TICS DECOMPOSITION

AN ANALYSIS OF EXISTING TICS PYTHON CODE FOR CONVERSION INTO C

Christopher McGeorge

Earth Resources Technology, Inc.

Revision 1.0.1.0

March 31, 2020

TABLE OF CONTENTS

Contents

INTRODUCTION

This document provides an overview of the current TICS Python code for translation into C. **Theater Ionosphere Characterization System (TICS)** is a Python-based application for scaling, interpolating, assimilating, and aggregating empirical ionospheric parameters from digisonde data sources. That is, TICS acquires and processes digisonde data from specified sources and determines the following parameters for a specified location and time:

- The critical frequency (in hertz or megahertz) of the ionospheric F2 layer (f0F2) and the critical frequencies of the F1, E, Es layers (f0F1, foE, foEs)
- The maximum height (in meters or kilometers) of the F2 layer (hmF2) and the height of the E and Es layer peak (hmE, hmEs)
- The lower thickness (in meters or kilometers) (B0)
- The shape parameter (dimensionless) (B1)
- The ionospheric gradients and tilts (beta_lat and beta_lon), which are essentially normalized gradients in foF2

Figure 1 shows sample parameter values that TICS provides.

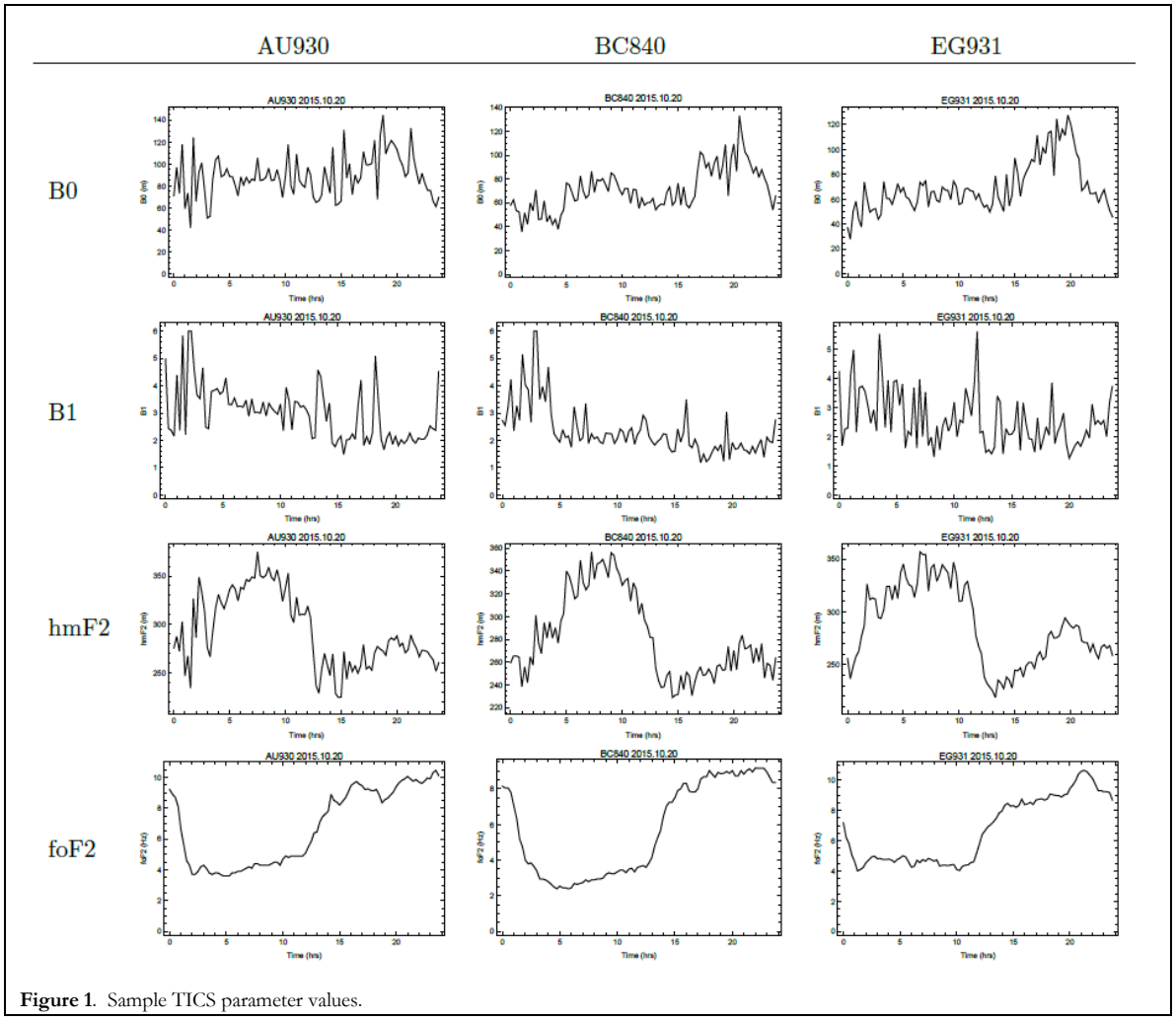


Figure 1. Sample TICS parameter values.

Figure 2 shows the flow of parameters in TICS (other sections in this document will describe the components that this figure shows):

- TICS (or more specifically the RIPE component of TICS) interpolates foF2, hmF2, B0, B1 using digisonde data
- TICS (or more specifically the ROAM component of TICS) calculates beta_lat and beta_lon
- TICS uses digisonde data directly for foEs and hmEs
- TICS uses IRI values for foE and hmE
- TICS currently uses IRI values for also foF1 but will eventually use values from digisonde data (either vertical or oblique)

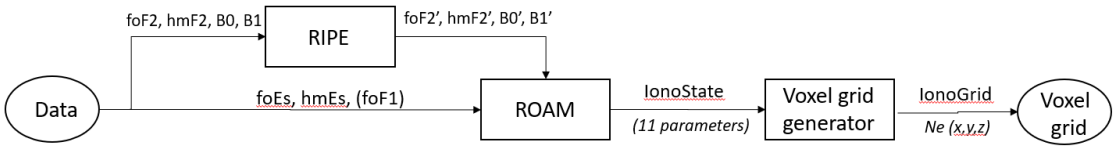


Figure 2. Flow of parameters in TICS.

Figure 3 shows a high-level class diagram of the existing TICS Python code.

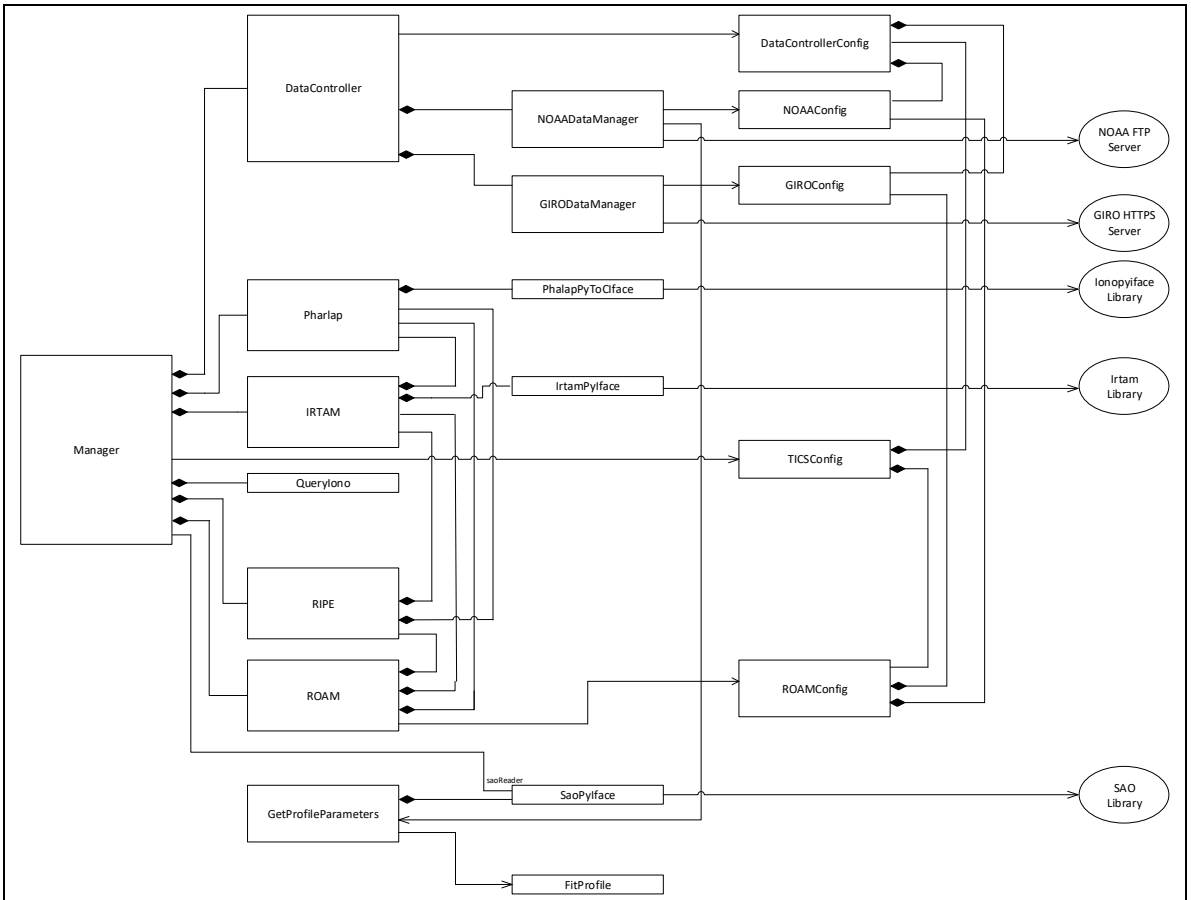
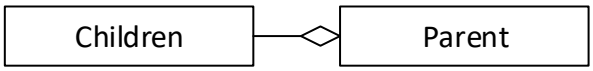


Figure 3. High-level class diagram showing the existing components of TICS.

The class diagram in Figure 3—and the other class diagrams throughout this document—use the following conventions:

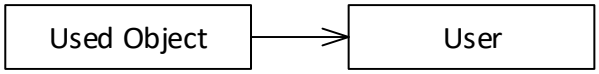
- A square indicates a class in the TICS code.
- An ellipse indicates a component, such as a server or a library, that is outside of TICS.
- An arrow that ends with a black diamond indicates a "parent-child" relationship (composition) between classes; that is, an instance of a child class belongs to an instance of a parent class:

- An arrow that ends with a white diamond indicates parent-child relationships (aggregations) with multiple children instances:



  (Although Figure 3 does not show these types of relationships, other class diagrams throughout this document do.)

- A pointed arrow indicates that an instance of one class uses an instance of another class, though does not own an instance as in a parent-child relationship):



The following sections in this document describe the classes in detail. The last section in this document provides an estimation of the effort to conversion of the TICS Python code into C.

IRI

The **International Reference Ionosphere (IRI)** is a project for providing monthly averages of ionospheric electron density, electron temperature, ion temperature, and ion composition for a given location, time, and altitude. Figure 4 shows sample IRI data.

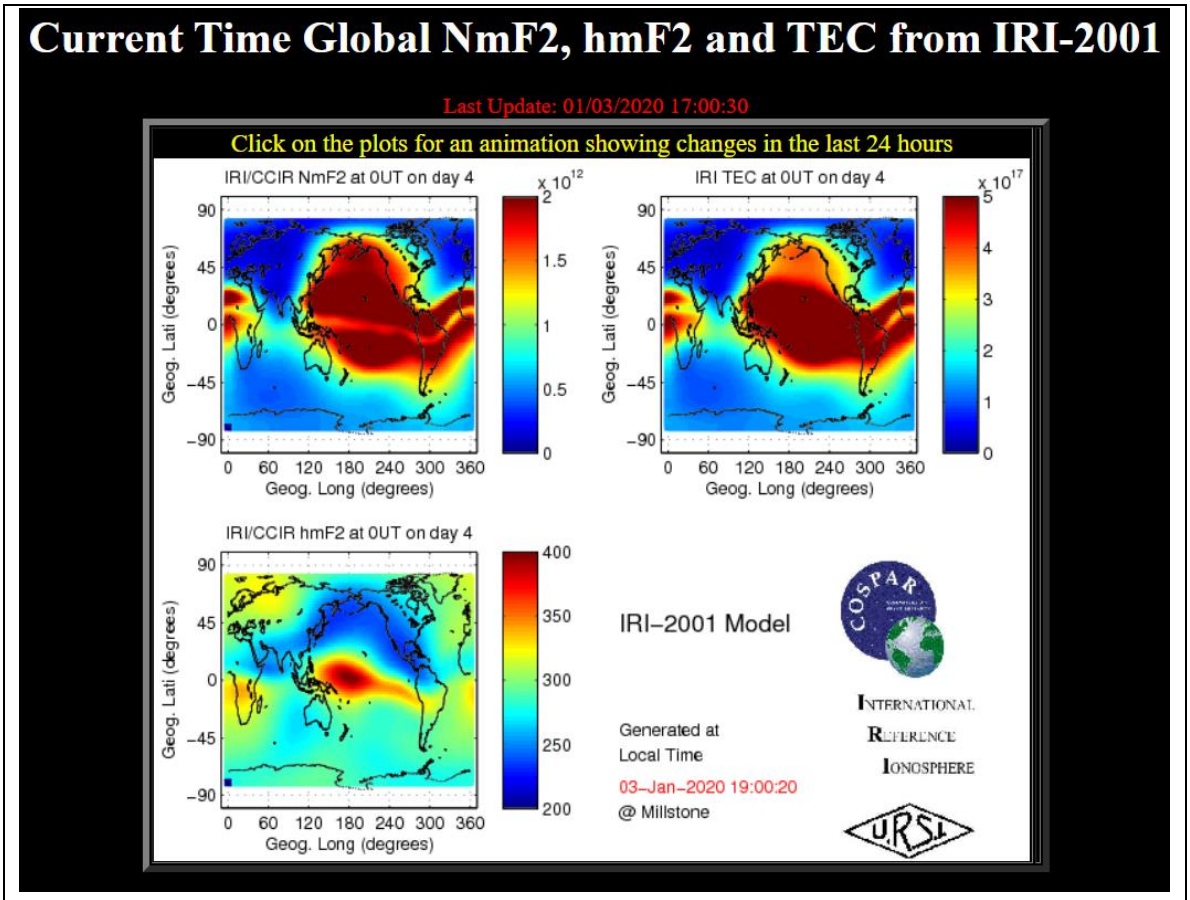


Figure 4. Sample IRI data.

Global Ionosphere Radio Observatory (GIRO) is a data center that provides a digisonde network from which IRI obtains its data. The **IRI-Based Real-Time Assimilative Model (IRTAM)** continually assimilates IRI data for f0F2, hmF2, B0, and B1 values. Figure 5 shows sample IRTAM dataset.

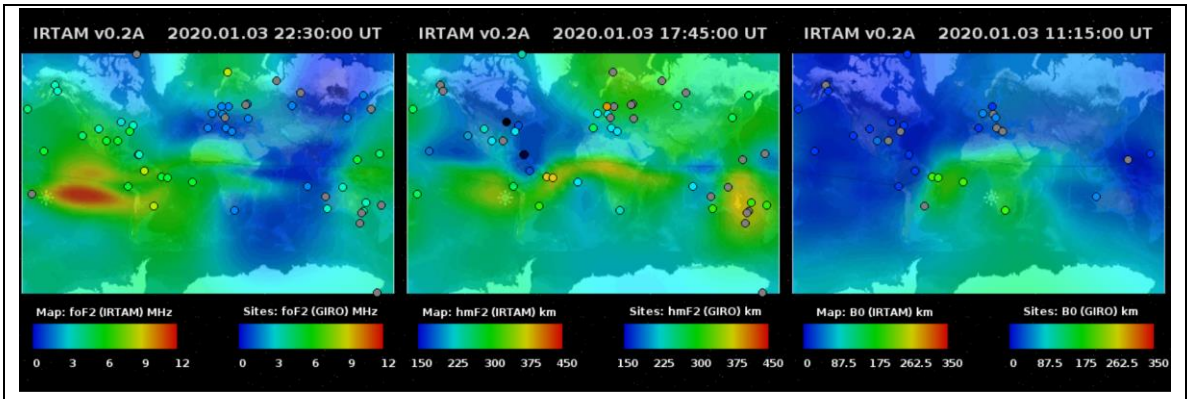


Figure 5. Sample IRTAM data.

TICS reads IRTAM data as text files. Figure 6 shows the IRTAM-related classes in TICS. As Figure 6 shows, the `IRTAM` class outputs f0F2, f0F2, hmF2, B0, and B1 values for a given latitude, longitude, and time, using specified IRTAM files. To provide these values, the `IRTAM` class uses an external library (`libirtampy_lin.so`) to process read IRTAM data. The `IrtamPyIface` and `processirtam` classes provide a way for the TICS Python code to interact with the C code in the external library.

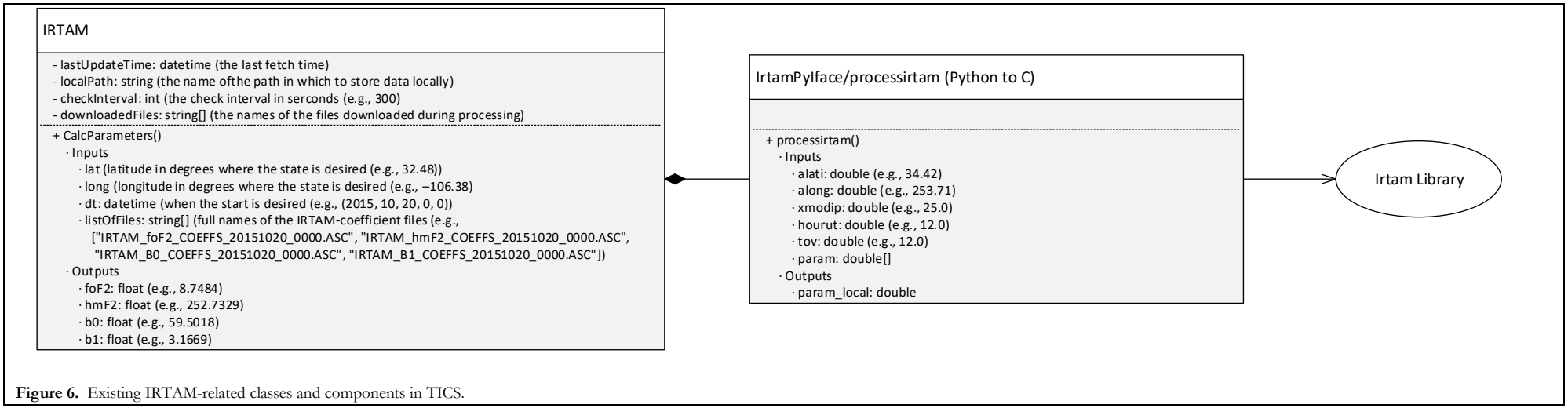


Figure 6. Existing IRTAM-related classes and components in TICS.

Provision of High-Frequency Raytracing Laboratory for Propagation Studies (PHaRLAP) is a toolkit for the modeling of high-frequency radio waves in the ionosphere. TICS uses PHaRLAP to obtain and process data from two sources: IRI (which the previous section described) and IGRF.

International Geomagnetic Reference Field (IGRF) is a series of mathematical models for describing the magnetic field of the Earth. The IGRF data that PHaRLAP uses comes from the **National Oceanic and Atmospheric Administration (NOAA)**, which has a center that provides public access to geophysical data.

NOAA file names have the naming convention *XXXXX*`_NOAA.TXT`, where *XXXXX* is the five-character International Union of Radio Science (URSI) code for the station. Figure 7 shows sample data from a NOAA file.

yyyy.MM.dd	(DDD)	HH:mm:ss	C-score	foF2	foF1	foE	foEs	h'Es	hmF2	hmF1	hmE	B0	B1	D1
2015.10.20	(203)	00:00:00	100	9.212	---	---	---	---	275.3	---	---	71.2	4.99	---
2015.10.20	(203)	00:15:00	100	8.900	---	---	---	---	287.4	---	---	96.9	2.44	---
2015.10.20	(203)	00:30:00	100	8.700	---	---	---	---	272.2	---	---	73.5	2.37	---
2015.10.20	(203)	00:45:00	100	8.100	---	---	---	---	302.7	---	---	117.9	2.16	---
2015.10.20	(203)	01:00:00	100	6.600	---	---	---	---	246.6	---	---	59.3	4.39	---
2015.10.20	(203)	01:15:00	100	5.600	---	---	---	---	267.1	---	---	73.6	2.37	---
2015.10.20	(203)	01:30:00	100	4.600	---	---	---	---	233.9	---	---	42.1	5.84	---
2015.10.20	(203)	01:45:00	100	4.300	---	---	---	---	326.6	---	---	124.0	2.21	---
2015.10.20	(203)	02:00:00	100	3.700	---	---	---	---	286.1	---	---	66.1	6.00	---
2015.10.20	(203)	02:15:00	100	3.700	---	---	---	---	348.8	---	---	92.6	6.00	---
2015.10.20	(203)	02:30:00	100	3.900	---	---	---	---	334.0	---	---	101.1	4.59	---
2015.10.20	(203)	02:45:00	100	4.200	---	---	---	---	314.1	---	---	77.0	3.66	---
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2015.10.20	(203)	20:15:00	100	9.650	---	---	---	---	277.2	---	---	104.4	1.81	---
2015.10.20	(203)	20:30:00	100	9.800	---	---	---	---	280.4	---	---	98.9	2.02	---
2015.10.20	(203)	20:45:00	100	9.900	---	---	---	---	270.5	---	---	88.6	1.89	---
2015.10.20	(203)	21:00:00	100	10.050	---	---	---	---	273.7	---	---	92.1	2.01	---
2015.10.20	(203)	21:15:00	100	9.900	---	---	---	---	288.9	---	---	132.7	2.27	---
2015.10.20	(203)	21:30:00	100	9.750	---	---	---	---	280.6	---	---	105.7	2.01	---
2015.10.20	(203)	21:45:00	100	9.850	---	---	---	---	275.1	---	---	93.1	1.90	---
2015.10.20	(203)	22:00:00	100	9.700	---	---	---	---	266.3	---	---	81.8	2.09	---
2015.10.20	(203)	22:15:00	100	9.600	---	---	---	---	273.1	---	---	91.7	2.04	---
2015.10.20	(203)	22:30:00	100	9.750	---	---	---	---	272.7	---	---	82.8	2.05	---
2015.10.20	(203)	22:45:00	100	9.950	---	---	---	---	271.4	---	---	76.1	2.19	---
2015.10.20	(203)	23:00:00	100	9.900	---	---	---	---	266.9	---	---	76.1	2.53	---
2015.10.20	(203)	23:15:00	100	10.300	---	---	---	---	263.1	---	---	66.1	2.43	---
2015.10.20	(203)	23:30:00	100	10.450	---	---	---	---	251.5	---	---	61.4	2.37	---
2015.10.20	(203)	23:45:00	100	10.113	---	---	---	---	260.3	---	---	69.8	4.52	---

Figure 7. Sample data from a NOAA file.

Figure 8 shows the PHaRLAP-related classes in TICS. As Figure 8 shows, the `Pharlap` class (which is a TICS Python class that acts as a "façade" to the external PHaRLAP library and is not a part of the external library itself) provides methods for obtaining IGRF and IRI data (and also for tracing rays, though TICS does not appear to use the ray-tracing capabilities of PHaRLAP directly). To provide this data, the `Pharlap` class uses an external library (`libionopyiface_lin.so`) to process read data. The `PharlapPytoCIface` and `IonoPyIface` classes provide a way for the TICS Python code to interact with the C code in the external library.

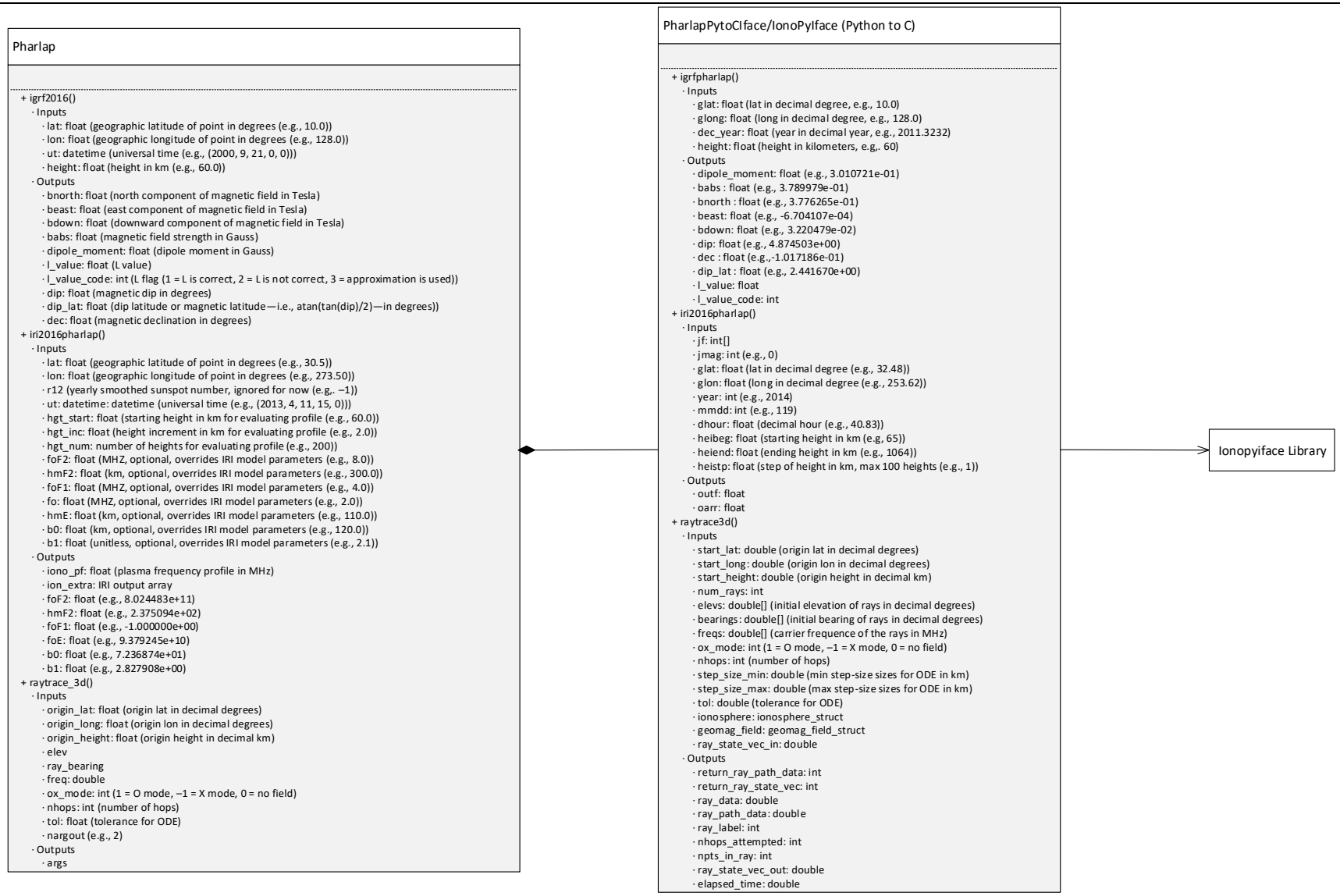


Figure 8. Existing PHaRLAP-related classes and components in TICS.

The **Regional Ionospheric Profile Estimation (RIPE)** component reads IRI data—through PHaRLAP—and IGRF data and interpolates f0F2, hmF2, B0, and B1 values for a given location and time. This component consists of the `RIPE` class, which Figure 9 shows.

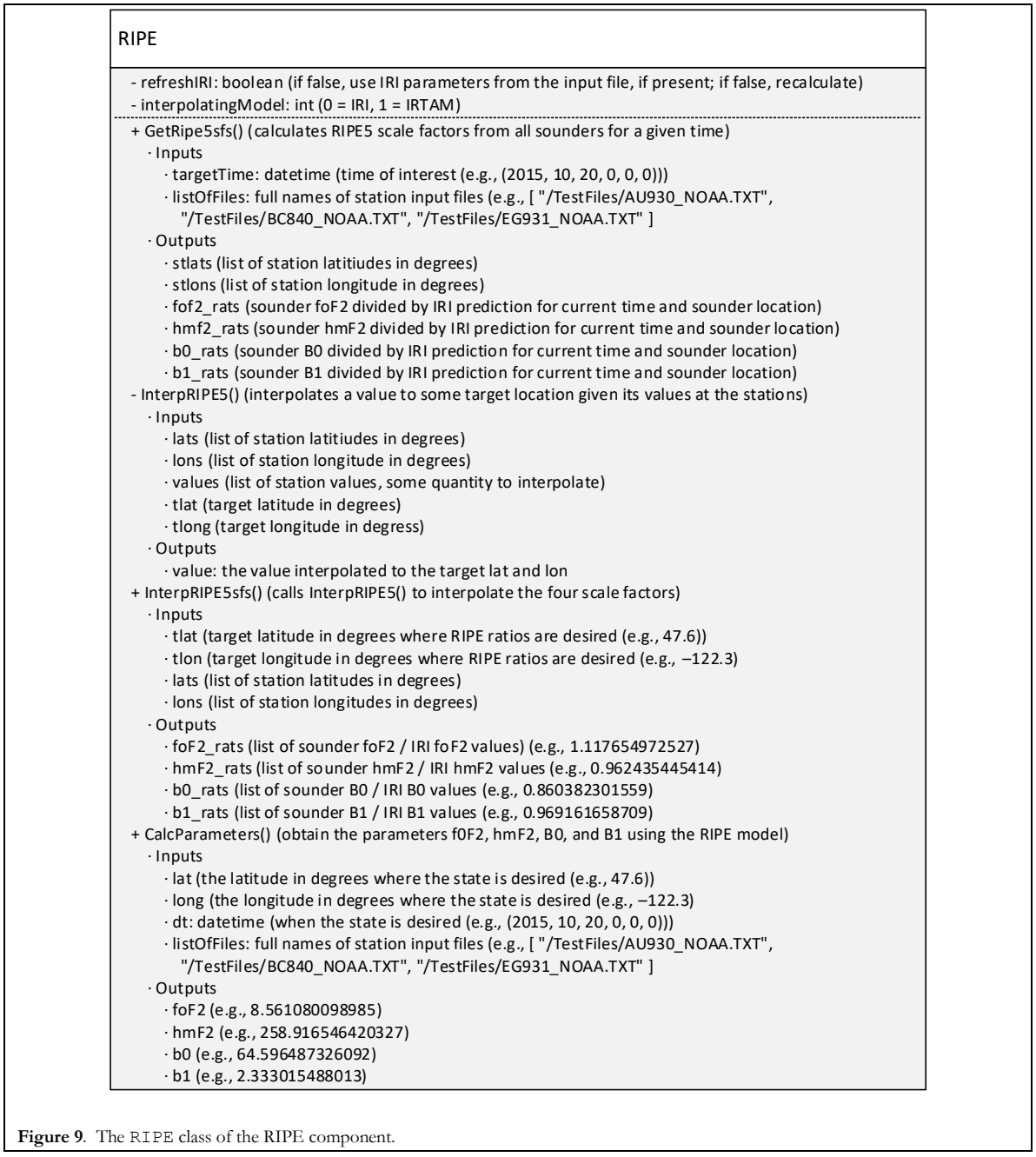


Figure 9. The `RIPE` class of the RIPE component.

Figure 10 shows how the `RIPE` class acquires and interpolates the values.

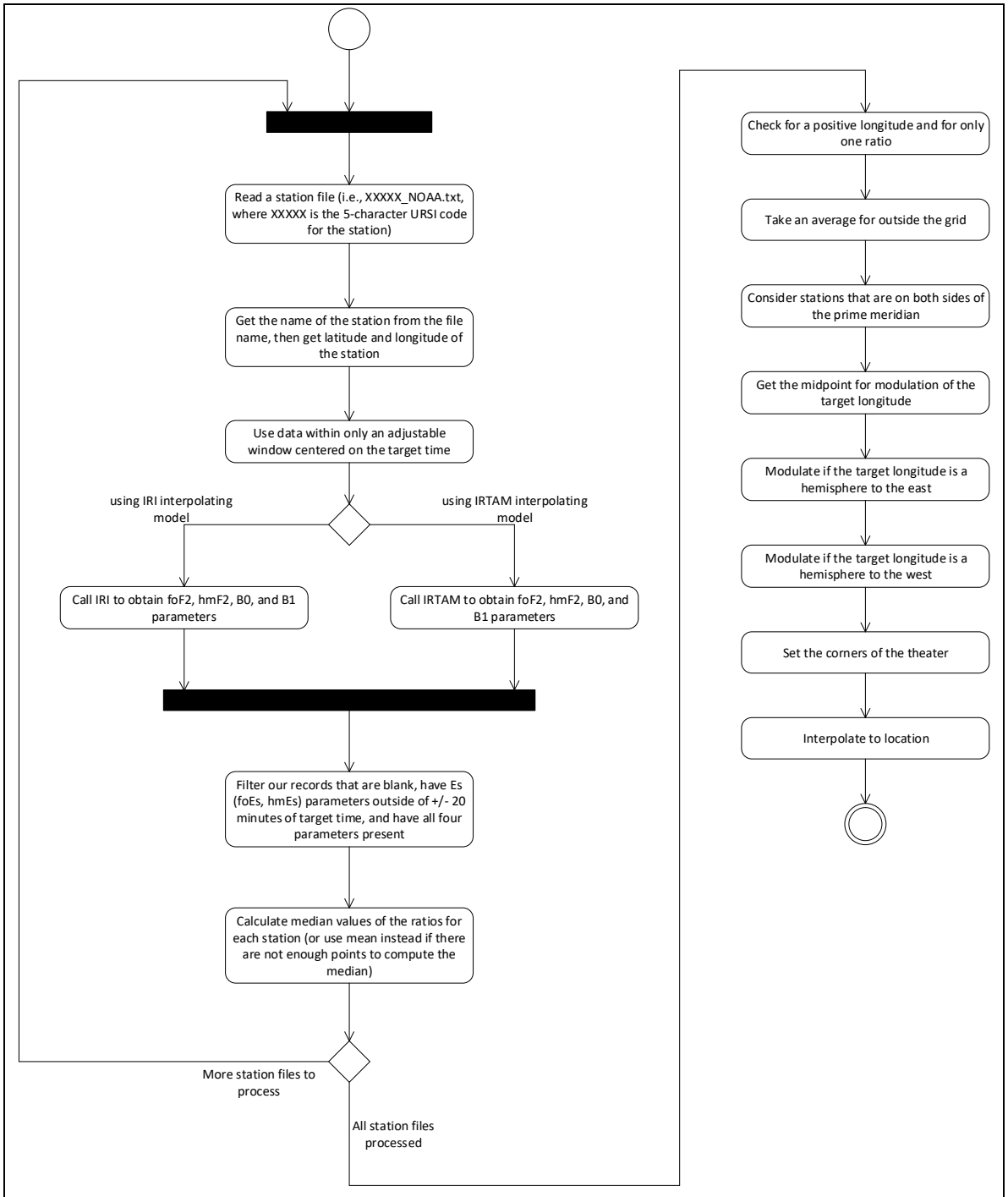


Figure 10. Diagram showing how RIPE acquires and interpolates f0F2, hmF2, B0, and B1 values.

Regional Optimal Assimilation Model (ROAM) is a component of TICS that refines ionospheric estimates and infers local gradients. This component consists of the ROAM class, which Figure 11 shows.

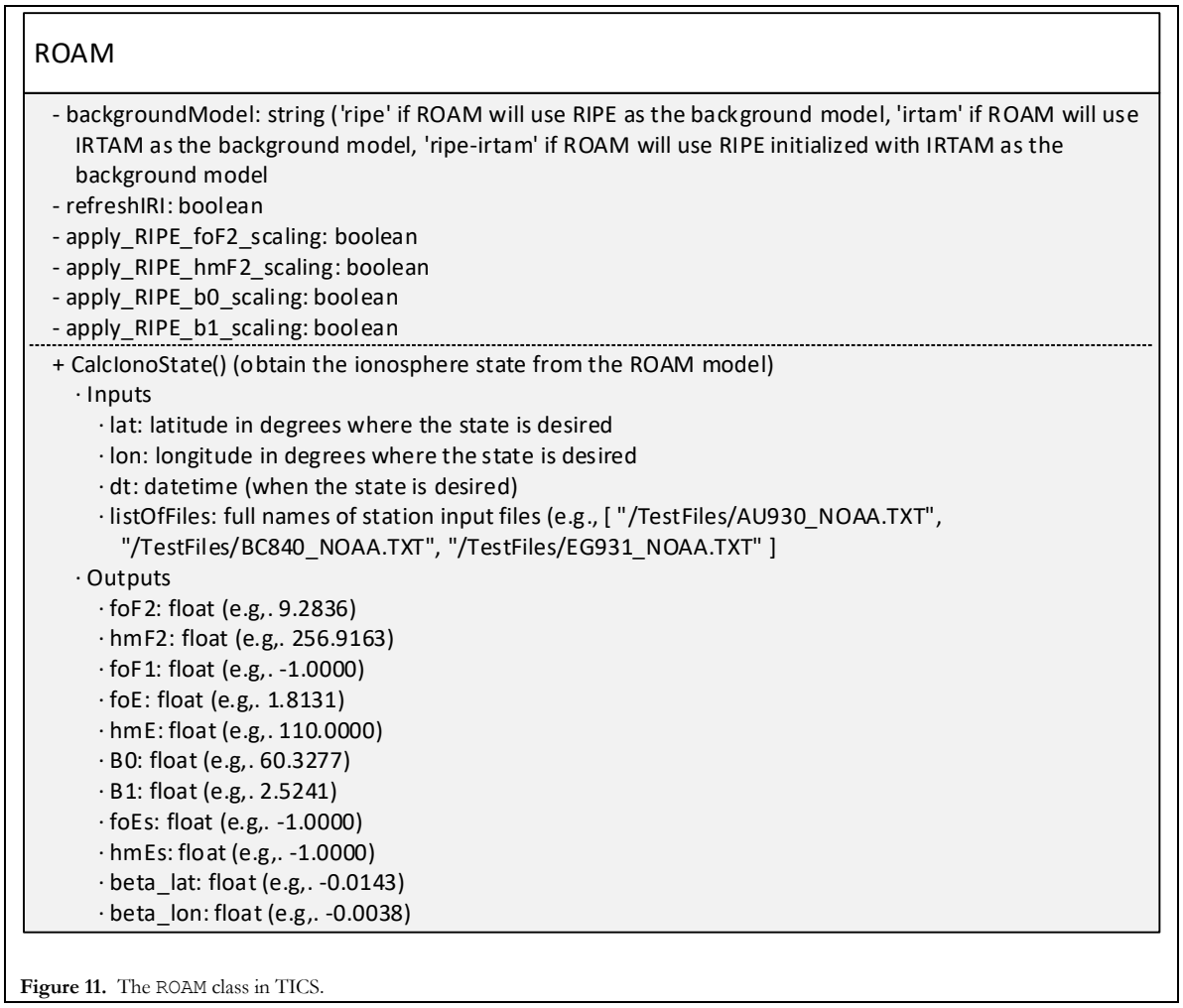


Figure 11. The ROAM class in TICS.

Figure 12 shows how ROAM works. As this figure shows, ROAM can work in three modes—that is, it can use any of three "background models:"

1. RIPE
2. IRTAM
3. TICS initialized with IRTAM

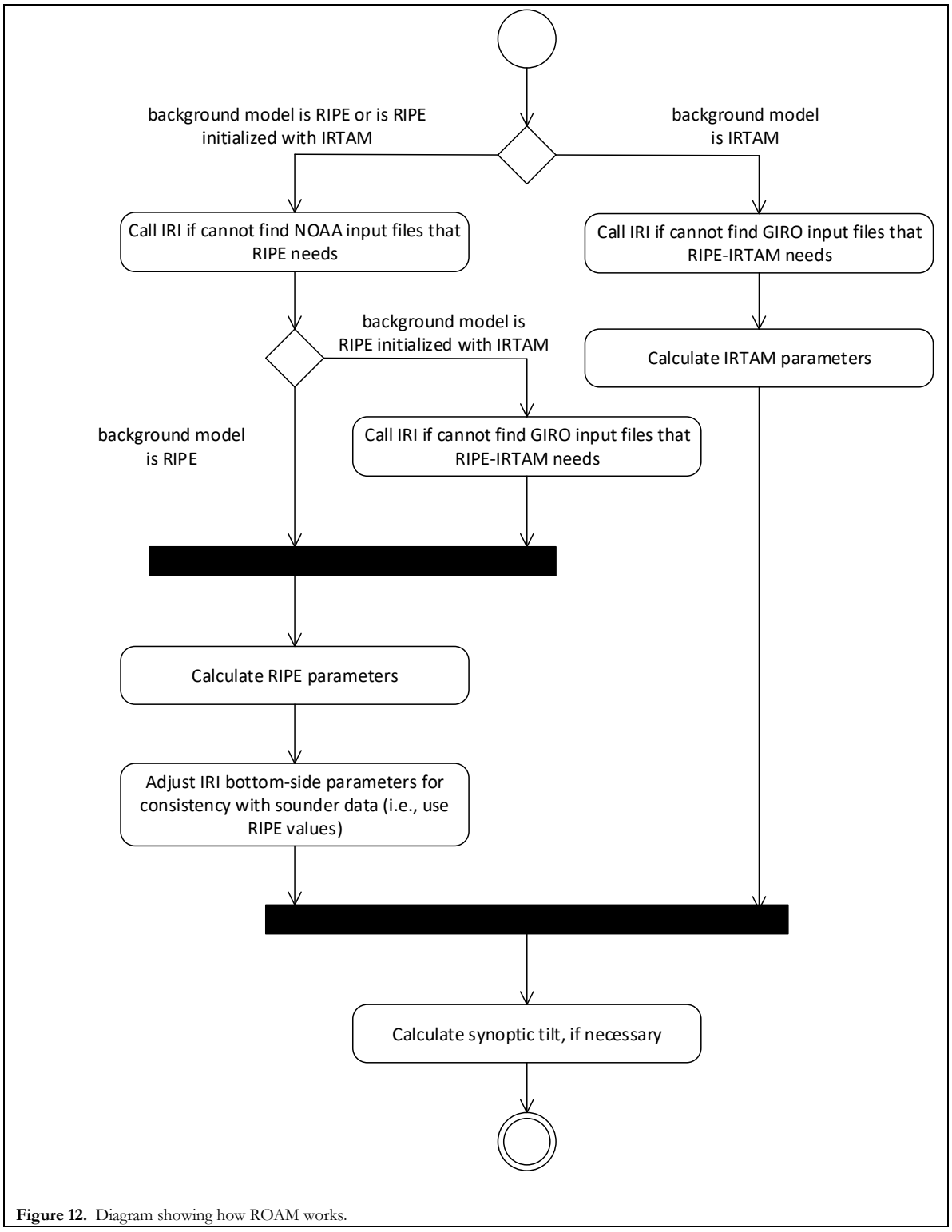


Figure 12. Diagram showing how ROAM works.

PROFILE FITTING

Figure 13 shows the profile-fitting component of TICS. As this figure shows, the `GetProfileParameters` class reads SAO data, does profile fitting—using the `FitProfile` class—, and saves the resulting parameters to output text files. **Standard Archiving Output (SAO)** data capture large-scale oscillations in ionospheric parameters. TICS reads SAO data as text files, using an external library (`libsaopy_lin.so`). The `SaoPyIface` class provides a way for the TICS Python code to interact with the Fortran code in the external library.

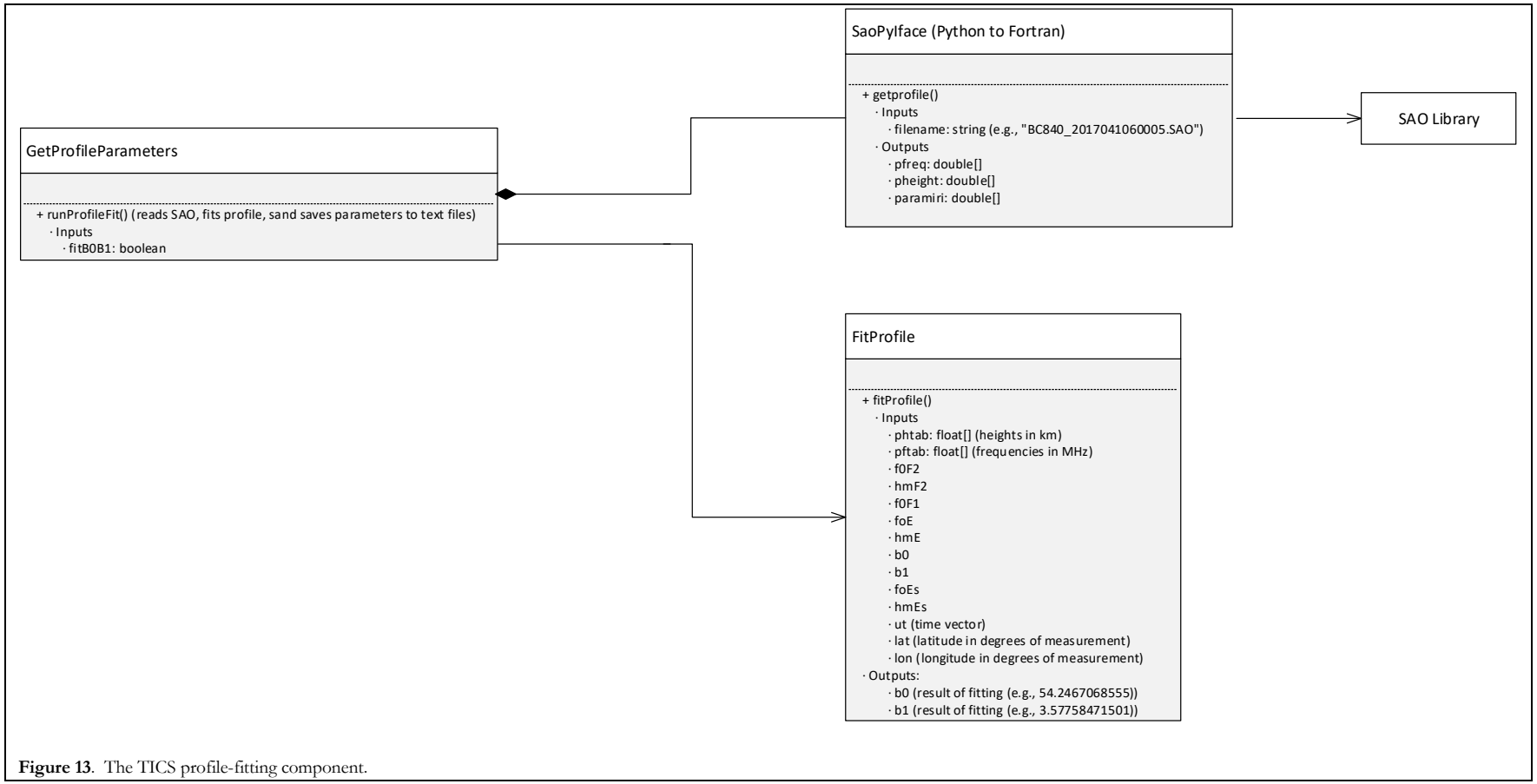


Figure 13. The TICS profile-fitting component.

TICS APPLICATION

Figure 13 shows the TICS-application component, which makes up the main executable portion of the TICS Python code. This component consists also of configuration classes that control various settings in other components in TICS and data-controller classes that fetch data from external sources. Figure 14 shows the configuration and data-controller classes.

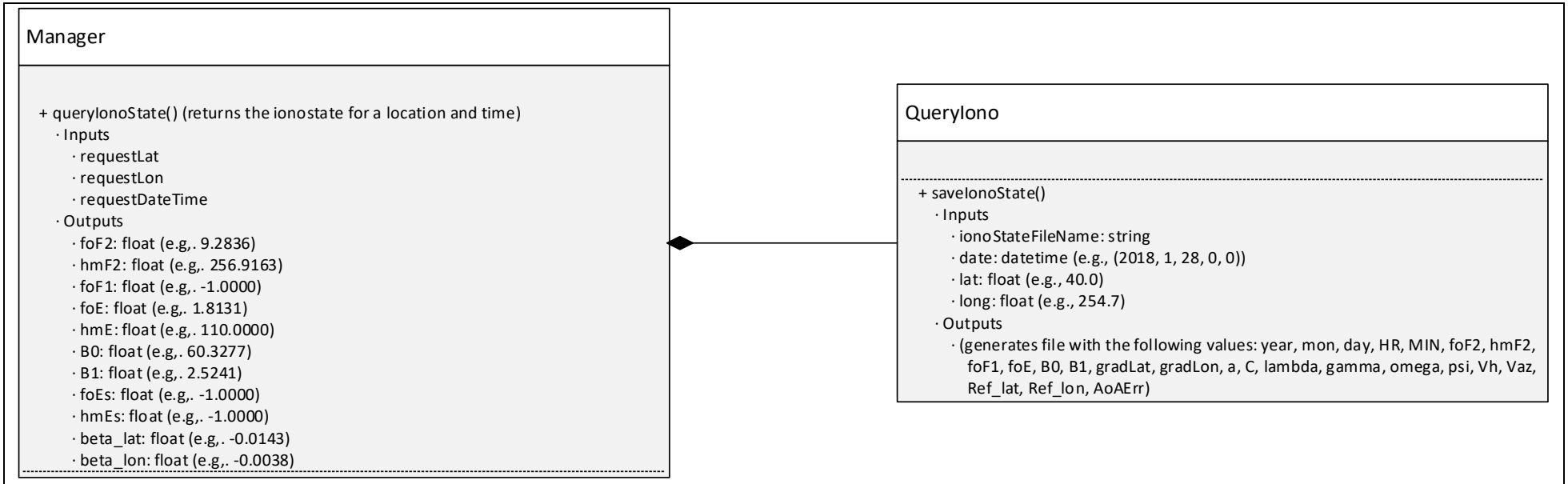


Figure 13. The main application component of TICS.

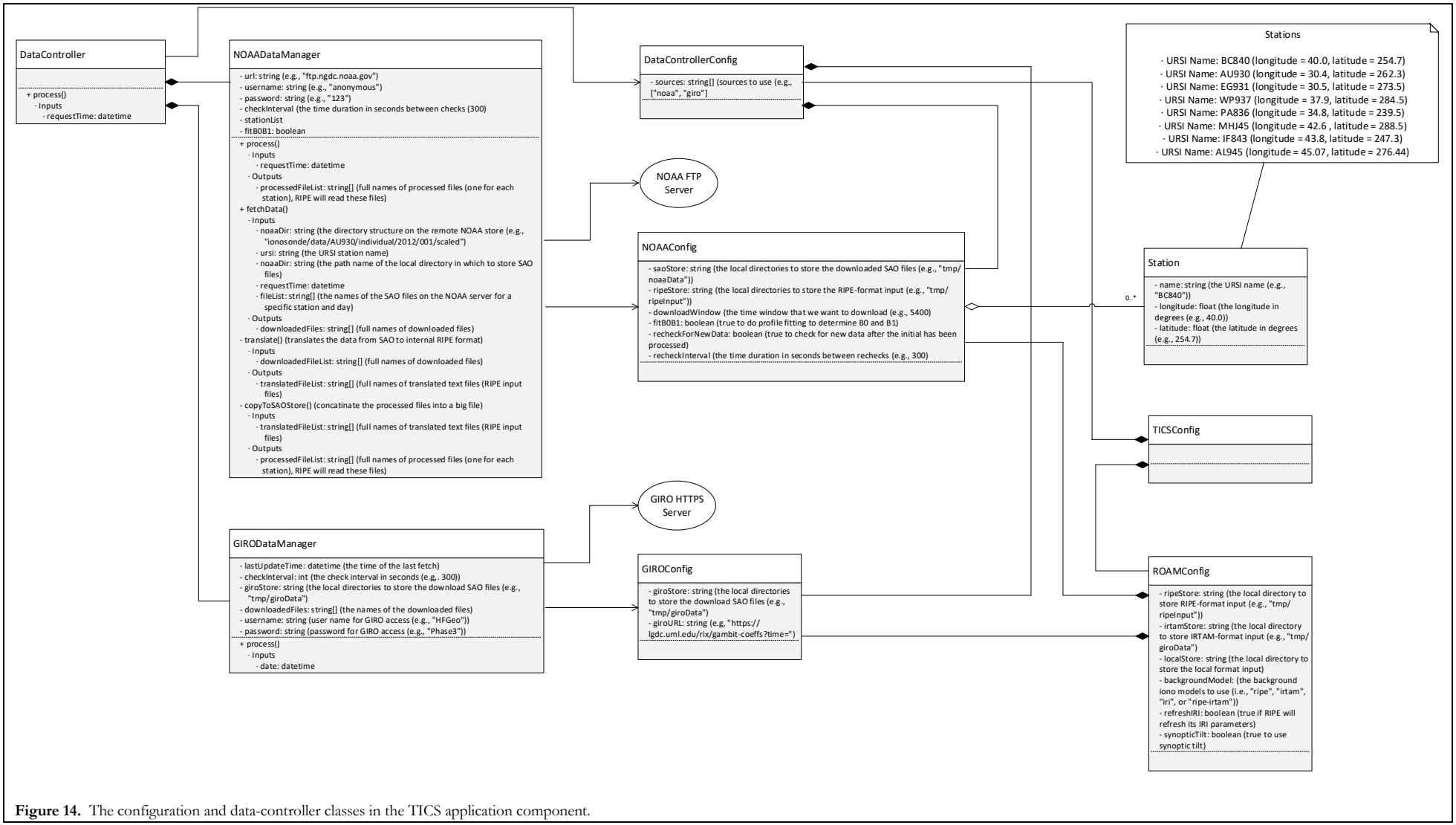


Figure 14. The configuration and data-controller classes in the TICS application component.

PYTHON TO C CODE CONVERSION

This section provides an estimate for the time for converting the existing TICS Python code into C, in terms of the separate TICS components, under the following assumptions:

- The current TICS Python code uses the following NumPy functions:

    - `argmax()`: finds the maximum values along an axis
    - `arcsin()`: calculates the inverse sine for elements in arrays
    - `array()`: creates an array
    - `cos()`: calculates the cosine for elements in arrays
    - `degrees()`: converts angles from radians to degrees
    - `empty()`: creates a new, empty array
    - `exp()`: calculates the exponential of elements in arrays
    - `floor()`: calculates the floor of elements in arrays
    - `genfromtxt()`: loads data from a text file
    - `linspace()`: returns evenly spaced numbers over intervals
    - `loadtxt()`: loads data from text files
    - `log()`: calculates natural logarithm for elements in arrays
    - `power()`: calculates the power of elements in one array using the values in a second array
    - `radians()` (and equivalently `deg2rad()`): converts angles from degrees to radians
    - `reshape()`: changes the shapes of arrays without changing their data
    - `sin()`: calculates the trigonometric sine for elements in arrays
    - `sqrt()`: calculates the non-negative square roots of elements in arrays
    - `tile()`: creates arrays that consists of repeating elements
    - `zeros()`: creates arrays filled with zeros

    and the following SciPy functions:

    - `curve_fit()`: uses non-linear least squares to fit a function to data
    - `griddata()`: interpolates unstructured *d*-dimensional data
    - `interp1d()`: interpolates a one-dimensional function
    - `minimize()`: minimizes a scalar function of one or more variables

    and these functions are easily replicable using built-in C math functions, the GNU Scientific Library, or minor custom C code.

- The conversion will exclude IRTAM.

- There is some allowable way (e.g., external libraries) to make FTP and HTTPS requests from C.

- The conversion does not include converting the existing external libraries. Instead the converted TICS C code will call these external libraries similarly to how the existing TICS Python code does.

- Threading is unnecessary in the converted C code but is available (e.g., by using the POSIX Threads) for possible speed improvements.

- The converted C code will not include oblique ionospheric processing, which is currently in development.

- The following external components are installed or available and outside of the scope of Python to C code-conversion efforts:
    - NOAA FTP Server
    - GIRO HTTPS Server
    - Ionopyiface Library
    - Irtam Library
    - SAO Library

Under these assumptions, Table 1 shows the estimated time to convert the existing Python components of TICS into equivalent C components.

Component	Estimated Weeks
PHaRLAP	3
TICS Calculator	3
ROAM	3
Profile Fitting	3
TICS Application: Configuration	1
TICS Application: Data Controllers	4
TICS Application: Main	1
Total	*18*

Table 1. Estimation of time for one developer to convert the Python components of TICS into C.

Table 2 shows risks associated with the estimation, ordered from highest to lowest severity.

Number	Risk Description	Likelihood	Impact	Severity	Workaround
1	External libraries may not be allowable on the target system	High	High	High	?
2	PHaRLAP, which requires the Intel Parallel Studio Runtime, may not be usable on the target system	High	High	High	Compile using the Intel Parallel Studio Runtime on a system other than the target system
3	FTP and HTTPS connectivity to NOAA and GIRO may not be available	High	Medium	Medium	Download NOAA and GIRO files separately from TICS
4	FTP and HTTPS libraries may not be available to the version of C to use	High	Medium	Medium	Make system calls (FTP, Curl) instead of using libraries to download files

Table 2. Risks in the estimation of the TICS Python to C code-conversion efforts.

INDEX