Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Separation of Temperature and Emissivity Algorithm Module

Final Report

Vincent Rivet
AEREX Avionique Inc.
36, du Ruisseau,
Suite 102 Breakeyville,
Quebec, G0S 1E2

AEREX Report Number: 2008-42362-1

Scientific Authority: Pierre Lahaie (418) 844-4000 ext. 4815

Defence R&D Canada – Valcartier
Contract Report
DRDC Valcartier CR 2008-226
March 2008

Canada

AEREX AVIONIQUE INC.

Separation of Temperature and Emissivity Algorithm Module

Final Report

AEREX report number: 2008-42362-1

version: 1.0

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada.

Author: Vincent Rivet — March 27th 2008

Vincent Rivet, Eng. (date)

Approved for publication by: Daniel Pomerleau — March 27th 2008

Daniel Pomerleau, Eng., Ph.D. (date)

President, AEREX Inc.

SOMMAIRE

Ce document présente le résultat des travaux de mise au point de programmes informatiques pour l'élaboration et le test d'algorithmes de séparation température-émissivité. Ces travaux ont été entrepris dans le cadre du contrat numéro W7701-042362/001/QCA intitulée "Amélioration du logiciel STEAM et validation des algorithmes de TES" émis par Travaux publics Canada pour le compte de Recherche et développement pour la Défense Canada – Valcartier.

Les algorithmes de séparation température-émissivité développés précédemment ont été ré-implémentés sous forme d'objets COM. De plus, un objet COM a été développé spécialement pour énumérer les algorithmes de TES présents sur le système.

La nouvelle version de STEAM (v2.1) possède essentiellement les caractéristiques de la version précédente (v1.0), mais l'expérience usager est améliorée : l'interface graphique est plus conviviale, une gestion des données par projet permet de regrouper logiquement les résultats obtenus par les différents algorithmes, et la stabilité du logiciel a été grandement améliorée. De plus, les calculs relatifs à l'atmosphère ont été refaits du début et validés à l'aide de Modtran. Finalement quelques fonctions ont été ajoutées.

EXECUTIVE SUMMARY

This document describes algorithms and software programs developed in the framework of Public Works and Government Services Canada contract number W7701-042362/001/QCA "Improvement of the STEAM software and validation of TES algorithms".

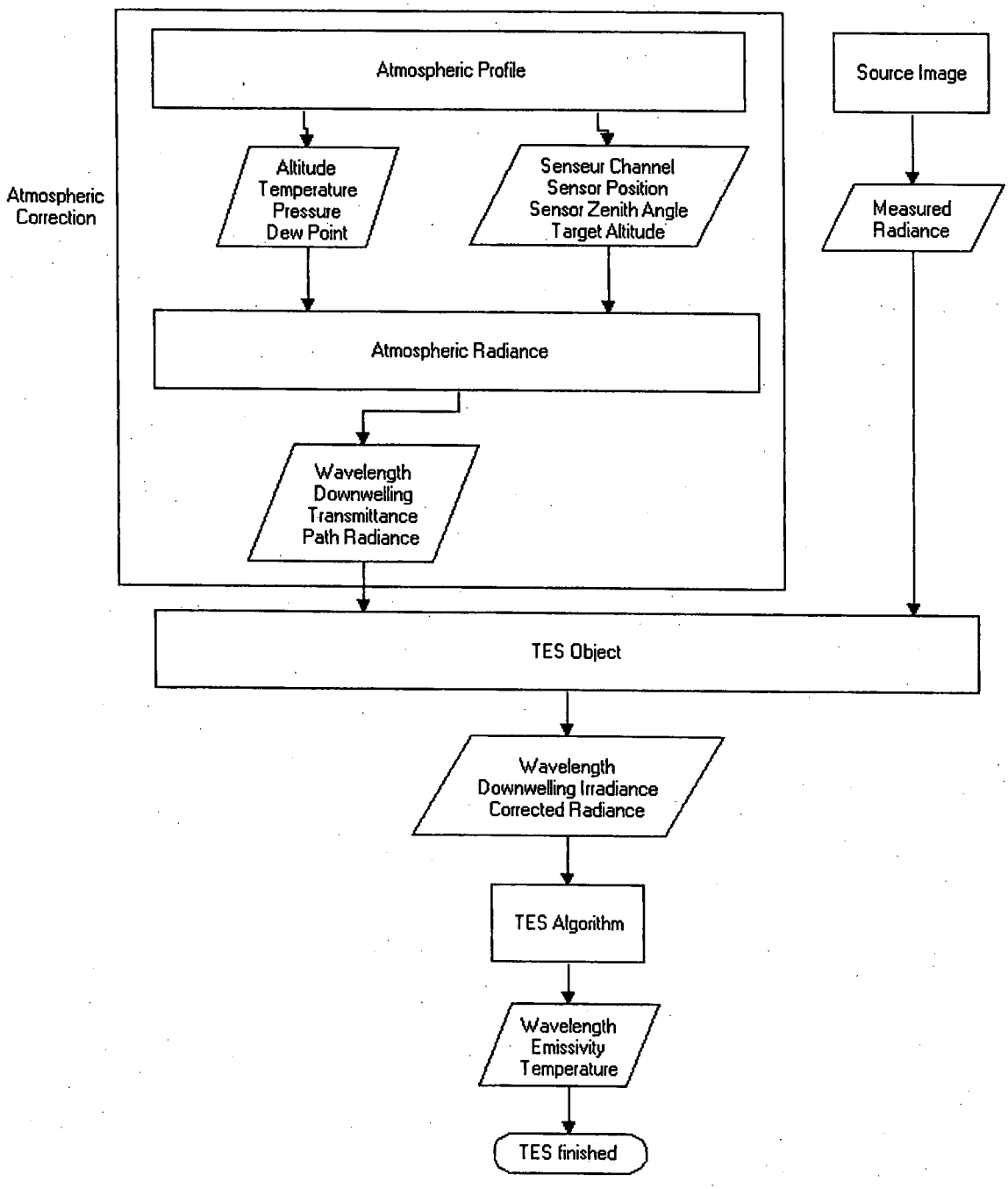
The temperature-emissivity separation algorithms developed previously were re-implemented as COM objects. Also, a different COM object was developed specially to enumerate the TES algorithms available on the system.

The new version of STEAM (v2.1) has mostly the same features as the previous version (v1.0), but the user experience has been improved: the graphical user interface has been made friendlier, project-based data management makes it possible to group results from different algorithms logically, and the stability of the software has been improved. Also atmospheric calculations were redone from scratch and validated with Modtran. Finally, a few functions were added.

Table of Content

1. TES Implementation in STEAM

In STEAM, a temperature-emissivity separation (TES) algorithm requires the corrected land-leaving radiances, the downwelling irradiances and the wavelength range for each of the channel that was used to acquire the source image.



As shown in this flowchart shown above, the source image provides measured radiances (not corrected) and wavelengths, while the atmospheric correction module provides wavelength, downwelling irradiance, transmittance and path radiance. We can calculate

the corrected land-leaving radiance by subtracting the path radiance from the measured radiance and dividing the result by the transmittance:
Corrected land-leaving radiance = (Measured radiance – path radiance)/transmittance.

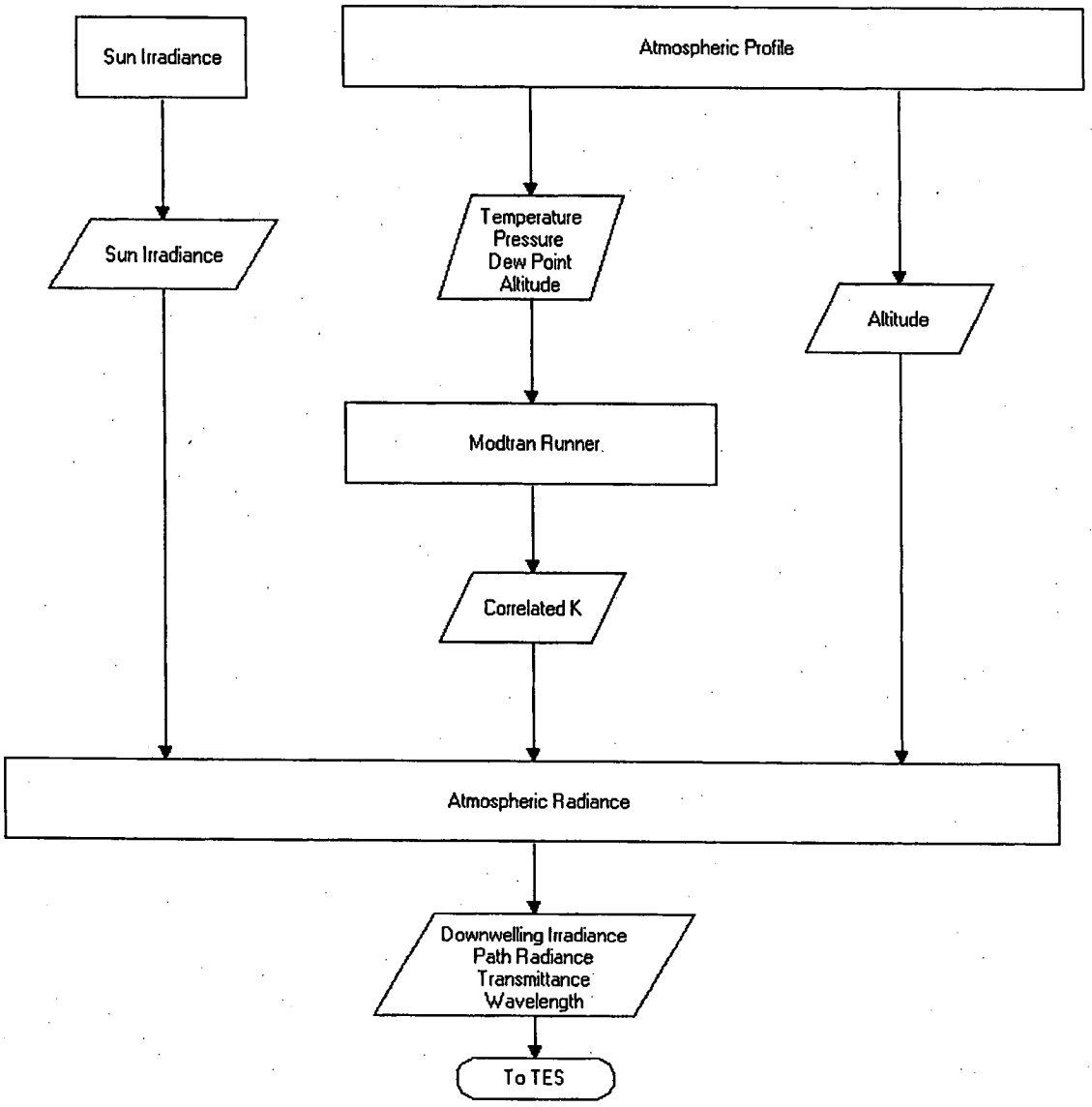
Once the corrected radiance is calculated, STEAM is ready to use the algorithms do to the actual TES. On multiprocessors or hyper-threading processors, STEAM can speed up the calculations by spawning multiple threads to do the calculations. Actually, STEAM spawns as many threads as the number of processors (or 2 for each hyper-threading processors) present on the machine.

When a project is created (i.e. when the first TES is being done), the downwelling irradiance, path radiance and transmittance are saved in a *.rdd* file. When a new TES is performed for the same project, the downwelling irradiance, path radiance and transmittance are retrieved from the file instead of being calculated again, which speeds up calculations even more.

When using the atmospheric profiles method, the profile comes partly from a weather balloon, and partly from a Modtran standard profile. Before atmospheric corrections are calculated, the two profiles are joined, so that it scales from a sea level altitude to 100km. The weather balloon atmosphere is privileged, meaning that if atmospheric data is available from the balloon profile and the standard profile, the balloon profile will be used. For example, if the balloon profile provides data from altitude of 1km to 5 km, the standard profile will be used from 0km to 1km and from 5km to 100km.

1.1 Atmospheric Correction

In STEAM v2.0, the atmospheric correction is the process of using the atmospheric profile to calculate the downwelling irradiance, path radiance and transmittance of the atmosphere at the time the source image was acquired by the sensor. The same method can be used in STEAM v2.1.



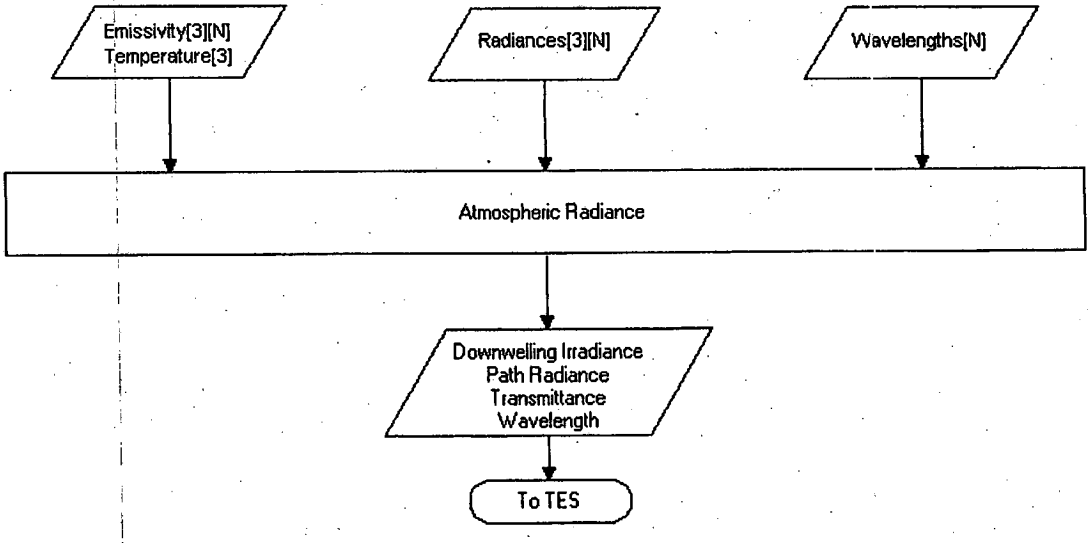
The first step in calculating the downwelling irradiance, path radiance and transmittance is to calculate the correlated-Ks related to the atmosphere. In order to do that, we create a .tp5 file using 60 layers of the atmosphere, from altitude of 0km to 100km, and then we run Modtran. Once Modtran has calculated the correlated-K, we interrupt it and extract the correlated-K.

With the correlated-K and the sun irradiance (file chkur.dat from the modtran directory), we can calculate the transmittance, path radiance and downwelling irradiance the same

way Modtran would. The difference is that it would require 9 runs of Modtran to calculate only the downwelling irradiance, while we use the correlated-K retrieved once to do every calculations.

1.2 ELM Method Implementation

In STEAM v2.1, a new method was implemented for atmospheric correction (note that it is still possible to use the atmospheric correction method described previously). The new method uses three known vectors (vectors for which we know the emissivity and the temperature) to calculate the three parameters. For each of the N wavelengths that will be used in TES, we obtain a "3 equations/3 unknowns" system. Schematically, the system looks like this:



E = Emissivity
R = Radiance
W = Wavelength
B(w, t) = Blackbody Radiance for wavelength w, temperature t
T = Temperature
PR = Path Radiance
DI = Downwelling Irradiance
TR = Transmittance
x = index of substance used (0 = hot water, 1 = cold water, 3 = aluminium)
y = index of wavelength for which we calculate PR, DI and TR.

We use the simplified equation for the measured radiance R:
R[x][y] = PR[y] + TR[y] * (E[x][y]*(B[W[y], T[x]] + (1-E[x][y])DI[y]))

Note that this equation doesn't take noise or radiance emitted by adjacent scene elements into account.

For transmittance we use 2 vectors of the same substance (currently water) with different temperatures. Since emissivity is the same for both vectors, E[0][y] = E[1][y], for every "y". Therefore, we have:

R[0][y] – R[1][y] = TR[y] * E[0][y]* (B[W[y], T[0]] - B[W[y], T[1]])
Or, by isolating transmittance:

TR[y] = (R[0][y] – R[1][y]) / (E[0][y]* (B[W[y], T[0]] - B[W[y], T[1]]))

For downwelling irradiance, we use the third vector, taken from a substance with the lowest possible emissivity. Using the transmittance found previously, we have:

PR[y] + TR[y]*(1-E[3][y]) * DI[y]) = R[3][y] – E[3][y] * B[W[y], T[3]] * TR[y] (Al)
PR[y] + TR[y]*(1-E[0][y]) * DI[y]) = R[0][y] – E[0][y] * B[W[y], T[0]] * TR[y] (H_2O)

Therefore :
R[3][y] – E[3][y] * B[W[y], T[3]] * TR[y] - R[0][y] – E[0][y] * B[W[y], T[0]] * TR[y] =
PR[y] + TR[y]*(1-E[3][y]) * DI[y]) - (PR[y] + TR[y]*(1-E[0][y]) * DI[y])) =
TR[y] * DI[y] * ((1-E[3][y]) – (1-E[0][y])) =
TR[y] * DI[y] * (E[0][y] – E[3][y]).
Or :
DI[y] =
R[3][y] – E[3][y] * B[W[y], T[3]] * TR[y] - R[0][y] – E[0][y] * B[W[y], T[0]] * TR[y] / TR[y] * (E[0][y] – E[3][y])

For path radiance, we already have TR[y] and DI[y], so the system is a simple 1 equation/1 unknown system. Using hot water:
R[0][y] = PR[y] + TR[y] * (E[0][y]*(B[W[y], T[0]] + (1-E[0][y])DI[y]))
becomes :
PR[y] = R[0][y] -TR[y] * (E[0][y]*(B[W[y], T[0]] + (1-E[0][y])DI[y]))

2. Future Algorithms Development

Each algorithm is a COM object that implements the IBaseAlgo interface. The COM object must be registered in the registry as a normal COM object (in HKCR\CLSID\[*CLSID*]). The COM object must use the Apartment threading model.

Also, when the COM server is registered, it must create two registry keys in HKLM\SOFTWARE\Aerex\tes\[*CLSID*]
where [*CLSID*] is the CLSID of the COM object. The two keys are
Description=[Description of the algorithm] and
Name=[Name of the algorithm],
both of type REG_SZ. A code to register and unregister algorithms is provided at annex 1.

2.1 IbaseAlgo Interface

IBaseAlgo is the only required interface for a TES algorithm. A COM server that implements a TES algorithm is free to also implement other interfaces, but STEAM will only use the IBaseAlgo interface to run the algorithm. To implement the IBaseAlgo interface, the following functions must be implemented:

```
interface IBaseAlgo  : public IUnknown
{
public:
  virtual HRESULT STDMETHODCALLTYPE GetNumberOfParams(long* nbParams/*[out]*/)
= 0; // [1]
  virtual HRESULT STDMETHODCALLTYPE Run(double* L/*[in]*/, double* S/*[in]*/,
double* W/*[in]*/, double* E/*[out]*/, double* Temperature/*[out]*/, long
nbBands/*[in]*/) = 0; // [2]
  virtual HRESULT STDMETHODCALLTYPE GetRequiredParam(long paramIndex/*[in]*/,
long* paramType/*[out]*/,  byte* Name/*[out]*/, long strSize/*[in]*/) = 0; //
[3]
  virtual HRESULT STDMETHODCALLTYPE GetDescription(byte*
szDescription/*[out]*/, long strSize/*[in]*/) = 0; // [4]
  virtual HRESULT STDMETHODCALLTYPE GetVersionNumber(double*
versionNumber/*[out]*/) = 0; // [5]
  virtual HRESULT STDMETHODCALLTYPE GetName(byte* szName/*[out]*/, long
strSize/*[in]*/) = 0; // [6]
  virtual HRESULT STDMETHODCALLTYPE SetParam(long index/*[in]*/, byte*
value/*[in]*/, long size/*[in]*/) = 0; // [7]
  virtual HRESULT STDMETHODCALLTYPE GetClassID(byte* pClassID/*[out]*/) = 0;
};
CLSID IID_IBaseAlgo = {0x2C7826B4, 0x57CA, 0x4D8C,{ 0x94, 0x75, 0xC3,0xBC,
0xD3, 0xE1,0x09, 0x33} };
```

- **GetNumberOfParams**: Used to get the number of required parameters for the algorithm.
  - nbParams[*out*]: Number of required parameters for the algorithm.
- **Run**: This is the main function of the object. It calculates TES. The parameters for the function are:
  - L[*in*]: Measured land-leaving radiance, for each sensor channel.
  - S[*in*]: Downwelling irradiance, at each sensor channel.

- W[*in*]: Wavelengths of each channels of the sensor.
- E[*out*]: Calculated emissivities, at each sensor channel.
- Temperature[*out*]: Calculated emperature.
- nbBands[*in*]: Number of channels for the sensor.

- **GetRequiredParam**: Used to get the type of a required parameter.
  - paramIndex[*in*]: Index of the parameter (the valid range is between 0 and the "nbParams" returned by GetNumberOfParams).
  - paramType[*out*]: Type of the parameter. This is a binary mask, with bit 0-3 representing the size of the parameter, bit 4 is for unsigned parameter, bit 5 is for floating point number, and bit 6 is for a variable number of values for the parameter (typically, an array). See annex 2 for a complete description of a parameter type.
  - Name[*out*]: Name of the parameter.
  - strSize[*out*]: Size of the "Name" parameter, in bytes. The COM server should not write anything beyond Name[strSize-1].
- **GetDescription**: Used to get the description of the algorithm.
  - szDescription[*out*]: Description of the algorithm.
  - strSize[*in*]: Size of the "Description" parameter, in bytes. The COM server should not write anything beyond Description[strSize-1].
- **GetVersionNumber**: Used to get the version of the algorithm.
  - versionNumber[*out*]: Version of the algorithm.
- **GetName**: Used to get the name of the algorithm.
  - szName[*out*]: Name of the algorithm.
  - strSize[*in*]: Size of the "Name" parameter, in bytes. The COM server should not write anything beyond Name[strSize-1].
- **SetParam**: Used to set the value of a parameter.
  - index[*in*]: Index of the parameter to set (the valid range is between 0 and the "nbParams" returned by GetNumberOfParams).
  - value[*in*]: New value of the parameter.
  - size[*in*]: Size, in bytes, of the new value of the parameter.
- **GetClassID**: Used to get the CLSID of the algorithm.
  - pClassID[*out*]: CLSID of the COM server.

3. STEAM and ENVI Information Sharing

It is possible to transfer an image from STEAM to ENVI, either a STEAM source image or the result of a TES. Although different processes use different memory spaces, it is possible for different processes to share informations in memory by using "memory-mapped file". The process of sending an image from STEAM to ENVI goes like this:

Action	STEAM	ENVI
1	Create a memory-mapped file.	
2	Write data (image+header) to the memory-mapped file.	
3	Wait until the first 64 bits of the memory-mapped file are all 0s. Check every 0,1 second.	Open the memory-mapped file.
4		Read data (image+header) from the memory-mapped file.
5		Write 0s in the first 64 bits of the memory-mapped file once reading is finished.
6	Close the memory-mapped file.	Display the image.
7	Finished	Finished

An obvious problem may arise: if the user doesn't read the image in ENVI, the last 64 bits of the memory-mapped file won't ever be marked as 0s, and STEAM will be in a deadlock. To solve this problem, STEAM will ask the user if he wishes to abort or retry sending the file to ENVI every 20 seconds until the file has been correctly transferred or the user chooses to abort.

4. File formats

4.1 Image header (.hdr)

This file is identical to an ENVI image header (see ENVI user guide, page 1015), with the following fields added (note that adding lines will not cause any problem for ENVI to open the file):
sensor file: name of the .sen file, either relative or absolute
profile: name of the .spr file for this image, either relative or absolute
sensor altitude: altitude, in kilometers, of the sensor at the time the image was acquired (floating point number)
target altitude: altitude, in kilometers, of the target at the time the image was acquired. This is usually the ground altitude (floating point number)
sensor angle: azimuth angle of the sensor to target, in degrees (i.e. if the sensor is directly above the target, sensor angle is 0 degrees)
scale factor: The factor by which Steam should multiply the values in the data portion of the image to get the real radiance. If this field is not added, the default value is 1.

Note that the wavelengths in the .hdr file are used for display only. For TES calculations, STEAM uses the sensor or the smile file.

4.2 Image data (no extension, .img, .int)

This is the data of the image. For a description, see ENVI user guide, p.43.

4.3 Steam profile (.spr)

This file contains 3 types of data:
Comments: A comment lines begins with a semi-column (;). A comment line is ignored by Steam.
Field=Value: Three values are necessary for a profile to be complete:

1. Latitude: Latitude of the target, in degrees. A positive value is in the northern hemisphere, while a negative value is in the southern hemisphere. The value is a floating point in degrees only, no minute or second. Therefore 18.75 is really 18 degrees and 75/100, not 18 degrees and 75 minutes (which wouldn't make sense anyway).
2. Longitude: Longitude of the target in degrees. A positive value is in the eastern hemisphere, while a negative value is in the western hemisphere. The value is a floating point in degrees only, no minute or second.
3. DateTime: The time and date when the target was acquired. The time must be UTC, with no daylight saving time. The time is 12 hours based. The time and date format must be "yyyy-mm-dd hh:mm:ss AM(PM)", with no quote, and only either AM or PM.

Atmospheric Layer: Each atmospheric layer is made of four space-separated floating-point numbers:

Altitude(km) Temperature(^{0}C) DewPoint(^{0}C) AtmosphericPressure(hPa)

4.4 Sensor (.sen)

A sensor is made of n channel, each containing m_x bands. Each band is made of a wavelength (in cm^{-1}) and a weight. A sensor file therefore begins with the number of channels(n), followed by the channels themselves. Each channel begins with the number of band in the channel (m_x), followed by the wavenumber-weight pairs:

n
m_0
Wavenumber[0] Weight[0]
Wavenumber[1] Weight[1]
[...]
Wavenumber[m_0-1] Weight[m_0-1]
m_1
Wavenumber[0] Weight[0]
Wavenumber[1] Weight[1]
[...]
Wavenumber[m_1-1] Weight[m_1-1]
[...]

4.5 Smile (.hdr)

A smile is a set of n sensor, where n is the number of columns in the source image. Therefore, the smile file is a normal ENVI file (header+data) with 1 row and n columns. In this case, each sensor is made of a single band. The values for each mono-band sensor are in the smile data file.

4.6 Smile data (no extension, .img, .int)

This file contains the sensors for each columns of the source image. For a description, see ENVI user guide, p.43.

4.7 Radiances (.rad)

Since Steam automatically generates radiances files, none should be provided. Radiances files should normally not be modified by hand.

The first line of a radiance file is the number of wavenumbers for which the radiances values were calculated. Each other line of the file is made of four space-separated floating-point numbers:

Wavenumber(cm-1) Path Radiance(W * SR^{-1} * cm^2 / cm^{-1}) Downwelling Irradiance(W * SR^{-1} * cm^2 / cm^{-1}) Transmittance (/1)

A radiance file therefore looks like this:

n
W_0 PR_0 S_0 T_0

W_1 PR_1 S_1 T_1
[...]
W_n PR_n S_n T_n

4.8 Project (.prj)

Steam also generates and modifies project files automatically. Project files should normally not be modified by hand.
A complete project is made of a project file (.prj), and a directory containing source datas, sensor, radiance and atmospheric datas, and results from previous TES. A project named "Hello" for which one TES has been done, using the "Dummy" algorithm, would have the following hierarchy:
Hello.prj
Hello <Directory>

- Source.hdr
- Source
- Sensor.sen
- Hello.rad
- Hello.spr
- Dummy[].hdr
- Dummy[]

A project file in itself contains the following Field=Value pairs:
Project: Name of the project. This is the filename of the project file, with no extension (i.e. if the project file is hello.prj, the project name is hello)
Source: This is the name of the ENVI header file (.hdr) used as a source for TES for the entire project. A project can have only a single source.
Smile: Name of the sensor file (.sen) or smile file (.hdr) used for the project. Since there can only be one source for a project, and each source can only have one sensor or smile file associated to it, a project contains only one sensor or one smile file.
Profil: Name of the atmospheric profile file (.spr) used for the project. Since there can only be one source for a project, and each source can only have one atmospheric profile associated to it, a project contains only one atmospheric profile file.
Radiances: Name of the radiances file (.rad) used for the project. Since there can only be one source for a project, and each source can only have one radiances file associated to it, a project contains only one radiances file.
AlgorithmName[parameters]: A project can contain any number of these fields (at least one). Each pair describes the algorithm used when doing a TES, and the name of the file (ENVI header, .hdr) generated as a result for the TES. By default, the names of the files are always the same as the Algorithm (with parameters between []), mostly because this is convenient when opening the files in an external program (for example, ENVI), but this is not a requirement. For example, a project file could contain the pair "Dummy[]=a.hdr", as long as the a.hdr file (and the data it comes with) are in the ".\project\" directory.

Annex 1: Registering a tes algorithm

```
bool RegisterAlgorithm(CLSID rclsid, const char* szName, const char*
szDescription)
{
        HKEY hHKLM_SOFTWARE;
        if(RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE", 0, KEY_ALL_ACCESS,
&hHKLM_SOFTWARE)!=0)
                return false;

        HKEY hHKLM_SOFTWARE_Aerex_tes;
        if(RegCreateKey(hHKLM_SOFTWARE, "Aerex\\tes",
&hHKLM_SOFTWARE_Aerex_tes) != 0)
                return false;

        HKEY hHKLM_SOFTWARE_Aerex_tes_CLSID;
        wchar_t *wstrCLSID = new wchar_t[128];
        char strCLSID[128];
        StringFromCLSID(rclsid, &wstrCLSID);
        wcstombs(strCLSID, (const wchar_t *) wstrCLSID, sizeof(strCLSID));
        delete [] wstrCLSID;

        if(RegCreateKey(hHKLM_SOFTWARE_Aerex_tes, strCLSID,
&hHKLM_SOFTWARE_Aerex_tes_CLSID))
                return false;
        if(RegSetValueEx(hHKLM_SOFTWARE_Aerex_tes_CLSID, "Name", 0, REG_SZ,
szName, strlen(szName) ))
                return false;
        if(RegSetValueEx(hHKLM_SOFTWARE_Aerex_tes_CLSID, "Description", 0,
REG_SZ, szDescription, strlen(szDescription) ))
                return false;

        RegCloseKey(hHKLM_SOFTWARE);
        RegCloseKey(hHKLM_SOFTWARE_Aerex_tes);
        RegCloseKey(hHKLM_SOFTWARE_Aerex_tes_CLSID);
        return true;
}

bool UnRegisterAlgorithm(CLSID rclsid)
{
        HKEY hHKLM_SOFTWARE;
        if(RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE", 0, KEY_ALL_ACCESS,
&hHKLM_SOFTWARE))
                return false;

        HKEY hHKLM_SOFTWARE_Aerex_tes;
        if(RegOpenKeyEx(hHKLM_SOFTWARE, "Aerex\\tes", 0, KEY_ALL_ACCESS,
&hHKLM_SOFTWARE_Aerex_tes)!=0)
                return false;

        wchar_t *wstrCLSID = new wchar_t[128];
        char strCLSID[128];
        StringFromCLSID(rclsid, &wstrCLSID);
        wcstombs(strCLSID, (const wchar_t *) wstrCLSID, sizeof(strCLSID));
        delete [] wstrCLSID;
        RegDeleteKey(hHKLM_SOFTWARE_Aerex_tes, strCLSID);//don't care if it
doesn't work
        return true;
}
```

Annex 2: Parameter Types

The following table contains the allowed parameter types for an algorithm.

Parameter Type	bits [0..3]	bit 4	bit 5	Data type	Basic C type
0x00000001	1	0	0	signed 8-bits	signed char
0x00000002	2	0	0	signed 16-bits	signed short
0x00000004	4	0	0	signed 32-bits	signed long
0x00000008	8	0	0	signed 64-bits	__int64
0x00000011	1	1	0	unsigned 8-bits	unsigned char
0x00000012	2	1	0	unsigned 16-bits	unsigned short
0x00000014	4	1	0	unsigned 32-bits	unsigned long
0x00000018	8	1	0	unsigned 64-bits	unsigned __int64
0x00000024	4	0	1	simple precision floating point	float
0x00000028	8	0	1	double precision floating point	double

These parameter types are used for single value parameters. If a parameter should contain many values (i.e. an array containing the band numbers to exclude for the algorithm), the same parameter type can be used, but with bit 6 set to 1. For example, to use an array of simple precision floating point values, the parameter type would be:
0x00000024 | 0x00000040 = 0x00000064.

Note that one could easily use binary operators to find informations about a parameter. For example, to find the size (in bytes) of a parameter:
size = type & 0x0000000F

The SetParam function of the IBaseAlgo interface requires a "size" argument, which is defined as the "size, in bytes, of the new value of the parameter". Although the COM server could use the parameter type to figure out its size for a single value parameter, the "size" argument is necessary for multiple value parameters.

Annex 3: Code documentation

Insert CD with all the code documentation here.

Liste de distribution

Document No.: DRDC Valcartier CR 2008-226

1re PARTIE DE LA LISTE : Distribution interne par le Centre :

3 Bibliothèque des documents
1 Pierre Lahaie
1 Jean-Marc Garneau
1 Eldon Puckrin
1 Caroline Turcotte

7 NOMBRE TOTAL – 1ère PARTIE DE LA LISTE

2e PARTIE DE LA LISTE : Distribution externe par la DSIGRD :

1 DRDKIM (fichier PDF)
1 Library and Archives Canada

2 NOMBRE TOTAL – 2e PARTIE DE LA LISTE

9 NOMBRE TOTAL D'EXEMPLAIRES REQUIS

DOCUMENT CONTROL DATA

1. ORIGINATOR (name and address)	2. SECURITY CLASSIFICATION (Including special warning terms if applicable)
Defence R&D Canada Valcartier 2459 Pie-XI blvd. North Quebec City, QC G3J 1X8	Unclassified

3. TITLE (Its classification should be indicated by the appropriate abbreviation (S, C, R or U)
Separation of temperature and emissivity algorithm module - final Report - Version 1.0 (U)

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)
V. Rivet

5. DATE OF PUBLICATION (month and year)	6a. NO. OF PAGES	6b .NO. OF REFERENCES
March 2008	20	0

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. Give the inclusive dates when a specific reporting period is covered.)
Contract report

8. SPONSORING ACTIVITY (name and address)
15ev11 - 15ed01

9a. PROJECT OR GRANT NO. (Please specify whether project or grant)	9b. CONTRACT NO.
15ev11 - 15ed01	W7701-042362/001/QCA

10a. ORIGINATOR'S DOCUMENT NUMBER	10b. OTHER DOCUMENT NOS
CR 2008-226	N/A

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

- ☒ Unlimited distribution
- ☐ Restricted to contractors in approved countries (specify)
- ☐ Restricted to Canadian contractors (with need-to-know)
- ☐ Restricted to Government (with need-to-know)
- ☐ Restricted to Defense departments
- ☐ Others

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)
No limitation

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This document describes software programs developed in the framework of Public Porks and Government Services Canada contract number W7701-042362/001/QCA "Improvement of the STEAM software and validation of TES algorithms".

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Temperature Emissivity separation

Atmospheric compensation

Atmospheric correction

Hyperspectral imaging