPAIR_IMAGE.sml …

If you are already familiar with SML functions and syntax … and you just want to Run REPAIR_IMAGE.sml, this Quick Guide is designed to help you.

BEFORE you run REPAIR_IMAGE.sml …

- Import the source MS imagery to a **TNTmips .RVC** (Project File).
- Use the suggested Band-Code Names (CB, BL, GL, YL, RL, RE, NA, NB, MA, MB, MC, MD, ME, MF, and/or MG as defined in *Table A7*.
- Be sure to set the Null value equal to **0** when you import these images.

AFTER you run the script, the script will ask you to provide or to accept specific information items via a series of **Popup Windows**, in the following order:

- **CONSOLE-WINDOW ADJUSTMENT:** Use your mouse to adjust the size and placement of the Console Window. Under TNTmips Version. 7.1, you only need to do this the first time you use SML.
- **IMAGER-NUMBER SELECTION:** Read the list. Then, select the Imager Number. Type that number in the box, e.g., 4 for Landsat 7 ETM+.
- **SELECT RASTER OBJECTS:** Navigate to the location of the requested raster (e.g., GL) and select it as an input raster object. **Repeat** this action for all requested rasters.
- **The script runs to completion.** Changes to the input rasters are made in place.
- The text report in the Console Window will list how many “errant” pixels were found in each spectral band raster that was processed by this script. The subject pixels were fixed by substituting the indicated SUBSTITUTE VALUE (e.g., 1) for the errant value (e.g., 0). Note that the BL band (blue light) is skipped as it never has an errant value (due to high path-radiance values in this band). You may use a right-click action in the Console Window to save the information as a text file.

B1. What is **SRFI***?*

SRFI is a 16-bit unsigned integer that is equal to the Standardized Reflectance Factor (SRF, in %) multiplied times 100 (see *A16*). SRFI values range up to 65,535, which corresponds to the SRF value of 655.35%.

The numeric range of SRFI values retains the radiometric precision of the source DNs. For example, QuickBird (QB) MS DNs range up to 2047, which corresponds approximately to a SRF value of 100% to 150%, i.e., a SRFI value of 10,000 and 15,000. Since the conversion and correction operations used in SRFI.sml are all linear, SRFI values can easily be converted back to the source DN values without any significant error or loss of precision. The related conversion parameters and equations are in the processing report.

As will be seen, SRFI may be adjusted in many ways before the resulting values are finally used for information maps or categorical analyses. For example, TERCOR.sml makes local adjustments to SRFI values to account for the effects of slope and aspect of terrain relative to the elevation and azimuth angle of the sun. This kind of adjustment can be carried out without having to re-do processing steps that go from original image DNs to estimates of surface reflectance.

SRFI.sml contains quantitative physical radiant-energy quantities associated with 14 MS imaging systems. The user provides up to four items of information:

1. The collection date (YYYYMMDD)
2. The processing date (required only for some sensors)
3. The processing place (required only for some sensors)
4. The sun elevation angle (degrees)

If justified, the user may override the default values of three other processing parameters: The histogram-edge parameter (**`delcf`**), the MS scaling factor for all c-factors (**`msfac`**), and/or the initial c-factor for the red-light band (**`icRL`**).

SRFI has a consistent, calibrated scale that is the same for all MS imagers. This is an important attribute when dealing with different sources of MS data.

B2. What Must Be Done Before Running **SRFI.sml***?*

From www.microimages.com, get a copy of the *most recent* SRFI.sml script.

Image DNs must have a linear relationship to spectral radiance at the sensor (i.e., at the top of the atmosphere): SRtoa. In some cases, the absolute radiometric characteristics of greater-than-8-bit data are irreversibly altered when the data provider converts them to an 8-bit format.

Import the source imagery into a TNTmips project file. Name the new image rasters by simple names such as BL, GL, RL, NA, MB, MC … see *Table A7* for a complete list of generic MS band-image names.
When you import an MS image file, you **must** set the Null Value to 0, enable the Standard Lossless Compression and the Pyramid creation options.

Hint B2. You can use a long, informative name for the TNTmips Project File. You can also put verbose information in the description line associated with each imported raster. But, keep the raster name simple, e.g., BL, GL, RL & NA. Avoid ambiguous names like "blue," "green," "red," and NIR. Also, don't use band numbers, e.g., "Band_1," …, "Band_4." Band number sequences sometimes do not start with 1, e.g., MSS that goes from 4 to 7. MODIS bands are out of order, i.e., from shortest to longest wavelengths, the MODIS band-number sequence is 3, 4, 1, 2, 5, 6, and 7 (for BL, GL, RL, NA, MA, MB, and MC, respectively).

B3. How is the **SRFI.sml** *Script Opened?*

From the TNTmips Main Menu:

- Click the *Process* button, then the *S M L* button, and then the *Edit Script…* button. The *SML Editor* window opens.
- Click the *File* button, then the *Open* button, and finally the **.sml File* button.
- Navigate to the location of the SML file on your computer and select SRFI.sml. This script then appears in the *SML Editor* window.
- Starting with TNTmips Version 7.1, you can open an SML file from Windows by double clicking its icon. TNTmips will start with the SML Editor being opened automatically.

B4. Why do Different Parts of the SML Have Different Colors and Fonts?

Colorized text indicates the role of each SML word or phrase. Please follow along in the SRFI.sml script (in the SML Editor window) as you read the rest of this tutorial. Note the following:

- Comment lines start with the # symbol and are displayed in red type. Comments are present to *help you understand the script and its logic*. This is important during de-bugging activities. Each comment line is ignored when the script runs. A comment line may follow an executable statement on the same line. But, this option is not in these SML scripts.
- **Expressions**, **names of variables**, and **names of objects** are displayed in **black type**. Each **expression** should end with a **semicolon** (**;**). Many **expressions** contain colorized elements such as **declarations**, **keywords**, procedures, functions, and text.
- **Declarations** are preceded by **keywords** displayed in **bold blue type**.
- Procedures and functions are displayed in non-bold blue type.
- Strings of text characters are "enclosed by quotation marks and are displayed as non-bold greenish text."

There are about 1946 lines of programming statements and comments in SRFI.sml. So, the author won't be commenting on each line. But, he will be making comments on groups of lines related to important or complex operations.

B5. What are the **$warnings** *Options?*

SML has three **$warnings** options:

1. Warns you about the use of any function or class which has been deprecated and scheduled for retirement.
2. Same as Option 1.
3. Warns you about variables and objects that have not been declared. Warns you about statements that do not end with a semicolon (;). The default **$warnings** is Option 3.

B6. Why Shouldn't Warning Option 1 or Option 2 be Used?

Shakespeare wrote, "To err is to be human. To forgive, divine." TNTmips is unforgiving. Let the SML Editor help find your typing errors. Use Option 3.

B7. What is a **string***?*

The **string** keyword declares that each variable that follows it will contain **text** characters. In my SMLs, each **string** variable ends with a dollar sign (**$**).

> **Hint B7a.** While ending the name of a **string** variable with **$** is not required by SML, the author strongly recommends that you use this practice so that **string** variables are easily recognized as such in the script.

Near the beginning of a SML script, a bunch of **string** variables are declared with names like **p1$, p2$, … p21$**. Each of these **string** variables is then assigned a single line of **text**. Then, an assignment **statement**, e.g., **p$ = p1$ + p2$ + p3$**, is used to concatenate the **text** content of many **string** variables into a single **string** variable. An example of this approach is in SRFI.sml right after the first set of **string** variables have been declared. Almost all lines of **text** in this example contain **\n** (the line feed command) at the end of the **text**.

> **Hint7b.** Devote a single **line of output text** to a single line of SML script **text**. This practice lets you easily see the relative alignment of **multiple lines of text** as they will appear in a related window or file. See *B9*.

B8. What is the clear**();** *Function?*

This function clears any text that happens to be in the Console Window. The empty parentheses, **(),** indicate that this function has no related parameters.

B9. What is the PopupMessage *Function?*

The PopupMessage function causes a Message window to "pop up" with **text content** controlled by the **text** in the **string** variable, **p$**. See next page.

Figure B9. SML Popup Message Window.

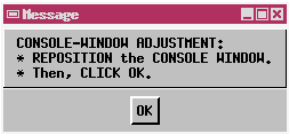


Figure B9 shows that the **text** content of this PopupMessage window is similar in content and format to the related SML script:

```
p1$ = "CONSOLE-WINDOW ADJUSTMENT\n";
p2$ = "* REPOSITION the CONSOLE WINDOW.\n";
p3$ = "* Then, CLICK OK.";
```

But, neither the **\n** control nor the quotation marks are printed into the PopupMessage window. The **string** variable, **p$**, contains a long **string** of concatenated **text** as follows:

```
"CONSOLE-WINDOW ADJUSTMENT\n* REPOSITION the CONSOLE
WINDOW.\n* Then, CLICK OK."
```

Characters in this **text string** run together. And, it is difficult to anticipate how this long **string** of continuous characters would look in a **text** window. The author's approach of using separate lines of **text** followed by concatenation solves this problem.

In response to the Message, you should take the requested actions:

- To Move the Console Window as a Unit: Move your mouse cursor into the red title bar. Then, hold down your left mouse button. Then, drag the window to a different location. Release the button when you are happy with the new location.
- To Move an Edge of the Console Window: Move your mouse cursor into white edge or corner. Then, hold down your left mouse button. Then, drag the edge or corner to a different location. Release the button when you are happy with the new location.
- Then, you can close the Message window by Clicking OK, as suggested.

One configuration for the Console Window is shown on *Page B4*. This shows that the SML Editor window **SML script text** is narrow enough to position the Console Window next to it and at the same size.

B10. What is the **class** **DATETIME cdate$** *Statement?*

TNTmips SML has a long list of pre-defined **class** types. A **class** variable is declared as shown in the statement. In this case, the **class** variable is called **cdate$** (for imager data **c**ollection **date**). **cdate$** contains a long **string** of **text** that describes the full date and full time in English.

cdate$ is associated with the **class** type called **DATETIME**. Its assigned value is based on the value of a **numeric** variable called **cdate**. Note that this **numeric** variable name does not end with the **$**.

Later in SRFI.sml, the user provides a **numeric** value for **cdate** in the **YYYYMMDD** format. This is done through a PopupNum function (see *B17*). Then, the script uses **cdate$** to compute the **numeric** Day of the Year (**doy**). **doy** is used to estimate of the earth-sun distance (**esd**). **esd** is a factor in the model that predicts the Spectral Irradiance at the top of the atmosphere (**sitoa**). **sitoa** is then used in the calculation of SRFtoa and SRFI. This kind of serial calculation constitutes an algorithm for calculating the desired output value of SRFI for each pixel and band.

B11. What is the **class CONTRAST smlContrast** *Statement?*

The **class CONTRAST** variable, called **smlContrast**, allows SML to set parameter values related to the creation of a contrast lookup table for output rasters and then to create (compute) this as a subobject in each output raster.

B12. What is **proc writeTitle()***?*

This part of the script defines a customized procedure that the author wrote. The absence of parameters inside of the parentheses indicates that this procedure has no input or output parameters. In fact, **writeTitle()** contains nothing but a series of printf functions. The printf function is discussed in *B15*.

This particular procedure, **writeTitle()**, writes information into the Console Window about the SML's title, about the SML's version date, about the SML's purpose, about where to read about details, about the SML's author, and about the allowed use.

Customized procedures are all declared by the keyword, **proc**. The opening of a procedure is indicated by the keyword, **begin**. The closing of a procedure is indicated by the keyword, **end**. The author prefers using **begin** and **end** to bracket all of my functions, procedures, and logic loops. But, SML also allows the use of **{** and **}** for this same purpose. Frankly, the author finds it much easier to find loop keywords like **begin** and **end** than loop keyword symbols like **{** and **}**.

B13. What is the **printf** *function?*

This function writes formatted text into the Console Window. Its general format is: printf**(format$,value1,value2 etc.);**
Examine how printf is used in the first 10 printf functions in the **proc** called **writeTitle()**.

- It only prints text to the Console Window.
- **\n** causes a line feed to occur in the Console Window.
- No values are passed to the Console Window through variables.

More complicated printf statements are used later in this script. They often use one or more markers (the **%** symbol) and an associated specification to control insertions in the printed text from indicated variables. The content of each insertion comes from a "value" parameter that is listed in the latter parts of the printf statement.

Markers and related insertion formats are:

- **%s** marks the insertion of a string of text. **%d** marks the insertion of an integer number. **%f** marks the insertion of a floating-point number.
- **%d** can include a specification concerning the number of digits to allocate to the printed number; e.g., **%4d** allocates four digits for an integer.
- **%f** can include a specification concerning the number of digits and the number of decimal places after the point (**.**), e.g., **%6.2f** allocates six digit/characters with two digits after the decimal point, which is one of the characters.

You can see by these examples that the script sometimes continues printing to the same line in the Console Window by using a series of two or more printf functions. To do this, don't include the **\n** command until the last printf function.

B14. What are the Choices for Declared Names?

In SRFI.sml, 8 of the 11 possible **declaration** keywords are used. These 8 **declaration** keywords are **array**, **class**, **func**, **local**, **numeric**, **proc**, **raster**, and **string**. Since SRFI.sml does not deal with **cad** objects, **tin** objects, or **vector** objects, these TNTmips object-related **declaration** keywords do not appear in this SML.

Declarations must be made BEFORE the related object or variable is first assigned a value. Declarations can be made for a list of names with each name separated from others by a comma. The author uses this option for almost all of the declared names in SRFI.sml. A few exceptions occur in the beginning of SRFI.sml.

For variables related to an array, you must use a declaration method such as **array numeric vBL[maxip1];**, where **maxip1** is the size of the **array** called **vBL** (including the possible **array** index value, 0).

B15. Why are Nearly All of the Declaration Statements Up Front?

This was the author's choice. The author also inserted short defining comment lines. But, when he started to write a SML, he declared the variables *just ahead* of where the variable or object is first assigned a value.

Hint B15a. It is a good general practice, in any case, to *put declarations on separate lines.* But, they can occur in the same expression that assigns a value to the declared variable. You can see that both methods are used.

Then, near the end of the creation process, I use a Cut and Paste operation (highlight then Ctrl<C> or Ctrl<X> and then Ctrl<V> at a new location) to *move* the declaration statements next to each other near the top of the SML.

Hint B15b. Be sure to use the Syntax / Check… button often. Then, use the File / Save buttons to save the successfully modified script often. On rare occasions, SML will simply lock up. When this happens, you may suffer the loss of all of the editing that you did (perhaps over a long period of time). To prevent this kind of lost effort, **check syntax and save often**. Also, it is much easier to debug errors if you **check the syntax often**, e.g., after each editorial change! It doesn't take that long to do this. So, do this!

Organize declarations into logical associations. This brings them all together so you can easily spot possible duplications and other errors.

Hint B15c. Checking for errors is easier if you use easily recognizable names that are consistent with quantities and models that you are using in the algorithms.

string variables are easier to recognize as such if you end them with the dollar sign (**$**). **numeric** variables are easier to recognize as such if you start with a lower-case letter. And, **raster** objects are easier to recognize as such if you start with an upper-case letter. Examples in SRFI.sml are:

```
numeric pF,pCB,pBL,pYL,pRE,pNA,pNB;
numeric pMA,pMB,pMC,pMD,pME,pMF,pMG,nc;
numeric i,lin,col,nlins,ncols;
numeric cCB,cBL,cGL,cYL,cRL,cRE,cNA,cNB;
numeric cMA,cMB,cMC,cMD,cME,cMF,cMG,cRLm1,pc;
```

pF, …, pMB have a **numeric** value of either **0** (for **False**) or **1** (for **True**). The "**p**" here stands for "Do you want to process this specific MS band?" Boolean variables are used in conditional statements, e.g., **if** (**pBL**) **then** take a specified action. If **pBL** is equal to **1**, the condition is **True**. If **pBL** is **0**, the condition is **False**. In reality, these Boolean **numeric** variables are floating point numbers.

The **numeric** variables called **i**, **lin**, **col**, **nlins**, and **clins**, have only integer values. But, in reality, they too are floating-point numbers.

The **numeric** variables called **cCB, …**, **cMG**, contain the floating-point values of the c-factor for each of the 15 generic MS bands: **CB**, **BL**, **GL**, **YL**, **RL**, **RE**, **NA**, **NB**, **MA**, **MB**, **MC**, **MD**, **ME, MF**, and **MG** (see *Table A7*). **cRLm1** is also in this group; it will have an assigned a value equal to **cRL – 1**. This factor

occurs in the model that predicts c-factor values for each MS band. **pc** is a power-law parameter in the model that predicts the values of the c-factors.

B16. How Does a User Provide Input Information to the Running SML?

The author chose to use SML's Popup functions as the way to get input and control information from the user. There are other ways to do this.

He could have inserted a series of assignment statements near the beginning of the script; in that case, the user would have to edit the script before running it. This is dangerous approach; the user might make a typo or some other error.

Using Popup functions is better that embedded assignment statements. They allow the script to provide a default value and to specify a range of allowed inputs (for **numeric** variables).

Using XML-controlled interfaces is another way for the user to interact with the script for control and information inputs. The author may modify my SMLs later to add this kind of interface. XML is more complex than Popup functions.

In SRFI.sml, examine the expressions that have Popup functions in them. They start right after the end of all of the declaration statements. The first one is:

```
site$ = PopupString(p$,"NAME",t$);
```

Before this function is used, **t$** is loaded with a short **string** of **title text**, and **p$** is loaded with a long **string** of **prompt text**. Examine the script to see how this was done.

In plain English, the expression above asks the user to **ENTER** in **SITE NAME** as a **text**. The default **text** is NAME (**"NAME"**). When the user **CLICK**s **OK**, the PopupString function assigns the entered text string to the **string** variable called **site$**.

The next use of a **Popup** function is:

```
cdate = PopupNum(p$,dd,d1,d2,0);
```

Before this function is called, **text** is assigned to **p$** and **numeric** values to **dd, d1,** and **d2**. When the user **CLICK**s **OK**, the **PopupNum** function assigns (returns) the **numeric** value in the pane to the **numeric** variable called **cdate**.

In plain English, this expression asks the user to type in an integer number that indicates the collection date for the MS data being processed. SML assigns this number to the variable called **cdate**. The default value for **cdate** is assigned to **dd**. The minimum allowed value for **cdate** is **d1**, which is **19720723**; this is the date that Landsat 1 was launched. The maximum value for **cdate** is **d2**, which is **29991231** (December 31, 2999). **cdate** is an integer; so, the number of decimal places is specified to be **0** (indicated by the last number in the parentheses).

Hint 16. **PopupNum** is used again several times to get **numeric** values for **sunelevang**, **imager**, **atcor**, **delcf**, **msfac**, and **icRL**. In all of these instances, the user should accept the default values *until he or she has ample reason to modify them*. The latter, if ever necessary, requires an analysis of the outputs from SRFI.sml.

B17. What is the Role of **cdate***?*

The earth-sun distance varies with the day of the year (DOY), i.e., it varies with **cdate**. This parameter is in the model for SItoa. Also, the relationship between an image DN and the value for SRsensor may change gradually over time, i.e., as a function of **cdate**. For spacecraft-based systems, SRtoa is equal to SRsensor. In the case of QB, the relationship between DN and SRtoa is believed to be constant over time. The same is true for Ikonos and OrbView. But, Landsat MSS, Landsat TM, Landsat ETM+, ASTER, and MODIS experience gradually-changing relationships between DN and SRtoa over time. So, **cdate** is needed at a location in the script that is ahead of where **imager** number is selected. **cdate** then may be used inside of a loop that produces parameters related to the selected imager. In some cases, the processing date, **pdata**, is also required (as processing coefficients may vary with **pdata** and even from one processing center to another.

B18. Why is the Role of **sunelevang***?*

While the sun produces a steady and predictable level of direct solar spectral irradiance (dssi) at the top of the atmosphere, the local solar elevation angle (**sunelevang**) varies from place to place on the earth. This causes SItoa to vary, but in a predictable way. See *A15*.

B19. What is the **imager** *Number?*

The user selects the source imager type by picking an **imager** number from a list. Later in the script, imager-specific values are assigned to key radiant-energy and sensor parameters according to the user's choice for **imager** number. In some cases, more information is required from the user (as a function of **imager** number). **Special Case for ASTER Data:** When **imager** = 12 (ASTER option), two further options exist: **Option 1**: VNIR, which is GL, RL, and NA) only, or **Option 2**: All 9 Bands (GL, RL, NA, MB, MC, MD, ME, MF, and MG, preprocessed to geographically match each other.

B20. What is the **atcor** *Parameter?*

This **numeric** variable's name stands for atmospheric correction level (**atcor**). It controls the kinds of corrections that are applied to top of the atmosphere (TOA) estimates of SRF (Standardized Reflectance Factor). The choices for **atcor** are:

Level 1: No correction
Level 2: Atmospheric Reflectance (Correction) Only
Level 3: (Correct for) All Atmospheric Effects.

B21. What is the Function of the **if** (**expression**) **then action** *Statements?*

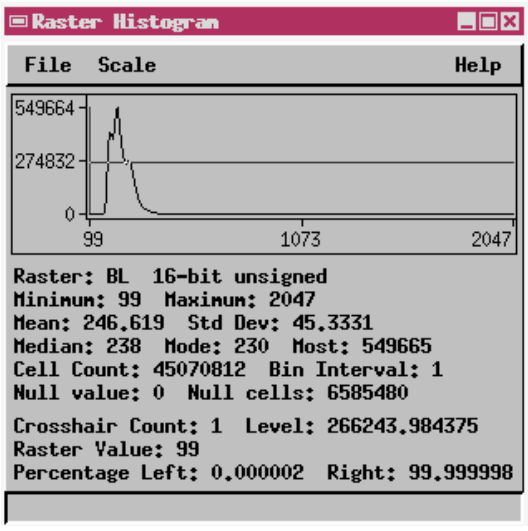
In many places in most SML scripts, a specific **action** requires that a specific **condition** be **True**. The **if** (**expression**) **then action** statement handles these situations. If the expression is **True** (i.e., has a value of **1**), then the **action** happens. The **action** may be a long series of expressions that start with the keyword, **begin**, and end with the keyword, **end**.

Conditional logic is used at many places in this script. Examples are:

Expression	Plain English Condition Being Tested
(imager == 1)	Is the **numeric** value of **imager** equal to 1?
(atcor > 1)	Is the **numeric** value of **atcor** greater than 1?
(pBL)	Is the **numeric** value of **pBL**, True (i.e., is **pBL** == 1)?
(atcor < 3)	Is the **numeric** value of **atcor** less than 3?

B22. What is the **delcf** *Parameter?*

This **numeric** variable's name stands for delta cumulative frequency (**delcf**). It controls how SRFI.sml finds DNpath, which is the image DN that is associated with the atmospheric path spectral radiance and which is also associated with SRFpath (the Standardized Reflectance Factor of the atmospheric path).



An expert analyst can estimate the value DNpath for each spectral band of an MS image data set by using the TNTmips histogram tool. This tool displays the histogram of the DN values for a selected MS band raster as a graph.

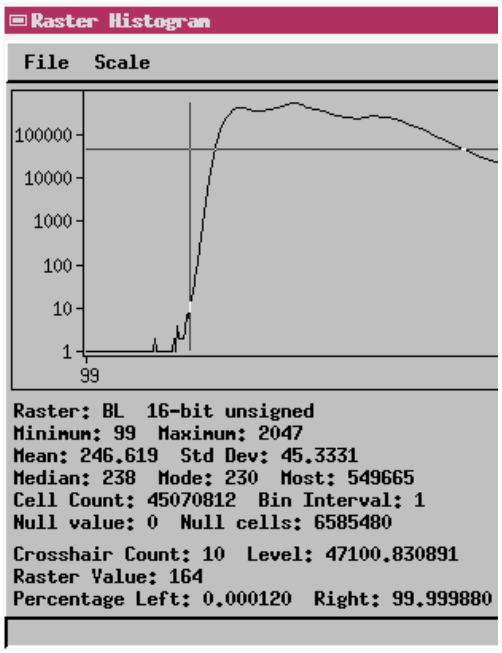
As indicated, using this tool is an interactive and expert activity. In SRFI.sml, an automatic way is included to find a candidate value for DNpath for each spectral band. To understand this, consider how a typical histogram looks (left).

For example, a histogram for a QB MS BL raster might look like the one on the previous page.

In this case, the Minimum DN value for the BL band is **99**. But, as the "Crosshair Count" parameter indicates, there is only 1 pixel in the BL raster that has a value of 99. This isolated value is not likely to be related to the true value of the DNpath for BL: dnpathBL. Imagers usually produce a few isolated noise pixels that have DNs less than true value for dnpathBL.

A better approach is for the analyst to look at the shape of the histogram plot. In particular, the analyst may find the DN where the frequency count (vertical scale of the histogram plot) begins to "rise rapidly" as you go from one DN to the next DN. To "see" this point in the histogram, an analyst usually changes the vertical scale of the histogram plot from a linear scale to a log scale. Also, the analyst might expand the horizontal axis so that every value of DN can be explored in the 1 to 2047 range. This would be done by making the Raster Histogram plot window as wide as the full screen will allow.

For the example histogram, a portion of the expanded log scale plot looks like:



In this case, when the DN value became 164, the frequency count jumped dramatically. Also, the cumulative frequency (**cfq**) rose rapidly. The **cfq** is indicated by the **"Percentage Left: 0.000120"** information in the Raster Histogram window. A number like 0.0000120 is a bit difficult to manage. So, an indicator value, called delcf.

The units of **cfq** are %; they range from 0 to 100%. So, this is a calibrated scale that is more reliable than the frequency count scale.

In these scripts, histogram data is handled though the ReadHistogram function. It transfers histogram frequency count values to an **array** called **v**. Before the ReadHistogram function is called, the values of **v** are reset to 0 by calling the ResizeArrayClear**(v,maxip1)** function. Then, a customized function (called dnHE) is used.

func deHE converts the frequency counts (**fq**) in the array **v** into a cumulative frequency (**cfq**). Then, it searches through the **cfq** values until the change of cfq (called **del**) exceeds the user-specified maximum (called **heloc**).

heloc is obtained through the single argument of the function. In the script, the value of **heloc** is equal to **he**, which is equal to **0.01 * delcf**.

The function, dnHE, then returns a value to the variable, **dnhe**. This is a value that might best correspond to the desired value for **srfipath**. As will be seen later, the final **srfipath** value has to meet a test of continuity with regard to changes as a function of wavelength. The **dnhe** value is a starting point for this final analysis (called the Chavez Power-Law Model).

B23. What is the **msfac** *Parameter?*

This is a user-selected rescaling factor that affects all MS bands through the values of the related **c** factors. Normally, the user will allow **msfac** to keep its default value of 1.000. If the user believes, based on firm, independent information, that the SRFI values produced by SRFI.sml are systematically too high or too low – for all bands – then **msfac** may be set equal to a value less than or greater than 1 to adjust for this. In any case, the default value (1) for **msfac** would be taken for the first use of SRFI.sml.

In SRFI.sml, the default values for **c** are adjusted: i.e., **cBL = cBL * msfac**. As you will learn, the adjustment factor, **msfac**, is nearly always kept at its default value of 1.000. There are other ways to adjust for overall scaling problems.

B24. What is the **icRL** *Parameter?*

This is a user-selected value. Its default value is 1.34. A **c**-factor is a floating-point number that is greater than one, but probably less than 2. It is used to correct for remaining atmospheric effects through a simple equation:

SRFsfcXX = cXX * SRFapcXX. **(B24a)**

As explained in detail in *A17*, **c** is an aggregate factor that includes several atmospheric effects into one linear operation:

c = 1 / [t2 * (t1 + Slsky / Sltoa)] **(B24b)**

where

- **t1**: the transmittance of the atmospheric path between the sun and the surface (a spectral radiance transmittance).
- **t2**: the transmittance of the atmospheric path between the surface and the sensor (a spectral radiance transmittance).
- **SIsky / SItoa** is the magnitude of **SIsky** compared to that of **SItoa**.

t1, **t2**, and **SIsky** are generally unknown. **SItoa** is known. So, the unknowns are combined into one unknown parameter, the c factor.

In addition to an imprecisely known atmospheric effect, the reflectance factor of the surface is affected by the irradiance and viewing angles. This effect is called the bidirectional reflectance effect. The clearest example of this is when a MS imager is viewing objects in a scene at a viewing angle that points toward the azimuth of the sun. Many 3-D objects in the scene have a shadowy side (away from the sun). So these objects appear to have less reflectance when looking back at the sun compared to nadir looking or looking in some other direction. But, some other objects, e.g., a water surface, will have a higher reflectance when seen at this same angle of viewing.

One way to account for the bidirectional effects on reflectance is to change **msfac** from its nominal value of **1**. Another way would be to change **cRL** from its nominal value of **1.34**. The effects of these changes on SRFI will be explored in a different section of this tutorial. But, for now, it can be noted that the value of **c** for MS bands other than the RL band are controlled by a wavelength-dependent power-law model where the c-associated power factor (**pc**) is set equal to a nominal value of 2.2714.

B25. What are the Sensor-Specific Parameters for QuickBird MS?

The statements below assign values to all of the QB-related processing parameters in this script.

```
if (imager == 1) then begin
    imager$ = "QuickBird MS";
    pBL=1; pNA=1; maxi=2100; dnlow=1; dnhigh=2047;
    wLenBL=0.482; wLenGL=0.548; wLenRL=0.654; wLenNA=0.809;
    skBL=0.2359; skGL=0.1453; skRL=0.1785; skNA=0.1353;
    dnbBL=0; dnbGL=0; dnbRL=0; dnbNA=0;
    dssiBL=1925; dssiGL=1843; dssiRL=1575; dssiNA=1250;
end
```

Setting **pBL=1** and **pNA=1** enables on all of the conditional statements and related actions that deal with these two MS bands. By default, GL and RL are always enabled. In other sensors, these might be disabled and others might be enabled. SRFI.sml is designed to be flexible with regard to MS bands that need to be processed.

Setting **maxi=2100** accounts adequately for the expected range of DNs for QB MS data. **dnlow** and **dnhigh** specify the expected range of the related QuickBird MS data (as 16-bit unsigned integers). Note that "0" is reserved as an image fill raster value.

The effective wavelengths (in μm) associated with each QB MS band are assigned to **wLenBL**, **wLenGL**, **wLenRL**, and **wLenNA**.

The spectral k factors (in $W\ m^{-2}\ sr^{-1}\ um^{-1}\ DN^{-1}$) associated with each QB MS band is assigned to **skBL**, **skGL**, **skRL**, and **skNA**.

Krause (2004) has published values for the **k** factors and the effective bandwidths (ebw) for each QB MS band. **skBAND = kBAND / ebwBAND**. His report also indicates that the values of **dnbBL**, dnb**GL**, dnb**RL**, and dnb**NA** are all zero.

The values for dssi (direct solar spectral irradiance) are from Weast (1985). See *A15* for the definition of dssi.

The parameters associated with each of the other 13 imagers are often more complicated than is the case for QuickBird 2 MS data (**imager** == 1). For example, in the case of IKONOS data (**imager** == 2), the **sk** parameters were changed by Space Imaging after the collection date of **20010222** (February 22, 2001). Here the processing date and the collection date are assumed to be the same. This kind of change is handled by a logic **if-then** statement within the (**imager** == 2) loop.

The most complex situation (so far in the SML) is the case of Landsat 7 ETM+ data (**imager** == 4). There are several changes related to processing date. And, there are differences between two processing centers (EarthExplorer and USGS). Also, the data in each spectral band may have been collect with a low gain or a high gain. To handle all of these possibilities, SRFI.sml has 127 lines of code within the (**imager** == 4) loop!

B26. What is the Model for c-Factors?

The following expressions predict the values for each c-factor:

```
if (pCB) then cCB = 1 + cRLm1 * (wLenRL/wLenCB)^pc;
if (pBL) then cBL = 1 + cRLm1 * (wLenRL/wLenBL)^pc;
cGL = 1 + cRLm1 * (wLenRL/wLenGL)^pc;
if (pYL) then cYL = 1 + cRLm1 * (wLenRL/wLenYL)^pc;
if (pRE) then cRE = 1 + cRLm1 * (wLenRL/wLenRE)^pc;
if (pNA) then cNA = 1 + cRLm1 * (wLenRL/wLenNA)^pc;
if (pNB) then cNB = 1 + cRLm1 * (wLenRL/wLenNB)^pc;
if (pMA) then cMA = 1 + cRLm1 * (wLenRL/wLenMA)^pc;
if (pMB) then cMB = 1 + cRLm1 * (wLenRL/wLenMB)^pc;
if (pMC) then cMC = 1 + cRLm1 * (wLenRL/wLenMC)^pc;
```

```
if (pMD) then cMD = 1 + cRLm1 * (wLenRL/wLenMD)^pc;
if (pME) then cME = 1 + cRLm1 * (wLenRL/wLenME)^pc;
if (pMF) then cMF = 1 + cRLm1 * (wLenRL/wLenMF)^pc;
if (pMG) then cMG = 1 + cRLm1 * (wLenRL/wLenMG)^pc;
```

This is a Power-Law Model. Based on the ratio of **wLenRL** to **wLenX** (for each named band, X), the value of each **c** is predicted from **cRLm1**. The default power factor, **pc**, is equal to 2.2714. Its value is based on c-factor values from many scenes and based on the notion that the value of **c** must decrease toward **1** as wLen gets longer (towards MG).

Recall also that the multiplicative factor, **msfac**, is applied to all modeled values of **c** after the power-law above is used.

In reality, **cRL**, **pc**, and **msfac** determine the values of **c**. Increasing **cRL** causes all c-factors to increase but at a higher rate at the BL end of the spectrum than at the MG end of the spectrum. Increasing **pc** leaves cRL unchanged, but increases cCB, cBL, cGL, and cYL and decreases cRE, cNA, cNB, cMA, cMB, cMC, cMD, cME, cMF, and cMG. Increasing **msfac** causes all c-factors to increase but in the same multiplicative way.

> The user has a few choices with regard to altering c-factors. However, alteration may not be necessary at all if the resulting SRFI values are viewed as being accurate enough to serve as a reliable frame of reference for finding biophysical features of interest for information extraction processes that work with SRFI values.

B27. What are the Global Functions?

After the c-factors have been predicted, SRFI.sml is ready to process the source data. GetInputRaster functions are used to open the input rasters for reading DN values. Note that a filter is operative here with respect to the number of lines, number of columns, and data type for these input rasters.

After the input rasters are opened, the GlobalMin function gets the minimum value in a target raster and assigns the value to a variable, e.g., **dnminBL**. The GlobalMax function gets the maximum value in a target raster and assigns the value to a variable, e.g., **dnmaxBL**. Later, these values are printed to the Console Window for the user to examine (to see if they are reasonable values).

B28. How are the Dark Edges of Histograms Found?

If atcor > 1, then DN values related to SRFpath need to be determined. The statements using the custom function called deHE are used in the first step of this process.

B29. How are dnpath and SRFpath Values Found?

After finding dnhe values, the script determines SItoa values for all bands. See *A16*. This part of the script contains the model that accounts for DOY-based variations in earth-sun distances (esd). This code is:

```
cdate$.SetDateYYYYMMDD(cdate);
doy = cdate$.GetDayOfYear();
esd = 1 - 0.01672 * cosd(0.9856 * (doy - 4));
sitoafac = sind(sunelevang) / esd^2;
```

The first expression above uses the **numeric** value of **cdate** to set the date and time as a **string** of **text** assigned to **cdate$**. The second expression above then extracts the Day of the Year **numeric** value (**doy**) from cdate$.

The function, **cosd**, returns the **numeric** value of the cosine of the argument (argument is in degrees). The function, **sind**, returns the **numeric** value of sine of the argument (argument is in degrees).

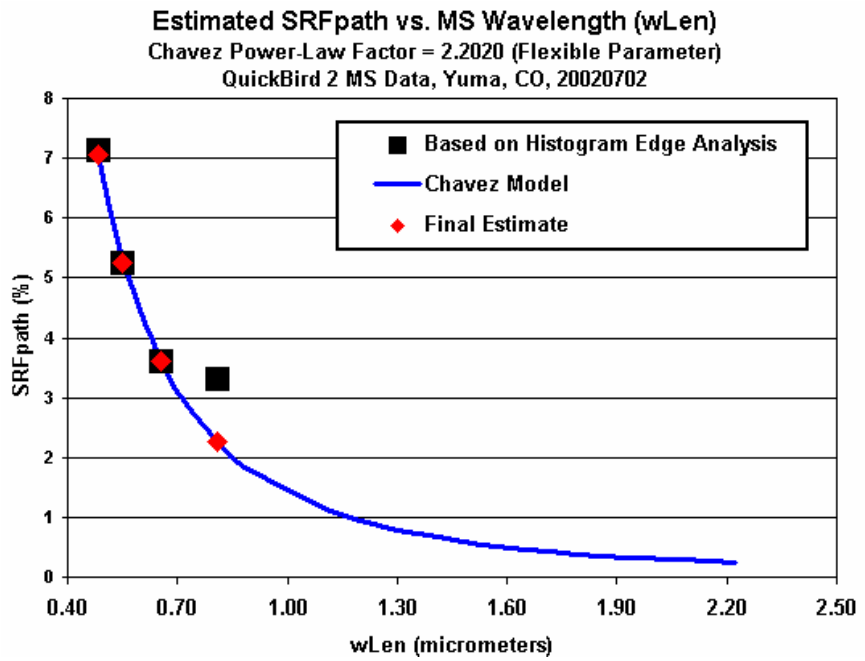
sitoafac is a factor that accounts for **esd** effects and **sunelevang** effects on **dssi** to yield **sitoa** values.

After this, conversion factors, called a-factors, are determined. These with dnb factors convert DNs to SRFtoa values. That is, for MS band XX, **srftoaXX = (dnXX – dnbXX) * aXX**. The **a**-factors and dnb factors also can be used to convert **srftoaXX** back to **dnXX** values.

Starting with provisional dnpathX1 values (from dnheX), the script converts the DNs to SRFtoa values (using a-factors and dnb factors). Then, it uses another power-law model to determine how the dnpathX1 values vary with wavelength (wLenX) according to a power law. The power law produces a second opinion for SRFpathX values (called srfpathX2). Then, the lower of two estimates (SetMin**(srfpathX1, srfpathX2)** is taken as the final value for srfpathX. Its related dnpathX is recovered. This produces values for the SRF of the atmospheric path (SRFpath) that are, in turn, used to convert SRFtoa to SRFapc.

The illustration below shows how these estimates relate to each in a plot of SRFpath versus wLen.

Figure B29. Chavez (1996) Power-Law Model



B30. How are Arrays Used to Speed the Production of SRFI Rasters?

Each QB MS image has millions of pixels. It would be inefficient to recalculate the value of SRFI from each DN value using redundant floating-point operations.

A more efficient, faster way is to pre-calculate each SRFI for each possible DN value and to store these results in an array.

There are only about 2100 possible QB DN values for each MS band. The script related to *B23* does this pre-calculation work. Then, the array results are applied to each pixel for all of the MS bands involved with the selected imager.

B31. Why Provide a Report about the Results of the Histogram Analyses?

Before creating new output rasters for SRFI and before processing each input MS band to fill those output rasters with values, it is useful to provide the user with information related to key elements of the processing.

In particular, the user needs to view the results from the automatic processing that SRFI.sml used to find DNpath values and their relationship to SRFpath values. The related report is generated by the script related to this question.

After the report has been printed to the Console Window, the user should look at these results and make a quality determination *before* opting to continue the program.

If the user elects to continue, then the rest of the processing is fully automatic.

If the user elects to **stop** the program (click the Cancel button), then the user can change the default parameter, **delcf**, and then re-run SRFI.sml to return to the quality checking point (Histogram Analyses Report).

The issue for the user will be: What are reasonable expectations regarding the parameters in the Histogram Analyses Report?

Look at a typical set Histogram Analyses Report (as printed out in the Console Window).

Here are quality control questions you should check out when looking at the information in the Histogram Analyses Report:

- ✓ Is the SITE NAME correct?
- ✓ Is the COLLECTION DATE correct? Must be correct. The DAY OF THE YEAR is calculated from this date. You do not need to check its accuracy.
- ✓ Is the PROCESSING DATE correct? Must be correct (for some sensors).
- ✓ Is the SUN ELEVATION ANGLE correct? Must be correct.
- ✓ Is the IMAGER correct? Must be correct.
- ✓ Is the GAINCODE correct? Must be correct.
- ✓ Is the DATA VENDOR correct? Must be correct (for some sensors).
- ✓ Is the CORRECTION LEVEL what you wanted it to be?
- ✓ Take note of the delcf. You may want to change it (when you optionally run the script again).
- ✓ Take note of the msfac parameter. You usually should keep it at **`1.0000`**. But, you may re-run the script with a different overall **`msfac`** scaling value.
- ✓ Take note of the icRL parameter. You may want to change it (when you optionally run the script again).
- ✓ Now, consider the data in the printed table:
  - ✓ Is DNheXX greater than DNminXX, but not too much greater?
  - ✓ Are the SFRpath1XX values near the SRFpathXX values for the visible bands?
  - ✓ Is the SRFpathXX value for near infrared (and middle infrared) bands as low as you would expect for the kind of atmosphere that you see in this image? They should be significantly lower than the SRFpath1 values for these non-visible region bands.
  - ✓ Does SRFpath decrease steadily from short-wavelength bands, e.g., from BL, to the longer-wavelength bands, e.g., NA?
  - ✓ Is the SRFIpath Model Parameter in the range from 1.5 to 5?

If all of these parameters look reasonable to you, then continue to select the output SRFI, PVI (if asked) and PBI (if asked). After you select them, SRFI.sml runs without further user interaction to completion. This takes about 10 minutes for a full scene of QB MS data.

We will now continue examining the rest of the script that automatically carries out processing to the end of SRFI.sml.

B32. How are SRFI Values Calculated and Handled by Arrays?

The equations for calculating a SRFI value for a given DN value for a given MS band are similar within each band. For example, for the BL band:

```
srftoaBL = (i – dnbBL) * aBL;
srfapcBL = srftoaBL - srfpathBL;
srfsfcBL = srfapcBL * cBL;
```

The 1st assignment assignment applies a conversion factor (**aBL**) to the indexed DN value (**i**) **- dnbBL** to get the value for SRFtoa, which is transferred to the variable called **srftoaBL**.

The 2nd assignment statement subtracts **srfpathBL** from **srftoaBL** and transfers the resulting value to **srfapcBL**. If the user had specified that no correction for atmospheric path reflectance should occur (i.e., if **atcor == 1**), then **srfpathBL** has been already assigned a value of zero. This condition results in **srfapcBL** being assigned to a value equal to **srftoaBL**.

The 3rd assignment statement applies a conversion factor (**cBL**) to **srfapcBL** and assigns the result to **srfsfcBL**. If the user had chosen **atcor** to be either 1 or 2, then **cBL** has already been assigned to a value of 1 before this part of the script. This **atcor** condition would result in **srfsfcBL** being assigned a value equal to either **srftoaBL** or **srfapcBL** (for atcor == 1 and atcor == 1, respectively).

Then, the resulting **srfsfcBL** value scaled up by a factor of 100. Then, it is rounded off to the nearest integer. That integer is checked for possible values less than 1. If so, it is reassigned a value of 1. Then, the resulting final integer (**srfiBL**) is assigned to the i[th] element of the array (**vBL[i]**).

```
srfiBL = round(srfsfcBL * 100);
if (srfiBL < 1) then srfiBL = 1;
vBL[i] = srfiBL;
```

In the above six assignment statements, there are 5 floating point operations, 6 assignment operations, 1 logical condition operation, and one call function, round(x). By storing the result in an array for each possible DN value, these operations are reduced from 13 to 2. This is a significant run-time savings.

The author usually tries to use pre-calculated arrays in processes that involve processing large numbers of pixels in rasters.

B33. How are Output Raster Objects Specified and Set Up?

The GetOutputRaster function is used to ask the user to name and find a location for new "output" rasters. The general format for this function is:

```
GetOutputRaster(Raster,nlins,ncols,rtype$);
```

Raster is the assigned raster name (in a selected .RVC file). nlins is the number of lines for the new Raster. ncols is the number of columns for the new Raster. rtype$ is a string that specified the data type for the new Raster.

After the user has selected the new Raster, the next set of statements specify desired conditions and subobjects for the new Raster.
SetNull(SRFIBL,0) specifies that the raster, **SRFIBL**, will have a null value and that this null value is 0.

B34. How Can You Transfer Common Subobjects to New Rasters?

CopySubobjects function automatically copies specified subobjects from a SourceRaster to a TargetRaster. The general format is:

```
CopySubobjects(SourceRaster,TargetRaster,control$);
```

In this case, the script is transferring the georeferencing ("GEOREF" subobject from RL to the SRFI raster for each indicated MS band.

Next, a pre-existing raster is opened: e.g., CLUTRL, in a pre-existing .RVC file, e.g., cluts.rvc located in the root directory of your C: drive. CLUTRL has one subobject, which is a default contrast look up table designed to go with SRFIRL. After the subobject copy is finished, the CLUTBL raster is closed with the CloseRaster(CLUTBL) function.

B35. How are Image DNs Actually Converted to SRFI Values?

An input image DN is obtained from the input raster via a statement such as: **dnBL = BL[lin,col]**. dnBL is checked to see if it is a null value, i.e., is it equal to 0. If not, the dnBL value (an integer) is then used as an index in the array called **vBL** raster to provide a pre-calculated value for SRFI. That value is immediately assigned to the same pixel in the output raster via a statement: **SRFI[lin,col] = vBL[dnBL]**.

The use of pre-calculated values in an array greatly speeds up the conversion from an image DN to the value for SRFI.

B36. What are PVI and PBI?

PVI is related to the Weighted Difference Vegetation Index (WDVI), which was defined by Clevers (1988). In the context of this tutorial,

wdvi = srfiNRsfc – bslineslope * srfiRLsfc **(B37a)**

where

- srfiNRsfc is the SRFIsfc value for a pixel in the NA or the NB band.
- srfiRLsfc is the SRFIsfc value for this pixel in the RL band.
- bslineslope is the slope of the Line of Bare Soils in a plot of srfiNRsfc vs. srfiRLsfc feature space.

Here, NR represents either NA or NB. bslineslope will change when the wavelengths associated with RL and/or NR change.

Rondeaux, *et al.*, (1996) found that the expected location for the Line of Bare Soils is specified by:

srfiNR = srfiNRint + bslineslope * srfiRL **(B37b)**

where

- srfiNRint = 254 (which is to say that SRFNRint = 2.54%)
- bslineslope = 1.086.

In the classic definition for WDVI, srfiNRint was wrongly assumed to be 0. This small error is easily corrected. Simply, re-define WDVI as:

wdvi = (srfiNR – srfiNRint) – bslineslope * srfiRL **(B37c)**

Another way to think about this is that a small subtractive transformation is needed to yield a slightly-different value, tsrfiNR.

tsrfiNR = srfiNR – srfiNRint **(B37d)**

The purpose of the bslineslope factor in Equation B36c is to transform also the SRFIRL scale in a way that places the Line of Bare Soils on the 1:1 diagonal line in a plot of tsrfiNR vs. srfiRL.

However, these empirical operations do not preserve the original SRFI units of the data. See the discussions in *A19* and *A20*. Another scale-preserving transformation places the Line of Bare Soils on the 1:1 diagonal line in a plot of tsrfiNR vs. tsrfiRL. This is a translation (T) and rotation (R) of the SRFI coordinates system in a counterclockwise direction by an angle equal to 45 degrees minus atan(bslineslope). See *A20* for details.

In SRFINR vs. SRFIRL feature space, let ang be the angle between the Line of Bare Soils and the SRFIRL axis line. ang is related to bslineslope by an arctangent (atan) function. Specifically,

ang = - atan(bslineslope) **(B37e)**

For example, if bslineslope = 1.086, then ang = **- 47.3608** degrees.

Then,

pbi = pfac * (srfiRL * cosang - tsrfiNR * sinang) **(B37f)**

and

pvi = pfac * (srfiRL * sinang + tsrfiNR * cosang) **(B37g)**

where

- **cosang = cos(angrad)**
- **sinang = sin (angrad)**
- pfac is a scaling factor (to be addressed below)

The choice of pfac is important if pvi and pbi are to be represented by an integer number.

A typical value of srfiNR and srfiRL for dense vegetation is 6000 and 300 (related to 60% for SFRNR and 3% for SRFRL). If these values are inserted into *Equation B37f* and if the desired value of pvi is 1000, then the resulting pfac value would be 0.2723659.

Note also that the value of pvi any where on the Line of Bare Soils is 0. Where pvi values are positive, there is a spectral mixture between bare soil and vegetation. However, some pixels represent materials not involved in this soil-vegetation mix. They would have pvi values less than 0 (i.e., negative values). Thus, it is prudent, therefore, to add a fixed offset to the pvi values from *Equation 37f* to achieve a final expression for pvi, which is:

pvi = pvioff + pfac * (srfiRL * sinang + tsrfiNR * cosang) **(B37h)**

A convenient value for pvioff is 1000. It is possible for pbi to take on large values, e.g., for very bright objects in a scene. Thus, a maxpvipbi value is selected (equal to 3000 in SRFI.sml) to control the overall range of pvi and pbi. Most surface materials will have pvi and pbi values much less than 3000.

This PVI and PBI model and its associated parameter values appear in SRFI.sml. It is important to note that the basic units of pvi and pbi are related to the units of SRFI, but scaled down, in a uniform way, by the pfac =

0.2723659. Without this scaling factor, the units of pvi and pbi would be the same as the units of SRFI. cosang and sinang factors in *Equation B37f* and *Equation B37g* cause a rotational transformation that does not affect the units of the resulting feature space.

B37. What Kinds of Diagnostic Analyses Can Be Done with SRFI, PVI, and PBI?

After SRFI.sml finishes, you have a number of SRFI rasters and possibly a pair of PVI and PBI rasters. They each contain a georeferencing subobject, a histogram subobject, a set of pyramid tier subobjects, and a contrast table subobject. So, they are ready to be viewed in various ways.

Quick-look diagnostic work could involve the following four data Quality Checking (QC) Activities:

1. View the SRFI rasters as a Color Infrared (CIR) image
2. View the SRFI rasters as a Natural Color (NC) image.
3. View the PVI and PBI rasters.
4. Use the Raster Correlation tool to examine the Tasseled Cap (TC) distribution in a scatterplot of SRFINA vs. SRFIRL.
5. Use the Raster Correlation tool to examine the TC distribution in a scatterplot of PVI vs. PBI.

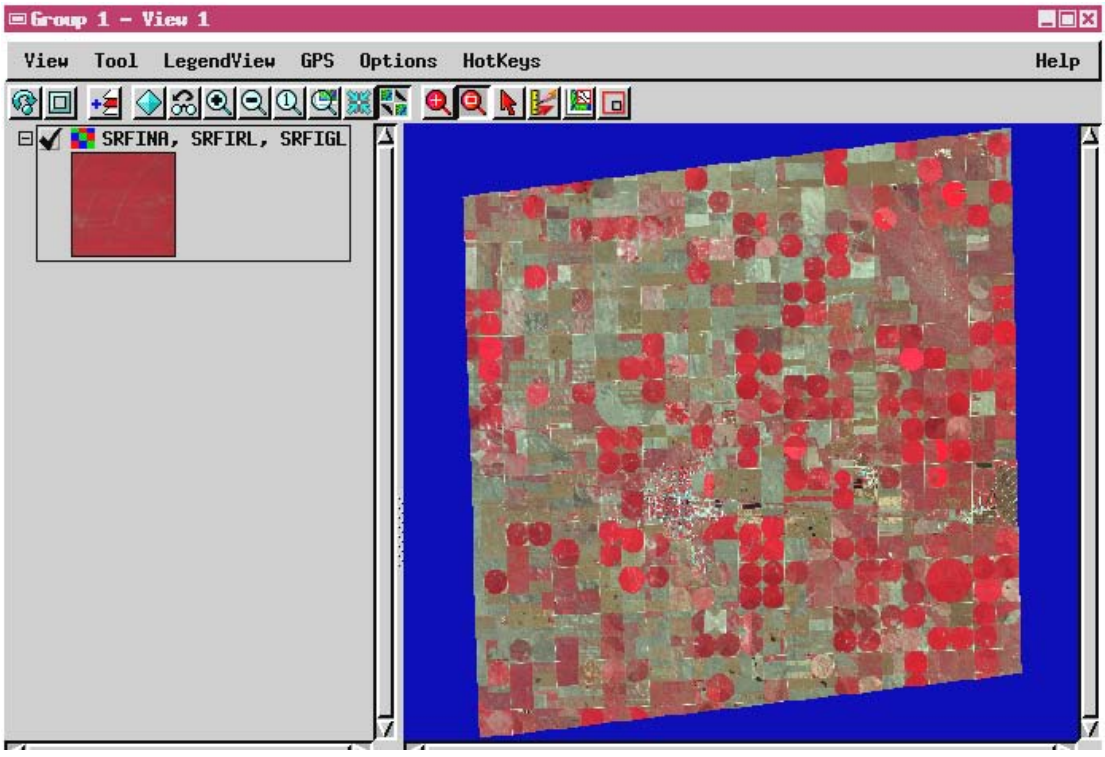
The next 8 pages contain hints about these five QC Activities. They also discuss how you can derive modified values for the control parameters in SRFI.sml: delcf, msfac, and/or icRL.

QC Activity 1:

- Open a TNTmips Display Spatial Data… New 2D Group.
- In the Group Controls window, use the Add Raster… button to get a pull-down menu.
- Select Quick-Add RGB.
- Navigate to the .rvc file that contains the SRFI rasters. Every MS data set includes at least a set of SRFIGL, SRFIRL, and SRFINA rasters OR a set of SRFIGL, SRFIRL, and SRFINB rasters
- Assign the Red Green Blue components of color to SRFINA, SRFIRL, and SRFIGL, respectively, OR to SRFINB, SRFIRL, and SRFIGL, respectively.

When the author did this for a QB MS SRFI data set for Yuma, CO (collected July 2, 2003), TNTmips produced the display in ***Figure B37a***.

Figure B37a: Color Infrared (CIR) Combination of SRFI Rasters



While there are no “true” colors for a CIR picture, you should expect to see a reasonable range of color saturation in the “red” colors associated with dense vegetation in a CIR picture. And, the expected the overall contrast should be excellent.

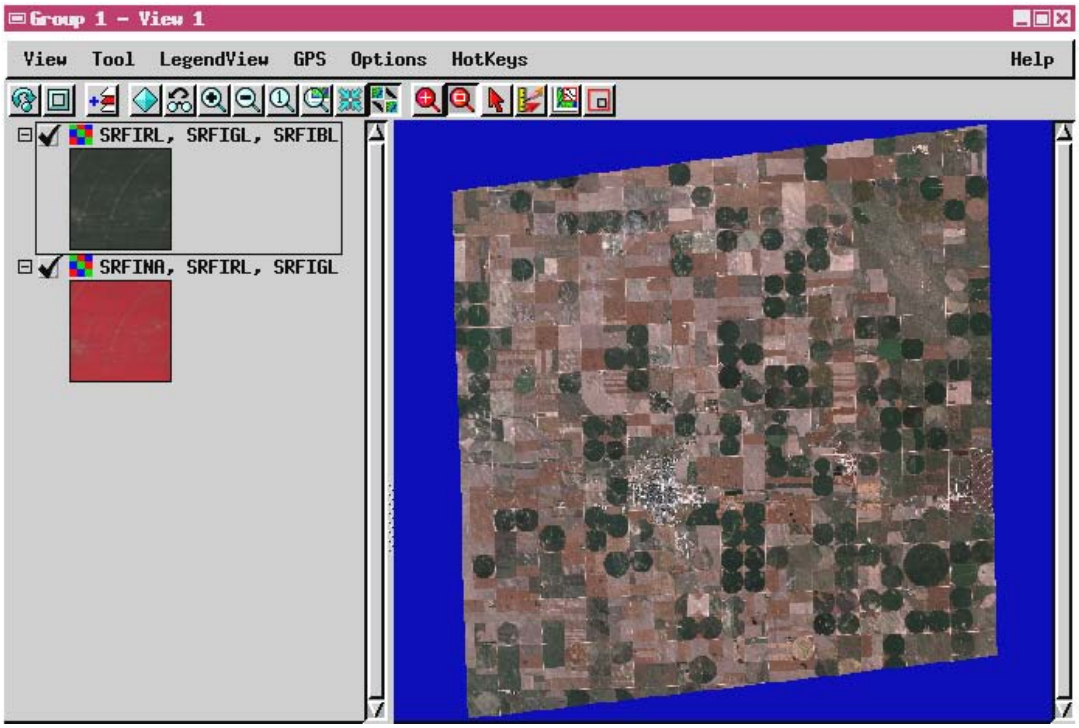
So, this image passes the QC Activity 1 test.

QC Activity 2:

- If you have a SRFIBL raster, then Quick-Add to add another RGB using SRFIRL, SRFIGL, and SRFIBL.

When the author did this with the same Yuma, CO, QB MS SRFI data set, TNTmips produced the display in ***Figure B37b***.

Figure B37b. Natural Color (NC) Combination of SRFI Rasters.



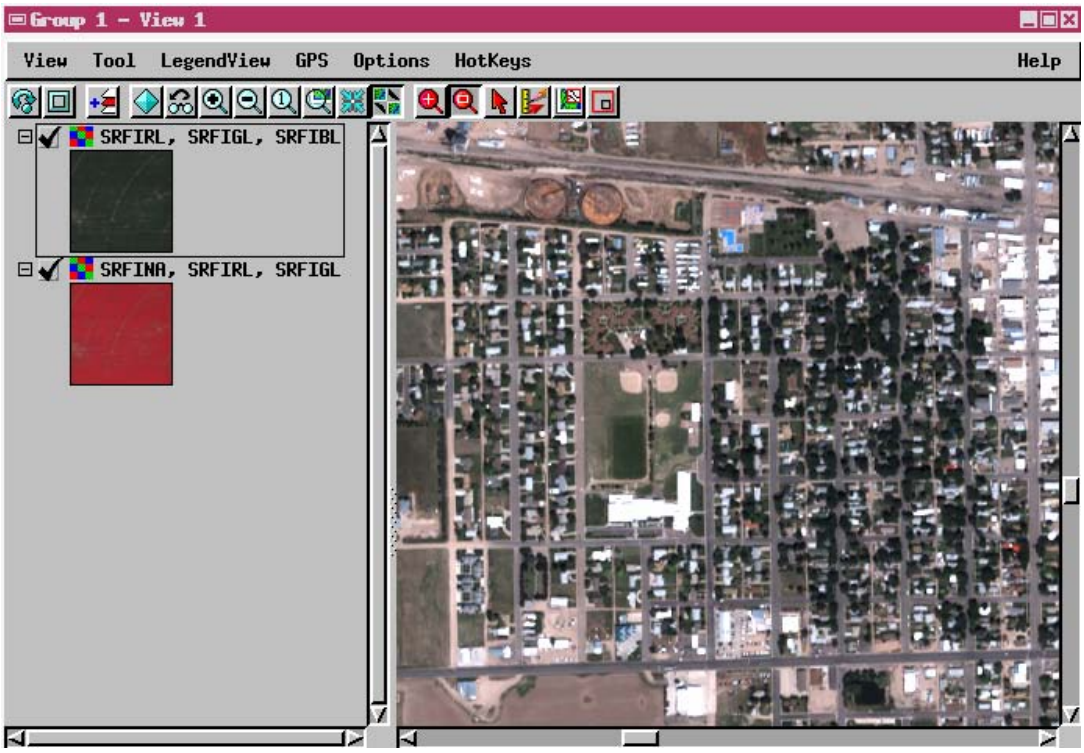
This NC image looks like it present true colors to me. So, it also seems to pass my QC Activity 2 test. But, let's look closer (at full resolution).

Zoom into the urban area of Yuma, CO, (west of center and south of center). You can find some manmade objects there that have a variety of colors that you can use to test the quality of the visible-band SRFI values.

Note that the standard contrast lookup tables included with SRFI.sml are designed to compensate for the fact that skylight and attenuated sunlight are strongest for the BL band than for the RL band. We humans do not see reflectance spectra without the influence of the color of illuminating light. Also, my standard contrast lookup tables have an exponential form that boosts the darker reflectances to compensate for the response of the human eye as it relates to RGB primary colors on your computer's RGB monitor.

Figure B37c shows the computer's monitor display when zoomed in to full resolution.

Figure B37c. Full Resolution View of Yuma, CO, as a Natural Color Display.



This display looks right to me. So, it appears that the standard contrast tables do a good job of displaying SRFI as a NC image. The author calls attention to familiar objects such grass in the parks, trees around the houses, the dirt infield of the softball fields, the blue color of the recreational pools, occasional red roofs in the town, and the white industrial building roof tops. Having the right natural colors with good color saturation probably means that the SRFI rasters are approximately correct.

If you look carefully at this scene, you can see the shadows of tall objects falling in a northwesterly direction. And, you can see the same tall objects appearing to lean toward the south southwest away from the observer.

Since I have a QB PAN image, you can easily verify metadata information regarding the **sun's elevation and azimuth angles** and the **imager's look direction angles** (target azimuth and nadir angle), as seen in the figure on the next page.

Figure B37d. Close-Up (8X) View of Yuma, CO, July 2, 2003, QB PAN.

Sun Azimuth Angle = 128 deg Heading
Shadows fall away from the Sun.
Sun Elevation Angle = 65.34 deg.

Look-Direction Azimuth Angle = 208 deg Heading
Vertical edges of buildings lean away from the observer (QuickBird).
Look-Direction Nadir Angle = 25 deg (from nadir).

These four angles, related two directions (one for the sun and the other for the observer), have significant effects on SRFI. This is the bi-directional reflectance effect.

In the case of this particular combination of sun direction and look direction, the resulting SRFI values are much lower than what they would be if this same scene were viewed at nadir, rather than at 25 deg off nadir looking toward a heading of 208 deg. Many objects in this scene are 3D objects, e.g., buildings, houses, water towers, tall row crops, individual plants, and trees. Thus, they will appear to have relatively-low SRFI values due to the observed shading on significant parts of the object. Shading in this PAN image is obvious for this large building. But, the same shading effects happen for all 3D objects regardless of their size, e.g., individual corn plants, row furrows, and even individual clogs of dirt on rough surfaces.

If the look direction is directly away from the sun, then the imager cannot see the shaded sides of 3D objects. This causes SRFI values to be greater than when the object is viewed at nadir. This is called the “hot spot” effect, which is a feature of bidirectional reflectance distributions with respect to sun angle and look angle.

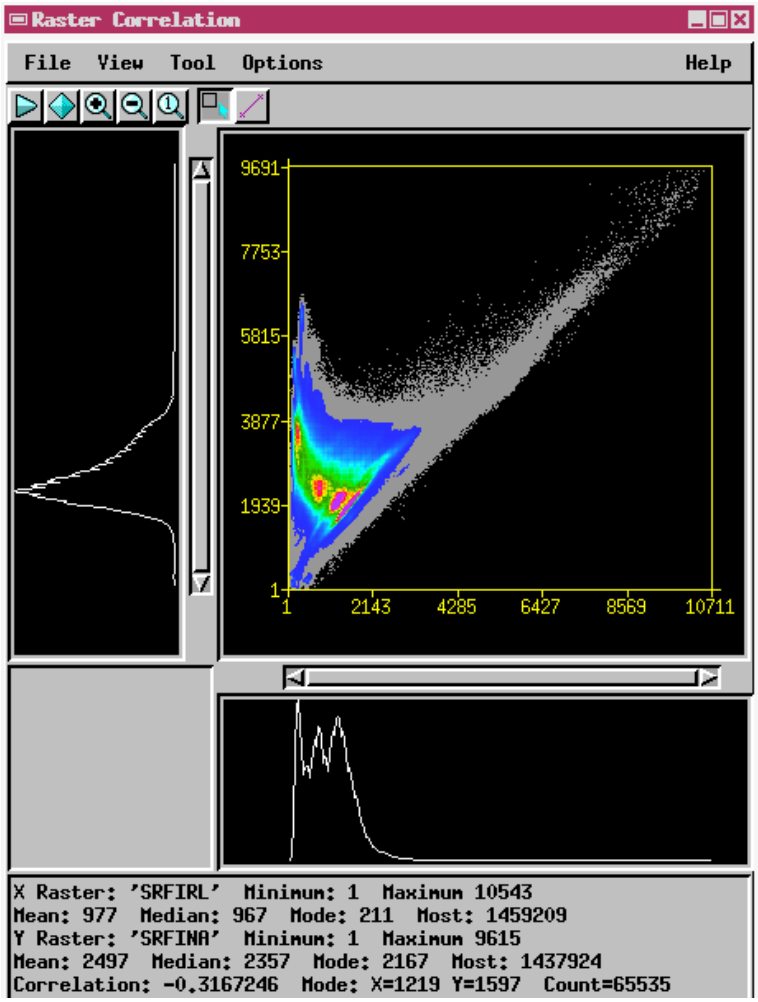
QC Activity 3:

- Quick-Add the PVI raster and the PBI raster to the group.
- If SRFIRL and SRFINA are accurate, then bare soil will have a nearly constant PVI value near 1000 (usually with the range from 950 to 1050).
- In contrast to this narrow range, mixtures of vegetation and soil will have a wide range of PVI values that are spread out from 1000 to 2000 (and perhaps a bit higher than 2000).
- You can move your mouse cursor over the scene to check out the values for PVI (via the DataTips option) as you point to different kinds of land cover.
- If the automatic name of the DataTips of wrong, then change it (to PVI or PBI).
- You can turn off the DataTips for the CIR and NC images.
- There is one crop that has PVI values over 2000 (maximum PVI = 2095).
- There are many bare fields that all have PVI values near 1000. This is true even when the soil brightness (PBI) varies greatly over any of the bare fields.
- In contrast, corn crops have relatively low PVI and low PBI values. This is caused by the presence of stalks and shading effects noted above (bidirectional reflectance effects).

QC Activity 4:

- Another tool that is useful for examining raster data like these is the Raster Correlation tool. It is in the Tools pull-down menu associated with each Raster layer in the Group.
- Select Raster Correlation from the PVI raster Tools menu. The tool comes up with a blank (black) scatterplot area.
- Use the File button to select a New pair of rasters to correlate.
- Assign the X Axis to SRFIRL and the Y Axis to SRFINA.
- After you Click OK, you will soon get a Raster Correlation plot, as displayed on the next page.

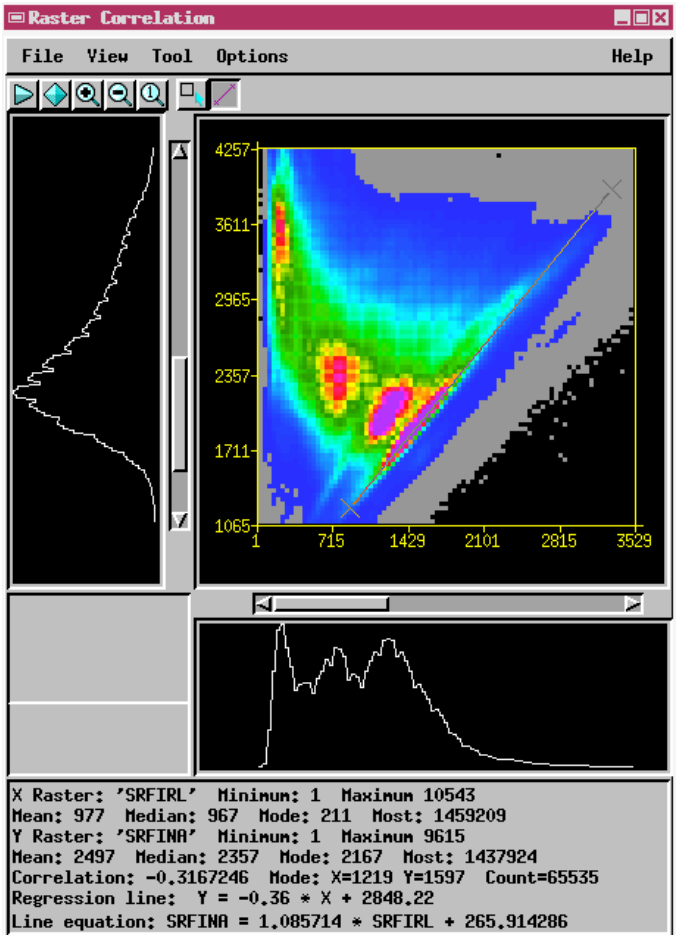
Figure B37e. Scatterplot of SRFINA vs. SRFIRL for Yuma, CO.



The black areas indicate the pairs of SRFINA vs. SRFIRL values that have no occurrences (no hits). The blue area is where there are a few pairs of SRFINA vs. SRFIRL hits. But, most of the hits are in the green and red areas. The density of the correlation plot is greatest in the red area. The shape of the **green & red pattern** has a name: Tasseled Cap (TC). The brim of the TC is the long, skinny red area. This is the domain of bare soil. Most consider this domain to be a line: The Line of Bare Soils. But, this "line" has a narrow, but finite thickness. The tip of the TC is where SRFINA is high and SRFIRL is low (upper left). If you point to this tip, you can see that SRFIRL ≈ 400 and SRFINA ≈ 6400. These SRFI values are equivalent to SRF values of 3% and 64%, respectively. This is similar to what you would expect for dense vegetation.

If you know how to use the Equation tool, you can stretch out a line that runs through the apparent location of the Line of Bare Soils. The Line equation information tells you that on this line: **Line equation: SRFINA = 1.097327 * SRFIRL + 234.885129**

Figure B37f. Scatterplot of SRFINA vs. SRFIRL for Yuma, CO with the Equation Line Enabled and Possitioned as Shown (White Line),



This equation is close to what you would expect from Rondeaux *et al.* (1996), which is

Rondeaux *et al.* Bare Soil Line: SRFINA = 1.086 * SRFIRL + 254

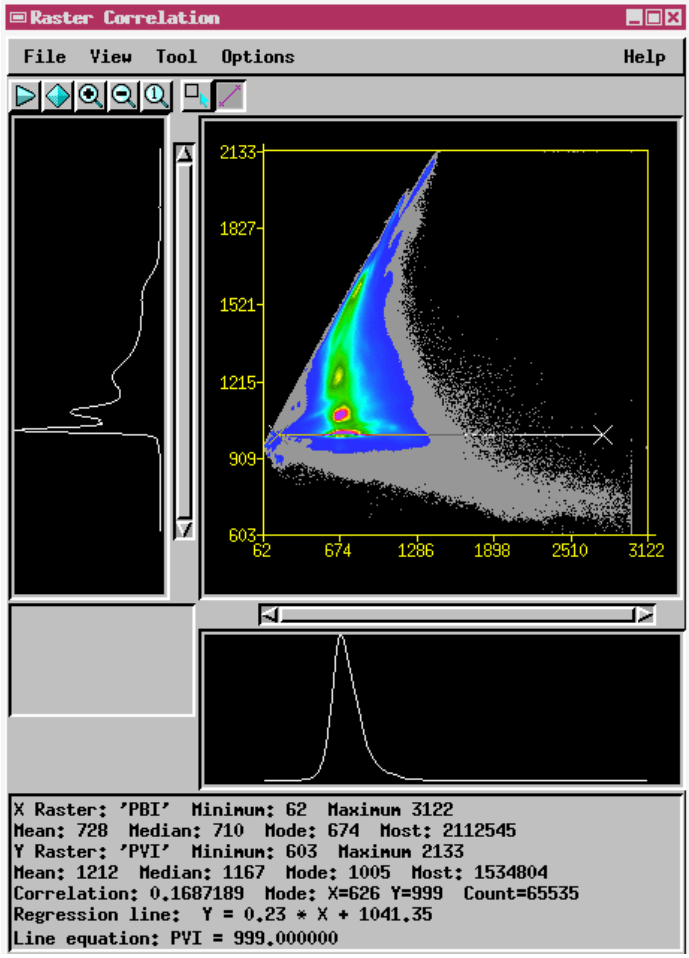
It looks like SRFI.sml did a good job of converting these QB MS DNs to values of QB MS SRFI.

The gaps (variations in data-cloud density) in this 2-Space plot are caused by the expansion from QuickBird DNs (11-bits) to 13 to 14 bits associated with corresponding SRFI values.

QC Activity 4:

- Another correlation plot of interest is when X Axis = PBI and Y Axis = PVI:

Figure B37g. Scatterplot of PVI vs. PBI for Yuma, CO.



As you can see, this looks like the previous Raster Correlation plot, but with a clockwise rotation of 47.36 degrees = atan(1.086). Now, the Line of Bare Soils is defined by PVI = 999. The shape of the Tasseled Cap feature is straight up with a curve to the right for high values of PVI in the PVI vs. PBI plot. The brim is at PVI = 1000.

The Yuma, CO, scene is a highly agricultural scene. It has very little urban, woodland, or open water material cover types. So, the TC distribution and the location of the Line of Bare Soils is easy to see without further processing.

However, in scenes, where these SRFI Feature Space features are not so easy to see, it is necessary to do further processing and analysis with an SML called DIAG.sml.
DIAG.sml will ask you to specify a set of search box parameters to guide it as it isolates "pure" pixels that are related to bare soil and dense vegetation. As

you can clearly see from *Figure B38g*, the Line of Bare Soils is in a search box that is near PVI = 1000 and in a range of PBI from about 400 to 900. Also, dense vegetation exists when PVI is greater than some lower threshold, e.g., PVI > 1500. When you look at a correlation plot of PVI vs. PBI, be sure to note these parameters for the scene that you are analyzing so that you can enter them into DIAG.sml when the script requests this information.

DIAG.sml requires that you make a binary mask raster, called MK, to isolate areas where diagnostic features are likely to exist. You can also exclude areas of no interest or areas when making the binary mask. The author will teach you how to make the MK binary raster in FAQs_by_Jack_C.doc.

B38. Should any Parameters Be Changed and SRFI.sml Be Re-Run?

In general, this is not required. In most cases, you will not have any quantitative data at hand for the scene you are analyzing to help you make quantitative adjustments to producing parameters.

There are some quantitative aspects you can look for. If you look at ***Figure B37e***, you may have noticed that the TC distribution fits against each axis without any gaps. Also, the histograms of the axis values do not pile up against the zero ends. Both of these characteristics indicate that the SRFpath estimates were good for these two MS bands.

In addition, the range of values for SRFINA for the dense-vegetation tip of the TC distribution appears to be about right: That is, SRFINA ranges from near 0 to about 6000 (60% in terms of SRF).

If the upper end of the distribution of SRFINA for dense vegetation is much lower than 6000, you can boost it (in a subsequent use of SRFI.sml) by adopting a value for **`msfac`** that is proportionally higher than 1.0000. Be aware of the fact that some scenes do not contain dense vegetation.

The best situation is where you have independent “ground truth” information about the SRFI spectrum for some object that is much larger than one pixel. This is a very rare situation. But, if you do have data like this, then a better value for msfac would be equal to SRFIgroundtruth / SRFIsfc for the same object (measured at the same time and date and ideally at the same looking angle).

The last indication of quantitative accuracy is the actual location of the Line of Bare Soils versus a pre-conceived notion of its location. This is not a very reliable test as this line will move around with different soils in ways that also depend on the viewing angle and illumination angle. As will be seen in a subsequent SML, it is better to simply note the position of the Line of Bare Soils in the SRFINA vs. SRFIRL frame of reference that was produced by SRFI.sml. This linear feature is the basis for most VIs.

An Analogy

A friend invited the author to a party at the friend's house out in the country northeast of Longmont. The friend gave precise instructions about how to get to his house. He referred to a major road intersection. From there, he said to go north for 2.0 miles. At that point, he said to look for an east-west paved road that was just beyond a silo on the right. Take that road to the east for 0.2 of a mile to the next road that went north. After turning, go to the second house on the right.

So, the friend's instructions were followed. When the trip meter reported exactly two miles from the major road intersection, neither the silo nor the road were seen. So, continuing a short distance up the road, the silo came into view. Then, a few hundred feet further was the east-west road. After turning right, the road to the north was seen a little bit beyond that point. After turning, the friend's house was found.

Getting to the friend's house is a little bit like navigating through reflectance-factor feature space to find a spectral feature at an expected set of reflectance coordinates.

If the friend' distance instructions had been followed precisely, the author would have turned east into a dirt field. His house appeared to be further than 2.0 miles north and a bit more than 0.2 miles east of the starting point. To make matters worse, the author's trip meter is not accurate. It reports distances that are longer than reality. This affects the speed calculations as well.

But, the friend's distance instructions were accurate enough, and the trip meter was accurate enough – to get close enough to the silo, road, and house features. At the end of the trip, the author used these features to find the friend's house. Having an approximately accurate frame of reference and an approximately accurate odometer was good enough to get to the right vicinity. After that, all that was necessary was to recognize the point features (silo and first house on the right) and the linear features (east-west road and first road to the north).

In Texas, directions were less precise. "Go down the road apiece. Turn right, and go a while … until you see my house." This is like trying to find spectral features in DN coordinates. It's much better to have SRFI coordinates. They may not be as precise as true reflectance coordinates. But, SRFI is close enough to get you to your goal. And, the true reflectance coordinates might be wrong also.

B39. Why is REPAIR_IMAGE.sml Necessary?

Raster values range from an allowed minimum value (e.g., 0) to an allowed maximum value (e.g., 255). Integer raster values between (and including) these two extremes:

- Indicate the *relative* brightness of the image in the subject spectral band, *OR*
- Indicate that an actual image-brightness value is absent at that line and column position in the raster.

The latter type of raster value (e.g., 0) is called the null value of the raster (in TNTmips). Raster cells having the null value are ignored by most TNTmips processes.

However, some providers of remotely-sensed imagery mistakenly use 0 as a legitimate image-brightness indicator. If 0 is also the declared null value, then some legitimate image-brightness values will wrongly be ignored by TNTmips. This problem exists for sets of multispectral rasters that include long-wavelength spectral bands such as MB, MC, etc. Path radiances are high enough in short-wavelength spectral bands to force the corresponding image-brightness related raster values to be greater than 0.

REPAIR_IMAGE.sml fixes this problem. It looks for illogical situations where a spectral-band raster has a value of zero in a long-wavelength band and a non-zero value in one or more short-wavelength bands. If a raster cell is truly a null value, it will have the declared null value at the related line and column position for each and every spectral-band raster.

If an errant pixel is detected by REPAIR_IMAGE.sml, the errant raster value (e.g., 0) will be set equal to the lowest allowed legitimate raster value (e.g., 1). Cells having the declared null value in all of the spectral bands will not be changed by this script.

REFERENCES

Chavez, P. S., Jr., 1996: Image-based atmospheric corrections revisited and improved. *Photogrammetric Engineering and Remote Sensing*, 62:1025-1036.

Clevers, J.G.P.W., 1988: The derivation of a simplified reflectance model for the estimation of leaf area index. *Remote Sensing of Environment*, 25, 53-69.

Krause, K., 2004: *Radiance Conversion of QuickBird Data.* Technical Note, www.digitalglobe.com., 4 pp.

Rondeaux, G., M. Steven, and F. Baret, 1996: Optimization of soil-adjusted vegetation indices. *Remote Sensing of Environment*, 55:95-107.

Weast, R. C., ed., 1985: *CRC Handbook of Chemistry and Physics*. 66th Ed., CRC Press, Inc., Boca Raton, FL.