

# **United States Meteorological Data: Daily and Hourly Files to Support Predictive Exposure Modeling**

**United States Meteorological Data**  
**Daily and Hourly Files to Support Predictive Exposure Modeling**

By

Lawrence A. Burns, Ph.D.  
Ecologist, Ecosystems Research Division  
U.S. Environmental Protection Agency  
960 College Station Road  
Athens, Georgia 30605

Luis A. Suárez, Ph.D.  
Pharmacokineticist, Ecosystems Research Division  
U.S. Environmental Protection Agency  
960 College Station Road  
Athens, Georgia 30605

Lourdes M. Prieto, B.S.  
Environmental Scientist, Ecosystems Research Division  
U.S. Environmental Protection Agency  
960 College Station Road  
Athens, Georgia 30605

U.S. Environmental Protection Agency  
Office of Research and Development  
Washington, DC 20460

## Notice

The U.S. Environmental Protection Agency through its Office of Research and Development funded and managed the research described here under GPRA Goal 4, *Preventing Pollution and Reducing Risk in Communities, Homes, Workplaces and Ecosystems*, Objective 4.3, *Safe Handling and Use of Commercial Chemicals and Microorganisms*, Subobjective 4.3.4, *Human Health and Ecosystems*, Task 6519, *Advanced Pesticide Risk Assessment Technology*. It has been subjected to the Agency's peer and administrative review and approved for publication as an EPA document. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

## Abstract

ORD numerical models for pesticide exposure include a model of spray drift (AgDisp), a cropland pesticide persistence model (PRZM), a surface water exposure model (EXAMS), and a model of fish bioaccumulation (BASS). A unified climatological database for these models has been assembled from several National Weather Service (NWS) datasets, including Solar and Meteorological Surface Observation Network (SAMSON) data for 1961-1990 (versions 1.0 and 1.1), combined with NWS precipitation and evaporation data. Together these NWS products provide coordinated access to solar radiation, sky cover, temperature, relative humidity, station atmospheric pressure, wind direction and speed, and precipitation. The resulting hourly and daily weather parameters provide a unified dataset for use in coordinated exposure modeling. The data files, which include some derived data of use to exposure modeling (e.g., short-grass crop standard evapotranspiration  $ET_0$ ) are publicly available (gratis) on EPA's Center for Exposure Assessment Modeling (CEAM) web site at <http://www.epa.gov/ceampubl/tools/metdata/index.htm>. By using observational data for models, "trace-matching" Monte Carlo simulation studies can transmit the effects of environmental variability directly to exposure metrics, by-passing issues of correlation (covariance) among external driving forces.

This report covers a period from May 2, 2001 to December 27, 2004 and work was completed as of December 27, 2004.

## **Foreword**

Environmental protection efforts are increasingly directed toward preventing adverse health and ecological effects associated with specific chemical compounds of natural or human origin. As part of the Ecosystems Research Division's research on the occurrence, movement, transformation, impact, and control of environmental contaminants, the Ecosystems Assessment Branch studies complexes of environmental processes that control the transport, transformation, degradation, fate, and impact of pollutants or other materials in soil and water and develops models for assessing the risks associated with exposures to chemical contaminants.

Eric Weber, Director  
Ecosystems Research Division  
Athens, Georgia

### List of Symbols [units]

$\acute{a}$  surface albedo or canopy reflection coefficient [dimensionless]. Standardized at 0.23 for the FAO hypothetical grass reference crop (Allen et al. 1998:23). The albedo of water surfaces for atmospheric radiation is 0.03, and for direct sunlight, 0.06 (Kohler and Parmele 1967).

$\ddot{A}$  slope of saturation vapor-pressure curve at temperature  $T$  [kPa °C<sup>-1</sup>];

$$\Delta = \frac{4098 \left[ 0.6108 \exp\left(\frac{17.27T}{T+237.3}\right) \right]}{(T + 237.3)^2} = \frac{[2503 \exp\left(\frac{17.27T}{T+237.3}\right)]}{(T + 237.3)^2}$$

For  $T$  in °F,  $\ddot{A}$  [inches Hg °F<sup>-1</sup>] is (Lamoreux 1962)

$$\Delta = \frac{7182.6}{(T + 398.36)^2} \exp(15.674) \exp \frac{-7482.6}{(T + 398.36)}$$

$\ddot{a}$  solar declination [radians]

$\tilde{a}$  psychrometric coefficient [kPa °C<sup>-1</sup>] =  $6.65 \times 10^{-4} P$

$\tilde{a}$  is weakly dependent on temperature as well.  $\tilde{a}$  can alternatively be calculated from

$$\gamma = c_p P / \lambda \epsilon$$

(Burman and Pochop 1994:28). In this equation,  $c_p$ , the specific heat at constant pressure of naturally occurring moist air, can be taken as 1.013 kJ kg<sup>-1</sup> K<sup>-1</sup> (ASCE 1996:128),  $\acute{a}$ , the ratio of the mole weight of water to air, is 0.622,  $P$  is station barometric pressure [kPa], and  $\ddot{e}$  [kJ/kg] is calculated (following Harrison 1963) from dew-point temperature

(Burman and Pochop 1994, ASCE 1996):  $\lambda = 2501 - 2361T_{\phi}$

$\tilde{a}_p$  psychrometric coefficient for Class A evaporation pan [kPa °C<sup>-1</sup>] = 0.001568 $P$

$\acute{a}$  emissivity

$\ddot{e}$  latent heat (enthalpy) of vaporization (= 2.45 MJ kg<sup>-1</sup> in FAO standard (Allen et al. 1998:31)), calculated from dew-point temperature via  $\lambda = 2501 - 2361T_{\phi}$

$\ddot{o}$  angle of the sun above the horizon [radians]

$\phi$	latitude [radians]
$\dot{\omega}$	solar time angle at midpoint of hourly period [radians]
$\sigma$	Stefan-Boltzmann constant = $7.88 \times 10^{-11}$ equivalent inches of evaporation $\text{cm}^{-2} \text{K}^{-4} \text{day}^{-1}$ (using FAO standard enthalpy of vaporization of water of 585 cal/g)
$e$	water vapor pressure [kPa]
$e_a$	atmospheric water vapor pressure, actual water vapor pressure [kPa] ( $\equiv e_d \equiv 0.01 e \text{ RH}$ )
$e_s$	saturation water vapor pressure [kPa] at temperature $T$ [ $^{\circ}\text{C}$ ]; $e_s(T) = 0.6108 \exp\left[\frac{17.27T}{T + 237.3}\right]$
$e_s - e_a$	vapor pressure deficit
$e_d$	saturation water vapor pressure [kPa] at dew-point temperature $T_d$ ( $\equiv e_a$ )
$E_a$	aerodynamic function in Penman equation [ $\text{mm day}^{-1}$ ]
$E_L$	evaporation from small lake or pond [ $\text{mm day}^{-1}$ ]
$E_p$	pan evaporation [ $\text{mm day}^{-1}$ ]
$ET_0$	reference or potential <i>daily</i> evapotranspiration for standard short-grass crop (FAO standard method) [ $\text{mm day}^{-1}$ ]
$ET_{0(hr)}$	reference or potential <i>hourly</i> evapotranspiration for standard short-grass crop (FAO standard method) [ $\text{mm hour}^{-1}$ ]
$G$	soil heat flux density [ $\text{MJ m}^{-2} \text{hour}^{-1}$ ]
$J$	day of the year (on January 1, $J=1$ ; on December 31, $J=365$ ; 366 in leap years)
$P$	atmospheric pressure [kPa]
$\text{RH}$	relative humidity (%) = $100 \times e_a/e_s$
$R_a$	solar radiation received on a horizontal surface at the top of the atmosphere
$R_n$	net broad-band radiation in units of equivalent depth of water evaporated [ $\text{mm day}^{-1}$ ]

$R_{n(hr)}$	net broad-band hourly radiation at the FAO reference short-grass surface [ $\text{MJ m}^{-2} \text{hour}^{-1}$ ]
$R_s$	global (direct + diffuse) solar radiation received on a horizontal surface [ $\text{J cm}^{-2} \text{day}^{-1}$ ]
$R_{so}$	short-wave radiation received on a horizontal surface on a clear-sky day [ $\text{J cm}^{-2} \text{day}^{-1}$ ]
$T$	temperature [ $^{\circ}\text{C}$ ]
$T_{hr}$	mean hourly air temperature [ $^{\circ}\text{C}$ ]
$T_a$	mean daily air temperature, dry-bulb temperature [ $^{\circ}\text{C}$ ]
$T_{dp}$	dew-point temperature [ $^{\circ}\text{C}$ ]
$T_0$	water-surface temperature [ $^{\circ}\text{C}$ ]
$u_2$	average hourly wind speed at 2 meter height [ $\text{m s}^{-1}$ ]
$u_4$	average hourly wind speed at 4 meter height [ $\text{m s}^{-1}$ ]
$u_{0.6}$	Class A evaporation pan wind speed at 0.6 m height (about 2 feet; 6 inches above rim of pan) [ $\text{m s}^{-1}$ ]
$u_p$	daily wind movement at Class A pan [ $\text{km day}^{-1}$ ] (at 0.6 m height)
$u_{4d}$	daily wind movement over open water at 4 m height [ $\text{km day}^{-1}$ ]
$u_{10}$	wind speed at 10 meter height [ $\text{m s}^{-1}$ ]
$u_{0.1}$	wind speed at 0.1 meter height over open water [ $\text{m s}^{-1}$ ]
$W$	precipitable water in the atmosphere [mm]
[ ]	dimensionless units for variable

## Contents

Notice.....	<a href="#">i</a>
Abstract.....	<a href="#">i</a>
Foreword.....	<a href="#">ii</a>
List of Symbols.....	<a href="#">iii</a>
Introduction.....	<a href="#">1</a>
Data Sources.....	<a href="#">2</a>
File and Data Formats.....	<a href="#">3</a>
Daily Data Files.....	<a href="#">3</a>
Hourly Data Files.....	<a href="#">3</a>
Data Assembly and Processing.....	<a href="#">6</a>
Assembly of Evaporation Data.....	<a href="#">6</a>
Assembly of Precipitation Data.....	<a href="#">7</a>
Estimation of standard crop potential evapotranspiration (reference evapotranspiration $ET_0$ ).....	<a href="#">7</a>
Estimation of Pan Evaporation and Free Water Surface Evaporation.....	<a href="#">11</a>
Pond and Lake Evaporation.....	<a href="#">13</a>
Conversion of wind speeds to standard heights.....	<a href="#">15</a>
Processing Sequence for Production of Daily Values Files (*.dvf).....	<a href="#">19</a>
Notes and Irregularities.....	<a href="#">22</a>
Fortran Processing Code.....	<a href="#">26</a>
BinaryTree.....	<a href="#">26</a>
dump.....	<a href="#">29</a>
dumpmet.....	<a href="#">49</a>
Et0.....	<a href="#">62</a>
Evap.....	<a href="#">90</a>
Fix_Data.....	<a href="#">99</a>
Gaps.....	<a href="#">103</a>
global.....	<a href="#">122</a>
LinkedList.....	<a href="#">134</a>
Make_R0.....	<a href="#">140</a>
Precip.....	<a href="#">142</a>

raw_data.....	<a href="#"><u>220</u></a>
read_info. ....	<a href="#"><u>231</u></a>
Red_Black. ....	<a href="#"><u>242</u></a>
samson. ....	<a href="#"><u>259</u></a>
setup.....	<a href="#"><u>277</u></a>
Stats. ....	<a href="#"><u>291</u></a>
Utils0. ....	<a href="#"><u>297</u></a>
Utils1. ....	<a href="#"><u>300</u></a>
Utils2. ....	<a href="#"><u>324</u></a>
Utils3. ....	<a href="#"><u>357</u></a>
Utils4. ....	<a href="#"><u>363</u></a>
Utils5. ....	<a href="#"><u>371</u></a>
Test Data for Laramie, Wyoming: average values for August 1987. ....	<a href="#"><u>376</u></a>
SAMSON Station Locations.....	<a href="#"><u>377</u></a>
References. ....	<a href="#"><u>386</u></a>

## Introduction

Exposure models (e.g., EXAMS (Burns 2000), PRZM (Carousel et al. 2005)) used in EPA chemical exposure assessments number weather data among their required input parameters. Although “weather generator” software (e.g., Hanson et al. 1993, 1994, Nicks and Gander 1994) is available and has been used for water resource and climate studies, it has been observed that the weather sequences thus generated are weak in their ability to capture the extreme events that are usually of greatest importance in exposure and risk assessments. For example, in a study (Johnson et al. 1996) of the USCLIMATE (Hanson et al. 1993, 1994) and CLIGEN (Nicks and Gander 1994) models, the authors remark that “[a]nnual and monthly precipitation statistics (means, standard deviations, and extremes) were adequately replicated by both models, but daily amounts, particularly typical extreme amounts in any given year, were not entirely satisfactorily modeled by either model. USCLIMATE consistently underestimated extreme daily amounts, by as much as 50%.” In a study (Semenov et al. 1998) of WGEN (Richardson and Wright 1984) (itself an element of USCLIMATE) and LARS-WG (Semenov and Barrow 1997) at 18 climatologically diverse sites in the USA, Europe, and Asia, the authors concluded that the Gamma distribution used in WGEN “probably tends to overestimate the probability of larger values” of rainfall. This result, although opposite in tendency to that of Johnson et al. (1996), is no less undesirable. Both models had a lower inter-annual variance in monthly mean precipitation than that seen in the observed data, and neither weather generator “performed uniformly well in simulating the daily variances of the climate variables.” Semenov et al. (1998) concluded that failures to represent variance in LARS-WG and WGEN were “likely to be due to the observed data containing many periods in which successive values are highly correlated...”

The problem of accurately preserving parameter variability, and covariance among parameters, can be bypassed by using observed synoptic data, and the pitfall of generating impossible input scenarios (e.g., days of heavy rainfall coupled with maximum drying potential) can be avoided, given accurate input datasets. Weather generator software can be problematic in this regard. For example, CLIGEN generates temperature, solar radiation, and precipitation independently of one another, so the covariance structure of daily sequences need not be preserved in the model outputs.

Here we report on the production of long-term (30-year) observed weather sequences for use in ecological exposure modeling. The Weather Station Data Files (available at <http://www.epa.gov/ceampubl/tools/metdata/index.htm>) contain weather data for 237 meteorological stations for the (nominal) period 1961-1990. Two types of files were generated: “\*.dvf” for daily values file, and “\*.hnn” for hourly values file. The daily weather files are used by the current versions of PRZM and EXAMS; they also include values potentially useful for modeling of spray drift of pesticides. The hourly data files serve as both source for the daily files and as a detailed data set publicly available for inclusion in simulation models. The name of each file is of the form “wnnnnnn”, where “nnnnn” represents the WBAN (Weather Bureau Army Navy) identification number of the weather station. Daily files are of file type “.dvf” and “nn” denotes the last two digits of the year for

each hourly file. For example, w25501.h65 contains 1965 hourly data for WBAN 25501 located in Kodiak, AK. More information on WBAN numbers is available at <http://lwf.ncdc.noaa.gov/oa/climate/stationlocator.html>

## Data Sources

The following sources provided data for the generation of the hourly files:

- ◆ Solar and Meteorological Surface Observation Network (SAMSON) 1961-1990 data sets, Version 1.0, Sept. 1993.
- ◆ National Solar Radiation Data base (NSRDB) version 1.1; NSRDB Hourly Data Files Text files downloaded from [http://rredc.nrel.gov/solar/old\\_data/nsrdb/hourly/](http://rredc.nrel.gov/solar/old_data/nsrdb/hourly/). These files provided updated solar radiation parameters for the SAMSON database.
- ◆ EarthInfo NCDC Summary of the Day and Surface Airways, 2001. Provided daily and hourly values for precipitation, and daily evaporation. Missing evaporation was calculated using the Kohler-Nordenson-Fox (Kohler et al. 1955, Burman and Pochop 1994) Class-A Evaporation Pan version of the Penman-Monteith equations.

SAMSON (Solar and Meteorological Surface Observation Network) data were obtained from NOAA (<http://nndc.noaa.gov/onlinestore.html>) as a three-volume CD-ROM disk set containing observational and modeled meteorological and solar radiation data for the period 1961-1990. SAMSON encompasses 237 NWS (National Weather Service) stations in the United States, plus offices in Guam and Puerto Rico. An additional five years of data (1991-1995) were acquired on CD from NCDC (National Climatic Data Center) as the HUSWO (Hourly United States Weather Observations) product. (HUSWO is nominally an update to the SAMSON files.) Combined data were then available for 234 stations over 1961-1995. Weather stations used to develop these data files are tabulated according to their standard WBAN (Weather Bureau Army Navy) number, along with station location (City, State), geographic (latitude and longitude) coordinates, and station elevation (m), later in this document.

The hourly SAMSON solar elements comprise extraterrestrial horizontal and extraterrestrial direct normal radiation; and global, diffuse, and direct normal radiation. Meteorological elements comprise total and opaque sky cover, temperature and dew point, relative humidity, station pressure, wind direction and speed, visibility, ceiling height, present weather, precipitable water, aerosol optical depth, snow depth, days since last snowfall, and hourly precipitation. Weather elements in the HUSWO files include total and opaque sky cover; temperature and dew point; relative humidity; station pressure; wind direction and speed; visibility; ceiling height; present weather; ASOS (Automated Surface Observing System) cloud layer data; snow depth; and hourly precipitation for most stations. Stations for which hourly precipitation was not available are italicized in the station tabulation.

The original intent for this project was to develop a data base encompassing the entire 1961-1995 period. However, the HUSWO CD-ROM, albeit formatted the same as SAMSON, was discovered to lack solar radiation data, i.e., the fields are present but marked as “missing data.” The published EPA dataset was therefore restricted to the period 1961-1990.

## File and Data Formats

### Daily Data Files

The daily values were obtained by computing the mean over 24 hours, or by adding the values over one day, as appropriate, from the completed hourly values data files. “Daily Values File” data files (\*.dvf) have the following format:

Field	Columns	Description	Units	Type	Format
1	1	blank	N/A	Character	1x
2	02 - 07	Date	mmddy	Integer	3i2
3	08 - 17	Precipitation	cm/day	Real	f10.2
4	18 - 27	Pan Evaporation	cm/day	Real	f10.2
5	28 - 37	Temperature (mean)	degrees Centigrade	Real	f10.1
6	38 - 47	Wind Speed @10 meter	cm/second	Real	f10.1
7	48 - 57	Solar Radiation	Langleys/day	Real	f10.1
8	58 - 63	FAO Short Grass Eto	mm/day	Real	f6.1
9	64 - 73	Daylight Station Pressure	kiloPascal	Real	f10.1
10	74 - 77	Daylight Relative Humidity	percent	Integer	i4
11	78 - 80	Daylight Opaque Sky Cover	tenths of sky covered	Integer	i3
12	81 - 90	Daylight Temperature	degrees Centigrade	Real	f10.1
13	91 - 96	Daylight Broadband Aerosol	optical depth	Real	f6.3
14	97 -102	Daylight Prevailing Wind Speed @10 meter	meter/second	Real	f6.1
15	103 -106	Daylight Prevailing Wind Direction	degrees (N=0, E=90, ...)	Integer	i4

Daily values Fortran file format:

(1x,3i2, t8,f10.2, t18,f10.2, t28,f10.1, t38,f10.1, &  
t48,f10.1, t58,f6.1, t64,f10.1, t74,i4, t78,i3, &  
t81,f10.1, t91,f6.3, t97,f6.1, t103,i4)

Fields 03 - 08 are daily totals or mean values (i.e., mean Wind Speed and mean Temperature)

Fields 03 - 07 units preserved from earlier PRZM meteorological files for compatibility

Fields 09 - 15 are mean values over daylight hours only (to support photochemical and spray drift algorithms)

### Hourly Data Files

Hourly values files (\*.hnn) contain hourly data for the year 19nn. For example, w25501.h65 contains 1965 hourly data for Kodiak, AK. The hourly values file has a header containing identifying information.

Example header record:

25501 Kodiak AK +9 N 57 45 W 152 30 5 2002-05-17 22:47:31

Header field description:

Field	Columns	Description	Type	Format
1	1	blank	Character	1x
2	02 - 06	WBAN id	Character	a5
3	08 - 37	City where the station is located.	Character	a30
4	39 - 40	State where the station is located.	Character	a2
5	42 - 44	Time Zone	Character	a3
6	47 - 54	Latitude of the station		
	47 - 47	N: North of the Equator S: South	Character	a1
	48 - 51	Degrees	Integer	i4
	53 - 54	Minutes	Integer	i2
7	57 - 64	Longitude of the station		
	57 - 57	W: West E: East	Character	a1
	58 - 61	Degrees	Integer	i4
	63 - 64	Minutes	Integer	i2
8	67 - 70	Station Elevation(meter)	Integer	i4
9	74 - 92	Generation date of the file, i.e, yyyy-mm-dd hh:mm:ss where yyyy: year mm: month dd: day of the month hh: hour (24-hour clock) mm: minutes ss: seconds	Character	a19

Header format: (1x, a5, 1x, a30, 1x, a2, 1x, a3, & 2x, a1, i4, 1x, i2, & 2x, a1, i4, 1x, i2, & 2x, i4, 3x, a19)

Hourly data fields:

Field	Columns	Description	Units	Type	Format
1	1	blank	N/A	Character	1x
2	002 - 011	Date	yyyy-mm-dd	Integer	i4,1x,i2,1x,i2
3	012 - 014	*Hour of the day	Hour	Integer	i3
4	016 - 021	Extraterrestrial Horizontal Radiation (Ra)	*Wh/m2	Integer	i5,a1
5	023 - 028	Extraterrestrial Direct Normal Radiation	Wh/m2	Integer	i5,a1
6	030 - 037	Global Horizontal Radiation (Rs)	Wh/m2	Integer	i5,a3
7	039 - 046	Direct Normal Radiation	Wh/m2	Integer	i5,a3
8	048 - 055	Diffuse Horizontal Radiation	Wh/m2	Integer	i5,a3
9	057 - 059	Total Sky Cover	tenths of sky covered	Integer	i2,a1
10	061 - 063	Opaque_Sky_Cover	tenths of sky covered	Integer	i2,a1
11	065 - 070	Dry Bulb Temperature	degrees Centigrade	Real	f5.1,a1

12	072 - 077	Dew Point Temperature	degrees Centigrade	Real	f5.1,a1
13	079 - 082	Relative_Humidity	percent	Integer	i3,a1
14	084 - 089	Station_Pressure	*kPa	Real	f5.1,a1
15	091 - 094	Wind_Direction	degrees(N=0,E=90,...)	Integer	i3,a1
16	096 - 101	Wind_Speed @10meter	m/s	Real	f5.1,a1
17	103 - 109	Horizontal Visibility	km	Real	f6.1,a1
18	111 - 117	Ceiling Height	m	Integer	i6,a1
19	119 - 120	Observation Indicator	N/A	Integer	i1,a1
20	122 - 131	Present_weather	N/A	Character	a9,a1
21	133 - 136	Precipitable Water	mm	Integer	i3,a1
22	138 - 144	Broadband Aerosol	Optical_Depth	Real	f6.3,a1
23	146 - 150	Snow Depth	cm	Integer	i4,a1
24	152 - 155	Days since last Snowfall	day	Integer	i3,a1
25	157 - 164	Hourly Precipitation	cm	Real	f6.2,a2
26	166 - 172	Eto, FAO Short Grass	mm/day	Real	f6.2,a1
27	174 - 180	Ep, Class A pan Evaporation	mm/day	Real	f6.2,a1

\*Hour of the day: The range of the hour of the day is 1h to 25h:

1h-24h: line contains data for the hour

25h: line contains daily values

\*Wh/m2: Watt hour meter<sup>-2</sup>

Watt hour meter<sup>-2</sup> is equivalent to 3.6e-3 MJoule meter<sup>-2</sup>

Watt hour meter<sup>-2</sup> is equivalent to 8.59845e-02 Langley

\*kPa: kiloPascal (1 kiloPascal is equivalent to 10 millibar)

Fields 26 and 27 represent daily totals. These fields are populated only when the hour of the day (field 3) is 25.

Generally, each value has one or more character flags associated with it. When possible, the flags associated with the original datum are transferred to the Hourly Values File (\*.hnn).

“W” = Measured value (SAMSON flag)

“B” = Calibrated (SAMSON flag)

“D” = Deleted (SAMSON flag)

“E” = Estimated

“\_” = Missing value

“R” = Datum from EarthInfo data 2001

“S” = Datum from SAMSON version 1.0

“T” = Datum from SAMSON version 1.1

“U” = Unlimited in Visibility or Ceiling Height, or short-gap interpolation with either of the endpoints being Unlimited or Cirriform.

“Z” = Cirriform in Ceiling Height data

“?” = Value is undefined, given the context, e.g., Dew point is undefined if the Relative Humidity is zero.

“#” = If the station is above the Arctic Circle (Latitude 66.5 degrees North), then the region will be in darkness for a period of the year. This prevents the computation of, e.g., Daylight value of Global Horizontal Radiation and Reference Crop Evapotranspiration (Et<sub>0</sub>)

“A” = Accumulation (SAMSON flag). The precipitation is distributed in the observation interval.

Missing values are denoted by “—” in the numeric value field and flag value of “-”

Hourly values Fortran file format:

```
(1x,i4,1x,i2,1x,i2, i3, t16,i5,a1, t23,i5,a1, &  
t30,i5,a3, t39,i5,a3, t48,i5,a3, &  
t57,i2,a1, t61,i2,a1, t65,f5.1,a1, t72,f5.1,a1, &  
t79,i3,a1, t84,f5.1,a1, t91,i3,a1, t96,f5.1,a1, &  
t103,f6.1,a1, t111,i6,a1, t119,i1,a1, t122,a9,a1, &  
t133,i3,a1, t138,f6.3,a1, t146,i4,a1, t152,i3,a1, &  
t157,f6.2,a2, t166,f6.2,a1, t174,f6.2,a1)
```

## Data Assembly and Processing

### Assembly of Evaporation Data

GIS software (ArcInfo) was used to facilitate station identification and data assembly. GIS evaporation coverages for Alaska, Hawaii, Guam and Puerto Rico were created by extracting from the 2001 NCDC Summary of the Day Earthinfo CD-ROMS the coordinates of the stations that had evaporation data for each pertinent state or region. An evaporation coverage for the conterminous United States was created by extracting from the 1998 NCDC Summary of the Day Earthinfo CD-ROMS the coordinates of the stations that had evaporation data.

To obtain evaporation data for the SAMSON stations in the conterminous U.S. the following guidelines were used:

- when available, data were taken from the cooperative network reporting station that corresponded to the SAMSON station of interest
- when the corresponding station was not available or had no data for the period of interest, a nearby station was chosen according the following criteria:
  - The station had to be located in the same climate region as the SAMSON station in question. To determine the climate region, an image of the map: “Land Resource and Climate Regions of the United States for EPA - Pesticide Study,” provided by EPA's Office of Pesticide Programs, was used. This map was developed for the U.S. Environmental Protection Agency, Office of Pesticide Programs, Environmental Fate and Effects Division under a contract with Oak Ridge National Lab in 1999-2000. ORNL subcontracted to USDA/NRCS through the Conservation Technology Information Center (CTIC) at Purdue. The work was carried out by NRCS staff Glenn Weesies (West Lafayette, IN), Dave Lightle (Lincoln, NE) and Ken Pfeiffer (Portland, OR). The EFED/ORNL project was entitled: “C-factor zone map of the U.S. to localize RUSLE applications for various crops developed and supporting database completed which includes RUSLE crop, climate, field operation, and C-factors.”
- Out of the stations located in the same climate region, we picked the closest one that had an station elevation difference with the SAMSON station of interest of no more than 500 feet.

- If the station chosen fell within 20 km of the SAMSON station, the station was considered to be “on target”. Stations that were more than 20 km away were considered to be “not on target”. The maximum allowable distance for “not on target” stations was 200 km.

To obtain evaporation data for the SAMSON stations located in Alaska, Hawaii, Guam and Puerto Rico the above guidelines were followed except that the station chosen had to be located in the same Major Land Resource Area (MLRA) as the SAMSON station in question, since no corresponding climate region map was available for these areas.

### Assembly of Precipitation Data

Daily precipitation files were created by extracting data for each station from the 2001 NCDC Summary of the Day Earthinfo CD-ROMS.

Hourly precipitation files were created by extracting hourly data for each station from the 2001 NCDC Hourly Precipitation Earthinfo CD-ROMS.

### Estimation of standard crop potential evapotranspiration (reference evapotranspiration $ET_0$ )

The FAO Penman-Monteith equation for hourly time steps (Allen et al. 1998:74) is:

$$ET_{0(hr)} = \frac{0.408\Delta(R_{n(hr)} - G) + \gamma \frac{37}{T_{hr} + 273} u_2 (e_s(T_{hr}) - e_a)}{\Delta + \gamma(1 + 0.34u_2)}$$

where  $ET_{0(hr)}$  is hourly reference evapotranspiration [ mm hour<sup>-1</sup>]

$R_{n(hr)}$  is hourly net radiation at the grass surface [MJ m<sup>-2</sup> hour<sup>-1</sup>]

$G$  is soil heat flux density [MJ m<sup>-2</sup> hour<sup>-1</sup>]

$T_{hr}$  is mean hourly air temperature [°C]

$\Delta$  is slope of the saturation vapor pressure curve at  $T_{hr}$  [kPa °C<sup>-1</sup>];

$$\Delta = \frac{\left[ 2503 \exp\left(\frac{17.27 T_{hr}}{T_{hr} + 237.3}\right) \right]}{(T_{hr} + 237.3)^2}$$

$\tilde{a}$  is the psychometric constant [kPa °C<sup>-1</sup>]. The SAMSON data files report station barometric pressure in millibars (1mB = 0.1 kPa); after conversion to kPa,  $\tilde{a} = 6.65 \times 10^{-4} \times P$  (Allen et al. 1998:32).

$e_s(T_{hr})$  is the saturation vapor pressure at  $T_{hr}$

$e_a$  is average hourly actual vapor pressure [kPa]

$u_2$  average hourly wind speed at 2 m height [m s<sup>-1</sup>]

This reference is accessible online at <http://www.fao.org/docrep/X0490E/x0490e00.htm>.

From the relative humidity measurements reported in the SAMSON data files, actual vapor pressure  $e_a$  was calculated as

$$e_a = e_s(T_{hr}) \frac{RH_{hr}}{100}$$

where  $RH_{hr}$  is hourly relative humidity [%], and  $e_s(T_{hr})$ , the saturation vapor pressure at temperature  $T_{hr}$ , is calculated from (Allen et al. 1998:36)

$$e_s(T_{hr}) = 0.6108 \exp \left[ \frac{17.27 T_{hr}}{T_{hr} + 237.3} \right]$$

Net radiation  $R_{n(hr)}$  is the difference between net shortwave radiation  $R_{ns}$  and the net longwave radiation  $R_{nl}$  at the hourly time steps:

$$R_{n(hr)} = R_{ns} - R_{nl}$$

The SAMSON data files report hourly values of global horizontal radiation  $R_s$  [ $\text{Wh m}^{-2}$ ]. Net hourly shortwave radiation  $R_{ns}$  [ $\text{MJ m}^{-2} \text{hour}^{-1}$ ] is the balance between incoming and reflected solar radiation, given by

$$R_{ns} = 3.60 \times 10^{-3} (1 - \alpha) R_s = 3.60 \times 10^{-3} (1 - 0.23) R_s$$

The factor  $3.60 \times 10^{-3}$  converts from [ $\text{Wh m}^{-2}$ ] to [ $\text{MJ m}^{-2}$ ]; the albedo of the standard short-grass crop is 0.23 (Allen et al. 1998:23).

Net longwave radiation  $R_{nl}$  was calculated as

$$R_{nl} = \sigma [T_{hr} + 273.15]^4 \left( 0.34 - 0.14 \sqrt{e_a} \right) \left( 1.35 \frac{R_s}{R_{so}} - 0.35 \right)$$

in which the Stefan-Boltzman constant  $\sigma$  has the value  $2.043 \times 10^{-10} \text{ MJ m}^{-2} \text{ hour}^{-1}$ . The ratio  $R_s/R_{so}$ , the ratio of actual global horizontal radiation to the equivalent (theoretical) clear-sky shortwave radiation, represents the effect of cloud cover and atmospheric aerosol. For calculation of  $R_{nl}$  for hourly periods during nighttime hours, the ratio  $R_s/R_{so}$  is set equal to  $R_s/R_{so}$  calculated for a time 2-3 hours before sunset, before the sun angle becomes small. The hourly period 2-3 hours before sunset was identified in the SAMSON files from positive (non-zero) values in the global horizontal radiation field.

For the calculation of  $R_{so}$  (short-wave radiation on a clear-sky day),

$$R_{so} = 3.60 \times 10^{-3} (K_B + K_D) R_a$$

where the factor  $3.60 \times 10^{-3}$  converts  $R_a$  from  $[\text{Wh m}^{-2}]$  to  $[\text{MJ m}^{-2}]$ , and

$K_B$  = the clearness index for direct beam radiation []

$K_D$  = the corresponding index for diffuse radiation []

$R_a$  = extraterrestrial radiation on a horizontal surface, reported in SAMSON in  $[\text{Wh m}^{-2}]$

The clearness index  $K_B$  was calculated from

$$K_B = 0.98 \exp \left[ \frac{-0.00146P}{K_t \sin \phi} - 0.075 \left( \frac{W}{\sin \phi} \right)^{0.4} \right]$$

where

$P$  = atmospheric pressure [kPa];  $P$  converted to [kPa] from SAMSON file hourly values [millibars]

$W$  = precipitable water in the atmosphere [mm], hourly values read from SAMSON files

$K_t$  is a turbidity coefficient [],  $0 < K_t < 1.0$ ;  $K_t = 1.0$  for clean air and  $K_t = 0.5$  for extremely turbid air. The parameters of the present version of this equation have been modified from those of Eq. 3-17 of Allen et al. (1998:227) to conform with the "ASCE Standardized Reference ET Calculation," as suggested in (Allen 2000), based on studies of 30 U.S. stations. For calculation of  $R_{so}$ ,  $K_t$  was taken as 1.0 (clean air), as suggested in (Allen 1996).

$\phi$  = angle of the sun above the horizon (radians);  $\phi$  is calculated as

$$\sin \phi = \sin \varphi \sin \delta + \cos \varphi \cos \delta \cos \omega$$

where

$\delta$  = solar declination [radians]

$\varphi$  = station latitude [radians]

$\omega$  = solar time angle at midpoint of hourly period [radians]

The solar declination  $\delta$  is calculated from

$$\delta = 0.409 \sin \left( \frac{2\pi}{365} J - 1.39 \right)$$

where  $J$  is the day of the year.

The solar time angle  $\omega$  at the midpoint of the period is

$$\omega = \frac{\pi}{12} [t + 0.06667(L_z - L_m) + S_c - 12]$$

where

$t$  = standard clock time at the midpoint of the hour [hours, based on 24-hour clock]

$L_z$  = longitude of the center of the local time zone [degrees west of Greenwich] (see Table 1)  
 $L_m$  = longitude of the measurement site [degrees west of Greenwich]  
 $S_c$  = seasonal correction for local solar time [hour]

Table 1. Standard U.S. Time Zones

Time Zone	Letter	U.S. Name	Central Meridian
+4	Q	Atlantic	60W
+5	R	Eastern	75W
+6	S	Central	90W
+7	T	Mountain	105W
+8	U	Pacific	120W
+9	V	Alaska	135W
+10	W	Hawaii-Aleutian	150W
+11	X	Samoa	165W
-10	K	Chamorro (proposed)	150E

"Time Zone" indicates hours to be added to local standard time to arrive at Universal Time, Coordinated (UTC). Time zones in the USA are defined in the U.S. Code, Title 15, Chapter 6, Subchapter IX - Standard Time. The U.S. Department of Transportation is responsible for time zone boundaries.

The seasonal correction for local time  $S_c$  is (Allen et al. 1998:48)

$$S_c = 0.1645 \sin(2b) - 0.1255 \cos(b) - 0.025 \sin(b)$$

$$b = \frac{2\pi(J - 81)}{364}$$

where  $J$  is the day of the year.  $J$  can be determined (Allen 1996) by

$$J = \text{int} \left( 275 \frac{M}{9} - 30 + D \right) - 2$$

where  $M$  is month number (1-12) and  $D$  is day in month; with the proviso that if  $M < 3$ ,  $J = J + 2$ , and during leap years, if  $M > 2$ ,  $J = J + 1$ . Alternatively,  $J$  can be determined from (Allen 2000)

$$J = D_M - 32 + \text{int}\left(275 \frac{M}{9}\right) + 2 \text{int}\left(\frac{3}{M+1}\right) + \text{int}\left(\frac{M}{100} - \frac{\text{mod}(Y,4)}{4} + 0.975\right)$$

where

$D_M$  is the day of the month (1 - 31)

$M$  is the number of the month (1-12)

$Y$  is the year (e.g., 1990)

The diffuse radiation index  $K_D$  was then calculated from  $K_B$  (Allen et al. 1998:227):

$$K_D = 0.35 - 0.36 K_B \text{ for } K_B \geq 0.15 \text{ (parameters altered per (Allen 2000))}$$

$$K_D = 0.18 + 0.82 K_B \text{ for } K_B < 0.15.$$

Soil heat flux  $G$  is important for hourly calculations. Hourly  $G$  during daylight periods is approximated as  $G_{hr} = 0.1 R_n$ , and during nighttime periods as  $G_{hr} = 0.5 R_n$  (Allen et al. 1998:55). The coefficients in these equations assume a constant surface resistance  $r_s$  of 70 s/m during all periods. This may cause some under-prediction of  $ET_0$  during some daytime periods and over-prediction of  $ET_0$  during evening hours. Precise estimates of  $ET_0$  for specific hourly periods would require the use of aerodynamic stability functions and estimation of  $r_s$  as a function of radiation, humidity, and temperature. When hourly values are summed to 24-hour totals, however, these hourly differences compensate for one another (Allen et al. 1998). Such functions were therefore not used in assembling this dataset.

## Estimation of Pan Evaporation and Free Water Surface Evaporation

Evaporation from the standard Class A pan is measured at numerous Weather Bureau stations. These pans provide a measure of the integrated effect of radiation, temperature, humidity, and wind on evaporation from an open water surface. The pans are, however, subject to some systematic differences in their behavior from that of natural water bodies or crops. These differences include energy exchange with the pan ambient environment through the pan walls, the differing albedo of water and cropped agricultural surfaces, heat storage and thermal inertia differences between lakes, crop lands, and evaporation pans, and differences in aerodynamic properties of the air immediately above the respective surfaces. Adjustment of observed pan evaporation to estimate lake evaporation and crop water use remains a useful and common procedure, however, and Field 4 of the database retains measured/modeled pan evaporation in cm/day.

Observations of water lost from Class A pans are commonly suspended during Winter months, and are subject to errors during rainfall events due to splash-out even when the pans are meticulously maintained. For this project, the Kohler-Nordenson-Fox equation (Kohler et al. 1955, Burman and Pochop 1994) was used to estimate pan evaporation from meteorological data. Where possible, the estimates were correlated with observed pan evaporation on rain-free days at the station, from which monthly correction/calibration factors were developed. Database field 4 (daily pan evaporation, cm) was filled with observed data where possible, supplemented with modeled data (corrected by month-specific calibration factors) to fill in missing values.

The underlying equation is an adaptation of the Penman (Penman 1948, 1963) equation to the task of estimating Class A pan evaporation  $E_p$  [mm day<sup>-1</sup>], as

$$E_p = \frac{\Delta R_n + \gamma_p E_a}{\Delta + \gamma_p}$$

In (Kohler et al. 1955), a psychometric coefficient  $\tilde{a}_p$  specific to the Class A pan was created by inclusion of an empirical adjustment that accounts for sensible heat conducted through the sides and bottom of the pan. The psychometric coefficient is calculated from mean daily station barometric pressure [kPa] via  $\tilde{a}_p = 0.001568P$ ; the values calculated with this relationship exceed theoretical values of  $\tilde{a}$  as a consequence of the incorporated heat transfer component.

The aerodynamic function  $E_a$  [mm day<sup>-1</sup>] in the Penman equation, specific to Class A pan evaporation, was developed by Kohler et al. (1955) from data at four locations representing a variety of climatic regimes (Vicksburg, Mississippi; Silver Hill, Maryland; Boulder City, Nevada; Lake Hefner, Oklahoma). For this project,  $E_a$  [mm day<sup>-1</sup>] was thus calculated as

$$E_a = 25.4 \{0.295(e_s - e_a)\}^{0.88} (0.37 + 0.00255u_p)$$

where

$e_s - e_a$  is the mean daily vapor pressure deficit [kPa] computed from hourly values of relative humidity  $RH_{hr}$  and air temperature  $T_{hr}$

$u_p$  is daily wind movement [km day<sup>-1</sup>] calculated from hourly wind speeds at 0.6 m height

Effective net radiation for the Class A pan (the term  $\Delta R_n$ ) is given by (Lamoureux 1962, Burman and Pochop 1994:185)

$$(\Delta R_n) = 154.8 \exp[(1.8T_a - 180)(0.1024 - 0.01066 \ln(0.086R_s))] - 0.01548$$

where

$R_s$  is global horizontal radiation [Wh m<sup>-2</sup> day<sup>-1</sup>], derived from SAMSON by summation over the day, and

$T_a$  is mean daily air temperature [°C]

Here  $\Delta$  [kPa/°C], the slope of the saturation vapor pressure versus temperature curve, was calculated by the method recommended at (Burman and Pochop 1994:24):

$$\Delta = \frac{2503}{(T_a + 237.3)^2} \exp\left[\frac{17.27T_a}{T_a + 237.3}\right]$$

### *Pond and Lake Evaporation*

Although based on limited data, the ratio of *annual* Class A pan evaporation to evaporation from natural water bodies (e.g., small ponds) can be taken as 0.70, provided (Kohler et al. 1955)

1. Any net energy advection into the pond is balanced by the change in energy storage
2. air temperature adequately represents surface water temperature, and, if an observation pan is the data source (rather than a computed value as  $E_p$ )
3. The net transfer of sensible heat through the observing pan is negligible.
4. The observing pan exposure is representative of pond conditions.

To account for the third point above, a “theoretical” pan (i.e., one not subject to sensible heat transfer through its walls) is constructed by replacing the pan-specific coefficient  $\tilde{a}_p$  with a standard psychrometric coefficient  $\tilde{a}$  calculated as  $\tilde{a}=0.000665P$  (Allen et al. 1998:32), giving

$$E_{FWS} = (0.70) \frac{\Delta R_n + \gamma E_a}{\Delta + \gamma}$$

where  $\Delta R_n$  and  $E_a$  are calculated as for pan evaporation above.

Evaporation calculated by this equation is generally designated “free water surface” (FWS) evaporation in order to emphasize its somewhat theoretical nature, i.e., it can only apply in its uncorrected form to small ponds meeting the four requirements listed above. This equation was used in the production of the NWS *Evaporation Atlas for the Contiguous 48 United States* (Farnsworth et al. 1982).

As might be expected from the admixture of a generalized annual coefficient (0.70) with shorter-term calculations, the pan coefficient, i.e., the ratio  $E_{FWS}/E_p$ , can vary considerably from month to month. The value of the pan coefficient depends on climatic conditions in the area affecting thermal properties of the exposed evaporation pan. Pan coefficients in the contiguous 48 United States were observed to vary from 0.64 to 0.88 for the period May through October (Farnsworth et al. 1982:5). Pan coefficients for colder months (November through April) were usually smaller than those of warmer months. Coastal southern California offers an “extreme” example, in which pan coefficient values range from 0.88 for the warmer months to 0.64-0.68 for the colder months.

The assumptions of the equations for free water surface evaporation (Penman 1948, Kohler et al. 1955, Penman 1963) were reexamined by Kohler and Parmele (Kohler and Parmele 1967). Because several empirical factors were inferred from data, the units in use at the time were maintained for this analysis. Their modified equation for evaporation  $E_{FWS}$  [mm day<sup>-1</sup>] from a free water surface (FWS) is

$$E_{FWS} = 25.4 \frac{(R_n - \epsilon\sigma(T_a + 273.15)^4)\Delta + E_a \left[ \gamma + \frac{4\epsilon\sigma(T_a + 273.15)^3}{f(u)} \right]}{\Delta + \gamma + \frac{4\epsilon\sigma(T_a + 273.15)^3}{f(u)}}$$

where

$E_{FWS}$  is free-water-surface evaporation in mm day<sup>-1</sup> (converted via 25.4 mm/inch)

$\epsilon$  is the broad-band emissivity of the water surface ( $\approx 0.97$  (Kohler et al. 1955:11))

$\sigma$  is the Stefan-Boltzmann constant in units of equivalent depth of evaporation, here with a value of  $7.87 \times 10^{-11}$  [inches cm<sup>-2</sup> K<sup>-4</sup> day<sup>-1</sup>] (Kohler and Parmele 1967) using the FAO standard value of  $\epsilon$  of 2.45 MJ/kg = 585 cal/g

$R_n$  is daily broad-band net radiation, expressed as [inches cm<sup>-2</sup> day<sup>-1</sup>] of equivalent evaporation. The SAMSON data files include hourly global horizontal radiation ( $R_s$ , [Wh m<sup>-2</sup>]), and hourly diffuse horizontal radiation ( $R_{dif}$ , [Wh m<sup>-2</sup>]). Taking the albedo of atmospheric radiation as 0.03 and the albedo of incident direct solar radiation as 0.06 (Kohler and Parmele 1967), net radiation is  $(0.97 R_{dif} + 0.94(R_s - R_{dif}))$ . Daily net radiation is then the sum of the hourly values, converted to Langley's (cal/cm<sup>2</sup>), divided by the enthalpy of vaporization (585 cal/g), and converted to equivalent inches of water evaporated (1g water /cm<sup>2</sup> = 0.3937 inches of evaporated water):

$$R_n = 0.086 \times 0.3937 \times \frac{0.97R_{dif} + 0.94(R_s - R_{dif})}{585} = 5.79 \times 10^{-5} (0.97R_{dif} + 0.94(R_s - R_{dif}))$$

$T_a$  is mean daily air temperature [°C]

$f(u)$  is the wind function in the aerodynamic equation i.e.,  $E_a = f(u)[e_s - e_a]$

The wind function  $f(u)$ , in which  $u_{4d}$  is daily wind movement at 4 meter height [km day<sup>-1</sup>], is  $f(u) = 0.181 + 0.00147u_{4d}$

$e_s - e_a$  is the mean daily vapor pressure deficit [inches of Hg] computed from hourly values of relative humidity  $RH_{hr}$  and air temperature  $T_{hr}$ . This quantity can also be calculated from (Lamoureux 1962):

$$e_s - e_a = 6.4133 \times 10^6 \left[ \exp \frac{-7482.6}{1.8T_a + 430.36} - \exp \frac{-7482.6}{1.8T_d + 430.36} \right]$$

$T_d$  mean daily dewpoint temperature [°C]

$u_{4d}$  is daily wind run [km day<sup>-1</sup>] calculated from hourly wind speeds at 4 m height

$\tilde{a}$  is the psychrometric coefficient, taken as a constant 0.0105 [inches of Hg °F<sup>-1</sup>], or calculated from mean daily station barometric pressure [kPa] via  $\tilde{a} = 0.000108P$ .

$\tilde{A}$  is the slope of the saturation vapor pressure versus temperature curve, here in units of [inches of Hg °F<sup>-1</sup>]

Evaporation from ponds and lakes of any considerable size will vary significantly from FWS evaporation. This can be appreciated by noting that larger lakes remain noticeably cooler than small ponds for an extended period early in the year, and remain noticeably warmer for an extended period during the Fall. Maximum lake evaporation can thus lag maximum pond or pan evaporation by several months. The Lake Hefner study ((Harbeck and Kennon 1954), cited from (Merkel 1988))

reported one to three years of evaporation data for four lakes, from which monthly multipliers for  $E_{FWS}$  can be deduced. The lakes are

- Lake Okeechobee, Florida (maximum depth 5.2 m, mean depth 2.7 m, volume  $57.6 \times 10^7 \text{ m}^3$ )
- Lake Hefner, Oklahoma (maximum depth 29 m, mean depth 8.8 m, volume  $9.2 \times 10^7 \text{ m}^3$ )
- Fort Collins Reservoir, Colorado (maximum depth about 26 m), and
- Lake Elsinore, California (mean depth 7.5 m, volume  $10.7 \times 10^7 \text{ m}^3$ )

Monthly factors for these four lakes are given in Table 2 (Merkel 1988).

Table 2. Monthly factors for conversion of free water surface evaporation  $E_{FWS}$  to large lake evaporation

Month	Factor	Month	Factor
January	0.986	July	1.014
February	0.857	August	1.079
March	0.821	September	1.129
April	0.821	October	1.166
May	0.871	November	1.179
June	0.937	December	1.143

These factors are recommended (Merkel 1988) for use with lakes similar to the four lakes from which the data were derived. (Merkel (1988) does not report variances associated with these estimates.) Few data are available for larger or smaller lakes.

### Conversion of wind speeds to standard heights

Wind speeds  $u$  reported by the NWS and incorporated into SAMSON do not include a correction for the height of the anemometer above the ground surface, which in many cases has varied significantly over the history of the observing station. These histories were assembled using data at <http://lwf.ncdc.noaa.gov/oa/climate/surfaceinventories.html>.

For this project, extrapolation of observed wind speeds to several standard heights was required:

1. The international standard height of 10 meters was used for wind speed data entered in Field 6 of the database (<http://www1.ncdc.noaa.gov/pub/data/documentlibrary/tddoc/td6421.pdf>).
2. Standardization to 2 meter height was required for calculation of FAO standard potential evapotranspiration  $ET_0$ .
3. Standardization to 0.6 meters (“nearly 2 feet above the ground level” (Farnsworth et al. 1982:3)) was required for calculation of pan evaporation and free water surface (FWS) evaporation by the

methods of Kohler et al. (1955), based on the standard Class A evaporation pan anemometer.

4. Standardization to 4 m height under meteorological station conditions was required for calculation of free water surface (FWS) evaporation by the method of Kohler and Parmele (1967).

These corrections are usually made, in the absence of available detailed studies of the wind profile at a site, by translation of observed wind speeds  $u$  at heights  $z_1$  and  $z_2$  along a logarithmic wind profile defined by the equations (Brutsaert 1982:58):

$$\bar{u}_2 - \bar{u}_1 = \frac{u_*}{k} \ln\left(\frac{z_2}{z_1}\right)$$

$$\bar{u} = \frac{u_*}{k} \ln\left(\frac{z}{z_{0m}}\right)$$

for  $z \gg z_{0m}$ , where the subscripts refer to two levels in the wind profile,  $z_{0m}$  is the momentum roughness parameter,  $u_*$  is the friction velocity, and  $k$  is von Kármán's constant.

For rough surfaces (most natural surfaces) the momentum roughness is commonly written as  $z_{0m} \equiv z_0$  where  $z_0$  is referred to as *surface roughness length* or *roughness height* (Brutsaert 1982). Proper placement of the reference height  $d_0$  (at which  $z \equiv 0$ ) is somewhat uncertain for rough surfaces. For densely placed somewhat permeable obstructions, which describes most crops, the reference level  $d_0$  (the *zero plane displacement*) is somewhere between the ground level and the crop height  $h$ ; in the FAO standard methodology  $d_0$  is taken as  $2h/3$ . More generally, then

$$\bar{u} = \frac{u_*}{k} \ln\left(\frac{z - d_0}{z_{0m}}\right)$$

Thus, to calculate the wind profile, the value of two parameters must be established: the zero-plane displacement height  $d_0$  and the roughness height  $z_0$ , of both the observing instrument and the target profile. The windspeed  $u_c$  at a height  $z_c$  above a surface with a zero-plane displacement of  $d_c$  and roughness height  $z_{0c}$  is then conventionally calculated from the observed windspeed  $u_1$  at height  $z_1$ , zero-plane displacement  $d_1$  and roughness height  $z_{01}$ , from the relationship:

$$u_c = u_1 \frac{\ln\left[\frac{z_c - d_c}{z_{0c}}\right]}{\ln\left[\frac{z_1 - d_1}{z_{01}}\right]}$$

The FAO reference surface is a hypothetical grass reference crop with an assumed crop height  $h$  of 0.12 m, a fixed surface resistance of  $70 \text{ s}^{-1}$  and an albedo of 0.23 (Allen et al. 1998:23). For such a crop, the zero plane displacement height  $d_0$  (m) and the roughness length governing momentum transfer  $z_{0m}$  are estimated as  $d_0 = (2/3)h = 0.08\text{m}$ , and  $z_{0m} = 0.123 h = 0.01476\text{m}$ . The calculation of  $d$  as  $(2h/3)$  is fairly representative (Brutsaert 1982), although  $d/h$  doubtless varies as a function of the density of the planting. However, as  $d$  appears in  $(z-d)$ , the profile functions are not very sensitive to its exact value, so long as  $z \gg z_0$ . In what follows,  $u_{10}$  designates wind speed at 10 m height,  $u_4$  wind speed at 4 m height,  $u_2$  wind speed at 2 m height, and  $u_{0.6}$  wind speed at 0.6 m ( $\approx 2$  foot) height.

Table 3 gives momentum surface roughness and zero-plane displacements for selected surfaces.

Table 3. Aerodynamic parameters for wind speed computations

Surface	Roughness Length $z_0$ (m)	Zero Plane Displacement $d_0$ (m)	Reference
Open Flat Terrain (Used for Meteorological Stations)	0.03	0.0 (none; few isolated obstacles)	(EPA 2000)
Class A Pan Anemometer	0.01476	0.08	Assumed approx. same as FAO Short Grass
FAO Reference Short-Grass Crop	0.01476	0.08	(Allen et al. 1998)
Open sea, fetch > 5km	0.0002	Depends on sea state	(EPA 2000)
Large Water Surfaces	0.0001-0.0006 0.000228	Depends on sea state	(Brutsaert 1982)

For wind speeds at 10 m height derived from SAMSON observations, extrapolation along an observing station site logarithmic wind profile was accomplished by taking  $d_c = d_l = 0\text{m}$ , and  $z_{0c} = z_{0l} = 0.03\text{m}$ . Inserting these numerical values in the equation for calculated wind speed  $u_c$  at 10 meter height (i.e., calculating  $u_{10}$  from SAMSON observations of meteorological observing station (airport) wind speed  $u_a$  at anemometer height  $z_a$  [m]) yields

$$u_{10} = u_a \frac{\ln\left(\frac{10.0-0.0}{0.03}\right)}{\ln\left(\frac{z_a-0.0}{0.03}\right)} = u_a \frac{5.81}{\ln(z_a / 0.03)}$$

Similarly, for wind speed at 2 meter height above the FAO reference short-grass crop surface  $u_2$ ,

$$u_2 = u_a \frac{\ln\left(\frac{2.00-0.08}{0.01476}\right)}{\ln\left(\frac{z_a-0.0}{0.03}\right)} = u_a \frac{4.87}{\ln(z_a / 0.03)}$$

For computation of meteorological station wind speed at 4 m height (for use in calculation of free water surface evaporation by the Kohler-Parmele equation (Kohler and Parmele 1967)),

$$u_4 = u_a \frac{\ln\left(\frac{4.00-0.0}{0.000228}\right)}{\ln\left(\frac{z_a-0.0}{0.03}\right)} = u_a \frac{9.77}{\ln(z_a / 0.03)}$$

For wind speed at the Class A pan anemometer height of 0.6 m ( $\approx 2$  feet)  $u_{0.6}$ ,

$$u_{0.6} = u_a \frac{\ln\left(\frac{0.6-0.08}{0.01476}\right)}{\ln\left(\frac{z_a-0.0}{0.03}\right)} = u_a \frac{3.56}{\ln(z_a / 0.03)}$$

For EXAMS, the meteorological station wind speeds at 10 m height must be translated to a height of 0.1 m above an open water surface. This transformation is executed automatically by EXAMS during the course of reading the meteorological data file:

$$u_{0.10} = u_{10} \frac{\ln\left(\frac{0.10-0.0}{0.000228}\right)}{\ln\left(\frac{10.0-0.0}{0.03}\right)} = 1.05u_{10}$$

## Processing Sequence for Production of Daily Values Files (\*.dvh)

- ◆ The program “make\_r0” generates all *dvh* files. The data structures are defined in GLOBAL.F90. The derived type for each datum contains the source of the item, any flags associated with the item, and its value.
- ◆ Subroutine Driver0 (in file RAW\_DATA.F90) reads several files containing general information pertaining to each weather station.
  - “SAMSON STATION NOTES.TXT” : Contains, for each weather station: the name of the station, its WBAN number, location (State), latitude and longitude (degrees and minutes), elevation (meters), and time zone.
  - “ANEMOMETER HEIGHTS.PRN” : Contains, for each station, the height of the anemometer (feet) during a given period (yyyy-mm-dd). The data were collected from <http://lwf.ncdc.noaa.gov/oa/climate/surfaceinventories.html>.

Example:

03103	30	1950-01-12
	20	1965-10-07

For weather station 03103 (Flagstaff, AZ), the height of the anemometer was 30 feet during the period Jan 1, 1950 to October 6, 1965, and 20 feet from October 7, 1965 to present date. The elevation data is used to normalize the SAMSON Wind\_Speed value to 10 meters. If no instrument height was available, assume the measurement was made at 30 feet (9.1 meter). A reference height of 10 meters or about 30 feet is internationally recommended as the standard, and anemometers are usually mounted as close to this height as is practical. Station histories were not available for 24 stations; these (24) instruments were assumed to be sited at a standard height to minimize changes to the observed data.

- Driver0 calls an “internal standard” module (Internal\_Standard, file RAW\_DATA.F90), to verify that certain parts of the program are behaving correctly.
  - Finally, Driver0 calls Read\_R0\_List, which reads the list of stations to be processed (contained in 1.SAMSON.LIST.TXT) and calls Process\_One\_WBAN\_Station to process each station.
- ◆ Process\_One\_WBAN\_Station (in RAW\_DATA.F90)
  - ◆ Subroutine Station\_With\_Missing\_Data: determines the completeness of the precipitation data. The SAMSON format document (file samson\_format.txt) identifies some stations as “Stations with Little or No Hourly Precipitation Data ”.

- ◆ Subroutine Read\_SAMSON\_v1x reads and stores each SAMSON data file (versions 1.0 and 1.1). The module verifies that the values read are within the range described in the SAMSON format document (file samson\_format.txt). The source of the datum and any associated flags are stored. Values outside the prescribed range are considered missing. Any errors detected are issued to the log file.
  - Unit conversion is performed (e.g., hourly precipitation: from hundredths of an inch to cm; station pressure: from mbar to kPa; wind speed normalized to a height of 10 meters.
  - The module also determines the maximum Ceiling Height and the maximum Horizontal Visibility (over all years, nominally 1961-1990). (See Process\_Set, *vide infra*.)
  - The undocumented value “990” was sometimes present for Wind\_Direction (valid range 0-360; missing value flag: 999). The observation was considered missing.
  
- ◆ Subroutine Issue\_Years: determines which years will be output based on the precipitation record. If the hourly precipitation was be incomplete (or missing), the daily precipitation record EarthInfo NCDC Summary of the Day and Surface Airways, 2001 was examined to determine which years had complete daily records. The year ranges present in the subroutine were arrived “by hand”: a previous run of “make\_r0” would show incomplete hourly precipitation for a particular station. For that station, the NCDC Summary of the day would be examined and if complete, the records would be exported so that “make\_r0” would supplement the precipitation on subsequent executions.
  
- ◆ Subroutine Process\_Set:
  - Replaces the radiation data from SAMSON v 1.0 with version 1.1.
  - Read hourly and daily precipitation from EarthInfo CD.
  - Coordinates the output of other processing modules.
  - Horizontal Visibility field nominal range: 0.0-160.9 (kilometers). SAMSON used the value “777.7” to denote unlimited visibility. The flag value was replaced with 110% of the maximum unlimited visibility observed during the nominal period 1961-1990. The datum was flagged with “U”.
  - Ceiling Height field nominal range: 0-30450 (meters). The value “77777” denoted unlimited ceiling height; “88888” denoted cirroform.. The flag values were replaced with 110% of the maximum ceiling height observed during the nominal period 1961-1990. The datum was flagged with “U” (unlimited) or “Z” (cirroform).
  - Gaps in the hourly data record were filled according to methods described in National Solar Radiation Data Base User's Manual (1961-1990), NSRDB Volume 1, September 1991, Section 5.2.1. The document may be downloaded from [http://rredc.nrel.gov/solar/pubs/NSRDB/NSRDB\\_index.html](http://rredc.nrel.gov/solar/pubs/NSRDB/NSRDB_index.html). In general:
    - Gaps of less than 6 hours were filled by linear interpolation between points at both sides of the gap.
    - Gaps of 6 hours to 49 hours were filled by copying data from the previous (or following) day.

- Longer gaps of 50 hours to 8784 hours (one leap year) were filled by copying data for the same period from some other year.

- ◆ Standardize\_ppt – ‘D’ (deleted) points are flagged “missing”; periods of missing data are identified and all intervening hours are flagged missing. Accumulation periods, i.e., periods in which only the total accumulation is known (e.g., 0.12 cm fell over five hours) are recorded for later processing.
- ◆ Process\_BAOD (Broadband aerosol optical\_depth (broadband turbidity)) – BAOD is not present during nighttime. See fragment file below (13893\_61.txt). To generate nighttime values: starting at 1 h, find the first non-missing BAOD value, call it "rv". Replace all missing values during that day with "rv". For the example, all missing values (99999.) for 1961-01-01 will be replaced with 0.034.

yy	mm	dd	hh	BAOD	
61	1	1	1	99999.	
61	1	1	:	99999.	! hours 2-6
61	1	1	7	99999.	
61	1	1	8	.034	
61	1	1	:	.034	! hours 9-16
61	1	1	17	.034	
61	1	1	18	99999.	
61	1	1	:	99999.	! hours 19-6
61	1	2	7	99999.	
61	1	2	8	.104	
61	1	2	9	.104	! etc.

- ◆ Process\_Days\_since\_last\_Snowfall – The SAMSON value of (DSLS was sometimes missing or not consistent with fields 4 and 5 of the Present\_Weather flag. Algorithm: accept non-missing values of DSLS, otherwise utilize field 4 (Occurrence of Snow, Snow Pellets, or Ice Crystals) or field 5 (Occurrence of Snow Showers, or Snow Squalls) to determine if snow occurred during the 24 hour period.
- ◆ Daylight Prevailing wind direction: indicates the most frequent wind direction observed during the daylight hours of a given day. (Wind direction indicates where the wind is coming from.).

Algorithm: First, determine the most frequent quadrant, which is defined as the quadrant with the most observations. If there is a tie, among the tied quadrants select the quadrant with the largest wind speed. If tied, select the quadrant where the maximum wind speed falls closest to noon. The prevailing wind direction is the median of the wind directions that fell in the most frequent quadrant. The prevailing wind speed is the mean of the wind speeds that fell in the most frequent quadrant. See subroutine DAYLIGHT\_PREVAILING\_WIND for complete details.

## Notes and Irregularities

Unless stated otherwise, these comments pertain to the SAMSON CDs.

- 1 Two files (years) missing from the SAMSON CDs: Burns, OR (94185): 1989, and Miles City, MT (24037) 1990. In addition, the SAMSON CDs also contained empty data files.
- 2 The files for WBAN Station Number 14847 (SAULT STE. MARIE, MI) were duplicated in two CDs: 1961-1990: Z:\nrel0001\data\14847, Z:\nrel0002\data\14847. The files were identical: same date & size; A file by file comparison using *cmp* showed both sets were identical. All other "MI" stations were present only in "nrel0002". So, the files in Z:\nrel0001\data\14847 were ignored.
- 3 Some SAMSON/HUSWO data files contained the undocumented value "E" (estimated) for the precipitation flag. The SAMSON present weather flag had values not described in the SAMSON document. These flags are described in Appendix C (Pages 48-49) of Database Guide for EarthInfo CD^2 NCDC Surface Airways. The SAMSON document was augmented to include the missing flags.
- 4 Expected number of observations: 8760 (365\*24) or 8784 (Leap year). Some HUSWO stations are missing the observation for 1995-12-31 24h.
- 5 For some years, observations were taken every three hours.
- 6 SAMSON v 1.0 files. The last fields (Hourly precipitation and Hourly precipitation flag) are not written in the original SAMSON file unless the fields are different from zero. For example  
1961010101 ...(other parameters)... 10  
1961010102 ...(other parameters)...  
e.g., it rained 10/100 inches during the first hour; no rain (implicit 0) during the second hour.
- 7 Wind Speed daily value == (Sum V<sub>i</sub>) / 24
- 8 Evaporation formula: wind run == Sum V<sub>i</sub>
- 9 If the station is above the Arctic Circle (Latitude 66.5 degrees North), then the region will be in darkness for a period of the year, preventing the computation of, e.g., daylight value of Global Horizontal Radiation, and Reference Crop Evapotranspiration (E<sub>to</sub>)
- 10 Evaporation files from EarthInfo. Determine the associated Cooperative Station for a given WBAN number (<http://lwf.ncdc.noaa.gov/oa/climate/stationlocator.html>) For the continental USA:
- 11 Cooperative Stations Index: <http://lwf.ncdc.noaa.gov/oa/climate/surfaceinventories.html>  
<http://lwf.ncdc.noaa.gov/oa/climate/stationlocator.html>  
<ftp://ftp.ncdc.noaa.gov/pub/data/inventories/COOP.TXT> Historical cooperative station index. Cooperative stations are U.S. stations operated by local observers which generally report max/min temperatures and precipitation. National Weather Service (NWS) data are also included in this

dataset. The data receive extensive automated + manual quality control. The index includes a county location cross-reference. Over 8000 stations are currently active across the country.

12 All stations: <http://lwf.ncdc.noaa.gov/oa/climate/stationlocator.html>

#24. Longer Gaps. <A NAME="Note\_24">  
See <A HREF="Gaps.f90#Fill\_Medium\_Gaps">

#25. 14 Feb 2002 4:34 pm. <A NAME="Note\_25"> <A HREF="0notes.txt#Note\_25">  
See <A HREF="samson.f90#Missing years">  
See <A HREF="4.missing.years.bat#\$1">  
See <A HREF="Utils2.f90#Missing years">  
24013 !!! Minot, ND;  
Minot is missing years 1989 and 1990.  
We will copy  
-> 24013\_89.txt  
-> 24013\_90.txt  
Run make\_r0, and manually delete those missing years  
from the r0 directory, and eliminate those years from  
the MET file.

\* 15 Feb 2002 5:02 pm; I modified make\_r0 to  
#1. determine missing files (on the fly)  
#2. fill missing block of data  
#3. not to produce the r0 file associated with the missing year(s)  
#4. not to generate the MET records for the missing year(s)

#26. <A NAME="Note\_26"> <A HREF="0notes.txt#Note\_26">  
Dew Point.  
See <A HREF="Utils1.f90#DewPoint">

#27. <A NAME="Note\_27"> <A HREF="0notes.txt#Note\_27">  
Accumulation flag in hourly precipitation field --  
The description and example presented in  
<A HREF="e:\5\3met\Docs\samson\_format.txt#\$1">  
is different to the actual usage in the SAMSON files.  
See <A HREF="Accum.f90#Fill\_Accumulation">

#28. Hourly Precipitation: Cut Bank, MT (WBAN: 24137)  
26 Mar 2002 10:02 am.  
Relevant SAMSON entries:  
1983-12-09 11h 0M ! Begin missing run  
1983-12-23 12h 0M ! End missing run  
1983-12-30 9h 22A ! End accumulation run  
The SAMSON file contains a terminating accumulation run flag without the concomitant starting

flag.

The EarthInfo record for that period --

```

                                HOURLY
Station      CUT BANK FCWOS      % Coverage      90
PO Code      MT                  Latitude      N48:36:30     Begin M/Yr 07/1948
Station ID   2173                  Longitude    W112:22:34    End M/Yr   12/2000
County       GLACIER                Elevation    3838          # Record Years 53
-----
                                Prcp (in.) -----
                                +000    +100    +200    +300    +400    +500
12/09/1983  0100      .        .        .        .        .        .
              0700      .        .        .        .        .        .
              1300     ---     ---     ---     ---     ---     ---
              1900     ---     ---     ---     ---     ---     ---
12/23/1983  0100      .        .        .        .        .        .
              0700      .        .        .        .        .        .
              1300     ---     ---     ---     ---     ---     ---
              1900     ---     ---     ---     ---     ---     ---
12/30/1983  0100     ---     ---     ---     ---     ---     ---
              0700     ---     ---     0.22 A   .        .        .
              1300      .        .        .        .        .        .
              1900      .        .        .        .        .        .
```

i.e, 1983-12-09 11 h begins the missing run (m).

There is no terminating M for this run.

1983-12-23 12 h begins the accumulation run (a), which terminates  
on 1983-12-30 9h (A)

So we will edit the SAMSON file Z:\nrel0003\data\24137\24137\_83.z  
to read:

```
1983-12-09 11h 0M ! Begin missing run (entry unchanged)
1983-12-23 11h 0M ! *** new entry: End missing run.
1983-12-23 12h 0A ! *** altered entry: Begin accumulation run
1983-12-30 9h 22A ! End accumulation run (entry unchanged)
```

WIN> ! File extracted using WinZip

```
DOS> mkszip -v 24137_83.txt ! Compress file, output to 24137_83.txt.gz
! Rename 24137_83.txt.gz -> 24137_83.z
```

#29. Hourly Precipitation: Williamsport, PA (WBAN: 14778)

27 Mar 2002 8:55 am

Relevant SAMSON entries:

```
1984-04-21 10h 99999A ! Begin accumulation run
...                  ! many intervening Begin/End Missing data pairs
1986-07-31 24h 99999M ! End missing run -- correctly paired.
End-of-data reached (1990) without matching the "Begin accumulation run"
of 1984-04-21 10h
```

The SAMSON file contains a terminating accumulation run flag without the concomitant starting flag.  
The EarthInfo precipitation files show no hourly data, only monthly data.

So we will edit the last record of the SAMSON file

Z:\nrel0001\data\14778\14778\_90.z to read:

```
90 12 31 24 <stuff> .9999999 01A
```

This will fix the missing matching "A".

An accumulation value of Zero will cause problems, therefore we will use 1, i.e., 1/100 inch. The precipitation data will be compared with the EarthInfo hourly and daily precipitation, and fixed when necessary.

WIN> ! File extracted using WinZip

DOS> mkszip -v 14778\_90.txt ! Compress file, output to 14778\_90.txt.gz

! Rename 14778\_90.txt.gz -> 14778\_90.z

#30. <A NAME="Note\_30"> <A HREF="0notes.txt#Note\_30">

From: Prieto.Lourdes@epamail.epa.gov

Subject: Metadata for weather station data files

To: Suarez.Luis@epamail.epa.gov

DeliveredDate: 07/01/2002 12:12:26 PM

For anemometer heights:

<http://lwf.ncdc.noaa.gov/oa/climate/surfaceinventories.html>

National Weather Service Station Histories – file “station-hist.zip” at

<ftp://ftp.ncdc.noaa.gov/pub/data/inventories>

Zip-archival version of the NWS station histories file.

To check lat/long and station elevation, checked individual stations here:

<http://lwf.ncdc.noaa.gov/oa/climate/stationlocator.html>

# Fortran Processing Code

## BinaryTree

```
!      Last change:  LSR   6 Jun 2002   3:17 pm

Module Binary_Tree

  Use Global_Variables
  Use Linked_List
  Use Red_Black_Binary_Tree

  Implicit None

  ! Store info in a binary tree since we want sorted output.
  ! See
  ! [] Cooper Redwine. 1995. Upgrading to Fortran 90.
  !   Springer Verlag; ISBN: 0387979956; pages 333-343.

Contains

  Subroutine Initialize_Binary_Tree(Xstations)

    Implicit None
    Type(Site_Info), Pointer :: Xstations ! Pointer to root of tree

    Nullify(Xstations) ! Disassociate pointer to root node of tree
  End Subroutine Initialize_Binary_Tree

  Subroutine Get_Node(Xroot, WBAN_id, Node_Was_New, Xnew)

    ! *** [] Robin A. Vowels. 1998. Algorithms and data
    ! ***   structures in F and Fortran. Pages 98-130.
    ! ***   ISBN: 0-9640135-4-1
    ! ***
    ! ***   Figure 3.14 Algorithm for manipulating a
    ! ***           Red-Black binary tree
    ! ***
    !
    ! This procedure either:
    ! #1. Creates and inserts a new node (Xnew) for the given WBAN number, or
    ! #2. Points Xnew to an existing node in Xroot with the WBAN number.
    ! On output Xnew points to the node containing the WBAN number.

    Implicit None
    Type(Site_Info),   Pointer :: Xroot
```

```

Character(Len=*), Intent(In) :: WBAN_id
Logical,          Intent(Out) :: Node_Was_New
Type(Site_Info),  Pointer :: Xnew  ! Intent(Out)

Allocate(Xnew)      ! Create a node.

! Binary tree initialization.
Xnew%Color = Red      ! New nodes are always Red.
Xnew%WBAN = WBAN_id
Nullify(Xnew%Parent)
Nullify(Xnew%pLeft)
Nullify(Xnew%pRight)

Call Insert_In_Tree(Xroot, Xnew, Node_Was_New) ! Insert or point to WBAN
If (Node_Was_New) Then
    Call Initialize_Node(Xnew, WBAN_id)
End If

! Sanity Check.
If (Xnew%WBAN /= WBAN_id) Then
    Write (6,9130) '?? Xnew%WBAN ....: ', Trim(Xnew%WBAN)
    Write (6,9130) '?? WBAN_id .....: ', Trim(WBAN_id)
    Write (ULog,9130) '?? Xnew%WBAN ....: ', Trim(Xnew%WBAN)
    Write (ULog,9130) '?? WBAN_id .....: ', Trim(WBAN_id)
9130    Format (1x, a, ' ', a, ' ')
    Stop '?? Xnew%WBAN /= WBAN_id'
End If

End Subroutine Get_Node

Subroutine Initialize_Node(Xnew, WBAN_id)

! <A NAME="Initialize_Node">
! <A HREF="BinaryTree.f90#Initialize_Node">

Implicit None
Type(Site_Info),  Pointer :: Xnew      ! Intent(InOut)
Character(Len=*), Intent(In) :: WBAN_id

Xnew%item = 0      ! Unused. Remnant of Red_Black derived type.
Xnew%Color = .True. ! Unused. Remnant. Which can assume RED or BLACK.

Xnew%WBAN = WBAN_id
Xnew%State = ''
Xnew%Text = ''
Xnew%Lat = Coords('', 0, 0) ! Latitude 'N' in radians; e.g., N 40 1
Xnew%Lon = Coords('', 0, 0) ! Longitude 'W' in radians; e.g., W 105 15
Xnew%Lat_radians = 0.0
Xnew%Lon_radians = 0.0

```

```
Xnew%Elev = 0           ! 1634  
Xnew%TZ = ''           ! +7(T)  
Xnew%iTZ = 0           ! 7
```

```
Xnew%Nelev = 0
```

```
End Subroutine Initialize_Node
```

```
End Module Binary_Tree
```

## dump

! Last change: LSR 7 Jun 2002 10:06 am

Module Dump\_R0

```
!!!          1: Blank space
!!!          2- 11: Date: yyyy-mm-dd, e.g., 1961-12-31
!!!          12: Blank space
!!!          13- 14: Hour of the day (01-25)
```

! List generated by hvf\_Cols

```
!!!          1  16- 21: Extraterrestrial Horizontal Radiation
!!!          2  23- 28: Extraterrestrial Direct Normal Radiation
!!!          3  30- 37: Global Horizontal Radiation
!!!          4  39- 46: Direct Normal Radiation
!!!          5  48- 55: Diffuse Horizontal Radiation
!!!          6  57- 59: Total Sky_Cover
!!!          7  61- 63: Opaque Sky_Cover
!!!          8  65- 70: Dry Bulb Temperature
!!!          9  72- 77: Dew_Point Temperature
!!!         10  79- 82: Relative Humidity
!!!         11  84- 89: Station Pressure
!!!         12  91- 94: Wind Direction
!!!         13  96-101: Wind Speed
!!!         14 103-109: Visibility
!!!         15 111-117: Ceiling Height
!!!         16 119-120: Observation Indicator
!!!         17 122-131: Present_weather
!!!         18 133-136: Precipitable Water
!!!         19 138-144: Broadband Aerosol Optical_Depth
!!!         20 146-150: Snow_Depth
!!!         21 152-155: Days Since Last Snowfall
!!!         22 157-164: Hourly Precipitation
!!!         23 166-172: FAO Short Grass PET
!!!         24 174-180: Pan Evaporation
!!!         25 182-188: K-P FWS Evaporation
```

```
!Use Binary_Tree
!Use FileStuff
!Use GetNumbers
!Use Linked_List
!Use Read_Info
!Use SAMSON
!Use Strings
!Use Utils0
!Use Utils1
Use Date_Module
Use Global_Variables
```

```

Use IoSubs
Use Utils2
Use Utils3
Use Winteracter
Implicit None

```

```

Logical, Save :: first_time = .True.

```

Contains

```

Subroutine Dump_R0_One_Year(y4)

```

```

! This routine dumps previously generated data
! All fields will have a flag.

```

```

Implicit None

```

```

Integer,          Intent(In) :: y4

```

```

! The format statement has to changed by hand.
! Look for all instances of "NNtext".
! a30 below == a<Maximum_Text_Length> == a<NNtext>
Integer, Parameter :: NNtext = 30 ! Maximum_Text_Length
Character(Len=30)  :: f1 = '---- ' ! value with one flag
Character(Len=30)  :: f2 = '----- ' ! value with two flag positions
Character(Len=30)  :: f3 = '-----' ! value with three flag positions
Character(Len=MaxNamLen) :: r0_file, v0_file
Character(Len=NNtext) :: qtext
Character(Len=250) :: tbuf, q0buf
Integer :: Out_r0, In_v0, Out_v0
Integer :: k0, kl, klbase, j, jpar, ierr, n_missing
Integer :: tlen, iipar, LenF
Logical :: is_a_number
Integer :: nbase, iv, doy
Integer :: yyyy, mm, dd, hh
Integer :: jd_begin, jd_end, jd_today ! julian days
Logical :: xok, qdebug, w0, one_file, cols_now
Integer, Dimension(0:f_end) :: id_of_missing

```

```

ierr = 0
id_of_missing = 0

```

```

qdebug = .True.
!one_file = .True.
one_file = .False.
Select Case(y4)

```

```

!!! Case(1961, 1962, 1964, 1965, 1990) ! Fargo
Case(1968, 1987) ! Memphis
    w0 = .True.
Case Default
    w0 = .False.

```

```

End Select
w0 = (qdebug .And. w0) .And. (.False.)

If (w0) Then
  Call Decompress_and_Open_File(In_v0, Year_Names(y4)%Samson_v10)
  !Call IORead(In_v0, Year_Names(y4)%Samson_v11)
  If (one_file) Then
    ! Do nothing
  Else
    Write (v0_file, '(3a,"_",i2.2,".v10")') &
      '.', '/', Trim(pWBAN%WBAN), Modulo(y4,100)

    Call IOWrite(Out_v0, v0_file)
  End If
End If

Errors_Detected = .False.

r0_file = name_r0(y4)
Call IOWrite(Out_r0, r0_file, Ok=xok)
If (.Not. xok) Then
  Write (ULog, *) '?? Could not open r0 file ', Trim(r0_file)
  Errors_Detected = .True.
  Return
End If

!Call ToTTY('Dump_R0_One_Year '//Trim(Itoa(y4))//' '//Trim(r0_file))
!If (first_time) Call hvf_Cols('', 0, 0, '', Initialize=.True.)

LenF = Len(Xparam(f_OI)%Samson_v10(1)%f)

! Trim(pWBAN%WBAN), Trim(pWBAN%Text)
j = Index(pWBAN%Text, ',', Back=.True.)
k0 = j - 1      ! City
k1 = j + 2      ! State
qtext = pWBAN%Text(1:k0)
If (k0 > NNtext) Stop '?? Stopping in Dump_R0_One_Year: Len(Text) > NNtext'
If (k0 > Maximum_Text_Length) Stop '?? Stopping in Dump_R0_One_Year: Len(Text) > Maximum_Text_Length'

j = Index(pWBAN%TZ, '(') - 1

Write (Out_r0, 9130) &
  Trim(pWBAN%WBAN),          qtext, pWBAN%Text(k1:k1+1), &
  pWBAN%TZ(1:j), &
  pWBAN%Lat%Letter, pWBAN%Lat%degrees, pWBAN%Lat%minutes, &
  pWBAN%Lon%Letter, pWBAN%Lon%degrees, pWBAN%Lon%minutes, &
  Nint(pWBAN%Elev), &
  Trim(TimeStamp)

```

```

9130 Format (&
      1x, a5, 1x, a30, 1x, a2, &      ! a<NNtext>
      1x, a3, &
      2x, a1, i4, 1x, i2, &
      2x, a1, i4, 1x, i2, &
      2x, i4, &
      3x, a)

If (w0) Then
  Read(In_v0, '(a)') q0buf
  If (one_file) Then
    Write(Out_r0, '(a,a)') 'v0>', Trim(q0buf)
  Else
    Write(Out_v0, '(a,a)') Trim(q0buf)
  End If
End If

nbase = Hours_since_Jd0(y4)
iv = nbase
doy = 0

jd_begin = Jd(y4, 01, 01)
jd_end = Jd(y4, 12, 31)
jd_today = jd_begin
n_missing = 0

Do
  If (jd_today > jd_end) Exit

  ! Store data in a vector array so that we can find gaps easier.
  ! Assume a virtual array: vData(ndays, Nhours)
  !   Ndays = 365 or 366 == WFlx%Expected_Ndays
  !
  !   Ndays|  1    2  3  4  ...    24    25    Nhours = 25
  !   -----+-----
  !   1 |  1    2  3  4  ...    24    25
  !   2 | 26   27 28 29 ...    49    50
  !   : |   :   :  :  :  ...    :     :
  ! 365| 9101 :   :  :  ...   9124  9125
  ! 366| 9126 .   .  .  ...   9149  9420
  !
  ! Samson_v1x(iv) <==> vData(doy,hh)
  !   iv = (doy-1)*Nhours + hh
  !
  !   doy = (iv+Nhours-1) / Nhours
  !   hh = iv - (doy-1)*Nhours
  !
  ! nbase = Hours_since_Jd0(yyyy)
  ! iv = (doy-1)*Nhours + hh + nbase
  ! Xparam(f_EHR)%Samson_v10(iv)%v = EHR

```



```

Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i5, a1)') &
    Int(Xparam(f_EHR)%Samson_v10(iv)%v), &
    Xparam(f_EHR)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i5, a1)', Icount=4)

! 02 Extraterrestrial Direct      Amount of solar radiation in Wh/m2
!   Normal Radiation             0-1415 received on a surface normal to
!                               the sun at the top of the atmosphere
!                               during the 60 minutes preceding the
!                               hour indicated.
!
iipar = f_EDNR
k0 = k1 + 2
k1 = k0 - 1 + 6
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i5, a1)') &
    Int(Xparam(f_EDNR)%Samson_v10(iv)%v), &
    Xparam(f_EDNR)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i5, a1)')

! 03 Global Horizontal Radiation  Total amount of direct and diffuse solar
!   Data Value                   0-1415 radiation in Wh/m2 received on a
!   Flag for Data Source          A-H, ? horizontal surface during the 60 minutes
!   Flag for Data Uncertainty     0-9   preceding the hour indicated.

```

```

!
! Table 3-6. Solar Radiation Source Flags
!
! Flag Definition
!
! A Post-1976 measured solar radiation data as received from
! NCDC or other sources
! B Same as 'A' except the global horizontal data underwent a calibration
! correction
! C Pre-1976 measured global horizontal data (direct and diffuse were
! not measured before 1976), adjusted from solar to local time, usually
! with a calibration correction
! D Data derived from the other two elements of solar radiation using the
! relation,  $K_t = K_n + K_d$ 
! E Modeled solar radiation data using inputs of observed sky_cover
! (cloud amount) and aerosol optical_depth s derived from direct
! normal data collected at the same location
! F Modeled solar radiation using interpolated sky_cover and aerosol
! optical_depths derived from direct normal data collected at the
! same location
! G Modeled solar radiation data using observed sky_cover and aerosol
! optical_depths estimated from geographical relationships
! H Modeled solar radiation data using interpolated sky_cover and
! estimated aerosol optical_depths
! ? Source does not fit any of the above categories. Used for nighttime
! values, calculated extraterrestrial values, and missing data
!
!

```

```

! Table 3-7. Solar Radiation Uncertainty Flags
!

```

Flag	Uncertainty Range (%)
1	0 - 2
2	2 - 4
3	4 - 6
4	6 - 9
5	9 - 13
6	13 - 18
7	18 - 25
8	25 - 35
9	35 - 50
0	Not applicable

```

iipar = f_GHR
k0 = k1 + 2
k1 = k0 - 1 + 8 ! i5 + s(1:1) + f(1:2)
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
is_a_number = .False.
n_missing = n_missing + 1

```

```

    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(i5, a1, a2)' &
        Int(Xparam(f_GHR)%Samson_v10(iv)%v), &
        Xparam(f_GHR)%Samson_v10(iv)%s(1:1), &
        Xparam(f_GHR)%Samson_v10(iv)%f(1:2))
Else
    tbuf(k0:k1) = Adjustr(f3(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i5, a1, a2)')

! 04 Direct Normal Radiation          Amount of solar radiation in Wh/m2
!   Data Value                        0-1415 received within a 5.7o field of view
!   Flag for Data Source               A-H, ? centered on the sun, during the 60
!   Flag for Data Uncertainty         0-9 minutes preceding the hour indicated.
iipar = f_DNR
k0 = k1 + 2
k1 = k0 - 1 + 8 ! i5 + s(1:1) + f(1:2)
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(i5, a1, a2)' &
        Int(Xparam(f_DNR)%Samson_v10(iv)%v), &
        Xparam(f_DNR)%Samson_v10(iv)%s(1:1), &
        Xparam(f_DNR)%Samson_v10(iv)%f(1:2))
Else
    tbuf(k0:k1) = Adjustr(f3(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i5, a1, a2)')

! 05 Diffuse Horizontal Radiation     Amount of solar radiation in Wh/m2
!   Data Value                        0-1415 received from the sky (excluding the
!   Flag for Data Source               A-H, ? solar disk) on a horizontal surface,
!   Flag for Data Uncertainty         0-9 during the 60 minutes preceding the
!                                     hour indicated.
iipar = f_DHR
k0 = k1 + 2

```

```

k1 = k0 - 1 + 8 ! i5 + s(1:1) + f(1:2)
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i5, a1, a2)' &
    Int(Xparam(f_DHR)%Samson_v10(iv)%v), &
    Xparam(f_DHR)%Samson_v10(iv)%s(1:1), &
    Xparam(f_DHR)%Samson_v10(iv)%f(1:2))
Else
  tbuf(k0:k1) = Adjustr(f3(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i5, a1, a2)')

! 06 Total Sky_Cover          0-10    Amount of sky dome (in tenths)
!                               covered by clouds.
iipar = f_TSC
k0 = k1 + 2
k1 = k0 - 1 + 3
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i2, a1)' &
    Int(Xparam(f_TSC)%Samson_v10(iv)%v), &
    Xparam(f_TSC)%Samson_v10(iv)%s(1:1))
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i2, a1)')

! 07 Opaque Sky_Cover        0-10    Amount of sky dome (in tenths)
!                               covered by clouds that prevent
!                               observing the sky or higher cloud
!                               layers.
iipar = f_OSC

```

```

k0 = k1 + 2
k1 = k0 - 1 + 3
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(i2, a1)') &
        Int(Xparam(f_OSC)%Samson_v10(iv)%v), &
        Xparam(f_OSC)%Samson_v10(iv)%s(1:1)
Else
    tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i2, a1)')

! 08 Dry Bulb Temperature      -70.0 to   Dry bulb temperature in degrees C.
!                               60.0
iipar = f_DBT
k0 = k1 + 2
k1 = k0 - 1 + 6
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(f5.1, a1)') &
        Xparam(f_DBT)%Samson_v10(iv)%v, &
        Xparam(f_DBT)%Samson_v10(iv)%s(1:1)
Else
    tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f5.1, a1)')

! 09 Dew_Point Temperature     -70.0 to   Dew_point temperature in degrees C.
!                               60.0
iipar = f_DPT
k0 = k1 + 2
k1 = k0 - 1 + 6

```

```

Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(f5.1, a1)') &
    Xparam(f_DPT)%Samson_v10(iv)%v, &
    Xparam(f_DPT)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f5.1, a1)')

! 10 Relative Humidity          0-100   Relative humidity in percent.
iipar = f_RH
k0 = k1 + 2
k1 = k0 - 1 + 4
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i3, a1)') &
    Int(Xparam(f_RH)%Samson_v10(iv)%v), &
    Xparam(f_RH)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i3, a1)')

! 11 Station Pressure          70.0-110.0   Station pressure in kilopascals.
iipar = f_SP
k0 = k1 + 2
k1 = k0 - 1 + 6
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1

```

```

        id_of_missing(iipar) = id_of_missing(iipar) + 1
    Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        is_a_number = .False.
    Case Default
        is_a_number = .True.
    End Select
    If (is_a_number) Then
        Write(tbuf(k0:k1), '(f5.1, a1)') &
            Xparam(f_SP)%Samson_v10(iv)%v, &
            Xparam(f_SP)%Samson_v10(iv)%s(1:1)
    Else
        tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
    End If
    !If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f5.1, a1)')

! 12 Wind Direction          0-360      Wind direction in degrees.
!                               (N = 0 or 360, E = 90, S = 180,
!                               W = 270)
iipar = f_WD
k0 = k1 + 2
k1 = k0 - 1 + 4
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(i3, a1)') &
        Int(Xparam(f_WD)%Samson_v10(iv)%v), &
        Xparam(f_WD)%Samson_v10(iv)%s(1:1)
Else
    tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i3, a1)')

! 13 Wind Speed              0.0-99.0   Wind speed in m/s at z=10 meters
iipar = f_WS
k0 = k1 + 2
k1 = k0 - 1 + 6
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)

```

```

        is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(f5.1, a1)') &
        Xparam(f_WS)%Samson_v10(iv)%v, &
        Xparam(f_WS)%Samson_v10(iv)%s(1:1)
Else
    tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f5.1, a1)')

! 14 Visibility                0.0-160.9    Horizontal visibility in
!                               kilometers.
iipar = f_HV
k0 = k1 + 2
k1 = k0 - 1 + 7
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(f6.1, a1)') &
        Xparam(f_HV)%Samson_v10(iv)%v, &
        Xparam(f_HV)%Samson_v10(iv)%s(1:1)
Else
    tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f6.1, a1)')

! 15 Ceiling Height           0-30450      Ceiling height in meters.
iipar = f_CH
k0 = k1 + 2
k1 = k0 - 1 + 7
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.

```

```

End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i6, a1)') &
    Int(Xparam(f_CH)%Samson_v10(iv)%v), &
    Xparam(f_CH)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i6, a1)')

! 16 Observation Indicator      0 or 9      0 = Weather observation made.
!                               9 = Weather observation not
!                               made or missing.
!   [lsr] Thu 19 Oct 2000 11:58:19           If this field = 9 OR if field
!   This field appears to be                 13 (wind speed) = missing
!   column 96 of the input file.             (9999. or 99.0), then
!   The manual provides no data             fields 6, 7, 8, 10, 11, 17,
!   to support this assertion.              and 18 were all modeled and
!   [lsr] Wed Nov 21 09:33:59 2001          not actually observed.
!   Observation Indicator appears           only in SAMSON v 1.0 files.
!   only in SAMSON v 1.0 files.

! Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
! Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
iipar = f_OI
k0 = k1 + 2
k1 = k0 - 1 + 2
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i1, a1)') &
    Int(Xparam(f_OI)%Samson_v10(iv)%v), &
    Xparam(f_OI)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols('Observation Indicator', k0, k1, '(i1, a1)')

! 17 Present_weather      Present_weather conditions denoted by 9 indicators.
! See <A HREF="e:\5\3met\Docs\samson_format.txt#Present Weather Table">
iipar = f_OI
k0 = k1 + 2
k1 = k0 - 1 + LenF + 1

```

```

Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(a9, a1)') &
    Xparam(f_OI)%Samson_v10(iv)%f, &
    Xparam(f_OI)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols('Present_weather', k0, k1, '(a9, a1)')

! 18 Precipitable Water      0-100      Precipitable_water in
!                               millimeters.
iipar = f_pH2O
k0 = k1 + 2
k1 = k0 - 1 + 4
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i3, a1)') &
    Int(Xparam(f_pH2O)%Samson_v10(iv)%v), &
    Xparam(f_pH2O)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i3, a1)')

! 19 Broadband Aerosol      0.0-0.900  Broadband aerosol optical_depth
!      Optical_Depth        (broadband turbidity) on the
!                               day indicated.
iipar = f_BAOD
k0 = k1 + 2
k1 = k0 - 1 + 7
Select Case(Xparam(iipar)%Samson_v10(iv)%s)

```

```

Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(f6.3, a1)') &
    Xparam(f_BAOD)%Samson_v10(iv)%v, &
    Xparam(f_BAOD)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f6.3, a1)')

! 20 Snow_Depth          0-100      Snow_depth in centimeters on
!                          the day indicated.
iipar = f_SD
k0 = k1 + 2
k1 = k0 - 1 + 5
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(i4, a1)') &
    Int(Xparam(f_SD)%Samson_v10(iv)%v), &
    Xparam(f_SD)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i4, a1)')

! 21 Days Since Last Snowfall  0-88      Number of days since last snowfall.
!                          88 = 88 or greater days.
iipar = f_DSLS
k0 = k1 + 2
k1 = k0 - 1 + 4
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.

```

```

        n_missing = n_missing + 1
        id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(i3, a1)') &
        Int(Xparam(f_DSLS)%Samson_v10(iv)%v), &
        Xparam(f_DSLS)%Samson_v10(iv)%s(1:1)
Else
    tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(i3, a1)')

! 22 Hourly Precipitation      0.0 -      In cm
!                               25.4e+6   (See information below).
!
! Hourly Precipitation Flag      See explanation below.

! Xparam(f_HP)%Samson_v10(iv)%v = Hourly_Precipitation
! Xparam(f_HP)%Samson_v10(iv)%f = Hourly_Precipitation_Flag
iipar = f_HP
k0 = k1 + 2
k1 = k0 - 1 + 8
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    Write(tbuf(k0:k1), '(f6.2, a1, a1)') &
        Xparam(f_HP)%Samson_v10(iv)%v, &
        Xparam(f_HP)%Samson_v10(iv)%s(1:1), &
        Xparam(f_HP)%Samson_v10(iv)%f(1:1)
Else
    tbuf(k0:k1) = Adjustr(f2(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f6.2, a1, a1)')

If (hh == 25) Then
    ! 23 FAO Short Grass PET in mm/day
    ! Range: -2.101697E-02  11.9132

```

```

iipar = f_FAO_SG_PET
k0 = k1 + 2
k1 = k0 - 1 + 7
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(f6.2, a1)') &
    Xparam(f_FAO_SG_PET)%Samson_v10(iv)%v, &
    Xparam(f_FAO_SG_PET)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f6.2, a1)')

! 24 Class A pan Evaporation, mm/day
!   Range :
iipar = f_Ep
k0 = k1 + 2
k1 = k0 - 1 + 7
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  Write(tbuf(k0:k1), '(f6.2, a1)') &
    Xparam(f_Ep)%Samson_v10(iv)%v, &
    Xparam(f_Ep)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f6.2, a1)')

! 25 K-P FWS Evaporation in mm/day
!   Range :   -6.22314      5.46000
iipar = f_KP_FWS_Evaporation
k0 = k1 + 2

```

```

k1 = k0 - 1 + 7
Select Case(Xparam(iipar)%Samson_v10(iv)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  !Write(tbuf(k0:k1), '(f6.2, a1)') &
  !      Xparam(f_KP_FWS_Evaporation)%Samson_v10(iv)%v, &
  !      Xparam(f_KP_FWS_Evaporation)%Samson_v10(iv)%s(1:1)
Else
  tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))
End If
!!If (cols_now) Call hvf_Cols(FieldInfo(iipar)%Name, k0, k1, '(f6.2, a1)')
! 8 Feb 2002 5:41 pm: until we develop a better formulation,
!   kill KP_FWS_Evaporation.
tbuf(k0:k1) = ''

!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
!If (cols_now) Call hvf_Cols('', 0, 0, '', Last=.True.)
first_time = .False.
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
End If

tlen = Len_trim(tbuf(1:k1))

! Check that there are no '*'s in the line.
If (Index(tbuf(1:tlen), '*') > 0) Then
  ierr = ierr + 1
  Write (ULog, 9150) tbuf(1:tlen)
9150   Format (///, 1x, '?? Field overflow: ', /, 1x, a)
  Do jpar = 1, f_end
    Write (ULog, 9170) Trim(FieldInfo(jpar)%Name), Xparam(jpar)%Samson_v10(iv)
  End Do
9170   Format(1x, 3x, a, ': ', 1pg14.6, 1x, a, 1x, a)
End If

If (w0) Then
  If (hh /= NHours) Then
    Read (In_v0, '(a)') q0buf
  Else
    q0buf = ''
  End If

```

```

        If (one_file) Then
            Write (Out_r0, *)
            Write (Out_r0, '(a,a)') 'r0>', tbuf(1:tlen)
            Write (Out_r0, '(a,a)') 'v0>', Trim(q0buf)
        Else
            Write (Out_r0, '(a)') tbuf(1:tlen)
            Write (Out_v0, '(a)') Trim(q0buf)
        End If
    Else
        Write (Out_r0, '(a)') tbuf(1:tlen)
    End If

End Do

    jd_today = jd_today + 1.0
End Do

Call IOClose(Out_r0)
If (w0) Then
    Call IOClose(In_v0)
    If (.Not. one_file) Then
        Call IOClose(Out_v0)
    End If
End If

If (n_missing /= 0) Then
    Write (ULog, 9190) y4, n_missing
9190    Format (/ , 1x, '?? Dump_R0_One_Year: Year == ', i0, ', Missing values == ', i0)
    Do jpar = 1, f_end
        If (id_of_missing(jpar) > 0) Then
            Write (ULog, 9210) Trim(FieldInfo(jpar)%Name), id_of_missing(jpar)
9210            Format(1x, 6x, a, ': ', i0, ' hourly values missing.')
        End If
    End Do
End If

If (ierr /= 0) Then
    Errors_Detected = .True.
    !Call IOsDeleteFile(r0_file)
End If
End Subroutine Dump_R0_One_Year

End Module Dump_R0

```

## dumpmet

! Last change: LSR 17 May 2002 5:35 pm

Module Dump\_MET

! List generated by dvf\_Cols

```
!!!      3      8- 17: Precipitation [cm/day]
!!!      4     18- 27: Pan Evaporation [cm/day]
!!!      5     28- 37: Temperature mean [°C]
!!!      6     38- 47: Wind Speed @10 meter [cm/s]
!!!      7     48- 57: Solar Radiation [Langleys/day]
!!!      8     58- 63: Eto, FAO Short Grass PET [mm/day]
!!!      9     64- 73: Station Pressure [kiloPascal]
!!!     10     74- 77: Relative Humidity [%]
!!!     11     78- 80: Opaque_Sky_Cover [tenths]
!!!     12     81- 90: Daylight Temperature [°C]
!!!     13     91- 96: Broadband Aerosol [optical depth]
!!!     14     97-102: Daylight Mean Wind Speed @ 10 meters [m/s]
!!!     15    103-108: Maximum Daylight Mean Wind Speed @10m [m/s]
!!!     16    109-112: Direction of Maximum Daylight Wind [degrees]
!!!     17    113-118: Daylight Prevailing Wind_Speed @z=10m [m/s]
!!!     18    119-122: Daylight Prevailing Wind_Direction [Degrees]
```

```
Use Date_Module
Use Global_Variables
Use IoSubs
Use Utils2
Use Utils3
Use Winteracter
Implicit None
```

```
Logical, Save :: first_time = .True.
```

Contains

```
Subroutine Dump_MET_file()
```

```
! This routine dumps previously generated data in MET file format.
! Note and Warning: The met (*.dvf) contains only numbers.
```

```
Implicit None
```

```
Character(Len=350) :: tbuf
Character(Len=30)  :: f1 = '---'
Integer :: jday, hh01, hh24, hh25
Integer :: k0, k1, jpar, tlen, ierr, n_missing, ipar
Integer :: Out_MET
Integer :: yyyy, mm, dd, hh
```

```

Logical :: okay, is_daytime
Logical :: is_a_number
Integer :: ival
Real    :: rval, R_s

! d9*, n9* -- daylight quantity
Real    :: d9_pressure, d9_RH, d9_OSC, d9_Temp, d9_Broadband_Aerosol
Real    :: d9_mean_wind_speed, d9_max_wind_speed, d9_direction_of_max_wind_speed
Integer :: n9_pressure, n9_RH, n9_OSC, n9_Temp, n9_Broadband_Aerosol
Integer :: n9_mean_wind_speed

! d0*, n0* -- 1h-24h quantity
Real    :: d0_pressure, d0_RH, d0_OSC, d0_Temp, d0_Broadband_Aerosol
Real    :: d0_mean_wind_speed, d0_max_wind_speed, d0_direction_of_max_wind_speed
Integer :: n0_pressure, n0_RH, n0_OSC, n0_Temp, n0_Broadband_Aerosol
Integer :: n0_mean_wind_speed
Integer, Dimension(0:g_end) :: id_of_missing

Type(Val_and_Flag), Dimension(jd0:jd1) :: d9_pwd, d9_pws
Type(Val_and_Flag), Dimension(jd0:jd1) :: d0_pwd, d0_pws

Errors_Detected = .False.
id_of_missing = 0

Call IOWrite(Out_MET, name_met, Ok=okay)
If (.Not. okay) Then
  Write (ULog, *) '?? Could not open MET file ', Trim(name_met)
  Errors_Detected = .True.
  Return
End If

Call ToTTY('Dump_MET_file '//Trim(name_met))
!If (first_time) Call dvf_Cols('', 0, 0, Initialize=.True.)

ierr = 0
n_missing = 0

! First try to compute daylight values only,
Call Daylight_Prevailing_Wind(okay, d9_pwd, d9_pws, OnlyDaylight=.True.)
If (.Not. okay) Then
  ! Some days missing. Compute all-day values and merge results.
  Call Daylight_Prevailing_Wind(okay, d0_pwd, d0_pws, OnlyDaylight=.False.)
  Do jday = jd0, jd1
    ! If one of (pwd, pws) is missing, then both are missing.
    If (d9_pwd(jday)%s == T_Missing) Then
      ! The daylight value is missing: replace it with the all-hours value.
      d9_pwd(jday) = d0_pwd(jday)
      d9_pws(jday) = d0_pws(jday)
    End If
  End Do
End If

```

```

End If

! 15 Feb 2002  1:44 pm:
! Note: We are not using "Do jday = jd0, jd1"
! because we will modify "jday" when years are skipped.
jday = jd0 - 1
Do
  jday = jday + 1          ! step by day
  If (jday > jd1) Exit    ! end?

  hh01 = (jday-jd0)*Nhours + 1  ! First hour of the day
  hh24 = hh01 + 23             ! Last hour of the day (24th)
  hh25 = hh24 + 1             ! 25th hour of jday == (jday-jd0+1)*NHours
  Call Jd_to_ymd(jday, yyyy, mm, dd)

  If (.Not. Issue_This_Year(yyyy)) Then
    ! Year is missing. Do not print records associated with this year.
    ! Skip the rest of the year.
    jday = Jd(yyyy, mm=12, dd=31) ! Last day of the current year
    Cycle
  End If

  tbuf = ''

  k0 = 01
  k1 = 07
  Write(tbuf(k0:k1), '(1x,3i2.2)') mm, dd, Mod(yyyy,100)
  !If (first_time) Call dvf_Cols('mmdYY, Date', k0, k1)
  !If (first_time) Call dvf_MkFMT(k0, '(1x,3i2)')

  ! Precipitation [cm/day]; R0: [cm]
  iipar = g_Precipitation
  k0 = 08
  k1 = 17
  Select Case(Xparam(f_HP)%Samson_v10(hh25)%s)
  Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
  Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
  Case Default
    is_a_number = .True.
  End Select
  If (is_a_number) Then
    rval = Xparam(f_HP)%Samson_v10(hh25)%v
  Else
    rval = Zero
    !tbuf(k0:k1) = Adjustr(f1(1:k1-k0+1))

```

```

End If
Write(tbuf(k0:k1), '(f10.2)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1, Icount=3)
!If (first_time) Call dvf_MkFMT(k0, '(f10.2)')

! Class A pan Evaporation [cm/day]; R0: [mm/day]
! 1 mm = 0.10000 cm
iipar = g_Pan_Evaporation
k0 = 18
k1 = 27
Select Case(Xparam(f_Ep)%Samson_v10(hh25)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    rval = Xparam(f_Ep)%Samson_v10(hh25)%v * 0.10000    ! mm -> cm
Else
    rval = Zero
End If
Write(tbuf(k0:k1), '(f10.2)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f10.2)')

! Mean Temperature [°C]; R0: [°C]
iipar = g_Temperature_mean
k0 = 28
k1 = 37
Select Case(Xparam(f_DBT)%Samson_v10(hh25)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    rval = Xparam(f_DBT)%Samson_v10(hh25)%v
Else
    rval = Zero
End If
Write(tbuf(k0:k1), '(f10.1)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)

```

```

!If (first_time) Call dvf_MkFMT(k0, '(f10.1)')

! Wind speed z=10 meters [cm/sec]; R0: Wind_Speed @z=10m [m/s]
iipar = g_Wind_Speed
k0 = 38
k1 = 47
Select Case(Xparam(f_WS)%Samson_v10(hh25)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    rval = Xparam(f_WS)%Samson_v10(hh25)%v * meters_sec__to__cm_sec
Else
    rval = Zero
End If
Write(tbuf(k0:k1), '(f10.1)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f10.1)')

! Solar Radiation [Langleys/day]
! R0: Rs == Global Horizontal Radiation [Wh/m2/day]
! 1 watt hour m^-2 = 8.59845E-02 langley
iipar = g_Solar_Radiation
k0 = 48
k1 = 57
Select Case(Xparam(f_GHR)%Samson_v10(hh25)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    rval = Xparam(f_GHR)%Samson_v10(hh25)%v * 8.59845E-02 ! m/s -> cm/s
Else
    rval = Zero
End If
Write(tbuf(k0:k1), '(f10.1)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f10.1)')

```

```

! FAO Short Grass PET [mm/day]
! R0: same
iipar = g_FAO_Short_Grass
k0 = k1 + 1
k1 = k0 - 1 + 6
Select Case(Xparam(f_FAO_SG_PET)%Samson_v10(hh25)%s)
Case(T_Missing)
    is_a_number = .False.
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    is_a_number = .False.
Case Default
    is_a_number = .True.
End Select
If (is_a_number) Then
    rval = Xparam(f_FAO_SG_PET)%Samson_v10(hh25)%v
Else
    rval = Zero
End If
Write(tbuf(k0:k1), '(f6.1)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f6.1)')

! The following represent mean values for daylight hours only.
d9_pressure = Zero
n9_pressure = 0
d9_RH = Zero
n9_RH = 0
d9_OSC = Zero
n9_OSC = 0
d9_Temp = Zero
n9_Temp = 0
d9_Broadband_Aerosol = Zero
n9_Broadband_Aerosol = 0

d9_mean_wind_speed = Zero
d9_max_wind_speed = -Huge(Zero)
d9_direction_of_max_wind_speed = -Huge(Zero)
n9_mean_wind_speed = 0

! All hours values
d0_pressure = Zero
n0_pressure = 0
d0_RH = Zero
n0_RH = 0
d0_OSC = Zero
n0_OSC = 0
d0_Temp = Zero

```

```

n0_Temp = 0
d0_Broadband_Aerosol = Zero
n0_Broadband_Aerosol = 0

d0_mean_wind_speed = Zero
d0_max_wind_speed = -Huge(Zero)
d0_direction_of_max_wind_speed = -Huge(Zero)
n0_mean_wind_speed = 0

OneDay: Do hh = hh01, hh24

  If (Xparam(f_Rs)%Samson_v10(hh)%s == T_Missing) Then
    is_daytime = .False.
  Else If (Xparam(f_Rs)%Samson_v10(hh)%s == T_Undefined) Then
    is_daytime = .False.
  Else
    R_s = Xparam(f_Rs)%Samson_v10(hh)%v
    is_daytime = (R_s > Zero)
  End If

  Select Case(Xparam(f_WS)%Samson_v10(hh)%s)
  Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    ! Do nothing
  Case Default
    rval = Xparam(f_WS)%Samson_v10(hh)%v
    d0_mean_wind_speed = d0_mean_wind_speed + rval
    n0_mean_wind_speed = n0_mean_wind_speed + 1
    If (rval > d0_max_wind_speed) Then
      d0_max_wind_speed = rval
      d0_direction_of_max_wind_speed = Xparam(f_WD)%Samson_v10(hh)%v
    End If
    If (is_daytime) Then
      d9_mean_wind_speed = d9_mean_wind_speed + rval
      n9_mean_wind_speed = n9_mean_wind_speed + 1
      If (rval > d9_max_wind_speed) Then
        d9_max_wind_speed = rval
        d9_direction_of_max_wind_speed = Xparam(f_WD)%Samson_v10(hh)%v
      End If
    End If
  End Select

  Select Case(Xparam(f_SP)%Samson_v10(hh)%s)
  Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    ! Do nothing
  Case Default
    rval = Xparam(f_SP)%Samson_v10(hh)%v
    d0_pressure = d0_pressure + rval
    n0_pressure = n0_pressure + 1
    If (is_daytime) Then
      d9_pressure = d9_pressure + rval

```

```

        n9_pressure = n9_pressure + 1
    End If
End Select

Select Case(Xparam(f_RH)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    ! Do nothing
Case Default
    rval = Xparam(f_RH)%Samson_v10(hh)%v
    d0_RH = d0_RH + rval
    n0_RH = n0_RH + 1
    If (is_daytime) Then
        d9_RH = d9_RH + rval
        n9_RH = n9_RH + 1
    End If
End Select

Select Case(Xparam(f_OSC)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    ! Do nothing
Case Default
    rval = Xparam(f_OSC)%Samson_v10(hh)%v
    d0_OSC = d0_OSC + rval
    n0_OSC = n0_OSC + 1
    If (is_daytime) Then
        d9_OSC = d9_OSC + rval
        n9_OSC = n9_OSC + 1
    End If
End Select

Select Case(Xparam(f_DBT)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    ! Do nothing
Case Default
    rval = Xparam(f_DBT)%Samson_v10(hh)%v
    d0_Temp = d0_Temp + rval
    n0_Temp = n0_Temp + 1
    If (is_daytime) Then
        d9_Temp = d9_Temp + rval
        n9_Temp = n9_Temp + 1
    End If
End Select

Select Case(Xparam(f_BAOD)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    ! Do nothing
Case Default
    rval = Xparam(f_BAOD)%Samson_v10(hh)%v
    d0_Broadband_Aerosol = d0_Broadband_Aerosol + rval
    n0_Broadband_Aerosol = n0_Broadband_Aerosol + 1

```

```

        If (is_daytime) Then
            d9_Broadband_Aerosol = d9_Broadband_Aerosol + rval
            n9_Broadband_Aerosol = n9_Broadband_Aerosol + 1
        End If
    End Select
End Do OneDay

! Station Pressure [kilopascals]
iipar = g_Daylight_Station_Pressure
k0 = k1 + 1
k1 = k0 - 1 + 10
If (n9_pressure > 0) Then
    rval = d9_pressure / n9_pressure
Else If (n0_pressure > 0) Then
    rval = d0_pressure / n0_pressure
Else
    rval = Zero
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
End If
Write(tbuf(k0:k1), '(f10.1)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f10.1)')

! Relative Humidity [%]; Range: 0-100
iipar = g_Daylight_Relative_Humidity
k0 = k1 + 1
k1 = k0 - 1 + 4
If (n9_RH > 0) Then
    ival = Nint(d9_RH / n9_RH)
Else If (n0_RH > 0) Then
    ival = Nint(d0_RH / n0_RH)
Else
    ival = 0
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
End If
Write(tbuf(k0:k1), '(i4)') ival
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(i4)')

! Opaque Sky_Cover; Range: 0-10
! Amount of sky dome (in tenths) covered by clouds that prevent
! observing the sky or higher cloud layers.
iipar = g_Daylight_Opaque_Sky_Cover
k0 = k1 + 1
k1 = k0 - 1 + 3
If (n9_OSC > 0) Then
    ival = Nint(d9_OSC / n9_OSC)

```

```

Else If (n0_OSC > 0) Then
    ival = Nint(d0_OSC / n0_OSC)
Else
    ival = 0
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
End If
Write(tbuf(k0:k1), '(i3)') ival
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(i3)')

! Daylight temperature
iipar = g_Daylight_Temperature
k0 = k1 + 1
k1 = k0 - 1 + 10
If (n9_Temp > 0) Then
    rval = d9_Temp / n9_Temp
Else If (n0_Temp > 0) Then
    rval = d0_Temp / n0_Temp
Else
    rval = Zero
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
End If
Write(tbuf(k0:k1), '(f10.1)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f10.1)')

! Broadband aerosol optical_depth (broadband turbidity) on the day indicated.
! Range: 0.0-0.900
iipar = g_Daylight_Broadband_Aerosol
k0 = k1 + 1
k1 = k0 - 1 + 6
If (n9_Broadband_Aerosol > 0) Then
    rval = d9_Broadband_Aerosol / n9_Broadband_Aerosol
Else If (n0_Broadband_Aerosol > 0) Then
    rval = d0_Broadband_Aerosol / n0_Broadband_Aerosol
Else
    rval = Zero
    n_missing = n_missing + 1
    id_of_missing(iipar) = id_of_missing(iipar) + 1
End If
Write(tbuf(k0:k1), '(f6.3)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f6.3)')

!!!      ! Daylight Mean Wind Speed @ 10 meters [meters/second]
!!!      ! R0: Wind_Speed @z=10m [m/s]
!!!      iipar = g_Daylight_Mean_Wind_Speed
!!!      k0 = k1 + 1

```

```

!!!      k1 = k0 - 1 + 6
!!!      If (n9_mean_wind_speed > 0) Then
!!!          rval = d9_mean_wind_speed / n9_mean_wind_speed
!!!      Else If (n0_mean_wind_speed > 0) Then
!!!          rval = d0_mean_wind_speed / n0_mean_wind_speed
!!!      Else
!!!          rval = Zero
!!!          n_missing = n_missing + 1
!!!          id_of_missing(iipar) = id_of_missing(iipar) + 1
!!!      End If
!!!      Write(tbuf(k0:k1), '(f6.1)') rval
!!!      !If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!!!      !If (first_time) Call dvf_MkFMT(k0, '(f6.1)')
!!!
!!!      ! Maximum Daylight Mean Wind Speed @ 10 meters [m/s]
!!!      ! R0: Wind_Speed @z=10m [m/s]
!!!      iipar = d_Daylight_max_wind_speed
!!!      k0 = k1 + 1
!!!      k1 = k0 - 1 + 6
!!!      If (n9_mean_wind_speed > 0) Then
!!!          rval = d9_max_wind_speed
!!!      Else If (n0_mean_wind_speed > 0) Then
!!!          rval = d0_max_wind_speed
!!!      Else
!!!          rval = Zero
!!!          n_missing = n_missing + 1
!!!          id_of_missing(iipar) = id_of_missing(iipar) + 1
!!!      End If
!!!      Write(tbuf(k0:k1), '(f6.1)') rval
!!!      !If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!!!      !If (first_time) Call dvf_MkFMT(k0, '(f6.1)')
!!!
!!!      ! Direction of Maximum Daylight Wind [degrees]
!!!      ! R0: Wind direction in degrees. (N = 0 or 360, E = 90, S = 180, W = 270)
!!!      iipar = d_Daylight_direction_of_max_wind_speed
!!!      k0 = k1 + 1
!!!      k1 = k0 - 1 + 4
!!!      If (n9_mean_wind_speed > 0) Then
!!!          ival = Nint(d9_direction_of_max_wind_speed)
!!!      Else If (n0_mean_wind_speed > 0) Then
!!!          ival = Nint(d0_direction_of_max_wind_speed)
!!!      Else
!!!          ival = 0
!!!          n_missing = n_missing + 1
!!!          id_of_missing(iipar) = id_of_missing(iipar) + 1
!!!      End If
!!!      Write(tbuf(k0:k1), '(i4)') ival
!!!      !If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!!!      !If (first_time) Call dvf_MkFMT(k0, '(i4)')

```

```

! Daylight Prevailing Wind Speed @ 10 meters [m/s]
iipar = d_PWS
k0 = k1 + 1
k1 = k0 - 1 + 6
Select Case(d9_pws(jday)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  rval = d9_pws(jday)%v
Else
  rval = Zero
End If
Write(tbuf(k0:k1), '(f6.1)') rval
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(f6.1)')

! Daylight Prevailing Wind Direction [degrees]
! R0: Wind direction in degrees. (N = 0 or 360, E = 90, S = 180, W = 270)
iipar = d_PWD
k0 = k1 + 1
k1 = k0 - 1 + 4
Select Case(d9_pwd(jday)%s)
Case(T_Missing)
  is_a_number = .False.
  n_missing = n_missing + 1
  id_of_missing(iipar) = id_of_missing(iipar) + 1
Case(T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  is_a_number = .False.
Case Default
  is_a_number = .True.
End Select
If (is_a_number) Then
  ival = Nint(d9_pwd(jday)%v)
Else
  ival = 0
End If
Write(tbuf(k0:k1), '(i4)') ival
!If (first_time) Call dvf_Cols(MET_field(iipar)%Name, k0, k1)
!If (first_time) Call dvf_MkFMT(k0, '(i4)')

!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

```



## Et0

! Last change: LSR 6 Jun 2002 3:18 pm

Module ET0

```
Use Date_Module
Use Floating_Point_Comparisons
Use Global_Variables
Use LSQ2          ! Unconstrained linear least-squares
Use LSQ2_derived_type
Use Stats
Use Strings
Use Utils1
Implicit None

Private
Public :: ET0_et_al, Compute_Ep, Compute_E_fws, Compute_ET0
Public :: xRs_Over_Rso, Test_Et0
Type(LSQ_type), Dimension(13), Public, Save :: Buttner_Reg

Real, Public :: L_z, L_m
Real, Public :: Station_Latitude, SL_Sin, SL_Cos

Type(Stat_Block), Save :: ET0_d
Type(Stat_Block), Save :: Efws_d, Ep_d
Type(Stat_Block), Save :: Adjusted_Rs_ov_Rso
Private :: Vrmax
```

Contains

Subroutine Test\_Et0(Okay)

```
! FAO test cases for ET0 [FAO:75]
! file:///D:/555Luis/3MET/Docs/fao/www.fao.org/docrep/X0490E/x0490e08.htm#TopOfPage
!
! 22 Mar 2002 9:47 am: reproduces Et0 results of [FAO:75]

Implicit None
Logical, Intent(Out) :: Okay

Real :: FAO_gamma, t_gamma, u2_to_u10
Real :: Rs_over_Rso, Tdew, Ta, RH, u10
Real :: Rs, Ra, FAO_P, FAO_ET0, FAO_Tdew
Type(Val_and_Flag) :: ET0_vfs ! reference evapotranspiration [mm/day]

Character(Len=50) :: wtz, xHead
Integer          :: ierr, ndim
Integer          :: k, iv02, iv14, t_doy, FAO_Jdoy
```

```

Real          :: welev, rlat, rlon, rv
Type(Coords)  :: wlat, wlon
Type(Site_Info), Target :: xWBAN

Okay = .False.
ierr = 0
pWBAN => xWBAN    ! Global pointer.

! Given mean average hourly data between 02.00 and 03.00 hours and 14.00 and
! 15.00 hours on 1 October in N'Diaye (Senegal) at 16°13'N and 16°15'W and
! 8 m above sea level.

wlat  = Coords('N', 16, 13)
rlat  = (wlat%degrees + wlat%minutes/60.0) * Degrees_to_Radians
wlon  = Coords('W', 16, 15)
rlon  = (wlon%degrees + wlon%minutes/60.0) * Degrees_to_Radians
welev = 8 ! meters
wtz   = '+1(?)'

! New node initialization.
xWBAN%WBAN = '--001'
xWBAN%Text = "N'Diaye, Senegal"
xWBAN%Lat  = wlat
xWBAN%Lat_radians = rlat
xWBAN%Lon  = wlon
xWBAN%Lon_radians = rlon
xWBAN%Elev = welev
xWBAN%TZ   = wtz
k = Index(wtz, '(') - 1
Read(wtz(1:k), *) xWBAN%iTZ
!Call Store_Elevation(xWBAN, edate='1800-01-01', elevation_ft=30.0)

! Station Latitude [radians]
Station_Latitude = pWBAN%Lat_radians
SL_Sin = Sin(Station_Latitude)
SL_Cos = Cos(Station_Latitude)

! Longitude of the center of the local time zone [degrees west of Greenwich]
L_z = TimeZone_to_Central_Meridian(pWBAN%iTZ)

! Longitude of the measurement site [degrees west of Greenwich]
L_m = pWBAN%Lon_radians * Radians_to_Degrees

! Allocate a small "SAMSON" array environment.
! Array to cover dates 1 October (1961-10-01) 2h and 14h.
Call ymdh_to_iv(1961, 10, 01, 02, iv02, t_doy)
Call ymdh_to_iv(1961, 10, 01, 14, iv14, t_doy)

ndim = iv14
Call Allocate_SAMSON_arrays(Nelements=ndim)

```

```

FAO_Jdoy = 274      ! Day of year of 1 October 1961
If (FAO_Jdoy /= t_doy) Then
  ierr = ierr + 1
  Write (ULog, 9130) FAO_Jdoy, t_doy
9130   Format (1x, '?? Test_Et0 FAO_Jdoy /= t_doy: ', i0, 1x, i0)
End If

! Climatic data          02.00-03.00 h  14.00-15.00h  Units
! Thr: mean hourly temperature ..... = 28          38          °C
! RHhr: mean hourly relative humidity = 90          52          %
! u2: mean hourly wind speed ..... = 1.9          3.3          m/s
! Rs: total solar radiation ..... = -            2.450        MJ m-2 hour-1
! Ra: Extraterrestrial radiation .... = -            3.543        MJ m-2 hour-1
! Rs/Rso: ..... = 0.0          0.922        dimensionless
! ET0: ..... = 0.00          0.63          mm / hour
!
!
!           Dew Point          Et0          Gamma
! -----
! Test #1: FAO: 21.98350000000000  4.342611420981305E-03  6.729999799304995E-02
! Test #1: R0:  26.20274292447007  4.409919734092227E-03  6.757485681030362E-02
! Test #2: FAO: 21.98350000000000  0.6269095810990636   6.729999799304995E-02
! Test #2: R0:  26.41541277264032  0.6268624548236352   6.758877044376800E-02
! Test #2: R0 estimate of Rs_over_Rso == 0.8029946722967382
!
! z: Elevation above sea level = 8 meters
! Pressure [kPa] = 101.3 * ( (293 - 0.0065*z[meters])/293 ) ** 5.26 == 101.205 [kPa]
!
! [FAO:36] e0(T) = 0.6108 * Exp(17.27 * T / (T + 237.3))
! Rs(SAMSON) = Rs(FAO) / Watt_hour__to_MJoule
!
! Note that wind speed is at 2 meters; remember SAMSON stores u10.
! u2_to_u10 converts u2 -> u10
u2_to_u10 = 5.81 / 4.87

! *** Test #1
FAO_P = 101.205
Rs = Zero / Watt_hour__to_MJoule
Ra = Zero / Watt_hour__to_MJoule
u10 = 1.9 * u2_to_u10
Ta = 28
RH = 90
Tdew = DewPointF(Ta, RH)

! First, lets see how far off is our estimate of gamma, given
! the estimate of Tdew.
FAO_gamma = 0.0673
t_gamma = GammaF(FAO_P, Tdew)

If (Abs(FAO_gamma-t_gamma) > 3.0e-4) Then
  ierr = ierr + 1

```

```

Write (ULog, 9150) '#1: FAO_gamma, t_gamma: ', FAO_gamma, t_gamma, FAO_gamma-t_gamma
9150 Format (1x, '?? Test_Et0 #1: FAO_gamma /= t_gamma: ', lp2g14.6)
End If
! The gammas are not too far off (0.4%)

! Now, compute the Tdew that would generate FAO_gamma. We are doing
! this to compare our results with FAO without too much problem.
!
! Gamma = Cp * P / (eps * lambda)      <A HREF="Utils1.f90#GammaF">
! lambda(Tdew) = a - b*Tdew            <A HREF="Utils1.f90#LambdaF">
!
! Solved by Mathematica. <A HREF="E:\5\3MET\r0.f90\44.evap.nb">
! Note that Gamma is a function of Tdew and Pressure. Therefore,
! FAO_Tdew is the same for test cases #1 and #2 (since FAO_P is
! a constant).

FAO_Tdew = 21.9835
!Write (6, *) 'FAO_Tdew, Tdew: ', FAO_Tdew, Tdew, FAO_Tdew-Tdew
! FAO_Tdew, Tdew: 21.983500000000000 26.20274292447007 -4.219242924470066
! The Dew points are much different.
!
! GammaF(FAO_P, FAO_Tdew) - GammaF(FAO_P, Tdew) == 2e-9

Xparam(f_Rs)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, Rs, "")
Xparam(f_Ra)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, Ra, "")
Xparam(f_DBT)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, Ta, "")
Xparam(f_DPT)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, FAO_Tdew, "")
Xparam(f_SP)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, FAO_P, "")
Xparam(f_RH)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, RH, "")
Xparam(f_WS)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, u10, "")
Xparam(f_pH2O)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, Zero, "")

FAO_ET0 = 0.0
Rs_over_Rso = 0.8
Do k = 1, 2
  If (k == 1) Then
    rv = FAO_Tdew
    Xparam(f_DPT)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, FAO_Tdew, "")
    xHead = 'Test #1: FAO_Tdew = '
  Else
    rv = Tdew
    Xparam(f_DPT)%Samson_v10(iv02) = Val_and_Flag(T_SAMSONv10, Tdew, "")
    xHead = 'Test #1: Tdew = '
  End If

Call Compute_ET0(iv02, Rs_over_Rso, ET0_vfs)
If (Abs(FAO_ET0-ET0_vfs%v) > 5.0e-3) Then
  ierr = ierr + 1
  Write (6, 9170) FAO_ET0, ET0_vfs%v, FAO_ET0-ET0_vfs%v
  Write (ULog, 9170) FAO_ET0, ET0_vfs%v, FAO_ET0-ET0_vfs%v

```

```

9170      Format (1x, '?? Test_Et0 #1: FAO_ET0 /= ET0_vfs: ', lp3g14.6)
      End If
      !Write (6, *) Trim(xHead), rv, '; ET0_vfs: ', ET0_vfs, '; gamma = ', GammaF(FAO_P, rv)
End Do

! *** Test #2
Rs = 2.450 / Watt_hour__to_MJoule
Ra = 3.543 / Watt_hour__to_MJoule
u10 = 3.3 * u2_to_u10
Ta = 38
RH = 52
Tdew = DewPointF(Ta, RH)

! First, lets see how far off is our estimate of gamma, given
! the estimate of Tdew.
t_gamma = GammaF(FAO_P, Tdew)

If (Abs(FAO_gamma-t_gamma) > 3.0e-4) Then
  ierr = ierr + 1
  Write (ULog, 9190) '#2: FAO_gamma, t_gamma: ', FAO_gamma, t_gamma, FAO_gamma-t_gamma
9190  Format (1x, '?? Test_Et0 #2: FAO_gamma /= t_gamma: ', lp2g14.6)
End If
! The gammas are not too far off (0.4%)
!
! Now, compute the Tdew that would generate FAO_gamma. Bla bla.
! See comments for Test #1.

Xparam(f_Rs)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, Rs, "")
Xparam(f_Ra)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, Ra, "")
Xparam(f_DBT)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, Ta, "")
Xparam(f_DPT)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, FAO_Tdew, "")
Xparam(f_SP)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, FAO_P, "")
Xparam(f_RH)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, RH, "")
Xparam(f_WS)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, u10, "")
Xparam(f_pH2O)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, Zero, "")

!Call xRs_Over_Rso(iv14, FAO_Jdoy, rRatio)
!Write (6, *) 'Test #2: Estimated Rs_over_Rso == ', rRatio
! Test #2: R0 estimate of Rs_over_Rso == 0.8029946722967382

FAO_ET0 = 0.63
Rs_over_Rso = 0.922
Do k = 1, 2
  If (k == 1) Then
    rv = FAO_Tdew
    Xparam(f_DPT)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, FAO_Tdew, "")
    xHead = 'Test #2: FAO_Tdew = '
  Else
    rv = Tdew
    Xparam(f_DPT)%Samson_v10(iv14) = Val_and_Flag(T_SAMSONv10, Tdew, "")
  End If
End Do

```

```

        xHead = 'Test #2: Tdew = '
    End If
    Call Compute_ET0(iv14, Rs_over_Rso, ET0_vfs)
    If (Abs(FAO_ET0-ET0_vfs%v) > 4.0e-3) Then
        ierr = ierr + 1
        Write (6, 9210) FAO_ET0, ET0_vfs%v, FAO_ET0-ET0_vfs%v
        Write (ULog, 9210) FAO_ET0, ET0_vfs%v, FAO_ET0-ET0_vfs%v
9210     Format (1x, '?? Test_Et0 #1: FAO_ET0 /= ET0_vfs: ', lp3g14.6)
    End If
    !Write (6, *) Trim(xHead), rv, '; ET0_vfs: ', ET0_vfs, '; gamma = ', GammaF(FAO_P, rv)
End Do

    Call Deallocate_SAMSON_arrays()
    Okay = (ierr == 0)

End Subroutine Test_Et0

Subroutine ET0_et_al(Xok)

! Compute ET0, E_fws
!* <A NAME="ET0">
!* <A NAME="E_fws">

! References:
! [1] Crop evapotranspiration - Guidelines for computing crop water requirements
!     - FAO Irrigation and drainage paper 56 by Richard G. Allen, Luis S.
!     Pereira, Dirk Raes, and Martin Smith. Water Resources, Development and
!     Management Service, FAO - Food and Agriculture Organization of the United
!     Nations, Rome, 1998. ISBN 92-5-104219-5.
!
!     The book can be found online at
!     http://www.fao.org/docrep/X0490E/x0490e00.htm#Contents
!
!     See also
!     file:///D:/555Luis/3MET/Docs/fao/www.fao.org/docrep/X0490E/x0490e00.htm#Contents
!
! Daily time step: see Ref[1:72]
! Hourly time step: see Ref[1:74]
! Pan evaporation: see Ref[1:78]

Implicit None
Logical,          Intent(Out) :: Xok

Integer :: jday, hh01, hh24, hh25
Integer :: jyyyy, jmm, jdd, jdoy, kdoy
Integer :: ierr
Integer :: iSunset ! The last hour of the day for which Rs > 0;
Integer :: iSunrise ! The first hour of the day for which Rs > 0;
Integer :: dSunset, dSunrise ! delta hours after sunrise or before sunset

```

```

Real :: Dlat

Type(Val_and_Flag) :: ET0_vfs
Logical :: okay
Real :: dR_s, dR_a, Rs_over_Rso, ET0_day

Real :: FWS_Rs, FWS_Rdiff, FWS-Ta, FWS_Dew, FWS_u4d, FWS_RH, FWS_P
Real :: Efws_day

! Class A pan evaporation [mm/day]
Real :: pan_Ep_day
Real :: pan_Rs, pan_up6d, pan-Ta, pan_RH, pan_P

! Buttner regression:
!   Rs = Rso (1 - b [OSC])
!
! where
!   Rso and b are regression parameters.
!   Rs -- daily total radiation
!   [OSC] -- daily mean
!
! Fit:
!   Rs = a + m*[OSC]
!
! Then
!   Rso = a
!   b = -m/a
!
! Perform 13 regressions:
!   1:12 -- Jan, Feb, ..., Dec
!   13 -- all months
!
! Variables for LSQ
! nCoeff = 2, nvar = 1
! wt    Weight of each data point
! xx    Used to transfer data
Integer, Parameter :: MaxCoeffs = 2
Real(dp), Dimension(MaxCoeffs) :: xx
Real(dp) :: wt, estim_Rs0, estim_b
Character(Len=15), Dimension(MaxCoeffs) :: vname
Integer :: nCoeff, nvar
Logical :: fit_const
Character(Len=30) :: tmonth

ierr = 0

! Station Latitude [radians]
Station_Latitude = pWBAN%Lat_radians
SL_Sin = Sin(Station_Latitude)
SL_Cos = Cos(Station_Latitude)

```

```

! Longitude of the center of the local time zone [degrees west of Greenwich]
L_z = TimeZone_to_Central_Meridian(pWBAN%iTZ)

! Longitude of the measurement site [degrees west of Greenwich]
L_m = pWBAN%Lon_radians * Radians_to_Degrees

! Buttner regression
nCcoeff = 2
nvar = nCcoeff - 1 ! Do not count the constant
fit_const = .True. ! Change to .false. if fitting a model without a constant.
vname(1) = 'Constant'
vname(2) = '[OSC]'
wt = One ! Weight of each data point

!!!Buttner:Do ii = 1, 13
!!!Buttner: ! Initializes the QR-factorization
!!!Buttner: Call LSQ_startup(nvar, fit_const, Buttner_Reg(ii))
!!!Buttner: If (ii < 13) Then
!!!Buttner: tmonth = Month_Table(ii)
!!!Buttner: Else
!!!Buttner: tmonth = 'All months'
!!!Buttner: End If
!!!Buttner: Call LSQ_set_names(&
!!!Buttner: 'Buttner: Rs[Watt hour m^-2] = a + m * OSC[tenths], '//Trim(tmonth), &
!!!Buttner: vname, 'Rs', Buttner_Reg(ii))
!!!Buttner:End Do

!Call Sunset_Angle_Test()

Call Vrmax(Initialize=.True.)

! Determine sunrise and sunset hours
! See <A HREF="0notes.txt#Note_21">
Call Find_Sunrise_Sunset(1, 24, 1, dSunrise, iSunrise, dSunset, iSunset)

Write (Umath, *) '(* {ET0_day, Efws_day, pan_Ep_day} *)'
Dlat = Station_Latitude * Radians_to_Degrees ! Station Latitude [degrees]

By_Days: Do jday = jd0, jdl ! step by day
hh01 = (jday-jd0)*Nhours + 1 ! First hour of the day
hh24 = hh01 + 23 ! Last hour of the day (24th)
hh25 = hh24 + 1 ! 25th hour of jday == (jday-jd0+1)*NHours

! Determine the day of the year
Call Jd_to_ymd(jday, jyyyy, jmm, jdd)
kday = jdd - 32 + Int(275*jmm/9.0) + 2*Int(3/Real(jmm+1)) + &
Int(jmm/100.0 - Modulo(jyyyy,4)/4.0 + 0.975)
!jday = jday - Jd(jyyyy,01,01) + 1
!If (jday /= kday) Then
! Write(6,*) 'jday, jyyyy, jmm, jdd == ', jday, jyyyy, jmm, jdd

```

```

! Write(6,*) 'jdoy == ', jdoy
! Write(6,*) 'kdoy == ', kdoy
! Stop '?? Stop: jdoy /= kdoy, ET0_et_al'
!End If
jdoy = kdoy

! Initialize to "missing data"
ET0_vfs = Val_and_Flag(T_Missing, Missing_Data, '')
Xparam(f_FAO_SG_PET)%Samson_v10(hh25) = ET0_vfs
Xparam(f_KP_FWS_Evaporation)%Samson_v10(hh25) = ET0_vfs
Xparam(f_Ep)%Samson_v10(hh25) = ET0_vfs

ET0_day = Zero
Efws_day = Zero

! dR_s -- daily Global Horizontal Radiation, [Watt hour m^-2]
dR_s = Xparam(f_Rs)%Samson_v10(hh25)%v

! dR_a -- daily Extraterrestrial Horizontal Radiation, [Watt hour m^-2]
dR_a = Xparam(f_Ra)%Samson_v10(hh25)%v

! Any radiation today?
If ((dR_s > Eps0) .And. (dR_a > Eps0)) Then
    Call Et0_sub0() ! ET0_day == Sum ET0_hr

!!!Buttner:! Buttner regression:
!!!Buttner:!     Rs = a + m [OSC]
!!!Buttner:!
!!!Buttner:! Perform 13 regressions:
!!!Buttner:!     1:12 -- Jan, Feb, ..., Dec
!!!Buttner:!     13 -- all months

!!!Buttner:Select Case(Xparam(f_OSC)%Samson_v10(hh25)%s)
!!!Buttner:Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
!!!Buttner: ! Opaque sky cover missing. No data point available.
!!!Buttner:Case Default
!!!Buttner: ! A one is inserted as the first variable if a constant is being fitted.
!!!Buttner: xx = (/ one, Xparam(f_OSC)%Samson_v10(hh25)%v /)

!!!Buttner: Call LSQ_includ(weight=wt, xrow=xx, &
!!!Buttner:     yelem=dR_s, T=Buttner_Reg(13)) ! All months.
!!!Buttner: Call LSQ_includ(weight=wt, xrow=xx, &
!!!Buttner:     yelem=dR_s, T=Buttner_Reg(jmm)) ! Some month.
!!!Buttner:End Select

Else
    ! If there is no radiation during the day, then darkness.
    ! Darkness over the land. Is the station North of the Arctic Circle?
    If (Dlat >= Latitude_Arctic_circle) Then
        Xparam(f_FAO_SG_PET)%Samson_v10(hh25)%s = T_Perpetual_Darkness

```

```

        Xparam(f_KP_FWS_Evaporation)%Samson_v10(hh25)%s = T_Perpetual_Darkness
        Xparam(f_Ep)%Samson_v10(hh25)%s = T_Perpetual_Darkness
    Else
        ! Some other problem.
        Write (ULog, 9130) Trim(pWBAN%WBAN), Trim(pWBAN%Text), jyyyy, jmm, jdd
9130    Format (1x, '?? ET0_et_al: dRs=dRa=0 and not Arctic Circle for ', &
            a, ': ', a, '; ', i4, '-', i2.2, '-', i2.2)
        Write (6, 9130) Trim(pWBAN%WBAN), Trim(pWBAN%Text), jyyyy, jmm, jdd
        Write (6, *) Dlat, Latitude_Arctic_circle
        Stop '@@@'
    End If
    Cycle By_Days
End If

! When possible, use the daily values computed by
! <A HREF="Utils2.f90#Daily values">
FWS_Rs = Xparam(f_Rs)%Samson_v10(hh25)%v      ! Watt hour meter^-2 day^-1
FWS_Rdiff = Xparam(f_Rdiff)%Samson_v10(hh25)%v ! same units as Rs
FWS-Ta = Xparam(f_DBT)%Samson_v10(hh25)%v     ! °C
FWS_Dew = Xparam(f_DPT)%Samson_v10(hh25)%v    ! °C
FWS_P = Xparam(f_SP)%Samson_v10(hh25)%v       ! kPa

! The function returns wind speed (in m/s) at z=4 meters
! <A HREF="Utils1.f90#Wind Conversion Table">
FWS_u4d = Wind_Speed_F(Xparam(f_WS)%Samson_v10(hh25)%v, T_u4) * ms__to__kmd ! km/day

FWS_RH = Xparam(f_RH)%Samson_v10(hh25)%v      ! %; unused by the current formulation
Call Compute_E_fws(FWS_Rs, FWS_Rdiff, FWS-Ta, FWS_Dew, FWS_u4d, FWS_RH, FWS_P, Efws_day)

!Call Stat_Add_Point(Efws_d, Efws_day)
Xparam(f_KP_FWS_Evaporation)%Samson_v10(hh25)%v = Efws_day
Xparam(f_KP_FWS_Evaporation)%Samson_v10(hh25)%s = T_Estimated

! Computations for daily Ep --
! Use daily values for all computations.

pan-Ta = Xparam(f_DBT)%Samson_v10(hh25)%v ! mean daily temperature

! The function returns wind speed (in m/s) at z=0.6 meters
! <A HREF="Utils1.f90#Wind Conversion Table">
pan_up6d = Wind_Speed_F(Xparam(f_WS)%Samson_v10(hh25)%v, T_up6) * ms__to__kmd ! km/day

pan_Rs = Xparam(f_Rs)%Samson_v10(hh25)%v
pan_RH = Xparam(f_RH)%Samson_v10(hh25)%v
pan_P = Xparam(f_SP)%Samson_v10(hh25)%v

Call Compute_Ep(pan-Ta, pan_up6d, pan_Rs, pan_RH, pan_P, pan_Ep_day, okay)
If (okay) Then
    !Call Stat_Add_Point(Ep_d, pan_Ep_day)

```

```

Write (Umath, 9150) ET0_day, Efws_day, pan_Ep_day
9150   Format (3x, '{', f10.3, ', ', f10.3, ', ', f10.3, '},')

Xparam(f_Ep)%Samson_v10(hh25)%v = pan_Ep_day
Xparam(f_Ep)%Samson_v10(hh25)%s = T_Estimated
Else
  ierr = ierr + 1
  Xparam(f_Ep)%Samson_v10(hh25)%v = pan_Ep_day
  Xparam(f_Ep)%Samson_v10(hh25)%s = T_Missing
  Write (ULog, 9170) '?? daily value Rs Ta : ', jyyyy, jmm, jdd, NHours, pan_Rs, pan-Ta
9170   Format(1x, a, i4, '-', i2.2, '-', i2.2, i3, 'h', 3x, lp2g14.6)
End If

```

```

End Do By_Days
Xok = (ierr == 0)

```

```

!!!Buttner:      ! Perform Buttner regressions.
!!!Buttner:      Do ii = 1, 13
!!!Buttner:      Call LSQ_Simple_Regression(Buttner_Reg(ii), ULog)
!!!Buttner:      estim_Rs0 = Buttner_Reg(ii)%beta(1)
!!!Buttner:      estim_b = - Buttner_Reg(ii)%beta(2) / estim_Rs0
!!!Buttner:      Write (ULog, *)
!!!Buttner:      Write (ULog, 9330) 'Rs = Rso (1 - b [OSC])'
!!!Buttner:      Write (ULog, 9330) '  Rso', estim_Rs0, '[Watt hour m^-2]'
!!!Buttner:      Write (ULog, 9330) '  b ', estim_b, '[dimensionless]'
!!!Buttner:9330  Format (1x, a, :, ' = ', lpg14.6, :, 1x, a)
!!!Buttner:      End Do

```

```

Call Vrmax(Xprint=.True.)

```

Contains

```

Subroutine Et0_sub0()

! Compute the daily value of Eto.
! ET0_day == Sum {Eto(hourly)}

Implicit None
Integer :: nET0, N_missing, hh, ii
Logical :: is_daytime, was_nighttime, is_dawn
Real    :: ET0_hr, R_s
Logical :: nan_detected

nET0 = 0
was_nighttime = .True.
nan_detected = .False.

```

```

OneDay: Do hh = hh01, hh24

```

```

! If missing parameters, skip this hour
N_missing = 0
Select Case(Xparam(f_Rs)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_Rdiff)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_Ra)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_DBT)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_DPT)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_SP)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_RH)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_WS)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
Select Case(Xparam(f_pH2O)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  N_missing = N_missing + 1
End Select
If (N_missing > 0) Then
  Cycle OneDay
End If

! Test perpetual darkness: 27502: Barrow, AK

! R_s -- hourly Global Horizontal Radiation, [Watt hour m^-2]
R_s = Xparam(f_Rs)%Samson_v10(hh)%v
is_daytime = (R_s > Zero)

! It is dawn if the previous hour was nighttime and the
! current hour is daytime. Note that the sunset hour will

```

```

! not be updated during the interval [1, SunRise], because
! it is still nighttime. This is the correct behaviour.
! The sunrise and sunset hours should be updated only at dawn.
! was_nighttime will remain .False. until the do loop is
! restarted, next Julian day.
is_dawn = was_nighttime .And. is_daytime
If (iSunset == Tbogus) Then
    Call Find_Sunrise_Sunset(hh01, hh24, jdoy, &
        dSunrise, iSunrise, dSunset, iSunset)
Else If (is_dawn) Then
    was_nighttime = .False.
    Call Find_Sunrise_Sunset(hh01, hh24, jdoy, &
        dSunrise, iSunrise, dSunset, iSunset)
End If

! Compute the ratio Rs_over_Rso == Max(Rs/Rso, 1)
!
! If the current hour (i.e., hh) is contained in the closed interval
! [iSunrise+Delta_hours, iSunset-Delta_hours] (i.e., daytime)
! then the ratio Rs/Rso is computed at ii = hh.
! If iSunrise <= hh < iSunrise+Delta_hours (i.e., the hours
! after sunrise), then the ratio is computed at
!     ii = iSunrise+Delta_hours
! If iSunset-Delta_hours <= hh (i.e., a few hours before sunset),
! then the ratio is computed at
!     ii = iSunset-Delta_hours

If (is_daytime) Then
    ii = hh
    Call xRs_Over_Rso(ii, jdoy, Rs_over_Rso)
    If (Rs_over_Rso > One) Then
        If (hh <= dSunrise) Then
            ii = dSunrise
            Call xRs_Over_Rso(ii, jdoy, Rs_over_Rso)
        Else If (dSunset <= hh) Then
            ii = dSunset
            Call xRs_Over_Rso(ii, jdoy, Rs_over_Rso)
        End If
    End If
Else
    ! Nighttime. Use the ratio computed before sunset.
    ii = dSunset
    Call xRs_Over_Rso(ii, jdoy, Rs_over_Rso)
End If
Rs_over_Rso = Min(Rs_over_Rso, One)

If (IsNaN(Rs_over_Rso)) Then
    ierr = ierr + 1
    nan_detected = .True.
    Write(ULog,*) '?? ET0_et_al: Rs_over_Rso=NaN for ', &

```

```

        jyyyy, jmm, jdd

Else
    !Call Stat_Add_Point(Adjusted_Rs_ov_Rso, Rs_over_Rso)

    Call Compute_ET0(hh, Rs_over_Rso, ET0_vfs)

    Select Case(ET0_vfs%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        ! Not a usable number. Skip.

    Case Default
        If (.Not. IsNaN(ET0_vfs%v)) Then
            nET0 = nET0 + 1
            ET0_hr = ET0_vfs%v
            ET0_day = ET0_day + ET0_hr
        Else
            ierr = ierr + 1
            nan_detected = .True.
            Write (ULog, 9130) '?? ET0_et_al: ET0_hr==NaN at ', &
                jyyyy, jmm, jdd, Modulo(hh,NHours), hh
9130         Format(1x, a, i4, '-', i2.2, '-', i2.2, i3, 'h; iv:', i0)
            End If
        End Select
    End If
End Do OneDay

If (nET0 > 0) Then
    !Call Stat_Add_Point(ET0_d, ET0_day)
    Xparam(f_FAO_SG_PET)%Samson_v10(hh25)%v = ET0_day
    Xparam(f_FAO_SG_PET)%Samson_v10(hh25)%s = T_Estimated
Else If (nET0 == 0) Then
    ierr = ierr + 1
    Write(ULog,*) '?? ET0_et_al: nET0==0 for ', jyyyy, jmm, jdd
End If

End Subroutine Et0_sub0

End Subroutine ET0_et_al

Subroutine Sunset_Angle_Test()

    Implicit None

    Integer :: dSunrise, iSunrise, dSunset, iSunset
    Integer :: i_sunset, d_sunset, iv_offset, h_angle, h0
    Integer :: jj, kk, k0, jh_1_24, npts
    Integer :: jday, hh01, hh24, hh25, jdd, jmm, jyyyy, jday

```

```

Real    :: w_s, aaa, B, S_c, a0, a1, w_deg
Real    :: Solar_Declination, xx_Rs_over_Rso
Logical :: ok
Real    :: SL_Tan
Character(Len=2) :: xcoda
Integer, Dimension(-24:24) :: Ksunset = 0, Ksunrise = 0

Type(Stat_Block) :: Hour_Range, Sunset_Angle
Type(Stat_Block) :: Rs_ov_Rso_daytime

SL_Tan = SL_Sin / SL_Cos
Call Stat_Initialize(Sunset_Angle, 'Sunset angle - 90°')
Call Stat_Initialize(Hour_Range, 'Hour Range, Rs/Rso <= 1: ')
Call Stat_Initialize(Rs_ov_Rso_daytime, 'Rs/Rso, from sunrise to sunset')

Do jday = jd0, jd1          ! step by day
  hh01 = (jday-jd0)*Nhours + 1 ! First hour of the day
  hh24 = hh01 + 23          ! Last hour of the day (24th)
  hh25 = hh24 + 1          ! 25th hour of jday == (jday-jd0+1)*NHours

  ! Determine the day of the year
  Call Jd_to_ymd(jday, jyyyy, jmm, jdd)
  jdoy = jday - Jd(jyyyy,01,01) + 1

  Call Find_Sunrise_Sunset(hh01, hh24, jdoy, dSunrise, iSunrise, dSunset, iSunset)

! [FAO:75]
!
! Since the ratio Rs/Rso is used to represent cloud cover, when
! calculating Rnl for hourly periods during the nighttime, the ratio
! Rs/Rso can be set equal to the Rs/Rso calculated for a time period
! occurring 2-3 hours before sunset, before the sun angle becomes small.
! This will generally serve as a good approximation of cloudiness occurring
! during the subsequent nighttime. The hourly period that is 2 to 3 hours
! before sunset can be identified during computation of Ra as the period
! where w, calculated from Equation 31, is within the range
! (w_s - 0.79) <= w <= (w_s - 0.52), where w_s is calculated using
! Equation 25. As a more approximate alternative, one can assume
! Rs/Rso = 0.4 to 0.6 during nighttime periods in humid and subhumid
! climates and Rs/Rso = 0.7 to 0.8 in arid and semiarid climates.
! A value of Rs/Rso = 0.3 presumes total cloud cover.

Solar_Declination = 0.409 * Sin(Two_Pi*Real(Jdoy)/365.0 - 1.39) ! Radians

! The sunset hour angle, w_s, is given by Equation 25:
w_s = Acos( -SL_Tan * Tan(Solar_Declination) )

! where w is the Solar time angle at the midpoint of the period, [radians]
! t -- Standard clock time at the midpoint of the hour [hours] (24-hour based)

```

```

!      t = Modulo(Ihh,NHours) + 0.5
!
!      w = (Pi/12) * (t + 0.06667*(L_z-L_m) + S_c - 12)
!      = (Pi/12) * (Modulo(Ihh,NHours) + 0.5 + 0.06667*(L_z-L_m) + S_c - 12)
! Let
!      a = 0.5 + 0.06667*(L_z-L_m) + S_c - 12
! Then
!      w = (Pi/12)*(h+a)
! And
!      (w_s-0.79) <= w <= (w_s-0.52)
!      (w_s-0.79) <= (Pi/12)*(h+a) <= (w_s-0.52)
!      12/Pi*(w_s-0.79) - a <= h <= 12/Pi*(w_s-0.52) - a

! S_c -- Seasonal correction for local time [1:48]
B = Two_Pi * Real(Jdoy-81) / 364.0
S_c = 0.1645*Sin(2*B) - 0.1255*Cos(B) - 0.025*Sin(B)

aaa = 0.5 + 0.06667*(L_z-L_m) + S_c - 12
a0 = 12/Pi*(w_s-0.79) - aaa
a1 = 12/Pi*(w_s-0.52) - aaa

i_sunset = Modulo(iSunset,NHours)
d_sunset = Modulo(dSunset,NHours)
ok = (a0 <= d_sunset) .And. (d_sunset <= a1)
If (ok) Then
    xcoda = ''
Else
    xcoda = '??'
End If

h0 = Ceiling(a0)
iv_offset = dSunset / NHours
h_angle = h0 + NHours*iv_offset

! Collect statistics on the ratio Rs/Rso for
! the period sunrise to sunset.
Do jj = iSunset, iSunrise, -1
    Call xRs_Over_Rso(jj, Jdoy, xx_Rs_over_Rso)
    Call Stat_Add_Point(Rs_ov_Rso_daytime, xx_Rs_over_Rso)
End Do

! Starting from sunset and ending on sunrise,
! record the first hour for which Rs/Rso <= 1.
Do jj = iSunset, iSunrise, -1
    Call xRs_Over_Rso(jj, Jdoy, xx_Rs_over_Rso)
    If (xx_Rs_over_Rso .LessThanOrEqual. One) Then
        kk = jj - iSunset
        Ksunset(kk) = Ksunset(kk) + 1

        k0 = jj - iSunrise

```

```

        KSunrise(k0) = KSunrise(k0) + 1

        jh_1_24 = Modulo(jj,NHours)
        Call Stat_Add_Point(Hour_Range, Real(jh_1_24))

        Exit
    End If
End Do

w_deg = w_s * Radians_to_Degrees - 90
Call Stat_Add_Point(Sunset_Angle, w_deg)

End Do

npts = jdl - jd0 + 1

Write (ULog, '(//)')
Write (ULog, 9130) npts
9130 Format (1x, 'Total number of points: ', i0)

!!!Call Stat_Output(ULog, Sunset_Angle)
!!!Call Stat_Output(ULog, Rs_ov_Rso_daytime)

!!!      Write(ULog, 9570) 'Ksunset(Hour(Rs/Rso<=1) - h_angle)'
!!!      Write(ULog, 9570) 'Ksunset(Hour(Rs/Rso<=1) - dsunset)'
!!!      Write(ULog, 9570) 'Ksunset(Hour(Rs/Rso<=1) - isunset)'
!!!9570 Format (/ , 1x, a)
!!!
!!!      Do jj = Ubound(Ksunset,1), Lbound(Ksunset,1), -1
!!!          If (Ksunset(jj) /= 0) Then
!!!              Write(ULog, 9580) jj, Ksunset(jj)
!!!9580          Format (1x, 3x, 'Ksunset(', i3, ') == ', i0)
!!!              End If
!!!          End Do
!!!
!!!      jj = Sum(Ksunset)
!!!      Write (ULog, 9590) jj, npts-jj
!!!9590 Format (1x, 'Sum(Ksunset): ', i0, 3x, '; Total pts - Sum(Ksunset) == ', i0)
!!!
!!!      Write (ULog, *)
!!!      Write(ULog, 9550) Trim(Hour_Range%Header), Hour_Range%xmin, Hour_Range%xmax
!!!9550 Format (1x, a, 1x, 2(1x,1pg14.6))
!!!
!!!
!!!      Write(ULog, 9770) 'Ksunrise(Hour(Rs/Rso<=1) - isunrise)'
!!!9770 Format (/ , 1x, a)
!!!
!!!      Do jj = Ubound(Ksunrise,1), Lbound(Ksunrise,1), -1
!!!          If (Ksunrise(jj) /= 0) Then
!!!              Write(ULog, 9780) jj, Ksunrise(jj)

```

```

!!!9780      Format (1x, 3x, 'Ksunrise(', i3, ') == ', i0)
!!!      End If
!!!      End Do
!!!
!!!      jj = Sum(Ksunrise)
!!!      Write (ULog, 9790) jj, npts-jj
!!!9790      Format (1x, 'Sum(Ksunrise): ', i0, 3x, '; Total pts - Sum(Ksunrise) == ', i0)

!!!Stop '## Scheduled stop at Sunset_Angle_Test'
      End Subroutine Sunset_Angle_Test

```

```

Subroutine xRs_Over_Rso(Ihh, Jdoy, Rs_over_Rso)

```

```

! Compute the ratio Rs/Rso using data at hour Ihh

! Text from [FAO:75]
!
! Since the ratio Rs/Rso is used to represent cloud cover, when
! calculating Rnl for hourly periods during the nighttime, the ratio Rs/Rso
! can be set equal to the Rs/Rso calculated for a time period occurring 2-3
! hours before sunset, before the sun angle becomes small. This will
! generally serve as a good approximation of cloudiness occurring during
! the subsequent nighttime. The hourly period that is 2 to 3 hours before
! sunset can be identified during computation of Ra as the period where w,
! calculated from Equation 31, is within the range
!      (w_s - 0.79) <= w <= (w_s - 0.52),
! where w_s is calculated using Equation 25. As a more approximate
! alternative, one can assume Rs/Rso = 0.4 to 0.6 during nighttime periods
! in humid and subhumid climates and Rs/Rso = 0.7 to 0.8 in arid and
! semiarid climates. A value of Rs/Rso = 0.3 presumes total cloud cover.

```

```

Implicit None

```

```

Integer,          Intent(In)  :: Ihh      ! iv-type Hour
Integer,          Intent(In)  :: Jdoy     ! Day of the year
Real,             Intent(Out) :: Rs_over_Rso

```

```

Real :: Sin_Phi, Solar_Declination, Solar_time_angle
Real :: S_c, B, W, t, P_kPa, R_a, R_s, R_so
Real :: K_b, K_d, K_t

```

```

! Solar_Declination -- lower case delta
Solar_Declination = 0.409 * Sin(Two_Pi*Real(Jdoy)/365.0 - 1.39) ! Radians

```

```

! S_c -- Seasonal correction for local time [1:48]
B = Two_Pi * Real(Jdoy-81) / 364.0
S_c = 0.1645*Sin(2*B) - 0.1255*Cos(B) - 0.025*Sin(B)

```

```

! Solar time angle (lower case omega) at the midpoint of the period, [radians]

```

```

! t -- Standard clock time at the midpoint of the hour [hours] (24-hour based)
t = Modulo(Ihh,NHours) + 0.5
Solar_time_angle = (Pi/12.0) * (t + 0.06667*(L_z-L_m) + S_c - 12)

Sin_Phi = SL_Sin * Sin(Solar_Declination) + &
          SL_Cos * Cos(Solar_Declination) * Cos(Solar_time_angle)

! Atmospheric Pressure, [kPa]
! Units conversion was performed on input.
P_kPa = Xparam(f_SP)%Samson_v10(Ihh)%v

! Precipitable_water in millimeters
W = Xparam(f_pH2O)%Samson_v10(Ihh)%v

! K_b -- clearness index for direct beam radiation [dimensionless]
! K_t -- turbidity coefficient. K_t == 1 for clean air, 0.5 for extremely turbid air.
K_t = 1.0
K_b = 0.98 * Exp(-0.00146*P_kPa/(K_t*Sin_Phi) - 0.075*(W/Sin_Phi)**0.4)

! K_d -- Clearness index for diffuse radiation [dimensionless]
If (K_b .GreaterThanOrEqualTo. 0.15) Then
  K_d = 0.35 - 0.36 * K_b
Else
  K_d = 0.18 + 0.82 * K_b
End If

! R_so -- Short-wave radiation on a clear-sky day
! R_a -- Extraterrestrial Horizontal Radiation,
!           SAMSON: [Watt hour m^-2],
!           After Conversion: [MJoule]
R_a = Watt_hour__to__MJoule * Xparam(f_Ra)%Samson_v10(Ihh)%v
R_so = (K_b + K_d) * R_a

! Net longwave radiation [MJoule]
R_s = Watt_hour__to__MJoule * Xparam(f_Rs)%Samson_v10(Ihh)%v
Rs_over_Rso = R_s / R_so

!If (IsNaN(Rs_over_Rso)) Then
!  Write (6,*) '??? IsNaN(Rs_over_Rso)), ihh = ', ihh
!  stop
!End If

End Subroutine xRs_Over_Rso

Subroutine Find_Sunrise_Sunset(Hour01, Hour24, Jdoy, &
  dSunrise, iSunrise, dSunset, iSunset)

```

```

! Find sunrise and sunset hours in the interval [Hour01, Hour24]
Implicit None
Integer, Intent(In)  :: Hour01, Hour24
Integer, Intent(In)  :: Jdoy
Integer, Intent(Out) :: dSunrise, iSunrise, dSunset, iSunset

Integer, Parameter :: Delta_hours = 3
Integer :: jj, dl
Logical  :: xx_is_daytime

iSunset = Tbogus      ! The last hour of the day for which Rs > 0;
iSunrise = Tbogus     ! The first hour of the day for which Rs > 0;
dSunrise = Tbogus
dSunset = Tbogus

Do jj = Hour01, Hour24
  Select Case(Xparam(f_Rs)%Samson_v10(jj)%s)
  Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    Cycle

  Case Default
    xx_is_daytime = (Xparam(f_Rs)%Samson_v10(jj)%v > Zero)
    If (xx_is_daytime) Then
      iSunset = jj
      If (iSunrise == Tbogus) Then
        iSunrise = jj
      End If
    End If
  End Select
End Do

! If iSunset is set, so is iSunrise.
! Daylight period: [iSunrise, iSunset]
!       dSunrise <= dSunset
!       iSunrise + dl <= iSunset - dl
! ==>       dl <= (iSunset - iSunrise) / 2
If (iSunset /= Tbogus) Then
  dl = Min(Delta_hours, (iSunset-iSunrise)/2)
  dSunrise = iSunrise + dl
  dSunset = iSunset - dl
End If

End Subroutine Find_Sunrise_Sunset

Subroutine Vrmix(Initialize, Xprint)

Implicit None
Logical, Optional, Intent(In) :: Initialize
Logical, Optional, Intent(In) :: Xprint

```

```

If (Present(Initialize)) Then
  !Call Stat_Initialize(Adjusted_Rs_ov_Rso, 'Rs/Rso, forced<=1, all 24 hours')
  !Call Stat_Initialize(ET0_d, FieldInfo(f_FAO_SG_PET)%Name)
  !Call Stat_Initialize(Efws_d, FieldInfo(f_KP_FWS_Evaporation)%Name)
  !Call Stat_Initialize(Ep_d, FieldInfo(f_Ep)%Name)
  Return
End If

If (Present(Xprint)) Then
  !Write (ULog, '(/)')
  !Call Stat_Output(ULog, Adjusted_Rs_ov_Rso)
  !Call Stat_Output(ULog, ET0_d)
  !Call Stat_Output(ULog, Efws_d)
  !Call Stat_Output(ULog, Ep_d)
  Return
End If

End Subroutine Vrmax

Subroutine Dump_One_Day(Ihour, M) ! Ihour == iv

! <A NAME="Dump_One_Day">
! Dump M days of data: the day where "Ihour" lies and the previous day.

Implicit None
Integer,          Intent(In) :: Ihour
Integer, Optional, Intent(In) :: M      ! Number of days to dump
Integer :: yyyy, mm, dd, hh, jday, d0days, d1days
Integer :: jpar, iv0, iv1, n, iv
Real       :: xx_Rs_over_Rso

If (Present(M)) Then
  d0days = Abs(M)
  d1days = 2
Else
  d0days = 0
  d1days = 1
End If

! see <A HREF="ET0.f90#Dump_One_Day">
n = Floor(Ihour/Real(Nhours))
n = Max(n-d0days, 0)
iv0 = n*Nhours + 1
iv1 = (n+d1days)*Nhours - 1

Do iv = iv0, iv1
  Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
  jday = dd - 32 + Int(275*mm/9.0) + 2*Int(3/Real(mm+1)) + &
    Int(mm/100.0 - Modulo(yyyy,4)/4.0 + 0.975)

```

```

        Call xRs_Over_Rso(iv, jdoy, xx_Rs_over_Rso)

        Write (ULog, 9130) Iv, yyyy, mm, dd, hh, xx_Rs_over_Rso
9130    Format(/, 1x, '### Iv = ', i0, 3x, i4, '-', i2.2, '-', i2.2, i3, 'h, Rs/Rso = ', 1pg14.6)

        Do jpar = 1, f_end
            Write (ULog, 9150) Trim(FieldInfo(jpar)%Name), Xparam(jpar)%Samson_v10(iv)
        End Do
9150    Format(1x, 3x, a, ': ', 1pg14.6, 1x, a, 1x, a)
        End Do
    End Subroutine Dump_One_Day

```

```

Subroutine Vadd(Vnums, Rval)

```

```

    Implicit None
    Character(Len=*) , Intent(InOut) :: Vnums
    Real,          Intent(In)      :: Rval

    Integer :: i

    i = Len_trim(Vnums) + 1
    Write (Vnums(i:), '("," , f15.7)') Rval

```

```

End Subroutine Vadd

```

```

Subroutine Compute_E_fws(Rs, Rdiff, Ta, Tdew, u4d, RH, P, E_fws)

```

```

! Compute E_fws, free-water-surface evaporation [mm/day]
Implicit None
Real, Intent(In)  :: Rs      ! Global horizontal radiation, [W h m^-2 day^-1]
Real, Intent(In)  :: Rdiff ! Diffuse horizontal radiation, [W h m^-2 day^-1]
Real, Intent(In)  :: Ta     ! Ambient Temperature [°C]
Real, Intent(In)  :: Tdew  ! Dew point Temperature [°C]
Real, Intent(In)  :: u4d   ! Wind Speed @z=4 meters [km/day]
Real, Intent(In)  :: RH    ! Relative Humidity [%]
Real, Intent(In)  :: P     ! Pressure [kPa]
Real, Intent(Out) :: E_fws ! free-water-surface evaporation [mm/day]

```

```

! Epsilon_Water: Broad-band emmissivity of the water surface, [dimensionless]
Real, Parameter :: Epsilon_Water = 0.97

```

```

! Sigma: Stefan-Boltzman constant
! See Tinoco, Sauer and Wang, Physical Chemistry, 3rd edition. Page 23
!
! Energy flux per m^2 = J s^-1 m^-2 = Sigma * T^4
! T == Temperature [Kelvin]
! Sigma == Stefan-Boltzman constant = 5.67e-8 J s^-1 m^-2 K^-4

```

```

! Lambda == Latent heat (enthalpy) of vaporization of water
!
!           Sigma * T^4   kg[Water]
! Mass Water / m^2 /s == ----- * -----
!           Lambda       m^2 s 10^6
!
!           Sigma * T^4   mm[Water]   86400 s   3.93701e-2 inch
!           == ----- * ----- * ----- * -----
!           Lambda       s 10^6       1 day    1 mm
!
! Sigma * 86400 * 3.93701e-2
! ----- = 7.87222e-11 inches K^-4 day^-1
!           Lambda * 10^6

Real, Parameter :: Sigma_Inch = 7.87e-11

Real :: Rn, tKelvin, e_s, delta, es_ea
Real :: fu4d, Ea, gammaP
Real :: term1, term2

Rn = 5.79e-5 * (0.97*Rdiff + 0.94*(Rs-Rdiff))
tKelvin = Ta + 273.15

! e_s    saturation water vapor pressure [kPa] at temperature Tc
! Delta  slope of saturation water vapor pressure [kPa/°C]
!        1 kilopascal = 0.29530 inchHg
!        1 kilopascal/°C = 0.29530 inchHg * 100°C/180°F
!                      = 0.16406 inchHg/°F
Call Es_and_Delta(Ta, e_s, delta)
delta = delta * 0.16406 ! inchHg/°F

term1 = -7482.6 / (1.8*Ta+430.36)
term2 = -7482.6 / (1.8*Tdew+430.36)
es_ea = 6.4133e+06 * (Exp(term1) - Exp(term2))

fu4d = 0.181 + 0.00147*u4d
Ea = es_ea * fu4d

gammaP = 0.000108 * P ! inchHg/°F

term1 = (Rn - Epsilon_Water*Sigma_Inch*tKelvin**4) * delta
term2 = gammaP + 4*Epsilon_Water*Sigma_Inch*tKelvin**3 / fu4d

E_fws = 25.4 * (term1 + Ea*term2) / (delta+term2)

End Subroutine Compute_E_fws

Subroutine Compute_Ep(Ta, up6d, Rs, RH, P, Ep, okay)

```

```

! Compute Ep, Class A pan Evaporation [mm/day]
Implicit None
Real, Intent(In) :: Ta ! Temperature [°C]
Real, Intent(In) :: up6d ! Wind Speed @z=0.6 meters [km/day]
Real, Intent(In) :: Rs ! Rs, [W h m^-2 day^-1]
Real, Intent(In) :: RH ! Relative Humidity [%]
Real, Intent(In) :: P ! Pressure [kPa]
Real, Intent(Out) :: Ep ! Class A pan Evaporation [mm/day]
Logical, Intent(Out) :: okay

Real :: e_s, e_a, delta, es_ea
Real :: Ea, tmp0, delRn, gammap

okay = .False.

! Mean e_s ...
Call Es_and_Delta(Ta, e_s, delta)

! e_a Atmospheric water vapor pressure; actual water vapor pressure, [kPa]
e_a = 0.01 * e_s * RH

! Numerical noise made es_ea == -1.387778780781446E-17.
es_ea = Max(e_s - e_a, Zero)

Ea = 25.4 * (0.295*es_ea)**0.88 * (0.37 + 0.00256*up6d)

! If Rs == 0 and Ta <= 35°F, then delRn == 0.
! 35 °F = 1.6667 °C
! Reference:
! [] Lamoreux, Wallace W. 1962. Modern Evaporation Formulae adapted
! to computer use. Monthly Weather Review. January 1962, pages 26-28.
If (Rs > Zero) Then
  tmp0 = (1.8*Ta - 180) * (0.1024 - 0.01066*Log(0.0862*Rs))
  delRn = 154.8 * Exp(tmp0) - 0.01548
Else
  ! Rs <= 0
  If (Ta .LessThanOrEqual. 1.6667) Then
    delRn = Zero
  Else
    Ep = Missing_Data
    Return
  End If
End If

gammap = 0.001568 * P
Ep = (delRn + gammap*Ea) / (delta+gammap)
okay = .True.

End Subroutine Compute_Ep

```

```

Subroutine Compute_ET0(hh, Rs_over_Rso, ET0_vfs)

! Compute (hourly) ET0, FAO Penman-Monteith reference evapotranspiration
!
! hh iv-type hour, e.g., hh==9126 represents hour == 1
!
! Sin_Station_Latitude      Sin(Station Latitude [radians])
! Cos_Station_Latitude      Cos(Station Latitude [radians])
!
! L_z      Longitude of the center of the local time zone
!           [degrees west of Greenwich]
! L_m      Longitude of the measurement site [degrees west of Greenwich]
!
! ET0_vfs (Output) Hourly reference evapotranspiration [mm/day]

Implicit None
Integer,          Intent(In)  :: hh
Real,             Intent(In)  :: Rs_over_Rso
Type(Val_and_Flag), Intent(Out) :: ET0_vfs ! reference evapotranspiration [mm/day]

Real, Parameter :: albedo_standard_short_grass = 0.23 ! [dimensionless]

! Stefan-Boltzman constant
! See Tinoco, Sauer and Wang, Physical Chemistry, 3rd edition. Page 23
!
! Energy flux per m^2 = J s^-1 m^-2 = Sigma * T^4
!   T == Temperature [Kelvin]
! Sigma == Stefan-Boltzman constant = 5.67e-8 J s^-1 m^-2 K^-4
! Lambda == Latent heat (enthalpy) of vaporization of water
!
!
!           Sigma * T^4      kg[Water]
! Mass Water / m^2 /s == ----- * -----
!           Lambda           m^2 s 10^6
!
!           Sigma * T^4      mm[Water]      86400 s
! == ----- * ----- * -----
!           Lambda           s 10^6           1 day
!
! Sigma * 86400
! ----- = 1.99954e-9 mm K^-4 day^-1 == Sigma_mm
! Lambda * 10^6

Real, Parameter :: Sigma = 2.043e-10 ! MJ m^-2 hour^-1

Real :: Rs, Ta, RH, Ra, Tdew, P
Real :: e_s, e_a, delta, es_minus_ea, u2
Real :: R_ns, R_nl, R_n_hr, G_hr, gamma
Real :: tmp0, tmp1, ET0_xxx
Logical :: is_daytime

```

```

!                                     %s,          %v, %f
ET0_vfs = Val_and_Flag(T_Missing, Missing_Data, '')

! Rs -- Global Horizontal Radiation, [Watt hour m^-2]
Select Case(Xparam(f_Rs)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    Return
End Select
Rs = Xparam(f_Rs)%Samson_v10(hh)%v
is_daytime = (Rs > Zero)

! Ra -- Extraterrestrial Horizontal Radiation, [Watt hour m^-2]
Select Case(Xparam(f_Ra)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    Return
End Select
Ra = Xparam(f_Ra)%Samson_v10(hh)%v

! mean hourly air temperature, [°C]
Select Case(Xparam(f_DBT)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    Return
End Select
Ta = Xparam(f_DBT)%Samson_v10(hh)%v

! Relative Humidity [Percent]
Select Case(Xparam(f_RH)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    Return
End Select
RH = Xparam(f_RH)%Samson_v10(hh)%v

! Atmospheric Pressure, [kPa]
Select Case(Xparam(f_SP)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    Return
End Select
P = Xparam(f_SP)%Samson_v10(hh)%v

! Precipitable_water, [mm]
Select Case(Xparam(f_pH2O)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    Return
End Select

! e_s ...
Call Es_and_Delta(Ta, e_s, delta)

! Atmospheric water vapor pressure; actual water vapor pressure, [kPa]
e_a = 0.01 * e_s * RH

```

```

! Numerical noise made day_es_minus_ea == -1.387778780781446E-17.
es_minus_ea = Max(e_s - e_a, Zero)

! Net shortwave radiation
! 1 watt hour = 3.60000E-03 megajoules
R_ns = Watt_hour_to_MJoule * (1-albedo_standard_short_grass) * Rs

! Net longwave radiation
R_nl = Sigma * (Ta + 273.16)**4 * (0.34 - 0.14*sqrt(e_a)) * &
      (1.35*Rs_over_Rso - 0.35)

! Hourly net radiation at the grass surface [MJ m^-2 hour^-1]
R_n_hr = R_ns - R_nl

! Soil heat flux
If (is_daytime) Then
  G_hr = 0.1 * R_n_hr
Else
  G_hr = 0.5 * R_n_hr
End If

! The function returns wind speed (in m/s) at z=2 meters
! <A HREF="Utils1.f90#Wind Conversion Table">
Select Case(Xparam(f_WS)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  Return
End Select
u2 = Wind_Speed_F(Xparam(f_WS)%Samson_v10(hh)%v, T_u2)

! Dew point temperature, [°C]
Select Case(Xparam(f_DPT)%Samson_v10(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Perpetual_Darkness)
  Return
Case(T_Undefined)
  Gamma = Zero
Case Default
  Tdew = Xparam(f_DPT)%Samson_v10(hh)%v

  ! Gamma -- Psychrometric constant [kPa C^-1]; [kPa/°C]
  Gamma = GammaF(P, Tdew)
End Select

! The FAO Penman-Monteith equation for hourly time steps [1:74]
tmp0 = 0.408 * Delta * (R_n_hr - G_hr) + &
      Gamma * 37.0/(Ta+273.0) * u2 * es_minus_ea
tmp1 = Delta + Gamma * (1.0 + 0.34*u2)

```

```
ET0_xxx = tmp0 / tmp1
ET0_vfs = Val_and_Flag(T_Estimated, ET0_xxx, '')

End Subroutine Compute_ET0

End Module ET0
```

# Evap

! Last change: LSR 16 Jul 2002 2:37 pm  
Module Evaporation\_module

```
Use Date_Module
Use Floating_Point_Comparisons
Use Global_Variables
Use IoSubs
Use LSQ2_derived_type
Use LSQ2          ! Unconstrained linear least-squares
Use Strings
Use Utils1
Use Stats
Implicit None
```

```
Type(Stat_Block), Save :: ObsEp_all, ObsEp_set
Type(Stat_Block), Save :: teoEp_set
Type(LSQ_type), Save :: Ep_all, Ep_some
```

```
Private :: Vrmax
```

Contains

```
Subroutine Process_Evaporation_Data()
```

```
! The general form of the file is:
```

```
!
```

```
! 123456789 <-- column number
```

```
! !Some comment
```

```
!
```

```
DAILY
```

```
!
```

```
! Station FARGO WSO AP          Parameter      Evap          % Coverage      42
! PO Code ND                   Latitude     N46:55:31      Begin M/Yr      04/1963
! Stn ID 2859                   Longitude    W096:48:40     End M/Yr        09/1980
! County CASS                   Elevation    900            # Record Years  18
```

```
! ----- Evap (mm) -----
```

!	1963	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Annual
!	1	---	---	---	---	---	---	9	---	---	8	---	---	
!	2	---	---	---	---	9	---	8	---	---	7	---	---	
!	3	---	---	---	---	12	---	6	---	---	7	---	---	
!	4	---	---	---	---	7	---	8	---	---	8	---	---	
!	5	---	---	---	---	7	---	8	---	---	6	---	---	
!	6	---	---	---	3	10	---	3	---	---	4	---	---	
!	7	---	---	---	6	15	---	7	---	---	3	---	---	
!	8	---	---	---	2	12	---	8	---	---	3	---	---	
!	9	---	---	---	2	10	---	9	---	---	4	---	---	
!	10	---	---	---	3	12	---	7	---	---	3	---	---	
!	11	---	---	---	4	5	---	4	---	---	4	---	---	

```

! 12    ---   ---   ---   ---   6    ---   7    ---   ---   6    ---   ---
! 13    ---   ---   ---   4    ---   ---   7    ---   ---   7    ---   ---
! 14    ---   ---   ---   5    4    ---   3    ---   ---   5    ---   ---
!   :
! 29    ---   ---   ---   ---   6    ---   9    ---   ---   5    ---   ---
! 30    ---   ---   ---   ---   7    ---   7    ---   ---   2    ---   ---
! 31    ---   ---   ---   ---   9    ---   7    ---   ---   4    ---   ---
!
! Total  ---   ---   ---   82   218  ---   201  ---   ---   121  ---   ---   ---
! Extrm  ---   ---   ---   9    15   ---   13   ---   ---   8    ---   ---   ---
! ^L (sometimes)

```

```

! Ignore blank lines, lines with "^L", lines with a leading "!"
!
! The number of lines and the information contained in the lines before
! the '-----' line varies with the type of data in the file. However the block
!   1. a '-----' line with an identifier ('WD16(km)' in this example)
!   2. year and month
!   3. 31 lines with data
! is constant through all files.
!
! After each block there are summary lines which vary with the file type.
! These lines will be ignored.
!
! Year : The range is MinYear - MaxYear
! '---': denotes missing data

```

Implicit None

```

Character(Len=100) :: xbuf
Character(Len=50)  :: xid
Integer :: ios, n1, n2, jj, f0, f1, uin, ierror
Integer :: npos, beg_col, end_col
Integer :: yyyy, mm, dd, hh, iv, jd_today
Integer :: points_read, points_excluded, points_wo_parameters
Logical  :: ok, missing_params, exclude_point, in_range
Real    :: units_to_mm, minV, maxV

```

```

! maximum_wind_speed: 10.8 m/s
Real :: maximum_wind_speed = 9.33E+05 ! m/day

```

```

Real :: maximum_rainfall = 13.0 ! mm/day
Real :: up6

```

```

! Variables for LSQ
! Evap(Obs) = a*Evap(r0_estimate) + b
! nCoeff = 2, nvar = 1 (i.e., Evap(r0_estimate))
! We hope a == 1 & b == 0
Integer, Parameter :: MaxCoeffs = 2
Integer  :: nCoeff, nvar

```

```

Logical  :: fit_const  ! .false. if fitting a model without a constant.

! wt  Weight of each data point
! xx  Used to transfer data
! beta Regression coefficients
Real(dp), Dimension(MaxCoeffs) :: xx
Real(dp) :: wt, yobs, yteo
Character(Len=15), Dimension(MaxCoeffs) :: vname
Real(dp), Dimension(jd0:jd1) :: yyobs, yyteo
Character(Len=Len(Xparam(1)%Samson_v10(1)%s)) :: steo
Integer, Dimension(:), Pointer :: Days_in_Month

yyobs = Missing_Data
yyteo = Missing_Data

Write (ULog, *)

Call Vrmax(Initialize=.True.)
vname(1) = 'Constant'
vname(2) = 'Ep_r0'
wt = One ! Weight of each data point

! Find the evaporation file, e.g., 'v:\evaporation\T_14914.evp'
Call IORead(uin, name_Daily_Evap, ierror, ok=OK)
If (.Not. OK) Then
  Write (ULog, *) '## Process_Evaporation_Data: did not find ', Trim(name_Daily_Evap)
  ! Not finding the file is NOT an error. Just return.
  !Errors_Detected = .True.
  Return
Else
  Write (ULog, *) '## File: ', Trim(name_Daily_Evap)
  ! Call ToTTY('Process_Evaporation_Data: found '//Trim(name_Daily_Evap))
End If

! Range of acceptable values.
minV = FieldInfo(f_Ep)%minimum_value
maxV = FieldInfo(f_Ep)%maximum_value

! f0 points to the character after the last DirDelim of name_Daily_Evap, therefore
! name_Daily_Evap(f0:) contains only the name of the input file. This makes
! messages more readable.
! I changed my mind (24 Jan 2002  2:30 pm). In case of problems,
! I do not want to hunt for a file.
f0 = 1  ! + Index(name_Daily_Evap, DirDelim, Back=.True.)
f1 = Len_trim(name_Daily_Evap)

nCcoeff = 2
nvar = nCcoeff - 1  ! Do not count the constant
fit_const = .True.  ! Change to .false. if fitting a model without a constant.

```

```

! Initializes the QR-factorization
Call LSQ_startup(nvar, fit_const, Ep_all)
Call LSQ_startup(nvar, fit_const, Ep_some)

Call LSQ_set_names('Evaporation, all points', vname, 'Ep(obs)', Ep_all)
Call LSQ_set_names('Evaporation, some deletions', vname, 'Ep(obs)', Ep_some)

! Read in the data, one line at a time, and progressively update the
! QR-factorization.

wt = one

points_read = 0
points_excluded = 0
points_wo_parameters = 0
Read_One_Line: Do
  Read (uin, '(a)', iostat = ios) xbuf
  If (ios /= 0) Exit ! End-Of-File or Error

  ! Skip lines until we find the beginning of the data block, e.g.,
  ! ----- Evap (mm) -----
  If (xbuf(1:10) /= '-----') Cycle Read_One_Line

  ! Found the beginning of the data block.
  ! Find the name in the line, e.g., 'Evap (mm)'
  ! n1 points to the first character of the name
  ! n2-1 delimits the name. Note that the name may contain blanks.

  n1 = Verify(xbuf, '- ') ! Find first non '-' or blank, "E" for the example.
  n2 = Index(xbuf(n1:), '-') !
  If (n2 > 0) n2 = n2 + n1 - 1 - 1
  xid = xbuf(n1:n2) ! e.g., 'Evap (mm)'

  n1 = Index(xid, '(') + 1
  n2 = Index(xid, ')') - 1

  ! The program assumes evaporation is measured in mm.
  ! Convert (if needed) data file units to mm. I am told
  ! that the file units should be "mm", with
  ! (perhaps occasionally), "in" (inches)
  Select Case(xid(n1:n2))
  Case('mm')
    units_to_mm = 1 ! 1 mm -> 1 mm
  Case('in')
    units_to_mm = 25.400 ! 1 inch -> 25.400 mm
  Case Default
    Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
    Write (ULog, 9130) xid(n1:n2)
    Format (lx, '?? Process_Evaporation_Data: Expecting "mm" or ', &
           '"in", found "', a, '"')
9130

```

```

        Errors_Detected = .True.
        Return
    End Select

! The line following "-----" identifies the year and the columns:
! 1963      Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep      Oct      Nov      Dec      Annual

Read (uin, '(a)', iostat = ios) xbuf
If (ios /= 0) Then
    Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
    Write (ULog, *) '    Expecting a "year" line'
    Errors_Detected = .True.
    Return
End If

! Get year.
Read (xbuf(1:4), '(i4)', iostat = ios) yyyy
If (ios /= 0) Then
    Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
    Write (ULog, *) '    Line: "', Trim(xbuf), '"'
    Write (ULog, *) '    The first four characters Do not represent a year.'
    Errors_Detected = .True.
    Return
End If

ok = (MinYear <= yyyy) .And. (yyyy <= MaxYear)
If (.Not. ok) Then
    Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
    Write (ULog, *) '    The year ', yyyy, ' is not between ', &
        MinYear, ' and ', MaxYear
    Errors_Detected = .True.
    Return
End If

Days_in_Month => Number_of_Days_in_Month/yyyy)

! Now load the data block (the next 31 lines) in the array.
Do jj = 1, 31
    Read (uin, '(a)', iostat = ios) xbuf
    If (ios /= 0) Then
        Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
        Write (ULog, *) '    Error: Expecting month-day ', jj
        Errors_Detected = .True.
        Return
    End If

    ! Get the day
    Read (xbuf, '(i3)') dd

    ! Sanity check: jj must be equal to dd

```

```

If (jj /= dd) Then
  Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
  Write (ULog, *) ' Error: jj /= dd; jj, dd == ', jj, dd
  Errors_Detected = .True.
  Return
End If

! 1963      Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec   Annual
!  1        ---   ---   ---   ---   ---   ---   9    ---   ---   8    ---   ---
!  2        ---   ---   ---   ---   9    ---   8    ---   ---   7    ---   ---
! 28         0     6    31     0     0     0     0     0     0     0     0     0
! 29         0    ---     1     0     0     0     0     0     0     0     0     0
! 30         0    ---     0     0     0     0     0     0     0     0     0     1
! 31         0    ---     3    ---     0    ---     0     2    ---     0    ---     4

hh = NHours ! the 25-th hour is the daily value
npos = 4    ! Start processing at this column.
Month_Loop: Do mm = 1, 12
  Call GetQwordCols(xbuf, npos, beg_col, end_col, ierror)
  If (ierror /= 0) Then
    Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
    Write (ULog, *) ' Line: "', Trim(xbuf), '"'
    Write (ULog, *) ' contains less than twelve months, mm == ', mm
    Errors_Detected = .True.
    Return
  End If

  ! Make sure the day of the month is valid for this month.
  ! Example: 1963-Feb has 28 days. If we call ymdh_to_iv
  ! for 1963-Feb-29, the iv returned is for 1963-March-01
  ! (i.e., one day after 1963-Feb-28). Ditto for 30-day months.
  If (dd > Days_in_Month(mm)) Then
    ! Invalid date. Go to next (column) month.
    Cycle Month_Loop
  End If

  If (xbuf(beg_col:end_col) == '---') Then
    ! Missing data
    Cycle Month_Loop
  End If

  Read(xbuf(beg_col:end_col), *, iostat = ios) yobs
  If (ios /= 0) Then
    Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
    Write (ULog, *) ' Line: "', Trim(xbuf), '&
    ' "', substr: "', xbuf(beg_col:end_col), '"'
    Write (ULog, *) ' Not a number.'
    Errors_Detected = .True.
    Return
  End If

```

```

! Before converting to appropriate units, make sure
! this value is not a "flag" value.
! Undocumented flag values are in the range: [-8323 , -8290]
! 24 Mar 2002 11:55 am: anything less than -8000 will be
! considered to be a flag value and silently skipped.
If (yobs <= -8000) Then
  ! some flag value.
  Cycle Month_Loop
End If

! Convert input units to mm
yobs = yobs * units_to_mm

! value in range?
in_range = ((minV <= yobs) .And. (yobs <= maxV))
If (.Not. in_range) Then
  Write (ULog, *) '?? Input file: ', name_Daily_Evap(f0:f1)
  Write (ULog, *) ' Value not in range, ignored v,minV,maxV : ', yobs, minV, maxV
  Cycle Month_Loop
End If

points_read = points_read + 1
Call Stat_Add_Point(ObsEp_all, yobs)

! Compute the day-index iv
Call ymdh_to_iv(yyyy, mm, dd, hh, iv)

! We have an Ep observation. Replace the estimated Ep (Class A
! evaporation) with the observed value.
!yteo = Xparam(f_KP_FWS_Evaporation)%Samson_v10(iv)%v
yteo = Xparam(f_Ep)%Samson_v10(iv)%v
steo = Xparam(f_Ep)%Samson_v10(iv)%s
Xparam(f_Ep)%Samson_v10(iv)%v = yobs
Xparam(f_Ep)%Samson_v10(iv)%s = T_EarthInfo
Xparam(f_Ep)%Samson_v10(iv)%f = ''

! if the estimated Ep is missing, no further analysis is possible.
Select Case(steo)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  Cycle Month_Loop
End Select

! Compute the Julian Day number of the current day
! and store the corresponding pair.
jd_today = Jd(yyyy, mm, dd)
yyteo(jd_today) = yteo
yyobs(jd_today) = yobs

! Exclude days when
! * pan is frozen: Xparam(f_DBT)%Samson_v10(iv)%v <= 0

```

```

! * rainfall: Xparam(f_HP)%Samson_v10(iv)%v > 0
! * high wind: Xparam(f_WS)%Samson_v10(iv)%v >= w0
! -- all of which interfere with observation.
missing_params = &
    (Xparam(f_DBT)%Samson_v10(iv)%s == T_Missing) .Or. &
    (Xparam(f_HP)%Samson_v10(iv)%s == T_Missing) .Or. &
    (Xparam(f_WS)%Samson_v10(iv)%s == T_Missing) !.Or. &
    (Xparam(f_KP_FWS_Evaporation)%Samson_v10(iv)%s == T_Missing)
If (missing_params) Then
    ! 8 Feb 2002 5:41 pm: until we develop a better formulation,
    ! kill KP_FWS_Evaporation.
    points_wo_parameters = points_wo_parameters + 1
    !Write (ULog, *) '## Missing KP_FWS_Evaporation parameters for ', yyyy, mm, dd
    Cycle Month_Loop
End If

! The function returns wind speed (in m/s) at z=0.6 meters
! up6 -- Wind Speed in meters/sec at z=0.6 meters
! (approx 2 feet, the height of a Class A pan Anemometer)
! <A HREF="Utils1.f90#Wind Conversion Table">
up6 = Wind_Speed_F(Xparam(f_WS)%Samson_v10(hh)%v, T_up6)

!!!Call LSQ_includ(weight=wt, xrow=xx, yelem=yobs, T=Ep_all)

exclude_point = &
    (Xparam(f_DBT)%Samson_v10(iv)%v <= Zero) .Or. &
    (Xparam(f_HP)%Samson_v10(iv)%v >= maximum_rainfall) .Or. &
    (up6 >= maximum_wind_speed)

If (exclude_point) Then
    points_excluded = points_excluded + 1
    Cycle Month_Loop
End If

Call Stat_Add_Point(ObsEp_set, yobs)
Call Stat_Add_Point(teoEp_set, yteo)

! vars = {yteo}; matrix "data" contains {<vars>, yobs}
Write (Umath, 9150) yteo, yobs
9150 Format (3x, '{', f10.3, ', ', f10.3, '}',)

! Evap(Obs) = a*Evap(r0_estimate) + b
xx = (/ one, yteo /) ! A one is inserted as the first
! ! variable if a constant is being fitted.

! *** WARNING Array xx is overwritten by LSQ_includ.
Call LSQ_includ(weight=wt, xrow=xx, yelem=yobs, T=Ep_some)
End Do Month_Loop

```

```

        End Do
    End Do Read_One_Line

    ! All data was read. Close the file ...
    Call IOClose(uin)
    Call Vrmx(Xprint=.True.)

    ! ... and perform the linear least squares.

    Write (ULog, 9170) points_read, points_excluded, &
        points_wo_parameters, Ep_some%nobs
9170 Format (//, &
        1x, 'Points read ..... : ', i0, /, &
        1x, 'points_excluded ..... : ', i0, /, &
        1x, 'points_wo_parameters ... : ', i0, /, &
        1x, 'No. of regression points : ', i0)

    Call LSQ_Simple_Regression(Ep_some, ULog)

End Subroutine Process_Evaporation_Data

Subroutine Vrmx(Initialize, Xprint)

    Implicit None
    Logical, Optional, Intent(In) :: Initialize
    Logical, Optional, Intent(In) :: Xprint

    If (Present(Initialize)) Then
        Call Stat_Initialize(ObsEp_all, 'Obs Evap, all values')
        Call Stat_Initialize(ObsEp_set, 'Obs Evap, values used for regression only')
        Call Stat_Initialize(teoEp_set, 'Estimated Ep at "regression points"')
        Return
    End If

    If (Present(Xprint)) Then
        !!Write (ULog, '(//)')

        Call Stat_Output(ULog, ObsEp_all)
        Call Stat_Output(ULog, ObsEp_set)
        Call Stat_Output(ULog, teoEp_set)

        Return
    End If
End Subroutine Vrmx

End Module Evaporation_module

```

## Fix\_Data

! Last change: LSR 16 May 2002 5:55 pm

Module Fix\_Data\_Records

```
Use Date_Module
Use Global_Variables
Use Utils1
Implicit None
```

```
Private
Public :: Fix_SAMSON
```

Contains

Subroutine Fix\_SAMSON(Xok)

```
! Fix_SAMSON performs manual correction of the data records.
! Be careful when fixing precipitation records, in particular,
! with runs of missing, deleted, or accumulated values.
! Make sure the beginning or end of such a run is not lost
! on transfer.
```

```
Implicit None
Logical, Intent(Out) :: Xok
```

```
Xok = .True.
If (pWBAN%WBAN == '03940') Then
  Call Jackson_MS(Xok)
```

```
Else If (pWBAN%WBAN == '22516') Then
  Call Kahului_HI(Xok)
```

```
Else If (pWBAN%WBAN == '14922') Then
  Call Minneapolis_St_Paul_MN(Xok)
```

```
End If
```

End Subroutine Fix\_SAMSON

Subroutine Kahului\_HI(Xok)

```
! 22516: Kahului, HI
```

```
Implicit None
Logical, Intent(Out) :: Xok
```

```

Type(Val_and_Flag), Dimension(:), Pointer :: OI

OI => Xparam(f_OI)%Samson_v10      ! Observation Indicator

! Observation Indicator  0 or 9  0 = Weather observation made.
!                               9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
!
!   Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
!   Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
!   Xparam(f_OI)%Samson_v10(iv)%s = data_source

! The SAMSON record for Days since last snow has typos (see below).
!
! "39" -- "3" == Light snow pellets -- In Hawaii in July? At sea level?
! "97" -- "7" is an illegal value. The legal range is 0-5, 9.
!
! Since most of the Observation Indicator / Present_weather
! are missing, we will set these records missing also.
! (In particular, OI(iv=1) == OI(1961-01-01 lh) is missing.)

! Observation Indicator == 0:  for  160245 1978-07-20 20h; OI(hh)%f(f4:f5) == "39"
OI(160245) = OI(1)

! Observation Indicator == 0:  for  162123 1978-10-03 23h; OI(hh)%f(f4:f5) == "97"
OI(162123) = OI(1)

! Observation Indicator == 0:  for  166673 1979-04-03 23h; OI(hh)%f(f4:f5) == "39"
OI(166673) = OI(1)

Xok = .True.

End Subroutine Kahului_HI

Subroutine Jackson_MS(Xok)

! 03940 Jackson, MS is missing SAMSON hourly precipitation
! data for 1961-1963. This routine adds EarthInfo hourly
! values to the SAMSON record.

! The subroutine assumes that both the hourly SAMSON records
! and the hourly EarthInfo records have been read.

Implicit None
Logical, Intent(Out) :: Xok

Integer :: jv0, jv1

```

```

Type(Val_and_Flag), Dimension(:), Pointer :: HP => Null() ! Hourly precipitation

If (.Not. Have_ppt_Obs_hourly_data) Then
  ! We need EarthInfo hourly data.
  Xok = .False.
  Return
End If

HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

! The dates were gotten by examining the records.
Call ymdh_to_iv(1961, 01, 01, hh=01, iv=jv0)
Call ymdh_to_iv(1963, 07, 31, hh=24, iv=jv1)

! This statement also transfers the daily values, but we
! do not care.
HP(jv0:jv1) = Obs_ppt(jv0:jv1)

Xok = .True.

End Subroutine Jackson_MS

Subroutine Minneapolis_St_Paul_MN(Xok)

  ! #===# 1: Processing 14922: Minneapolis/St. Paul, MN

  ! The subroutine assumes that the hourly EarthInfo
  ! records have been read.

  Implicit None
  Logical, Intent(Out) :: Xok

  If (.Not. Have_ppt_Obs_Daily_data) Then
    ! We need EarthInfo summary of the day data.
    Xok = .False.
    Return
  End If

  ! Hourly Precipitation [cm] Out-of-range value for 218925 1984-12-22 25h: 832.053
  ! The EarthInfo summary of the day shows:

  !!! Station MINNEAPOLIS AIRPORT      Parameter      Prcp          % Coverage      100
  !!! PO Code MN                       Latitude      N44:52:59      Begin M/Yr      01/1891
  !!! Stn ID 5435                       Longitude    W093:13:44      End M/Yr        12/2000
  !!! County HENNEPIN                   Elevation      834            # Record Years  110
  !!! ----- Prcp (in) -----
  !!! 1984      Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep      Oct      Nov      Dec      Annual
  !!! 1

```

```
!!! :  
!!! 22    0.00  0.00  0.00  0.00  0.01  0.35  0.00T 0.00  0.31  0.00  0.00 327.58A  
  
    ! Neither the SAMSON nor the EarthInfo hourly records for that day show precipitation.  
    ! Zero the entry.  
  
    Obs_ppt(218925) = Val_and_Flag(T_Estimated, Zero, '')  
  
    Xok = .True.  
  
End Subroutine Minneapolis_St_Paul_MN  
  
End Module Fix_Data_Records
```

## Gaps

! Last change: LSR 16 May 2002 5:55 pm

Module Process\_Gaps

```
Use Global_Variables
Use Utils1
Use Floating_Point_Comparisons
Implicit None
```

Type :: Type\_Gap

```
! Filled -- truth of "gap was resolved". Values were
!           interpolated or gotten from previous years.
! Col_Beg -- iv of begin of gap
! Col_End -- iv of end of gap
! Col_Prev -- the hour previous to Col_Beg.
!           if Col_Beg represents an hour in 2..24,
!           then
!               Col_Prev = Col_Beg - 1
!           Else
!               Col_Beg is the 1st of the day and
!               Col_Prev represents the 24th hour of the previous day.
! Col_Post -- the hour after Col_End.
!           if Col_End represents an hour in 1..23,
!           then
!               Col_Post = Col_Beg + 1
!           Else
!               Col_End is the 24th hour of the day and
!               Col_Post represents the 1st of the next day.
```

```
Logical :: Filled = .False.
Integer :: Col_Beg = 0
Integer :: Col_End = 0
Integer :: Col_Prev = 0
Integer :: Col_Post = 0
```

End Type Type\_Gap

Logical, Private :: print\_now = .False.

Contains

Subroutine Find\_i(VF, Hour01, Hour24, iFirst, iLast)

```
! Find iFirst and iLast hours in the interval [Hour01, Hour24]
! such that iFirst is the first hour of the day that is not
! missing, and iLast is the last hour not missing.
Implicit None
```

```

Type(Val_and_Flag), Dimension(:), Intent(In)  :: VF
Integer,                               Intent(In)  :: Hour01, Hour24
Integer,                               Intent(Out) :: iFirst, iLast

Integer :: jj

iLast = 0   ! The last hour of the day with non-missing data;
iFirst = 0  ! The first hour of the day with non-missing data;
Do jj = Hour01, Hour24
  Select Case(VF(jj)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
      Cycle
    Case Default
      iLast = jj
      If (iFirst == 0) Then
        iFirst = jj
      End If
    End Select
  End Do

End Subroutine Find_i

Subroutine Find_Gaps(k_id, Release_Storage)

! Fill gaps.
! 16 Feb 2002  9:49 am: At this time I think that
! since I will be using these arrays for all SAMSON
! parameters, I will leave the arrays allocated, rather
! than allocating and deallocating after each call.
! This subroutine will allocate arrays that will
! stay allocated between calls (this is a feature).
!
! The call Find_Gaps(0, Release_Storage=.True.)
! will deallocate the arrays *only* and exit.
! (The first two argument slots need to be present
! eventhough will not be used.)
! Since we check only for the presence of "Release_Storage",
! this argument should not be present during "regular"
! calls to this subroutine.

Implicit None
Integer,          Intent(In) :: k_id
Logical, Optional, Intent(In) :: Release_Storage

Integer, Parameter :: Size_Small  = 1
Integer, Parameter :: Size_Medium = Size_Small + 1
Integer, Parameter :: Size_Large  = Size_Medium + 1

Integer :: current_gap_size, i__max, size_order

```

```

Integer :: i_prev, col_beg, col_end
Integer :: iv, ig
Logical :: in_gap, point_is_missing, okay, first_message
Type(Val_and_Flag), Dimension(:), Pointer :: VF

Logical, Save :: first_time = .True.
Integer, Save :: Gap_dim = 0
Integer :: ngaps
Type(Type_Gap), Dimension(:), Pointer :: Gap_Info => Null()

If (first_time) Then
  Gap_dim = 50
  Allocate(Gap_Info(Gap_dim))
  first_time = .False.
End If

If (Present(Release_Storage)) Then
  If (Associated(Gap_Info)) Deallocate(Gap_Info)
  Gap_dim = 0
  first_time = .True.
  Return
End If

! Is it sensible to fill gaps?
Select Case(k_id)
Case(f_OI)
  ! Observation Indicator, Present_weather:
  ! It makes no sense to fill gaps in these parameters.
  ! We should not be here in the first place.
  Write(6, 9130)
  Write(ULog, 9130)
9130  Format(//, 1x, '?? Find_Gaps: called with k_id == ', &
        'Observation Indicator, Present_weather', //)
  Stop '?? Find_Gaps: called with k_id == Observation Indicator'
End Select

VF => Xparam(k_id)%Samson_v10

in_gap = .False.
ig = 0          ! No gaps so far.
current_gap_size = 0
i_prev = 0

i__max = Ubound(VF,1) + 1

! We need to process gaps in the following order:
!   a) first Small gaps
!   b) Medium gaps
!   c) Large gaps
! This will allow the results of the smaller fills

```

```

! to be used for the larger data gaps.
!
! One can make the argument that we should use only
! observed data (rather than estimated) to fill gaps.
!
! Algorithm: Find and record all gaps, then pass
! over the arrays Gap_Beg and Gap_End three times,
! processing the gaps in the order indicated.

Do iv = 1, i__max

    ! Skip the 25th hour
    If (Modulo(iv,25) == 0) Then
        Cycle
    End If

    ! We need to loop one extra time to correctly process
    ! a gap occurring at the end of the array. Duplicating
    ! the code for (in_gap) also works but I always forgot
    ! to propagate the changes to both instances of the code.

    If (iv < i__max) Then
        ! At this stage, if the point is missing, the point
        ! is missing ALL the time, otherwise the algorithm
        ! will interpolate with the assumption that the
        ! value of the point is correct!
        Select Case(VF(iv)%s)
            Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
                point_is_missing = .True.
            Case Default
                point_is_missing = .False.
        End Select
    Else
        ! iv == i__max
        point_is_missing = .False.
        ! The point is not missing -- it does not exist.
    End If

    If (in_gap) Then
        If (point_is_missing) Then
            current_gap_size = current_gap_size + 1
        Else
            ! The gap run just ended. Store terminating info.
            ! The gap finished on the previous column.
            Gap_Info(ig)%Col_End = i_prev
            Gap_Info(ig)%Col_Post = iv

            in_gap = .False.
            current_gap_size = 0
        End If
    End If

```

```

Else
  If (point_is_missing) Then
    ! Start of a new gap
    If (ig >= Gap_dim) Then
      Gap_dim = 2 * Gap_dim
      Gap_Info => Reallocate_Gaps(Gap_Info, Gap_dim)
    End If
    ig = ig + 1
    Gap_Info(ig)%Filled = .False.
    Gap_Info(ig)%Col_Beg = iv
    Gap_Info(ig)%Col_Prev = i_prev ! "iv - 1"
    Gap_Info(ig)%Col_End = 0
    Gap_Info(ig)%Col_Post = 0
    in_gap = .True.
    current_gap_size = 1
  Else
    ! No missing value and no gap.
    ! Do nothing.
  End If
End If

! Previous column to the next column.
! Note that next column is iv+1, unless iv is a 24th hour.
! In that case the next column is iv+2 (i.e., the first
! hour of the next day.
i_prev = iv
End Do

! Now, we need to process all the small gaps first,
! the medium gaps second, and the large gaps third.
! What is the best way (more efficient, faster) of
! passing through the arrays three times?
! I can have essentially the same code replicated
! three times, or the code once with if statements.
!
! Sat 16 Feb 2002 10:22 am. At this time I am selecting
! "if" statements. Replicated code has the tendency
! of getting out of synch with each other.

ngaps = ig
Do size_order = Size_Small, Size_Large

  ProcessGaps: Do ig = 1, ngaps

    ! Was this gap was processed?
    If (Gap_Info(ig)%Filled) Cycle ProcessGaps

    okay = .False.
    col_beg = Gap_Info(ig)%Col_Beg
    col_end = Gap_Info(ig)%Col_End

```

```

! Do not count the 25-th hour(s) as part of the gap.
current_gap_size = col_end - col_beg + 1 - Multiples_of(NHours, col_beg, col_end)

! Process all small gaps first, medium gaps second, and large gaps third.
! Tests:
! 23063: Eagle, CO has small, medium, and large gaps.
! 26451: Anchorage, AK has gaps containing the array boundaries.

Select Case(current_gap_size)
Case(1:5)
  If (size_order == Size_Small) Then
    Call Fill_Small_Gaps(VF, Gap_Info(ig), k_id, okay)
    Gap_Info(ig)%Filled = okay ! Gap filled?
  End If

Case(6:49)
  If (size_order == Size_Medium) Then
    Call Fill_Medium_Gaps(VF, current_gap_size, Gap_Info(ig), &
      k_id, move_back=.True., okay=okay)
    Gap_Info(ig)%Filled = okay ! Gap filled?
  End If

Case Default
  If (size_order == Size_Large) Then
    Call Fill_Large_Gaps(VF, current_gap_size, Gap_Info(ig), &
      k_id, okay)
    Gap_Info(ig)%Filled = okay ! Gap filled?
  End If

End Select

End Do ProcessGaps
End Do

! Now one more time (with feeling). This time we want to
! ensure that all gaps were filled

first_message = .True.
Do ig = 1, ngaps

  If (Gap_Info(ig)%Filled) Then
    Cycle
  End If

  col_beg = Gap_Info(ig)%Col_Beg
  col_end = Gap_Info(ig)%Col_End
  current_gap_size = col_end - col_beg + 1 - Multiples_of(NHours, col_beg, col_end)

  If (first_message) Then
    Write (ULog, *)

```

```

        first_message = .False.
    End If

    Write (ULog, 9150) ig, Trim(FieldInfo(k_id)%Name), &
        current_gap_size, &
        Str_iv_to_ymdh(col_beg), &
        Str_iv_to_ymdh(col_end)
9150    Format(1x, 6x, i0, ': ', a, ': gap of ', i0, ' hours from ', &
        a, ' to ', a)
    End Do

End Subroutine Find_Gaps

Function Reallocate_Gaps(pOld, Nnew) Result(pNew)
    Implicit None
    Type(Type_Gap), Dimension(:), Pointer :: pOld, pNew
    Integer, Intent(In) :: Nnew
    Integer :: nold, ierr

    Allocate(pNew(1:Nnew), STAT=ierr)
    If (ierr /= 0) Stop "?? Reallocate_Gaps error"

    If (.Not. Associated(pOld)) Return

    nold = Min(Size(pOld), Nnew)
    pNew(1:nold) = pOld(1:nold)
    Deallocate(pOld)
End Function Reallocate_Gaps

Subroutine Fill_Small_Gaps(VF, Gap_Info, k_id, okay)

! <A NAME="Fill_Small_Gaps"> <A HREF="Gaps.f90#Fill_Small_Gaps">
! gap <= 5 h --> linear interpolation between points
! at either side of the gap.

    Implicit None
    Type(Val_and_Flag), Dimension(:), Intent(InOut) :: VF
    Type(Type_Gap), Intent(In) :: Gap_Info
    Integer, Intent(In) :: k_id
    Logical, Intent(Out) :: okay

    Integer :: Col_Beg, Col_End ! Col_Beg <= Col_End
    Integer :: xi, x0, xn, irange
    Real :: vn, v0, vslope, dx
    Logical :: within_range
    Character(Len(VF(1)%s)) :: t_code

```



```

Else
  ! The terminating column is within bounds.
  x0 = Gap_Info%Col_Post
  xn = x0
  v0 = VF(x0)%v
  vslope = 0
End If
End If

t_code = T_Small_Gap
Select Case(k_id)
Case(f_OI)
  ! Observation Indicator, Present_weather
  ! It makes no sense to fill gaps in these parameters
  ! We should not be here in the first place.
  Write(ULog, 9130)
9130  Format(//, 1x, '?? Warning: Fill_Small_Gaps with ', &
        'Observation Indicator, Present_weather', //)
  Return

Case(f_HV) ! Horizontal Visibility
  ! Be careful filling gaps when
  ! Visibility == unlimited visibility.
  If ((VF(x0)%s == T_Unlimited) .Or. (VF(xn)%s == T_Unlimited)) Then
    t_code = T_SGU
  End If

Case(f_CH) ! Ceiling Height
  ! Be careful filling gaps when
  ! Ceiling Height == unlimited ceiling height, or
  ! == cirroform.
  ! We have this problem for 14914 (Fargo), 1965 1 1 12h-15h
  If ((VF(x0)%s == T_Unlimited) .Or. (VF(xn)%s == T_Unlimited)) Then
    t_code = T_SGU
  Else If ((VF(x0)%s == T_Cirroform) .Or. (VF(xn)%s == T_Cirroform)) Then
    t_code = T_SGC
  End If

Case Default
  ! Do nothing.
End Select

Do xi = Col_Beg, Col_End
  ! Skip the 25th hour
  If (Modulo(xi,25) == 0) Then
    Cycle
  End If

  dx = xi - x0 - Multiples_of(NHours, x0, xi)
  VF(xi)%v = vslope*dx + v0

```

```

VF(xi)%s = t_code
VF(xi)%f = ''
End Do
okay = .True.
End Subroutine Fill_Small_Gaps

```

```

Subroutine Fill_Medium_Gaps(VF, Gap_Length, Gap_Info, &
    k_id, move_back, okay)

```

```

! <A NAME="Fill_Medium_Gaps"> <A HREF="Gaps.f90#Fill_Medium_Gaps">
! Longer gaps of 6 to 49 hours: Copy data from previous day (period).

```

```

Implicit None
Type(Val_and_Flag), Dimension(:), Intent(InOut) :: VF
Integer, Intent(In) :: Gap_Length
Type(Type_Gap), Intent(In) :: Gap_Info
Integer, Intent(In) :: k_id
Logical, Intent(In) :: move_back
Logical, Intent(Out) :: okay

```

```

Integer :: Col_Beg, Col_End ! Col_Beg <= Col_End
Integer :: j, jmax, jmin, jv0, jv1, iv
Integer :: nh, ntimes
Integer :: jMidnight, jSunset, jSunrise
Logical :: period_found, moving_back
Character(Len(VF(1)%s)) :: t_code

```

```

okay = .False.
Col_Beg = Gap_Info%Col_Beg
Col_End = Gap_Info%Col_End

```

```

! If moving back j periods:
!   jv0 = Col_beg - j*Nhours
!   jv1 = Col_End - j*Nhours
!
! subject to
!   1 <= jv0 <= jv1 < Col_beg
!
! Let
!   jmin = 1 + Floor((Col_End-Col_Beg)/Nhours)
!   jmax = Floor((Col_Beg-1)/Nhours)
!
! Then, the set of possible periods j is: jmin <= j <= jmax
!
!
! Similarly, if moving forward j periods:
!   jv0 = Col_beg + j*Nhours
!   jv1 = Col_End + j*Nhours
!

```

```

! subject to
!   Col_beg < jv0 <= jv1 <= M
!
! Where
!   M = UBound(VF)
!
! Let
!   jmin = 1 + Floor((Col_End-Col_Beg)/Nhours)
!   jmax = Floor((M-Col_Beg)/Nhours)
!
! Then, the set of possible periods j is:  jmin <= j <= jmax

moving_back = move_back
period_found = .False.
t_code = T_Medium_Gap

! Only two options: move forward or backwards.
! If we have failed twice, there is no more to do.
xDirection: Do ntimes = 1, 2
  If (period_found) Exit xDirection

  jmin = 1 + Floor((Col_End-Col_Beg)/Real(Nhours))
  If (moving_back) Then
    ! search backwards.
    nh = -Nhours
    jmax = Floor((Col_Beg-1)/Real(Nhours))
  Else
    ! search forward.
    nh = +Nhours
    jmax = Floor((UBound(VF,1)-Col_Beg)/Real(Nhours))
    jmax = Min(jmax, 2) ! we will look only in the immediate vicinity
  End If

  If (jmin > jmax) Then
    ! No periods in this direction. Switch directions and try again.
    moving_back = (.Not. moving_back)
    Cycle xDirection
  End If

  ! We can try in this direction.
  Find_period: Do j = jmin, jmax
    jv0 = Col_Beg + j*nh
    jv1 = Col_End + j*nh

    ! Evaluate the candidate period.
    ! Make sure there are no missing values.
    ! If this period has missing values, then try next period.
    Do iv = jv0, jv1
      ! Skip the 25th hour
      If (Modulo(iv,25) == 0) Cycle

```



```

End If

! Find the 25th hour          ! See example above
!hh = Modulo(Col_Beg, 25)    ! 19
!dl = NHours - hh           ! 6 (delta hours from 25h)
!jMidnight = Col_Beg + dl   ! 500
jMidnight = NHours * (1 + Col_Beg/NHours) ! 500

! Find jSunset and jSunrise, making sure they lie within the
! array bounds.
jSunset = Gap_Info%Col_Prev ! pointer to value before midnight
If (jSunset < Lbound(VF,1)) Then
    jSunset = Gap_Info%Col_Post
    If (jSunset > Ubound(VF,1)) Then
        ! The gap spans the whole array, nothing sensible to do.
        Return
    End If
End If

jSunrise = Gap_Info%Col_Post ! pointer to value after midnight
If (jSunrise > Ubound(VF,1)) Then
    jSunrise = Gap_Info%Col_Prev
    If (jSunrise < Lbound(VF,1)) Then
        ! The gap spans the whole array, nothing sensible to do.
        Return
    End If
End If

VF(Col_Beg:jMidnight)%v = VF(jSunset)%v
VF(Col_Beg:jMidnight)%s = t_code
VF(Col_Beg:jMidnight)%f = ''

VF(jMidnight+1:Col_End)%v = VF(jSunrise)%v
VF(jMidnight+1:Col_End)%s = t_code
VF(jMidnight+1:Col_End)%f = ''

okay = .True.

End Subroutine Fill_Medium_Gaps

Subroutine Fill_Large_Gaps(VF, Gap_Length, Gap_Info, k_id, okay)

! <A NAME="Fill_Large_Gaps"> <A HREF="Gaps.f90#Fill_Large_Gaps">
! Much Longer gaps of 50 to 8784 hours (one leap year).
! Copy data from the same period from some other year.

Implicit None
Type(Val_and_Flag), Dimension(:), Intent(InOut) :: VF

```

```

Integer,                               Intent(In)   :: Gap_Length
Type(Type_Gap),                        Intent(In)   :: Gap_Info
Integer,                               Intent(In)   :: k_id
Logical,                               Intent(Out)  :: okay

Integer :: Col_Beg, Col_End ! Col_Beg <= Col_End
Integer :: jv0, jv1, yyyy, n_min, iv
Integer :: beg_year, beg_mm, beg_dd, beg_hh
Integer :: nhours, d_min, d_max, ig, ic, k
Integer :: best_year, itimes, g_low, g_high
Integer :: best_jv0, best_jv1
Real    :: best_mean_diff, year_mean_diff
Real    :: best_sigma_diff, year_sigma_diff
Real    :: gM_k, gQ_k, cM_k, cQ_k
Real    :: M_kml, x_k
Real    :: g_mean, c_mean, g_variance, c_variance
Logical :: best_found, Lkeep
Character(Len(VF(1)%s)) :: t_code

okay = .False.
t_code = T_Large_Gap
Col_Beg = Gap_Info%Col_Beg
Col_End = Gap_Info%Col_End

! Array bounds
d_min = Lbound(VF,1)
d_max = Ubound(VF,1)

! Determine the date of the beginning of the gap.
Call iv_to_ymdh(Col_Beg, beg_year, beg_mm, beg_dd, beg_hh)

! According to the documentation, SAMSON may look at
! periods up to 4 weeks adjacent to the gap.
! Interval: 4 weeks * 7 days/week * 24h/day, prorated to a leap year
nhours = Max(10, 4*7*24*Gap_Length/8784)

! Numerically stable computation of the variance.
!
!
!           1      n
!   Sample variance = ----- Sum (x_i-x_mean)^2
!                   n - 1  i=1
! Reference:
! [2] Nicholas J. Higham. 1996. Accuracy and Stability of Numerical
!     Algorithms. SIAM (Society for Industrial & Applied Mathematics).
!     ISBN 0-89871-355-2. Page 13.
!
! Accumulate:
!
!           1      k
!   M_k = - * Sum x_i

```

```

!           k   i=1
!
!           k           k           1   k
!   Q_k = Sum (x_i - M_k)^2 = Sum (x_i)^2 - - (Sum x_i)^2
!           i=1           i=1           k   i=1
!
! Updating formulae:
!
!   M_1 = x_1
!   M_k = M_{k-1} + -----,           k = 2..n
!                   x_k - M_{k-1}
!                   k
!   Q_1 = 0
!   Q_k = Q_{k-1} + -----,           k = 2..n
!                   (k-1) (x_k - M_{k-1})^2
!                   k
! After which:
!
!   Sample Mean = M_n
!
!   Sample Variance = -----
!                   Q_n
!                   n - 1
!
! Note that the updating formulae can be written:
!
!   M_0 = 0
!   M_k = M_{k-1} + -----,           k = 1..n
!                   x_k - M_{k-1}
!                   k
!   Q_0 = 0
!   Q_k = Q_{k-1} + -----,           k = 1..n
!                   (k-1) (x_k - M_{k-1})^2
!                   k

! We will look for years whose pre- and post- gap mean and variance
! are closest to the gap's pre- and post- mean and variance.
!
! mean_diff = Abs(g_mean - c_mean)
! sigma_diff = Abs(g_Variance - c_Variance)

best_found = .False.           ! found a year?
best_year = 0                   ! in MinYear .. MaxYear
best_mean_diff = Huge(Zero)    ! best mean difference so far
best_sigma_diff = Huge(Zero)

By_Year: Do yyyy = MinYear, MaxYear

! Skip the year of the gap
! If (yyyy == beg_year) Then

```

```

    Cycle By_Year
End If

! Skip years not read, i.e., data for the whole year is missing.
If (Year_Data(yyyy)%SAMSON_v10 == 0) Then
    Cycle By_Year
End If

! Determine the location of the corresponding segment in this year.
Call ymdh_to_iv(yyyy, beg_mm, beg_dd, beg_hh, iv=jv0)
jv1 = jv0 + Col_End - Col_Beg

! Gap end falls outside array bounds ?
If (jv1 > d_max) Then
    Cycle By_Year
End If

! Any entries missing?
By_Hour: Do iv = jv0, jv1
    ! Skip the 25th hour
    If (Modulo(iv,25) == 0) Cycle By_Hour
    Select Case(VF(iv)%s)
        Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
            ! This segment has missing values. Perhaps next year ...
            Cycle By_Year
        End Select
    End Do By_Hour

! Found a candidate segment without missing values.
! Determine mean and variance for both segments.

k = 0          ! Number of points.
gM_k = Zero
gQ_k = Zero
cM_k = Zero
cQ_k = Zero

QInterval: Do itimes = 1, 2
    ! Execute twice to determine leading and trailing intervals.
    ! Make sure the leading and trailing intervals are within
    ! the array boundaries.

    ! n_min is the number of entries common to both intervals; n_min >= 0;

    If (itimes == 1) Then ! Determine leading interval.
        n_min = Min(nhours, Col_Beg-d_min, jv0-d_min)
        g_low = Col_Beg - n_min
        g_high = Col_Beg - 1
    Else ! Determine trailing interval.
        n_min = Min(nhours, d_max-Col_End, d_max-jv1)
    End If
End Do QInterval

```

```

    g_low = Col_End + 1
    g_high = Col_End + n_min
End If

! ig: counter of gap
! ic: counter of candidate
Xview: Do ig = g_low, g_high
    ic = ig + jv0 - Col_Beg

    ! The pre/post gap segments may contain missing values.
    ! Nothing we can do about that.
    Select Case(VF(ig)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        ! Missing values. Skip.
        Cycle Xview
    End Select

    Select Case(VF(ic)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        ! Missing values. Skip.
        Cycle Xview
    End Select

    k = k + 1

    x_k = VF(ig)%v
    M_kml = gM_k
    gM_k = M_kml + (x_k-M_kml)/Real(k)
    gQ_k = gQ_k + (Real(k-1)/Real(k))*(x_k-M_kml)**2

    x_k = VF(ic)%v
    M_kml = cM_k
    cM_k = M_kml + (x_k-M_kml)/Real(k)
    cQ_k = cQ_k + (Real(k-1)/Real(k))*(x_k-M_kml)**2

End Do Xview
End Do QInterval

g_mean = gM_k
c_mean = cM_k

If (k >= 2) Then
    g_variance = gQ_k / Real(k-1)
    c_variance = cQ_k / Real(k-1)
Else
    ! Variance not defined for k<=1
    g_variance = Huge(Zero)
    c_variance = Huge(Zero)
End If

```

```

year_mean_diff = Abs(g_mean-c_mean)
year_sigma_diff = Abs(g_variance-c_variance)

! #1. Choose the year closest to the mean of the gap.
! #2. If the means are equal, choose the year with the
!     closest variance.
Lkeep = .False.
If (year_mean_diff < best_mean_diff) Then
    ! Current year has the smallest mean difference. Keep it.
    Lkeep = .True.
Else If (Abs(year_mean_diff-best_mean_diff) < Eps0) Then
    ! Means are equal, i.e., year_mean_diff == best_mean_diff.
    ! Keep this year if its variance difference is smaller than the
    !     best variance difference so far.
    Lkeep = (year_sigma_diff < best_sigma_diff)
End If
If (Lkeep) Then
    best_found = .True. ! found a better year.
    best_year = yyyy    ! in MinYear .. MaxYear
    best_jv0 = jv0
    best_jv1 = jv1
    best_mean_diff = year_mean_diff
    best_sigma_diff = year_sigma_diff
End If
End Do By_Year

If (best_found) Then
    VF(Col_Beg:Col_End)%v = VF(best_jv0:best_jv1)%v
    VF(Col_Beg:Col_End)%f = ''
    VF(Col_Beg:Col_End)%s = t_code
End If
okay = best_found

End Subroutine Fill_Large_Gaps

```

```

Subroutine Fill_Dew_Point(Xok)

! Estimate missing Dew point

Implicit None
Logical, Intent(Out) :: Xok

Integer :: iv
Integer :: jyyyy, jmm, jdd, jhh
Logical :: missing_params

Xok = .False.
Do iv = 1, Ubound(Xparam(f_DPT)%Samson_v10,1)

```

```

! Skip 25th hour
If (Modulo(iv,25) == 0) Then
  Cycle
End If

! Skip non-missing Dew points
If (Xparam(f_DPT)%Samson_v10(iv)%s /= T_Missing) Cycle

! The estimation is a function of air temperature and Relative Humidity.
missing_params = &
  (Xparam(f_DBT)%Samson_v10(iv)%s == T_Missing) .Or. &
  (Xparam(f_RH)%Samson_v10(iv)%s == T_Missing)
! If missing parameters, skip this point
If (missing_params) Cycle

! If Relative Humidity == 0, then the Dew point is undefined.
If (Xparam(f_RH)%Samson_v10(iv)%v > Eps0) Then
  Xparam(f_DPT)%Samson_v10(iv)%v = DewPointF( &
    Ta=Xparam(f_DBT)%Samson_v10(iv)%v, &
    RH=Xparam(f_RH)%Samson_v10(iv)%v)

  Xparam(f_DPT)%Samson_v10(iv)%s = T_Estimated
  Xparam(f_DPT)%Samson_v10(iv)%f = ''
Else
  ! Relative Humidity == 0: Dew point is undefined.
  Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
  Write(ULog, 9130) jyyyy, jmm, jdd, jhh
9130  Format(1x, '?? Fill_Dew_Point: RH == 0 for ', i4, 2('-',i2.2), i3, 'h')
  Xparam(f_DPT)%Samson_v10(iv)%v = -Huge(Zero)
  Xparam(f_DPT)%Samson_v10(iv)%s = T_Undefined
  Xparam(f_DPT)%Samson_v10(iv)%f = ''
End If

End Do
Xok = .True.
End Subroutine Fill_Dew_Point

End Module Process_Gaps

```

## global

```
!      Last change: LSR   6 Jun 2002   3:18 pm
Module Global_Variables

! See <A HREF="e:\5\3met\r0.f90\0notes.txt#$1">
!
! For a description of the SAMSON file format see
!      <A HREF="e:\5\3met\Docs\samson_format.txt#$1">

Use Constants      ! F90Lib
Use IoSubs
Implicit None

! Errors and debug variables
Logical, Save :: Errors_Detected = .False.
Logical, Save :: Global_Debug = .False.
Integer, Save :: ULog = 0
Integer, Save :: Umetadata = 0
Integer, Save :: Umath = 0

! Location of files
Character(MaxNamLen), Save :: Log_dir = 'v:\Logs'
Character(MaxNamLen), Save :: Log_file = ''
Character(MaxNamLen), Save :: R0_root = 'v:\r0.by.State\'
Character(MaxNamLen), Save :: R0_full = '' ! Changes by station
Character(MaxNamLen), Save :: Raw_Data_dir = 'z:\'
Character(MaxNamLen), Save :: zcat = 'F:\mks\mksnt\zcat.exe'
Character(MaxNamLen), Save :: metadata_coda = 'v:\Docs\metadata.txt'

! Time stamp for generated files.
Character(Len=25), Save :: TimeStamp = ''
Character(Len=1), Save :: DirDelim = '\'

! Time periods
Integer, Parameter :: MinYear = 1961
Integer, Parameter :: MaxYear = 1990

! The latitude of the Arctic Circle. The latitude of anything North of
! the Arctic Circle is >= Latitude_Arctic_circle.
Real, Parameter :: Latitude_Arctic_circle = 66.5 ! degrees North

Integer, Save :: jd0 = 0 ! From Jd(MinYear,01,01)
Integer, Save :: jdl = 0 ! To Jd(MaxYear,12,31)

! 1-24: hourly values
! 25: daily averages
Integer, Parameter :: Nhours = 25

! Epsilon(Zero) is too small. (26 Apr 2002 10:05 am)
```

```

! Use Eps0 for fuzzy comparisons.
!
! <A NAME="fuzzy comparisons">
! See <A HREF="global.f90#fuzzy comparisons">
!
! The initial precision of SAMSON and EarthInfo data is
! typically 1e-2, i.e., if the difference of two numbers A and B
! is less than 1e-2, for all intents and purposes the numbers
! are equal. We were getting differences on the order
! of 1e-16 to 1e-18. If B = A +/- 1e-16, then the numbers are
! clearly equal and the difference is due to numeric flutter.
!
! The value for Eps0 below was chosen after examination of the
! behaviour of different sections of code of make_r0.
! See, <A HREF="Precip.f90#Reconcile_HP_samson_earthinfo">
! among others.
!
! Comparison:      Test:
!   A == B         Abs(A-B) < Eps0
!   A /= B         Abs(A-B) >= Eps0
!   A >= B         Abs(A-B) < Eps0 .Or. Eps0 < (A-B)
!
!                   ---+-----|-----+----
!                   B   A-Eps0   A
!                   B < A - Eps0
!                   Eps0 < A - B
!
!   A > B          (A /= B) .And. (A >= B)
!                   Abs(A-B) >= Eps0 .And. Eps0 < (A-B)
!
!   A <= B         Abs(B-A) < Eps0 .Or. Eps0 < (B-A)
!
!   A < B          Abs(B-A) >= Eps0 .And. Eps0 < (B-A)

Real, Parameter :: Eps0 = 1.0e-6

! Because of roundoff errors and the initial precision
! of the precipitation data, differences less than 0.02 cm
! are not significant (0.01 inch == 0.0254 cm).
Real, Parameter :: ppt_Eps = 0.02

! All legal values are non-negative.
Integer, Parameter :: Missing_Data = 999999
Real, Parameter :: Zero = 0.0
Real, Parameter :: One = 1.0

!Real, Parameter :: Pi = 3.141592653589793238462643383279502884197
Real, Parameter :: Degrees_to_Radians = Pi / 180.0
Real, Parameter :: Radians_to_Degrees = 180.0 / Pi
Real, Parameter :: Two_Pi = 2.0 * Pi

```

```
! A value outside the index range of both arrays Samson_v10 and Samson_v11.
Integer, Parameter :: Tbogus = -Huge(0)
```

```
! Conversion factors
```

```
! Watt hour = 3.6e-3 MJoule
Real, Parameter :: Watt_hour__to__MJoule = 3.6e-3
```

```
! Watt hour m^-2 = 8.59845E-02 Langley
Real, Parameter :: Watt_hour_per_m2__to__Langley = 8.59845E-02
```

```
! 1 millibar = 0.10000 kilopascal
Real, Parameter :: millibar__to__kilopascal = 0.10000
```

```
! 1 feet = 0.30480 meter
Real, Parameter :: feet__to__meter = 0.30480
```

```
! 1 inch = 25.400 millimeter
! 1 inch = 2.5400 cm
Real, Parameter :: inches__to__cm = 2.5400
```

```
! meters/second = 86.400 kilometers/day
Real, Parameter :: ms__to__kmd = 86.400
```

```
! 1 meter/second = 100.00 cm/second
Real, Parameter :: meters_sec__to__cm_sec = 100.0
```

```
! Wind Speed indicators
```

```
Integer, Parameter :: T_u10 = 1100
Integer, Parameter :: T_u2 = 1020
Integer, Parameter :: T_u4 = 1040
Integer, Parameter :: T_up6 = 1006
Integer, Parameter :: T_up1 = 1001
```

```
! field ids
```

```
Integer, Parameter :: f_EHR = 1      ! Extraterrestrial Horizontal Radiation
Integer, Parameter :: f_EDNR = 2     ! Extraterrestrial Direct Normal Radiation
Integer, Parameter :: f_GHR = 3      ! Global Horizontal Radiation
Integer, Parameter :: f_DNR = 4      ! Direct Normal Radiation
Integer, Parameter :: f_DHR = 5      ! Diffuse Horizontal Radiation
Integer, Parameter :: f_TSC = 6      ! Total Sky Cover
Integer, Parameter :: f_OSC = 7      ! Opaque Sky Cover
Integer, Parameter :: f_DBT = 8      ! Dry Bulb Temperature
Integer, Parameter :: f_DPT = 9      ! Dew Point Temperature
Integer, Parameter :: f_RH = 10      ! Relative Humidity
Integer, Parameter :: f_SP = 11      ! Station Pressure
Integer, Parameter :: f_WD = 12      ! Wind Direction
Integer, Parameter :: f_WS = 13      ! Wind Speed
Integer, Parameter :: f_HV = 14      ! Horizontal Visibility
```

```

Integer, Parameter :: f_CH = 15      ! Ceiling Height
Integer, Parameter :: f_pH2O = 16   ! Precipitable Water
Integer, Parameter :: f_baod = 17   ! Broadband Aerosol Optical Depth
Integer, Parameter :: f_SD = 18     ! Snow Depth
Integer, Parameter :: f_DSLS = 19   ! Days since last Snowfall
Integer, Parameter :: f_HP = 20     ! Hourly Precipitation (value + flags)
Integer, Parameter :: f_OI = 21     ! Observation_Indicator / Present Weather
Integer, Parameter :: f_SAMSON = 22 - 1 ! Last SAMSON parameter.

Integer, Parameter :: f_FAO_SG_PET = 22 ! FAO Short Grass PET, mm/day
Integer, Parameter :: f_Ep = 23       ! Class A pan Evaporation, mm/day
Integer, Parameter :: f_KP_FWS_Evaporation = 24 ! K-P FWS Evaporation, mm/day
Integer, Parameter :: f_end = 25 - 1   ! Last daily parameter.

Integer, Parameter :: f_Dry_Bulb_Temperature = f_DBT
Integer, Parameter :: f_Hourly_Precipitation = f_HP
Integer, Parameter :: f_Observation_Indicator = f_OI
Integer, Parameter :: f_Opaque_Sky_Cover = f_OSC
Integer, Parameter :: f_PW = f_OI      ! Present Weather
Integer, Parameter :: f_Ra = f_EHR    ! FAO name
Integer, Parameter :: f_Rdiff = f_DHR ! name used in Dr Burns' manuscript
Integer, Parameter :: f_Relative_Humidity = f_RH
Integer, Parameter :: f_Rs = f_GHR    ! FAO name
Integer, Parameter :: f_Station_Pressure = f_SP
Integer, Parameter :: f_Wind_Direction = f_WD
Integer, Parameter :: f_Wind_Speed = f_WS
!Integer, Parameter :: f_Pan_Evaporation = f_KP_FWS_Evaporation

! 19 Apr 2002 11:41 am :: I cannot remember why I selected
!      g_* vs d_* for the names.
Integer, Parameter :: g_Precipitation = 1
Integer, Parameter :: g_Pan_Evaporation = 2
Integer, Parameter :: g_Temperature_mean = 3
Integer, Parameter :: g_Wind_Speed = 4
Integer, Parameter :: g_Solar_Radiation = 5
Integer, Parameter :: g_FAO_Short_Grass = 6
Integer, Parameter :: g_Daylight_Station_Pressure = 7
Integer, Parameter :: g_Daylight_Relative_Humidity = 8
Integer, Parameter :: g_Daylight_Opaque_Sky_Cover = 9
Integer, Parameter :: g_Daylight_Temperature = 10
Integer, Parameter :: g_Daylight_Broadband_Aerosol = 11
Integer, Parameter :: g_Daylight_Mean_Wind_Speed = 12
Integer, Parameter :: d_Daylight_max_wind_speed = 13
Integer, Parameter :: d_Daylight_direction_of_max_wind_speed = 14
Integer, Parameter :: d_PWS = 15     ! Daylight Prevailing Wind Speed
Integer, Parameter :: d_PWD = 16    ! Daylight Prevailing Wind Direction
Integer, Parameter :: g_end = 17 - 1 ! Last MET parameter.

```

```
! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
```

```

Type :: Coords
  Character(Len=1) :: Letter = ''
  Integer :: degrees = 0
  Integer :: minutes = 0
End Type Coords

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: Samson_Header
  Character(Len=05) :: WBAN = '' ! 94018
  Character(Len=50) :: Text = '' ! Boulder, CO
  Type(Coords)      :: Lat      ! N 40 1
  Type(Coords)      :: Lon      ! W 105 15
  Real              :: Elev     ! 1634
  Integer           :: TZ       ! 7
End Type Samson_Header

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: FileInfo
  Integer :: yyyy = 0
  Integer :: Expected_Ndays = 0
  Type(Samson_Header) :: Head
!!!!!!Type(Errors_type), Pointer :: pHead => Null()
!!!!!!Type(Errors_type), Pointer :: pTail => Null()
  Integer, Dimension(:,), Pointer :: Every_Hour_Present => Null()
  Logical :: Check_Header_Info = .False.
End Type FileInfo

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: FieldInfo_type
  Character(Len=80) :: Name = ''

  ! The day has to have at least Minimum_obs_per_day,
  ! otherwise the missing count will be increased.
  Integer :: Minimum_obs_per_day = 0
  Real :: minimum_value = Zero
  Real :: maximum_value = Zero
  Real :: vmax = Zero
  Real :: vmin = Zero
End Type FieldInfo_type

!!Type(FieldInfo_type), Dimension(0:f_SAMSON), Target, Save :: FieldInfo
Type(FieldInfo_type), Dimension(0:f_end), Target, Save :: FieldInfo
Type(FieldInfo_type), Dimension(0:g_end), Target, Save :: MET_field

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: Year_type
  Integer :: SAMSON_v10 = 0
  Integer :: SAMSON_v11 = 0
End Type Year_type

```

```

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! See <A HREF="e:\5\3met\Docs\samson_format.txt#$1">
! Data source values used in the r0 files.
! <A NAME="Data Source Flag Parameters">
!
! * <A NAME="Missing_Values">
! See <A HREF="0notes.txt#Note_14">
!
! 19 Feb 2002  3:03 pm
! * Leave as missing: Hourly Wind Speed, Hourly Wind Direction, Snow Depth,
!   Days since last snowfall.
! * Daily Wind Speed -- fill with the monthly mean.
!   13 Mar 2002  3:44 pm: Not anymore. Use Fill_Gap algorithms.
! * Dew Point -- estimate using <A HREF="Utils1.f90#DewPoint">
! * GapLength = 49 hours included in the range of "Medium Gaps".
!
! T_Missing -- value missing
! T_Not_Applicable -- value was cleared. Do not count as missing, *but* print as a
!   regular missing value, e.g., '---'. Used, for example:
!   * 25-th hour wind direction. There is no daily value for WD.
!   * 25-th hour Present_weather and indicators. Ditto above.
!   * Broadband Aerosol Optical Depth (Wed Dec 12 14:48:31 2001)
!     BAOD is measured only during daylight.
! T_Undefined -- Value is undefined, given the context;
!   e.g., Dew point is undefined if RH == 0.
! T_Perpetual_Darkness -- Region is under complete darkness for some period
!   of the year. (Rs, Ra) == 0 for that period, which
!   prevents the computation of Eto and Ep.
! T_Accumulation -- resolution of an accumulation flag (Hourly precipitation)
!
! Space: used by SAMSON           @: Unset                               `:
!   !:                             A: used by SAMSON; Accumulation a:
!   ":                             B: used by SAMSON; Calibrated b:
!   #: Perpetual Darkness          C: used by SAMSON                   c:
!   $:                             D: used by SAMSON; Deleted d:
!   %:                             E: used by SAMSON; Estimated e:
!   &:                             F: used by SAMSON                 f:
!   ':                             G: used by SAMSON                 g:
!   (:                             H: used by SAMSON                 h:
!   ):                             I:                               i:
!   *: Not Applicable (N/A)        J:                               j:
!   +: Cumulative                  K:                               k:
!   ,:                             L:                               l:
!   -: Missing                     M: used by SAMSON                 m:
!   .:                             N:                               n:
!   /: Average                     O: ** do not use **           o:
!   0:                             P:                               p:
!   1:                             Q:                               q:
!   2:                             R: EarthInfo                     r:
!   3:                             S: SAMSON v 1.0                     s:

```

```

!   4:                T: SAMSON v 1.1                t:
!   5:                U: Unlimited                    u:
!   6:                V:                               v:
!   7:                W: Measured                    w:
!   8:                X:                               x:
!   9:                Y:                               y:
!   ::                Z: Cirroform                    z:
!   ;:                [:                               {:
!   <:                \:                               |:
!   =:                ]:                               }:
!   >:                ^:                               ~:
!   ?: used by SAMSON; Undefined    _:                DEL:

```

```

Character(Len=1), Parameter :: T_Unset      = '@'
Character(Len=1), Parameter :: T_Measured  = 'W'
Character(Len=1), Parameter :: T_Calibrated = 'B'
Character(Len=1), Parameter :: T_Deleted   = 'D'
Character(Len=1), Parameter :: T_Estimated = 'E' ! Estimated
Character(Len=1), Parameter :: T_Observed  = T_Measured ! Observed
Character(Len=1), Parameter :: T_Small_Gap = T_Estimated ! <A HREF="Gaps.f90#Fill_Small_Gaps">
Character(Len=1), Parameter :: T_Medium_Gap = T_Estimated ! <A HREF="Gaps.f90#Fill_Medium_Gaps">
Character(Len=1), Parameter :: T_Large_Gap  = T_Estimated ! <A HREF="Gaps.f90#Fill_Large_Gaps">
Character(Len=1), Parameter :: T_Missing    = '-' ! Missing data
Character(Len=1), Parameter :: T_EarthInfo  = 'R' ! EarthInfo data
Character(Len=1), Parameter :: T_SAMSONv10 = 'S' ! SAMSON version 1.0 files
Character(Len=1), Parameter :: T_SAMSONv11 = 'T' ! SAMSON version 1.1 files
Character(Len=1), Parameter :: T_Unlimited   = 'U' ! Unlimited in Visibility & Ceiling Height
Character(Len=1), Parameter :: T_SGU       = T_Unlimited ! Short-gap interpolation with Unlimited
Character(Len=1), Parameter :: T_SGC       = T_Estimated ! Short-gap interpolation with Cirroform
Character(Len=1), Parameter :: T_Cirroform = 'Z' ! Cirroform in Ceiling Height data
Character(Len=1), Parameter :: T_Undefined  = '?' ! Undefined
Character(Len=1), Parameter :: T_Perpetual_Darkness = '#'
Character(Len=1), Parameter :: T_Accumulation = 'A'

```

```

Character(Len=1), Parameter :: T_Average    = '/' ! Daily value is an average.
Character(Len=1), Parameter :: T_Cumulative = '+' ! Daily value is cumulative.
Character(Len=1), Parameter :: T_Not_Applicable = '*' ! Clear Daily value entry.

```

Type :: Val\_and\_Flag

```

! %s - Data_Source <A NAME="Data Source Flag">
!       T_EarthInfo, T_SAMSONv10, T_SAMSONv11,
!       T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness,
!       T_Estimated,
!       T_Unlimited, T_Cirroform,
!       T_Small_Gap (T_SGU, T_SGC), T_Medium_Gap, T_Large_Gap
!       T_Unset
!
! %v - real number value
!
! %f - flag associated with the value

```

```

!           %f(1:2): GHR, DNR, DHR
!           %f(1:9): Present_weather (f_OI)
!           %f(1:1): (ADME <blank>) Hourly_Precipitation_Flag (f_HP)
!           T_Accumulation,
!           EarthInfo precipitation flags

Character(Len=1) :: s    ! Data_Source
Real :: v
Character(Len=9) :: f    ! long enough to store "Present_weather"
End Type Val_and_Flag

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: Parameter_Type
! Store data in a vector array so that we can find gaps easier.
! Assume a virtual array: vData(ndays, Nhours)
!     Ndays = 365 or 366 == WFileInfo%Expected_Ndays
!
!   Ndays|  1    2  3  4  ...    24    25    Nhours = 25
!   -----+-----
!   1   |  1    2  3  4  ...    24    25
!   2   | 26   27 28 29 ...    49    50
!   :   |   :   :  :  :  ...    :     :
!  365 | 9101 :   :   :  ...  9124  9125
!  366 | 9126 .   .   .  ...  9149  9150
!
! Samson_v10(iv) <==> vData(doy, hh)
!   iv = (doy-1)*Nhours + hh
!
!   doy = (iv+Nhours-1) / Nhours
!   hh  = iv - (doy-1)*Nhours
!
! nbase = Hours_since_Jd0(yyyy)
! iv = (doy-1)*Nhours + hh + nbase
! Xparam(f_Ra)%Samson_v10(iv)%v = EHR
Type(Val_and_Flag), Dimension(:), Pointer :: Samson_v10 ! Dimension((jdl-jd0+1)*Nhours)
Type(Val_and_Flag), Dimension(:), Pointer :: Samson_v11 ! Dimension((jdl-jd0+1)*Nhours)
End Type Parameter_Type

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! Suite of files associated with one WBAN year.
Type :: Name_type
Character(MaxNamLen) :: Samson_v10 = ''
Character(MaxNamLen) :: Samson_v11 = ''
End Type Name_type

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! Store error messages.
Type :: Errors_type
Type(Errors_type), Pointer :: pNext => Null()
Character(Len=132) :: Error_Text = ''

```

```

End Type Errors_type

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! Elevation measured in millimeters.
Type :: ElevationBlock
  Integer :: Julian_Day = 0
  Real :: Elevation_meter = 0.0
End Type ElevationBlock

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! See Initialize_Node Read_SAMSON_Station_Notes for initialization
! * <A HREF="BinaryTree.f90#Initialize_Node">
Type :: Site_Info
  Integer :: item = 0 ! Unused. Remnant of Red_Black derived type.
  Logical :: Color ! Unused. Remnant. Which can assume RED or BLACK.
  Type(Site_Info), Pointer :: Parent => Null() ! Pointer to the parent.
  Type(Site_Info), Pointer :: pLeft => Null() ! Pointer to the left child.
  Type(Site_Info), Pointer :: pRight => Null() ! Pointer to the right child.

! Error messages linked list.
!!!!!! Type(Errors_type), Pointer :: pHead => Null() ! Head of linked list
!!!!!! Type(Errors_type), Pointer :: pTail => Null() ! Tail of linked list

Character(Len=05) :: WBAN = '' ! 94018
Character(Len=02) :: State = '' ! CO
Character(Len=50) :: Text = '' ! Boulder, CO

! Lat_radians = Latitude 'N' in radians;
! Lon_radians = Longitude 'W' in radians;
Type(Coords) :: Lat = Coords('',0,0) ! N 40 1
Type(Coords) :: Lon = Coords('',0,0) ! W 105 15
Real :: Lat_radians = 0.0
Real :: Lon_radians = 0.0

Real :: Elev = 0.0 ! 1634
Character(Len=10) :: TZ = '' ! +7(T)
Integer :: iTZ = 0 ! 7

Type(ElevationBlock), Dimension(7) :: Elev_Directives
Integer :: Nelev = 0
End Type Site_Info

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: Stat_Block
  Character(Len=80) :: Header = ''
  Integer :: k = 0
  Real :: M_k = Zero
  Real :: Q_k = Zero
  Real :: xmin = +Huge(Zero)
  Real :: xmax = -Huge(Zero)

```

```

    Real    :: xMean = -Huge(Zero)
    Real    :: xVariance = -Huge(Zero)
End Type Stat_Block

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: Year_Stats
    Integer :: k = 0
    Real    :: Total = Zero
End Type Year_Stats

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
Type :: Accum_type
    Integer :: ibeg = 0
    Integer :: iend = 0
    Real    :: Total = Zero
End Type Accum_type

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! Other variables.
Real,    Save :: Maximum_Horizontal_Visibility = Zero
Real,    Save :: Maximum_Ceiling_Height = Zero
Integer, Save :: Maximum_Text_Length = 0

! * <A HREF="Utils1.f90#Allocate_SAMSON_arrays">
Type(Stat_Block),    Dimension(0:f_end), Save :: Xp_ranges
Type(Parameter_Type), Dimension(0:f_end), Save :: Xparam
Type(Year_type),    Dimension(MinYear:MaxYear), Save :: Year_Data
Type(Name_type),    Dimension(MinYear:MaxYear), Save :: Year_Names
Type(Site_Info),    Pointer, Save :: pWBAN => Null()

! Observed data of various sorts, of various dimensions
! Obs_ppt -- observed precipitation [cm]

! *** WARNING: When reading additional data, consider if missing years
! ***          need to be filled for the purposes of analysis.
! ***          see <A HREF="Utils2.f90#Missing_years">

Type(Val_and_Flag), Dimension(:), Pointer, Save :: Obs_ppt
Logical, Save :: Have_ppt_Obs_daily_data = .False.
Logical, Save :: Have_ppt_Obs_hourly_data = .False.
Type(Accum_type), Dimension(:), Pointer, Save :: Accum_Samson => Null()
Type(Accum_type), Dimension(:), Pointer, Save :: Accum_EI => Null()

! <A HREF="Utils1.f90#Generate_File_names">
Character(MaxNamLen), Dimension(MinYear:MaxYear) :: name_r0 = ''
Character(MaxNamLen) :: name_met = ''
Character(MaxNamLen) :: name_txt = ''
Character(MaxNamLen) :: name_Daily_Evap = ''
Character(MaxNamLen) :: name_Daily_ppt = ''
Character(MaxNamLen) :: name_Hourly_ppt = ''

```

```

Logical, Dimension(MinYear:MaxYear), Save :: Issue_This_Year = .True.

! Format of the Daily values file (dvh) and
! Hourly values file (hvf).
Character(300), Save :: FMT_dvf = '()'
Character(500), Save :: FMT_hvf = '()'

! We want to analyze SAMSON(Hourly), EarthInfo(Hourly), and EarthInfo(daily)
Type(Year_Stats), Dimension(MinYear:MaxYear, 1:13), Save :: ppt_SH, ppt_EIH, ppt_EID

! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@
! @@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@===@@@

```

Contains

```

Subroutine ToTTY(TheMessage)

  Implicit None
  Character(Len=*) , Intent(In) :: TheMessage
  Character(Len=08) :: xdate = ' ' ! yyymmdd
  Character(Len=10) :: xtime = ' ' ! hhmmss.sss

  Call Date_and_time(Date = xdate, Time = xtime)

  ! E.g, 'Executing InitialSetUp at 13:29:00.216'
  !Write (ULog, 0010) '## '//Trim(TheMessage), xtime(1:2), xtime(3:4), xtime(5:10)
  Write (6, 9130) Trim(TheMessage), xtime(1:2), xtime(3:4), xtime(5:10)
  !Write (ULog, 0010) Trim(TheMessage), xtime(1:2), xtime(3:4), xtime(5:10)
9130 Format(1x, a, ' at ', 3(a, :, ':'))
  Call FLushAll()

End Subroutine ToTTY

Function IsNaN(x)

  ! IsNaN is a function to determine if x is Not-a-Number;

  Implicit None
  Real, Intent(In) :: x
  Logical          :: IsNaN

  IsNaN = (x /= x)
End Function IsNaN

```

```

Subroutine FLushAll()

  Implicit None

  Call FLush(6)
  Call FLush(ULog)
  Call Flush_LUNs()

End Subroutine FLushAll

Function IOF(Vfile) Result(uu)

  Implicit None
  Character(Len=*), Intent(In) :: Vfile
  Integer :: uu

  Logical :: Ok

  Call IOWrite(uu, Vfile, Ok=Ok)
  If (.Not. Ok) Then
    uu = 6
  End If
End Function IOF

End Module Global_Variables

```

## LinkedList

! Last change: LSR 16 May 2002 3:19 pm

Module Linked\_List

```
!Use Floating_Point_Comparisons
!Use IoSubs
!Use Module_Config
!Use mXgetargs
Use Date_Module
Use FileStuff
Use Global_Variables
Use Strings
```

Implicit None

```
! Store info in a binary tree since we want sorted output.
! See
! [] Cooper Redwine. 1995. Upgrading to Fortran 90.
! Springer Verlag; ISBN: 0387979956; pages 333-343.
```

```
Interface Add_Error
  Module Procedure Append_Text_to_List1
  Module Procedure Append_List2_to_List1
End Interface
```

Contains

Subroutine Save\_and\_Output(pHead, pTail, Tbuf, Jout)

```
Implicit None
Type(Errors_type), Optional, Pointer  :: pHead
Type(Errors_type), Optional, Pointer  :: pTail
Character(Len=*), Intent(In)  :: Tbuf
Integer, Optional, Intent(In)  :: Jout
```

Integer :: k

k = Len\_trim(Tbuf)

```
If (Present(pHead) .And. Present(pTail)) Then
  Call Add_Error(pHead, pTail, Tbuf(1:k))
End If
```

```
If (Present(Jout)) Then
  Write (Jout, '(1x, a)') Tbuf(1:k)
End If
```

```
End Subroutine Save_and_Output
```

```
Subroutine Append_Text_to_List1(plHead, plTail, Etext, qdebug)
```

```
! Add an item (Etext) to a linked list (pl*)
```

```
Implicit None
```

```
Type(Errors_type), Pointer :: plHead
```

```
Type(Errors_type), Pointer :: plTail
```

```
Character(Len=*), Intent(In) :: Etext
```

```
Logical, Optional, Intent(In) :: qdebug
```

```
Logical :: tdebug
```

```
Type(Errors_type), Pointer :: pprev
```

```
If (Present(qdebug)) Then
```

```
    tdebug = qdebug
```

```
Else
```

```
    tdebug = .False.
```

```
End If
```

```
If (tdebug) Then
```

```
    Call Prev_Node(pprev, plHead, plTail)
```

```
End If
```

```
!                               pTail --+
```

```
!                               |
```

```
!                               v
```

```
! pHead --> Node_1 +--> Node_2 +--> Node_3
```

```
!           Data_1 |   Data_2 |   Data_3
```

```
!           pNext --+   pNext --+   pNext --> disassociated
```

```
! Test whether the queue is currently empty (or not).
```

```
If (.Not. Associated(plHead)) Then
```

```
    ! Queue is empty: Create storage for the node; plHead points to it
```

```
    Allocate(plHead)
```

```
    plTail => plHead ! Point to the only node.
```

```
Else
```

```
    ! Queue is not empty: Create storage for new last node and point to it.
```

```
    Allocate(plTail%pNext)
```

```
    plTail => plTail%pNext
```

```
End If
```

```
plTail%Error_Text = Etext    ! Store data
```

```
plTail%pNext => Null()      ! Last node in the queue.
```

```
If (tdebug) Then
```

```
    Call Prev_Node(pprev, plHead, plTail)
```

```
End If
```

```
End Subroutine Append_Text_to_List1
```

```
Subroutine Append_List2_to_List1(p1Head, p1Tail, &  
    p2Head, p2Tail, qdebug)
```

```
! Append a second list (p2*) to the end of the first list (p1*).
```

```
Implicit None
```

```
Type(Errors_type), Pointer :: p1Head
```

```
Type(Errors_type), Pointer :: p1Tail
```

```
Type(Errors_type), Pointer :: p2Head
```

```
Type(Errors_type), Pointer :: p2Tail
```

```
Logical, Optional, Intent(In) :: qdebug
```

```
Logical :: tdebug
```

```
Type(Errors_type), Pointer :: pprev
```

```
If (Present(qdebug)) Then
```

```
    tdebug = qdebug
```

```
Else
```

```
    tdebug = .False.
```

```
End If
```

```
If (tdebug) Then
```

```
    Call Prev_Node(pprev, p1Head, p1Tail)
```

```
End If
```

```
! WARNING: [lsr] Tue 16 Oct 2001 18:12:59
```

```
! -----
```

```
! Short version. In the initial version I did not check if the  
! lists were non-null. The bug was very difficult to find because  
! it was not clear where the problem was located (it looked as an  
! event triggered by a missing file). The bug showed its ugly  
! little head in other parts of the program. It has taken  
! 9 hours to locate this bug.
```

```
!
```

```
! Take home message: Check that the tail points to the correct  
! node. And no matter how easy the pointer operation looks,  
! do some drawings.
```

```
!                               pTail ---+
```

```
!                               |
```

```
!                               v
```

```
! pHead --> Node_1 +--> Node_2 +--> Node_3
```

```
!           Data_1 |   Data_2 |   Data_3
```

```
!           pNext ---+   pNext ---+   pNext --> disassociated
```

```
If (Associated(p2Head)) Then
```

```

    If (Associated(p1Head)) Then
        ! Both lists are non-null. Append p2 to the end of p1
        p1Tail%pNext => p2Head
        p1Tail => p2Tail
    Else
        ! p1 is empty; Just transfer list p2 to p1
        p1Head => p2Head
        p1Tail => p2Tail
    End If
Else
    ! p2 is a null-list so there is nothing to append.
    ! p1 remains unchanged. There is nothing to do.
End If

If (tdebug) Then
    Call Prev_Node(pprev, p1Head, p1Tail)
End If
End Subroutine Append_List2_to_List1

```

```

Function Number_Of_Links(pHead) Result(NLinks)

    ! Determine the number of links in the list
    Implicit None
    Type(Errors_type), Pointer :: pHead
    Integer :: NLinks

    Type(Errors_type), Pointer :: ptmp

    ! temp node points to the first node of the queue
    ptmp => pHead

    NLinks = 0
    Do While (Associated(ptmp))
        NLinks = NLinks + 1
        ptmp => ptmp%pNext ! Point to next node
    End Do
End Function Number_Of_Links

```

```

Subroutine Prev_Node(pPrev, pHead, pTail, QMessage)

    ! Find the node before the pTail node.
    ! If Qmessage is present, a sanity check will be
    ! performed. This subroutine will abort
    ! if the test fails.

    Implicit None
    Type(Errors_type), Pointer :: pPrev

```

```

Type(Errors_type),      Pointer    :: pHead
Type(Errors_type),      Pointer    :: pTail
Character(Len=*), Optional, Intent(In) :: QMessage

Type(Errors_type), Pointer :: tmp
Logical :: pOK

If (Associated(pHead)) Then
    ! previous pointer has meaning only if pHead is associated.
    tmp => pHead
    pPrev => tmp

    Do While (Associated(tmp))
        pPrev => tmp
        tmp => tmp%pNext ! Point to next node
    End Do

    If (Present(Qmessage)) Then
        ! pTail and pPrev must point to the same place.
        pOK = Associated(pTail, pPrev)
        If (.Not. pOK) Then
            Write(ULog, 9130) Trim(Qmessage)
            Write(6, 9130) Trim(Qmessage)
9130         Format(1x, '?? Prev_Node: pTail /= pPrev at: ', a)
            Stop '?? pTail /= pPrev'
        End If
    End If
Else
    pPrev => Null()
End If
End Subroutine Prev_Node

Subroutine Release_Storage(pHead, pTail)

    ! Release list storage.

    Implicit None
    Type(Errors_type), Pointer :: pHead ! Intent(InOut)
    Type(Errors_type), Pointer :: pTail
    Type(Errors_type), Pointer :: current_item, previous_item

    previous_item => pHead
    current_item => pHead%pNext

    If (Associated(pHead)) Then
        Loop_Thru_Buckets: Do
            If (.Not. Associated(current_item)) Exit Loop_Thru_Buckets

            ! Delete "current_item" bucket. The new "current_item" will

```

```
        ! be the bucket following the deleted bucket.
        previous_item%pNext => current_item%pNext
        Deallocate(current_item)
        current_item => previous_item%pNext

    End Do Loop_Thru_Buckets
End If
Nullify(pHead)
Nullify(pTail)
End Subroutine Release_Storage

End Module Linked_List
```

## Make\_R0

```
! Last change: LSR 16 May 2002 3:19 pm
Program Make_R0
```

```
! History:
! * Fri Nov 16 13:53:42 2001
! Starting over.
!
! References:
! [1] (Short name: FAO)
! Crop evapotranspiration - Guidelines for computing crop water
! requirements - FAO Irrigation and drainage paper 56 by Richard G.
! Allen, Luis S. Pereira, Dirk Raes, and Martin Smith. Water Resources,
! Development and Management Service, FAO - Food and Agriculture
! Organization of the United Nations, Rome, 1998, ISBN 92-5-104219-5.
! The book can be found online at
! http://www.fao.org/docrep/X0490E/x0490e00.htm#Contents
! or locally at
! Docs\fao\index.html
!
! [2] Nicholas J. Higham. 1996. Accuracy and Stability of Numerical
! Algorithms. SIAM (Society for Industrial & Applied Mathematics).
! ISBN 0-89871-355-2.
!
! [3] Cooper Redwine. 1995. Upgrading to Fortran 90.
! Springer Verlag; ISBN: 0387979956; pages 333-343.
```

```
Use Date_Module
Use F2kCLI
Use Global_Variables
Use Process_Raw_Data
Use Setup
```

```
Implicit None
Character(Len=80) :: xexe = ''
Character(Len=80) :: ybegin_date, yend_date, q0tmp
Integer :: nargs ! Number of arguments in the command line
Integer :: nn
Real :: time_beg, time_end
```

```
Call CPU_Time(time_beg)
ybegin_date = Unix_Date()
```

```
Errors_Detected = .False.
Call Get_Command_Argument(0, xexe) ! Get application name.
```

```
! Decode/Process command line arguments.
nargs = Command_Argument_Count()
If (.False.) Then
```

```

    If (nargs /= 1) Then
        ! Remove possible path.
        nn = 1 + Index(xexe, DirDelim, Back=.True.)
        Write (6, "(3a)") ' Syntax: ', Trim(xexe(nn:)), &
            ' config_file_name'
        Errors_Detected = .True.
        Go To 9999
    End If
End If

! ULog is set first to "6" (the console) to write errors
! detected during input, then attached to a file to write
! errors detected during the processing of the files.
ULog = 6

!!! Call Get_Command_Argument(1, Config_file_name) ! e.g., mkmet.in
!!! If (Len_Trim(Config_file_name) == 0) Then
!!!     Config_file_name = 'MkMET.in'
!!! End If

Call InitialSetUp()
If (Errors_Detected) Go To 9999

Call Driver0()
If (Errors_Detected) Go To 9999

9999 Continue
Write (6, '(/lx,3a)') '### Check "', Trim(Log_file), '" for messages.'
If (Errors_Detected) Then
    Write (6, '(/lx,a)') '?? Errors detected.'
Else
    Write (6, '(/lx,a)') 'Program completed successfully.'
End If

yend_date = Unix_Date()
Call CPU_Time(time_end)

Call Elapsed_Time(time_end-time_beg, q0tmp)

Write (6, '(lx,a,a)') '## Started on ....: ', Trim(ybegin_date)
Write (6, '(lx,a,a)') '## Finished on ....: ', Trim(yend_date)
Write (6, '(lx,a,a)') '## Elapsed cpu time: ', Trim(q0tmp)

Write (ULog, '(///,a)') Repeat('_', 20)
Write (ULog, '(lx,a,a)') '## Started on ....: ', Trim(ybegin_date)
Write (ULog, '(lx,a,a)') '## Finished on ....: ', Trim(yend_date)
Write (ULog, '(lx,a,a)') '## Elapsed cpu time: ', Trim(q0tmp)

Close (ULog)
End Program Make_R0

```

## Precip

! Last change: LSR 4 Jun 2002 10:51 am

Module Precipitation\_module

```
Use Date_Module
Use Global_Variables
Use Reallocate_Module
Use Utils0
Use Utils1
Use Utils4
Use Utils5
!Use Floating_Point_Comparisons
Implicit None

Private
Public :: Print_HP
Public :: Process_Precipitation
Public :: Process_Precip_Records
Public :: Read_Daily_ppt
Public :: Read_Hourly_ppt
Public :: Reconcile_HP_samson_earthinfo
Public :: Standardize_ppt
Public :: Test_Accumulation
Public :: Yearly_Precip_Stats

Character(Len=*), Parameter :: Q_PW_Missing = 'm'
Character(Len=*), Parameter :: Q_PW_No_Rain = 'n'
Character(Len=*), Parameter :: Q_PW_Yes_Rain = 'r'
Character(Len=*), Parameter :: Q_Perhaps_Rain = '?'
Character(Len=*), Parameter :: X_Zero = 'z'
Character(Len=*), Parameter :: X_Missing = 'm'
Character(Len=*), Parameter :: X_Accum = 'a'
Character(Len=*), Parameter :: X_Delete = 'd'

Character(Len=*), Parameter :: Flag_Continue = 'c'
Character(Len=*), Parameter :: Flag_Done = 'y'
Character(Len=*), Parameter :: Flag_Error = 'e'

! The precision of the precipitation data is 1/100 inches,
! equivalent to 0.0254 cm.
Real, Parameter :: minimum_allocation = 0.02

Integer, Save :: dimV = 0
Real, Dimension(:), Pointer, Save :: Vmin=>Null(), Vmax=>Null(), Vval=>Null()
Character(Len=1), Dimension(:), Pointer, Save :: Vpw=>Null()
Logical, Dimension(:), Pointer, Save :: &
    Vmissing_hours=>Null(), Vzero_ppt=>Null(), vSelect=>Null(), &
    vOSC_1_to_4=>Null(), vOSC_5_to_10=>Null(), Vbounds=>Null(), &
```

```

Vused=>Null(), Vtemp1=>Null(), VpwRain=>Null(), VpwMissing=>Null()

Type(Val_and_Flag), Dimension(:), Pointer :: HP => Null() ! Hourly precipitation
Type(Val_and_Flag), Dimension(:), Pointer :: OI => Null() ! Observation_Indicator
Type(Val_and_Flag), Dimension(:), Pointer :: OSC => Null() ! Opaque Sky Cover

! Global debugging variables.
Logical, Save :: print_now = .False.

```

Contains

```

Subroutine Test_Accumulation(Xok)

! Observation Indicator 0 or 9 0 = Weather observation made.M
!                               9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
!
! Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
! Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
! Xparam(f_OI)%Samson_v10(iv)%s = data_source

Implicit None
Logical, Intent(Out) :: Xok

Xok = .True.
If (Xok) Then
  Write(6, 9130)
  Write(ULog, 9130)
9130  Format (1x, '## No accumulation test procedure.')
Else
  Write(6, 9130)
  Write(ULog, 9130)
End If

End Subroutine Test_Accumulation

```

```

Subroutine Read_Daily_ppt(Okay)

! The general form of this subroutine is stored in
! <A NAME="Read_Daily_ppt">
! <A HREF="Precip.f90#Read_Daily_ppt">

Implicit None
Logical, Intent(Out) :: Okay

! The general form of the file is:

```

```

!!! 123456789 <-- column number
!!! !Some comment
!!!
!!!
!!!
!!! Station FARGO WSO AP          Parameter      Evap          % Coverage      42
!!! PO Code ND                   Latitude      N46:55:31      Begin M/Yr  04/1963
!!! Stn ID 2859                   Longitude    W096:48:40      End M/Yr    09/1980
!!! County CASS                   Elevation    900             # Record Years 18
!!! ----- Prcp (in) -----
!!! 1963   Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec   Annual
!!! 1      ---   ---   ---   ---   ---   ---   9    ---   ---   8    ---   ---
!!! 2      ---   ---   ---   ---   9    ---   8    ---   ---   7    ---   ---
!!! 3      ---   ---   ---   ---   12   ---   6    ---   ---   7    ---   ---
!!! 4      ---   ---   ---   ---   7    ---   8    ---   ---   8    ---   ---
!!! 5      ---   ---   ---   ---   7    ---   8    ---   ---   6    ---   ---
!!! 6      ---   ---   ---   3    10   ---   3    ---   ---   4    ---   ---
!!! 7      ---   ---   ---   6    15   ---   7    ---   ---   3    ---   ---
!!! 8      ---   ---   ---   2    12   ---   8    ---   ---   3    ---   ---
!!! 9      ---   ---   ---   2    10   ---   9    ---   ---   4    ---   ---
!!! 10     ---   ---   ---   3    12   ---   7    ---   ---   3    ---   ---
!!! 11     ---   ---   ---   4    5    ---   4    ---   ---   4    ---   ---
!!! 12     ---   ---   ---   ---   6    ---   7    ---   ---   6    ---   ---
!!! 13     ---   ---   ---   4    ---   ---   7    ---   ---   7    ---   ---
!!! 14     ---   ---   ---   5    4    ---   3    ---   ---   5    ---   ---
!!! :
!!! 29     ---   ---   ---   ---   6    ---   9    ---   ---   5    ---   ---
!!! 30     ---   ---   ---   ---   7    ---   7    ---   ---   2    ---   ---
!!! 31     ---   ---   ---   ---   9    ---   7    ---   ---   4    ---   ---
!!!
!!! Total   ---   ---   ---   82   218  ---   201  ---   ---   121  ---   ---
!!! Extrm   ---   ---   ---   9    15   ---   13   ---   ---   8    ---   ---
!!! ^L (sometimes)

```

```
! Ignore blank lines, lines with "^L", lines with a leading "!"
```

```
!
```

```
! Year : The range is MinYear:MaxYear
```

```
! '---': denotes missing data
```

```

Character(Len=*),          Parameter :: Zheader = 'Prcp (in)'
Character(Len(T_EarthInfo)), Parameter :: Zsource = T_EarthInfo
Real,                      Parameter :: Zfac = inches__to__cm
! The array Obs_ppt has already been initialized to T_Unset.

```

```

Character(Len=100) :: xbuf
Character(Len=50)  :: xid, yflag
Integer :: ios, n1, n2, jj, f0, f1, uin, jerr, ierror
Integer :: npos, beg_col, end_col, kk, ip
Integer :: yyyy, mm, dd, hh, iv, nobis
Logical  :: in_range, file_was_open
Real     :: minV, maxV, yobs, y_original

```

```

Integer, Parameter :: Daily_hour = 25
Integer, Dimension(:), Pointer :: Days_in_Month

nobs = 0
ierror = 0
Okay = .False.
Have_ppt_Obs_daily_data = .False.

! f0 points to the character after the last DirDelim of name_Daily_ppt, therefore
! name_Daily_ppt(f0:) contains only the name of the input file. This makes
! messages more readable.
! I changed my mind (24 Jan 2002 2:30 pm). In case of problems,
! I do not want to hunt for the file.
f0 = 1 ! + Index(name_Daily_ppt, DirDelim, Back=.True.)
f1 = Len_trim(name_Daily_ppt)

Call ToTTY('Read_Daily_ppt: File: '//name_Daily_ppt(f0:f1))

Call IORead(uin, name_Daily_ppt, jerr, file_was_open)
If (.Not. file_was_open) Then
  Write (ULog, *) '## Read_Daily_ppt: did not find ', name_Daily_ppt(f0:f1)
  ! Not finding the file is NOT an error. Just return.
  Go To 9999 ! Jump to End-of-Subroutine
Else
  Write (ULog, *) '## File: ', name_Daily_ppt(f0:f1)
End If

! Range of acceptable values.
minV = -Huge(minV)
maxV = +Huge(maxV)

Read_One_Line: Do
  Read (uin, '(a)', iostat = ios) xbuf
  If (ios /= 0) Exit ! End-Of-File or Error

  ! Skip lines until we find the beginning of the data block, e.g.,
  ! ----- Prcp (in) -----
  If (xbuf(1:10) /= '-----') Cycle Read_One_Line

  ! Found the beginning of the data block.
  ! Find the name in the line, e.g., 'Prcp (in)'
  ! n1 points to the first character of the name
  ! n2-1 delimits the name. Note that the name may contain blanks.

  n1 = Verify(xbuf, '- ') ! Find first non '-' or blank, "P" for the example.
  n2 = Index(xbuf(n1:), '-') !
  If (n2 > 0) n2 = n2 + n1 - 1 - 1
  xid = xbuf(n1:n2) ! e.g., 'Prcp (in)'

  n1 = Index(xid, '(') + 1

```

```

n2 = Index(xid, ')') - 1 ! xid(n1:n2) == 'in'

!! Make sure the units are what we expect.
!!If (xid(n1:n2) /= Zunits) Then

! Make sure the header is what we expect.
If (Trim(xid) /= Zheader) Then
! This is a fatal error.
Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
Write (ULog, 9130) Zheader, Trim(xid)
9130 Format (lx, '?? expecting "', a, '"', found "', a, '"')
ierror = ierror + 1
Go To 9999 ! Jump to End-of-Subroutine
End If

!!! The line following "-----" identifies the year and the columns:
!!! 1963 Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Annual

Read (uin, '(a)', iostat = ios) xbuf
If (ios /= 0) Then
Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
Write (ULog, *) ' Expecting a "year" line'
ierror = ierror + 1
Go To 9999 ! Jump to End-of-Subroutine
End If

! Get year.
Read (xbuf(1:4), '(i4)', iostat = ios) yyyy
If (ios /= 0) Then
Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
Write (ULog, *) ' Line: "', Trim(xbuf), '"'
Write (ULog, *) ' The first four characters Do not represent a year.'
ierror = ierror + 1
Go To 9999 ! Jump to End-of-Subroutine
End If

in_range = (MinYear <= yyyy) .And. (yyyy <= MaxYear)
If (.Not. in_range) Then
Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
Write (ULog, *) ' The year ', yyyy, ' is not between ', &
MinYear, ' and ', MaxYear
ierror = ierror + 1
Go To 9999 ! Jump to End-of-Subroutine
End If

Days_in_Month => Number_of_Days_in_Month(yyyy)

! Now load the data block (the next 31 lines) in the array.
Do jj = 1, 31
Read (uin, '(a)', iostat = ios) xbuf

```

```

If (ios /= 0) Then
  Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
  Write (ULog, *) ' Error: Expecting month-day ', jj
  ierror = ierror + 1
  Go To 9999 ! Jump to End-of-Subroutine
End If

```

```

! Get the day
npos = 3
Read (xbuf(1:npos), '(i3)') dd
npos = npos + 1 ! Skip over the day of the month.

```

```

! Sanity check: jj must be equal to dd
If (jj /= dd) Then
  Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
  Write (ULog, *) ' Error: jj /= dd; jj, dd == ', jj, dd
  ierror = ierror + 1
  Go To 9999 ! Jump to End-of-Subroutine
End If

```

!!!	1963	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Annual
!!!	1	---	---	---	---	---	---	9	---	---	8	---	---	---
!!!	2	---	---	---	---	9	---	8	---	---	7	---	---	---
!!!	10	0	0T	0T	2	0	0	136A	0T	0T	0	5	0	0
!!!	21	---	389X	252	318	353	257	318	363	435	570	488	370	---
!!!	27	0.00T	0.00	0.02	0.12	0.00	0.00	0.00	0.30	0.00	0.00	0.00	0.74	---
!!!	28	0.00	0.06	1.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00T	0.00T	0.00
!!!	29	0.00	---	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
!!!	30	0.00	---	0.30	0.00	0.00	0.00	0.00	0.00	0.00T	0.00	0.00	0.01	---
!!!	31	0.00	---	0.83	---	0.00	---	0.00	3.12	---	0.00	---	0.54	---

```

!!!
!!! Notes:
!!! 1. First number: day of the month, 1 - 31
!!! 2. Next 12 numbers: January, February, ..., December
!!! 3. Missing Data denoted With '---'
!!! 4. Trailing 'X' means extrapolated value
!!! 5. '0T' - trace amount
!!! 6. Trailing 'A' - Accumulated value. Treat as "missing data".

```

```

Month_Loop: Do mm = 1, 12
  Call GetQwordCols(xbuf, npos, beg_col, end_col, jerr)
  If (jerr /= 0) Then
    Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
    Write (ULog, *) ' Line: "', Trim(xbuf), '"'
    Write (ULog, *) ' contains less than twelve months, mm == ', mm
    ierror = ierror + 1
    Go To 9999 ! Jump to End-of-Subroutine
  End If

```

```

! Is this a valid day for the month? Note that the table

```

```

! has 31 entries for each month, regardless of the number
! of days in the month.
! Example: 1963-Feb has 28 days. If we call ymdh_to_iv
! for 1963-Feb-29, the iv returned is for 1963-March-01
! (i.e., one day after 1963-Feb-28). Ditto for 30-day months.
If (dd > Days_in_Month(mm)) Then
  ! Bogus table filler date. Go to next month.
  Cycle Month_Loop
End If

```

```

! Compute iv address of the daily value.
Call ymdh_to_iv(yyyy, mm, dd, Daily_hour, iv)

```

```

If (xbuf(beg_col:end_col) == '---') Then
  ! Missing data
  Obs_ppt(iv) = Val_and_Flag(T_Missing, Missing_Data, '')
  Cycle Month_Loop
End If

```

```

kk = end_col
ip = Verify(xbuf(kk:kk), Set='0123456789')
! ip > 0 Then xbuf(kk:kk) is a non-digit.
If (ip == 0) Then
  yflag = ''
Else
  yflag = xbuf(kk:kk)
  kk = kk - 1
End If

```

```

!!!           Select Case (xbuf(kk:kk))
!!!           Case ('A')
!!!             ! Accumulated value. Treat as missing Data.
!!!             Cycle Month_Loop
!!!
!!!           Case ('X')
!!!             ! Extrapolated value. Remove the trailing 'X'.
!!!             xbuf(kk:kk) = ''
!!!
!!!           Case ('T')
!!!             ! Some of the precipitation Data is recorded as '0T', i.e., trace.
!!!             ! From http://airquality.tor.ec.gc.ca/natchem/precip/summary95.html -
!!!             ! # TRACE SAMPLING PERIODS: Number of trace sampling periods in the
!!!             ! summary period. A trace sampling period is defined as a sampling
!!!             ! period when both the standard gauge and the collector reported
!!!             ! trace precipitation depths, i.e., less than the instrumental
!!!             ! detection limits. These detection limits are 0.2 millimeters (mm)
!!!             ! for the standard gauge and 0.1 mm for the collector.
!!!             ! We will replace '0T' With half of the detection limit for the
!!!             ! standard gauge, i.e., 0.2 mm/2 = 0.1 mm = 0.01 cm.
!!!
!!!

```

```

!!!           xbuf(beg_col:end_col) = '0.1'
!!!
!!!           Case Default
!!!             ! Do nothing.
!!!             *** Undefined flag.
!!!           End Select
!!!         End If

Read(xbuf(beg_col:kk), *, iostat = ios) y_original
If (ios /= 0) Then
  Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
  Write (ULog, *) '   Line: "', Trim(xbuf), &
    '"', substr: '"', xbuf(beg_col:kk), '"'
  Write (ULog,9150) yyyy,mm,dd
9150   Format(1x, '   Not a number at ', i4, 2('-',i2.2))
  ierror = ierror + 1
  Go To 9999 ! Jump to End-of-Subroutine
End If

!! Before converting to appropriate units, make sure
!! this value is not a "flag" value.
!!If (y_original <= -8000) Then
!   ! some flag value.
!   Cycle Month_Loop
!End If

! Convert input units to whatever
yobs = y_original * Zfac

! value in range?
in_range = ((minV <= yobs) .And. (yobs <= maxV))
If (.Not. in_range) Then
  Write (ULog, *) '?? Read_Daily_ppt: Input file: ', name_Daily_ppt(f0:f1)
  Write (ULog, *) &
    '   Value not in range, ignored v,minV,maxV : ', &
    yobs, minV, maxV
  Cycle Month_Loop
End If

! Make sure we will not overwrite the previous good data.
! * <A HREF="Utils1.f90#Allocate_SAMSON_arrays">
!   Initialization: Obs_ppt = Val_and_Flag(T_Unset, Missing_Data, '')
!
! * well, not all previous data is good. I just added "T_Missing"
!   to the test below because some of the previous data read was
!   flagged as missing.

Select Case(Obs_ppt(iv)%s)
Case(T_Unset, T_Missing)
  nobs = nobs + 1

```

```

Obs_ppt(iv) = Val_and_Flag(Zsource, yobs, yflag)

Case Default
! We have a previous value.
If ((Abs(Obs_ppt(iv)%v - yobs) < Eps0) .And. &
(Obs_ppt(iv)%f == yflag)) Then
! Same value. Do nothing.
Else
Write (ULog, *) '?? Read_Daily_ppt: duplicate ', &
yyyy, mm, dd, Daily_hour
Write (ULog, *) ' Old: ', Obs_ppt(iv)
Write (ULog, *) ' New: ', Zsource, yobs, yflag
End If
End Select

End Do Month_Loop
End Do
End Do Read_One_Line

! Finished reading the file. At this stage anything unset
! will be declared missing. We will look only the 25h slot,
! i.e., the daily values.
Do iv = 25, Ubound(Obs_ppt,1), NHours
If (Obs_ppt(iv)%s == T_Unset) Then
! At this stage "unset" implies Zero precipitation.
Obs_ppt(iv) = Val_and_Flag(Zsource, Zero, '')

Else If (Obs_ppt(iv)%s == T_Missing) Then
! Nothing to do.

Else If (Obs_ppt(iv)%s == Zsource) Then
If (Abs(Obs_ppt(iv)%v - Missing_Data) < Eps0) Then
Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
Write (ULog, 9170) '%s ok but missing %v:', &
iv, yyyy, mm, dd, hh, &
Obs_ppt(iv)%s, Obs_ppt(iv)%v, Trim(Obs_ppt(iv)%f)
9170 Format(1x, a, ': Obs(', i0, i5, 2('-',i2.2), i3, 'h) == ', &
a, 1x, lpg14.6, ' "', a, '"')
End If
End If
End Do

!!! ! Verify there are no missing values
!!! Do iv = 25, Ubound(Obs_ppt,1), NHours
!!! If (Obs_ppt(iv)%s /= Zsource) Then
!!! Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
!!! Write (ULog, 6130) iv, jyyyy, jmm, jdd, jhh
!!!6130 Format(1x, '?? Read_Daily_ppt: ', &
!!! 'Obs_ppt(iv) is missing ', i7, 1x, &

```

```

!!!          i4, '-', i2.2, '-', i2.2, i3, 'h')
!!!          Write (ULog, *) ' ', Obs_ppt(iv)
!!!          End If
!!!          End Do

! All data was read. Close the file ...
9999 Continue
If (file_was_open) Call IOClose(uin)

! On Output:
!   Obs_ppt%s == T_Missing, Zsource
!   Obs_ppt%f == yflag: { X T A <blank> }

Have_ppt_Obs_daily_data = ((nobs > 0) .And. (ierror == 0))
Okay = (ierror == 0)

End Subroutine Read_Daily_ppt

Subroutine Read_Hourly_ppt(Okay)

! The general form of this Subroutine is stored in
!   <A NAME="Read_Hourly_ppt">
!   <A HREF="Precip.f90#Read_Hourly_ppt">

Implicit None
Logical, Intent(Out) :: Okay

! The general form of the file is:
! <A HREF="e:\5\3met\Docs\td3240.txt#Hourly Precipitation Data">

!!! 123456789 <-- column number
!!! !Some comment
!!!
!!!                                HOURLY
!!!
!!! Station      JACKSON WSFO AIRPORT          % Coverage    99
!!! PO Code      MS                Latitude     N32:19:11   Begin M/Yr 07/1963
!!! Station ID   4472              Longitude   W090:04:39 End M/Yr   12/2000
!!! County       RANKIN            Elevation   310        # Record Years 38
!!! ----- Prcp (in.) -----
!!!          +000    +100    +200    +300    +400    +500
!!! 01/01/1961  0100    0 g     .     .     .     .     .
!!!             0700    .     .     .     .     .     .
!!!             1300    .     .     .     .     .     .
!!!             1900    .     .     .     .     .     .
!!! 07/01/1963  0100    --- m   ---   ---   ---   ---   ---
!!!             0700    ---   ---   ---   ---   ---   ---
!!!             1300    ---   ---   ---   ---   ---   ---

```

```

!!!      1900      ---      ---      ---      ---      ---      ---
!!! 07/16/1963 0100      ---      ---      ---      ---      ---      ---
!!!      0700      ---      ---      ---      ---      ---      ---
!!!      1300      --- M      .      .      .      .      .
!!!      1900      0.01      0.02      .      .      .      .
!!! 07/17/1963 0100      .      .      .      .      .      .
!!!      0700      .      .      .      .      .      .
!!!      1300      .      .      .      .      .      0.08
!!!      1900      0.01      .      .      .      .      .
!!! 10/31/1967 0100      .      .      .      .      .      .
!!!      0700      .      .      .      .      .      .
!!!      1300      .      .      --- a      ---      ---      ---
!!!      1900      ---      ---      ---      ---      ---      --- A
!!! 11/01/1967 0100      --- ,      ---      ---      ---      ---      ---
!!!      0700      ---      ---      ---      ---      ---      ---
!!!      1300      ---      ---      0.15 A      .      0.01      .
!!!      1900      0.01      .      .      .      .      .
!!! 10/03/1975 0100      .      .      .      .      .      .
!!!      0700      .      .      .      .      .      .
!!!      1300      .      --- a      ---      ---      ---      ---
!!!      1900      0.93 A      .      .      .      .      .
!!! 10/04/1975 0100      .      .      .      .      .      .
!!!      0700      .      .      .      .      .      .
!!!      1300      .      --- a      ---      ---      ---      ---
!!!      1900      0.41 A      --- a      ---      ---      ---      ---
!!! 10/05/1975 0100      0.09 A      .      .      .      .      .
!!!      0700      .      .      .      .      .      .
!!!      1300      .      --- a      ---      ---      ---      ---
!!!      1900      0.19 A      --- a      ---      ---      ---      ---
!!! 10/06/1975 0100      0.03 A      .      .      .      .      .
!!!      0700      .      --- a      ---      ---      ---      ---
!!!      1300      0.05 A      --- a      ---      ---      ---      ---
!!!      1900      0.20 A      --- a      ---      ---      ---      ---
!!! 10/07/1975 0100      0.02 A      --- a      ---      ---      ---      ---
!!!      0700      0.04 A      .      .      .      .      .
!!!      1300      .      .      .      .      .      .
!!!      1900      .      --- a      ---      ---      ---      ---
!!! 10/08/1975 0100      0.04 A      .      .      .      .      .
!!!      0700      .      .      .      .      .      .
!!!      1300      .      .      .      .      .      .
!!!      1900      .      .      .      .      .      .

```

```

! Ignore all lines except lines of form above.
! '---': denotes missing Data

```

```

Character(Len=*),      Parameter :: Zunits = 'Prcp (in.)'
Character(Len(T_EarthInfo)), Parameter :: Zsource = T_EarthInfo
Real,      Parameter :: Zfac = inches_to_cm
! The array Obs_ppt has already been initialized to T_Unset.

```

```

Character(Len=100) :: xbuf
Character(Len=50)  :: xid
Character(Len=1)   :: d1, d2, yflag
Integer :: ios, n1, n2, jline, f0, f1, uin, ierror, nobS
Integer :: npos, kk, k0, k1, kf, i__max
Integer :: yyyy, mm, dd, hh, iv, iv_base
Integer :: jyyyy, jmm, jdd, jhh
Integer :: id_hour, h_val, ierrors
Logical  :: ok, in_range, are_eq
Logical  :: file_was_open
Logical  :: year_range, month_range, day_range
Real     :: minV, maxV, yobs, y_original
Type(Val_and_Flag) :: Current_Obs

Type :: Track_Date
      Integer :: last_iv = 0
      Character(Len(X_Zero)) :: last_action
End Type Track_Date

Integer, Parameter :: Max_BackTracks = 2
Type(Track_Date), Dimension(0:Max_BackTracks) :: pDate
Integer :: iLev

Character(Len(X_Zero)) :: current_action

! Days_in_Month Contains the days in the month for pyear.
! Rather than calling Number_of_Days_in_Month each time
! we decode a date, just Call it when yyyy /= pyear
Integer, Dimension(:), Pointer :: Days_in_Month
Integer :: pyear

nobS = 0
Okay = .False.
ierrors = 0
Have_ppt_Obs_hourly_data = .False.

! f0 points to the Character after the last DirDelim of name_Hourly_ppt, therefore
! name_Hourly_ppt(f0:) Contains Only the name of the input file. This makes
! messages more readable.
! I changed my mind (24 Jan 2002 2:30 pm). In Case of problems,
! I Do not want to hunt for the file.
f0 = 1 ! + Index(name_Hourly_ppt, DirDelim, Back=.True.)
f1 = Len_trim(name_Hourly_ppt)

Call ToTTY('Read_Hourly_ppt: File: '//name_Hourly_ppt(f0:f1))

Call IORead(uin, name_Hourly_ppt, ierror, file_was_open)
If (.Not. file_was_open) Then
  Write (ULog, *) '## Read_Hourly_ppt: did not find ', name_Hourly_ppt(f0:f1)
  ! Not finding the file is NOT an error. Just Return.

```

```

    Go To 9999 ! Jump to End-of-Subroutine
Else
    Write (ULog, *) '## File: ', name_Hourly_ppt(f0:f1)
    ! Call ToTTY('Read_Hourly_ppt: found '//name_Hourly_ppt(f0:f1))
End If

! Range of acceptable values.
minV = -Huge(minV)
maxV = +Huge(maxV)

pyear = MinYear
Days_in_Month => Number_of_Days_in_Month(pyear)

iLev = 0 ! 0:Max_BackTracks
pDate(iLev)%last_iv = 1
pDate(iLev)%last_action = X_Zero

Look_for_Date: Do
    Read (uin, '(a)', iostat = ios) xbuf
    If (ios /= 0) Exit ! End-Of-File or Error

    ! Skip comments.
    If (xbuf(1:1) == '!') Cycle Look_for_Date

    ! Skip lines With leading blanks.
    If (xbuf(1:1) == ' ') Cycle Look_for_Date

    ! If the line is a begin-of-Data-Block, verify it Contains
    ! the right units.
    ! ----- Prcp (in.) -----
    If (xbuf(1:10) == '-----') Then

        ! Found the beginning of the Data Block.
        ! Find the name in the line, e.g., 'Prcp (in.)'
        ! n1 points to the first Character of the name
        ! n2-1 delimits the name. Note that the name may contain blanks.

        n1 = Verify(xbuf, '- ') ! Find first non '-' or blank, "P" for the example.
        n2 = Index(xbuf(n1:), '-') !
        If (n2 > 0) n2 = n2 + n1 - 1 - 1
        xid = xbuf(n1:n2) ! e.g., 'Prcp (in.)'

        ! Make sure the header is what we expect.
        If (Trim(xid) /= Zunits) Then
            ! This is a fatal error.
            Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
            Write (ULog, 9130) Zunits, Trim(xid)
            Format (1x, '?? Expecting "', a, '"', found '"', a, '"')
            ierrors = ierrors + 1
            Go To 9999 ! Jump to End-of-Subroutine
        End If
    End If
End Do

```

9130

```

    End If
    Cycle Look_for_Date
End If

! Try to decode a date.
! Skip lines that Do not start With a valid date "mm/dd/yyyy"
! 1234567890 <- Columns
! 01/01/1961 ...

npos = 10
Read (xbuf(1:npos), '(i2,a1,i2,a1,i4)', iostat = ios) mm, d1, dd, d2, yyyy
If (ios /= 0) Cycle Look_for_Date

! Update (If needed) the number of days in the month for the current year.
If (pyear /= yyyy) Then
    pyear = yyyy
    Days_in_Month => Number_of_Days_in_Month(pyear)
End If

year_range = (MinYear <= yyyy) .And. (yyyy <= MaxYear)
month_range = (1 <= mm) .And. (mm <= 12)
If (month_range) Then
    day_range = (1 <= dd) .And. (dd <= Days_in_Month(mm))
Else
    day_range = .False.
End If

ok = (month_range) .And. (d1 == '/') .And. &
    (day_range) .And. (d2 == '/') .And. (year_range)

If (.Not. ok) Then
    Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
    Write (ULog, *) '?? It looked like a date: ', &
        xbuf(1:npos), yyyy, mm, dd
    Cycle Look_for_Date
End If

! We have a date.

! # [lsr] 5 Apr 2002 11:13 am
! * careful with changes. I took two days (and many false
! starts) to get this code.
!
! Case 1:
! Consider last_iv == 24 (last hour set) and now we are
! reading the next day: iv_base == 25. We should not call
! Xupdate because the 25th hour is not going to be altered.
! See records (4) and (5) below.

```

```

!
! Case 2:
! If we just read a duplicated record, we may need to
! fix previous entries, e.g. see below.
!
! Case 3:
! 5 Apr 2002 11:01 am: I thought that a solitaire T_Missing
! was a bad omen of problems but for Jackson, MS, when
! records were merged, a single "missing" was produced.
! This is the correct behaviour.
!
! Note that the records below generate, among other things,
! a solitaire missing value at 1963-07-16 2h.
! A solitaire missing value is not part of a larger
! 'm' .. 'M' block.
!
! Record iv iv
! Number h=1 h=24 Date Data
! -----
! 1. 22776 22799 1963-07-01 'm' --- --- --- ---
! 2. 23151 23174 1963-07-16 --- 'M' . . .
! 3. 22776 22799 1963-07-01 0g . . .
! 4. 22851 22874 1963-07-04 . . 'm' --- ---
! 5. 22876 22899 1963-07-05 --- 'M' . 3 .
! 6. 23151 23174 1963-07-16 . 'm' --- 'M' .
! 7. 23226 23249 1963-07-19 . 1 . . .
!
! #0. Initialize:
!     iLev = 0
!     pDate(iLev) = (last_iv=1, last_action=X_zero)
!
! #1. Read record (1): 1963-07-01 1h :: iv = 22776;
!     iv_base = iv - 1
!     1.1 If (pDate(iLev)%last_iv < (iv_base - 1)) Then
!         Apply: pDate(iLev)%last_action
!         from: pDate(iLev)%last_iv
!         to..: iv_base - 1
!     End If
!     1.2 Process all 24 hours of current_date; iv = 22799
!     1.3 pDate(iLev=0) = (last_iv=(iv+2)122799+2, last_action=X_missing)
!         iv is a date at 24h
!         iv + 2 is next day's date at 1h
!
! #2. Read record (2): 1963-07-16 1h :: iv = 23151
!     2.1 Ditto 1.1
!     2.2 Process all 24 hours of current_date; iv = 23174
!     2.3 pDate(iLev=0) = (last_iv=23174+2, last_action=X_zero)
!
! #3. Read record (3): 1963-07-01 1h :: iv = 22776
!     3.1 If (pDate(iLev)%last_iv < (iv_base - 1)) -- False

```

```

!       3.2 Process all 24 hours of current_date; iv = 22799
!       3.3 If (iv < pDate(iLev)%last_iv)
!           22799 < 23176 -- True
!           Then: iLev = iLev + 1 ( == 1)
!       3.4 pDate(iLev=1) = (last_iv=22799+2, last_action=X_zero)
!
! #4. Read record (4): 1963-07-04 1h :: iv = 22851
!       4.1 If (pDate(iLev)%last_iv < (iv_base - 1))
!           22801 < 22851-2 ? True.
!           Then apply Xupdate ...
!       4.2 Process all 24 hours of current_date; iv = 22874
!       4.3 If (iv < pDate(iLev)%last_iv)
!           22874 < 22876 ? -- False
!       4.4 pDate(iLev=1) = (last_iv=22874+2, last_action=X_missing)
!       4.5 Do
!           If iLev == 0 Exit
!           If (pDate(iLev-1)%last_iv <= pDate(iLev)%last_iv) Then
!               pDate(iLev-1) = pDate(iLev)
!               iLev = iLev - 1
!           Else
!               Exit
!           End If
!       End Do
!       -- iLev unchanged.
!
! #5. Read record (5): 1963-07-05 1h :: iv = 22876
!       5.1 If (pDate(iLev)%last_iv < (iv_base - 1))
!           22876 < 22876-2 ? False.
!       5.2 Process all 24 hours of current_date; iv = 22899
!       5.3 If (iv < pDate(iLev)%last_iv)
!           22899 < 22876 ? -- False
!       5.4 pDate(iLev=1) = (last_iv=22899+2, last_action=X_zero)
!       5.5 Do
!           If iLev == 0 Exit
!           If (pDate(iLev-1)%last_iv <= pDate(iLev)%last_iv) Then
!               pDate(iLev-1) = pDate(iLev)
!               iLev = iLev - 1
!           Else
!               Exit
!           End If
!       End Do
!
! #6. Read record (6): 1963-07-16 1h :: iv = 23151
!       6.1 If (pDate(iLev)%last_iv < (iv_base - 1))
!           22901 < 23151-2 ? True.
!           Then apply Xupdate ...
!       6.2 Process all 24 hours of current_date; iv = 23174
!       6.3 If (iv < pDate(iLev)%last_iv)
!           23174 < 22901 ? -- False
!       6.4 pDate(iLev=1) = (last_iv=23174+2, last_action=X_zero)

```

```

!       6.5 Do
!           If iLev == 0 Exit
!           If (pDate(iLev-1)%last_iv <= pDate(iLev)%last_iv) Then
!               !-- 23176 <= 23176 ? True
!               pDate(iLev-1) = pDate(iLev)
!               !-- pDate(0) = (last_iv=23174+2, last_action=X_zero)
!               iLev = iLev - 1
!               !-- iLev = 0
!           End If
!       End Do
!
! #7. Read record (7): 1963-07-16 1h :: iv = 23226
!       7.1 If (pDate(iLev)%last_iv < (iv_base - 1))
!           22901 < 23176-2 ? True.
!           Then apply Xupdate ...
!       7.2 Process all 24 hours of current_date; iv = 23249
!       7.3 If (iv < pDate(iLev)%last_iv)
!           23249 < 23176 ? -- False
!       7.4 pDate(iLev=0) = (last_iv=23249+2, last_action=X_zero)
!       7.5 Do
!           If iLev == 0 Exit
!           If (pDate(iLev-1)%last_iv <= pDate(iLev)%last_iv) Then
!               Else
!                   Exit
!               End If
!           End Do
!
! The rest of this line (and the next 3 lines) compose an hourly record.
Call ymdh_to_iv(yyyy, mm, dd, hh=1, iv=iv_base)
iv_base = iv_base - 1
hh = 0

If (pDate(iLev)%last_iv < (iv_base - 1)) Then
    Call Xupdate(ifrom=pDate(iLev)%last_iv, &
                ito=iv_base - 1, &
                The_action=pDate(iLev)%last_action, &
                Zsource=Zsource)
End If

! A record is split over four lines (see sample below).
Read_One_Record: Do jline = 1, 4      ! Read and process four lines.

! xbuf Contains the line to process.
Read (xbuf(13:16), '(i4)', iostat = ios) id_hour
If (ios /= 0) Then
    Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
    Write (ULog, *) '   Line: ', Trim(xbuf), '"', '"', xbuf(13:16), '"'
    Write (ULog, *) '   Expecting Id_hour 0100, 0700, 1300, 1900'
    Write (ULog, *) '   Date = ', yyyy, mm, dd

```

```

        ierrors = ierrors + 1
        Go To 9999 ! Jump to End-of-Subroutine
End If

!           1           2           3           4           5           6           7           8
! 123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
! 01/01/1961 0100      0 g      .      .      .      .      .
!           0700      1      1      .      .      2      2      4
! 123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
!           1           2           3           4           5           6           7           8
! 123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
!           1300      --- m      ---      ---      ---      ---      ---
!           1900      --- M      .      .      .      .      .
!
!       Jline  id_hour
!           1      0100 h
!           2      0700 h
!           3      1300 h
!           4      1900 h

h_val = 100 * ((jline-1)*6 + 1)
If (h_val /= id_hour) Then
    Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
    Write (ULog, *) '    Expecting ', h_val, ' found ', id_hour, &
        '; Date = ', yyyy, mm, dd
    ierrors = ierrors + 1
    Go To 9999 ! Jump to End-of-Subroutine
End If

! Get 6 values from the line
! k = 1 .. 6
! the kth number begins in column (k+1)*10 : 20, 30, 40, ...
! the kth number ends   in column (k+1)*10+3: 23, 33, 43, ...
! the kth flag   is     in column (k+1)*10+5: 25, 35, 45, ...
By_Col: Do kk = 1, 6
    k0 = 10 * (kk+1) ! number begins here
    k1 = k0 + 3      ! number ends here
    kf = k0 + 5      ! flag column

    hh = hh + 1
    iv = iv_base + hh
    yflag = xbuf(kf:kf)

!Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
!ok = (yyyy == jyyyy) .And. (mm == jmm) .And. &
!     (dd == jdd) .And. (hh == jhh)
!If (.Not. ok) Then
!   Write(ULog,0102) yyyy, mm, dd, hh, jyyyy, jmm, jdd, jhh
!   Write(6, 0102) yyyy, mm, dd, hh, jyyyy, jmm, jdd, jhh
!   Stop '?? Read_Hourly_ppt: Date mismatch'

```

```

!End If
9150 Format(1x, '?? Read_Hourly_ppt: Date mismatch ', &
      2(i4, '-', i2.2, '-', i2.2, i3, 'h; '))

! Perhaps the combination " 7 M" is invalid (i.e., non-missing
! value with a M-flag). However, EarthInfo documentation does not
! explicitly preclude. Certainly SAMSON was not afraid to use it.
! We will make sure that if we have a number, it is decoded.
Select Case(xbuf(k0:k1))
Case(' . ')
! EarthInfo Documentation: Times having no reported events
! (presumably zero precipitation) simply have decimal point
! place-markers in their cells.
y_original = Zero
yobs = Zero

Case(' ---')
! Missing value display value.
y_original = Missing_Data
yobs = Missing_Data

Case Default
Read(xbuf(k0:k1), *, iostat = ios) y_original
If (ios /= 0) Then
Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
Write (ULog, *) ' Line: ', Trim(xbuf), &
      '"', substr: "'", xbuf(k0:k1), ''
Write (ULog,9170) yyyy,mm,dd
9170 Format(1x, ' Not a number at ', i4, 2('-',i2.2))
ierrors = ierrors + 1
Go To 9999 ! Jump to End-of-Subroutine
End If

!!! !! Before converting to appropriate units, make sure
!!! !! this value is not a "flag" value.
!!! !! If (y_original <= -8000) Then
!!! !! ! some flag value.
!!! ! Cycle By_Col
!!! !End If

! Convert input units to whatever
yobs = y_original * Zfac

! value in range?
in_range = ((minV <= yobs) .And. (yobs <= maxV))
If (.Not. in_range) Then
ierrors = ierrors + 1
Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
Write (ULog, *) &
      ' Value not in range, ignored v,minV,maxV : ', &

```

```

        yobs, minV, maxV
    Write (ULog, *) '    Date = ', yyyy, mm, dd
    Cycle By_Col
End If
End Select

! Process current point (observation)
If (Abs(yobs - Missing_Data) < Eps0) Then
    ! yobs == Missing_Data
    nobs = nobs + 1
    Current_Obs = Val_and_Flag(T_Missing, yobs, '')
    If ((yflag == 'm') .Or. (yflag == ' ')) Then
        current_action = X_Missing
    Else If (yflag == 'M') Then
        current_action = X_Zero
    Else If (yflag == 'a') Then
        current_action = X_Accum
    Else If (yflag == 'A') Then
        ! An accumulation run with missing accumulation value
        ! Fake an accumulation run with zero accumulation.
        current_action = X_Zero
!!! ***
!!! *** 11 Apr 2002 11:11 am
!!! *** Warning. At this time we are ignoring
!!! ***    accumulation runs in EarthInfo files
!!! ***
!!! *** <A NAME="ignoring accumulation note #1">
!!! *** <A HREF="Precip.f90#ignoring accumulation note #1">
!!! *** <A HREF="Precip.f90#ignoring accumulation note #2">
!!! ***
        Current_Obs = Val_and_Flag(T_Missing, yobs, '')
    Else If (yflag == 'd') Then
        current_action = X_Delete
    Else If (yflag == 'D') Then
        current_action = X_Zero
    Else If (yflag == ',') Then
        current_action = X_Missing
    Else
        ! Issue a warning:
        ierrors = ierrors + 1
        Write (ULog, *) '    Line: ', Trim(xbuf), &
            ', substr: ', xbuf(k0:k1), ''
        Write (ULog, *) &
            '    yobs == Missing_Data and yflag "', &
            yflag, '" not {<blank> mM aA dD , }'
        Write (ULog, *) '    Date = ', yyyy, mm, dd, hh
        Cycle By_Col
    End If

```

```

Else
  ! yobs /= Missing_Data
  nobs = nobs + 1
  Current_Obs = Val_and_Flag(Zsource, yobs, '')
  If ((yflag == 'g') .Or. (yflag == ' ') .Or. (yflag == 'E')) Then
    current_action = X_Zero
  Else If (yflag == 'A') Then
    current_action = X_Zero

!!! ***
!!! *** 11 Apr 2002 11:11 am
!!! *** Warning. At this time we are ignoring
!!! ***      accumulation runs in EarthInfo files
!!! ***
!!! *** <A NAME="ignoring accumulation note #2">
!!! *** <A HREF="Precip.f90#ignoring accumulation note #1">
!!! *** <A HREF="Precip.f90#ignoring accumulation note #2">
!!! ***

      Current_Obs = Val_and_Flag(T_Missing, yobs, '')
Else
  ! Issue a warning:
  ierrors = ierrors + 1
  Write (ULog, *) '   Line: "', Trim(xbuf), &
    '"', substr: '"', xbuf(k0:k1), '"'
  Write (ULog, *) &
    '   yobs /= Missing_Data and yflag "', &
    yflag, '" not {<blank> g A E}'
  Write (ULog, *) '   Date = ', yyyy, mm, dd, hh
  Cycle By_Col
End If

!Else ! E I
!   Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
!   Write (ULog, *) '   Line: "', Trim(xbuf), '"'
!   Write (ULog, *) '   Do not know how to handle flag "', yflag, '"'
!   Write (ULog, *) '   Date = ', yyyy, mm, dd
!   ierrors = ierrors + 1
!   Go To 9999 ! Jump to End-of-Subroutine
End If

! * <A HREF="Utils1.f90#Allocate_SAMSON_arrays">
! Initialization: Obs_ppt = Val_and_Flag(T_Unset, Missing_Data, '')

! Make sure we will not overwrite previous good (non-missing) data.
! (input) Old%s == T_Unset, T_Missing, Zsource
! Current%s ... == T_Missing, Zsource
! (output) Old%s == T_Missing, Zsource
!
! (input/output) %f == yflag: { <blank> }

If ((Obs_ppt(iv)%s == T_Unset) .Or. (Obs_ppt(iv)%s == T_Missing)) Then

```

```

! Obs_ppt(iv) is unset or missing.
! Current value just replaces the old value.
Obs_ppt(iv) = Current_Obs

Else If (Obs_ppt(iv)%s == Zsource) Then
! We have a previous observation.

If (Current_Obs%s == T_Missing) Then
! If current_obs is missing we keep the old obs.
! Nothing to do.
Else
! If either of the values is Zero, keep the non-zero value.
If (Abs(Obs_ppt(iv)%v) < Eps0) Then
! Obs is zero. Replace it with the current observation.
Obs_ppt(iv) = Current_Obs
Else If (Abs(Current_Obs%v) < Eps0) Then
! Current observation is zero.
! Keep the old value (i.e., nothing to do).
Else
! Else current_obs has data: Unless both obs are identical,
! we have a problem.
are_eq = (Obs_ppt(iv)%s == Current_Obs%s) .And. &
(Abs(Obs_ppt(iv)%v - Current_Obs%v) < Eps0) .And. &
(Obs_ppt(iv)%f == Current_Obs%f)
If (.Not. are_eq) Then
ierrors = ierrors + 1
Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
Write (ULog, 9190) 'Duplicate Old:', iv, jyyyy, jmm, jdd, jhh, &
Obs_ppt(iv)%s, Obs_ppt(iv)%v, Trim(Obs_ppt(iv)%f)
Write (ULog, 9190) ' New:', iv, jyyyy, jmm, jdd, jhh, &
Current_Obs%s, Current_Obs%v, Trim(Current_Obs%f)
End If
End If
End If
End If
End Do By_Col

9190 Format(1x, a, ': Obs(', i0, i5, 2('-',i2.2), i3, 'h) == ', &
a, 1x, lpg14.6, ' "', a, '"')

! Read next line of the hourly Block.
If (jline /= 4) Then
Read (uin, '(a)', iostat = ios) xbuf
If (ios /= 0) Then
Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
Write (ULog, *) ' EOF: Expecting line ', jline+1, &
' / 4 of hourly Block ', yyyy, mm, dd
ierrors = ierrors + 1
Go To 9999 ! Jump to End-of-Subroutine
End If

```

```

      End If
End Do Read_One_Record

! Finished 24-hour record.

If (iv < pDate(iLev)%last_iv) Then
! We moved backwards in time when we read this observation.
! Push date onto stack.
If (iLev >= Max_BackTracks) Then
  Write (ULog, *) '?? Read_Hourly_ppt: Input file: ', name_Hourly_ppt(f0:f1)
  Write (ULog, *) '  Increase Max_BackTracks == ', Max_BackTracks
  Write (ULog, *) '  Date: ', yyyy, mm, dd
  ierrors = ierrors + 1
  Go To 9999 ! Jump to End-of-Subroutine
End If
  iLev = iLev + 1
End If

! iv is a date at 24h; iv + 2 is next day's date at 1h;
! 2 == Nhours - 24 + 1
pDate(iLev)%last_iv = iv + Nhours - 24 + 1
pDate(iLev)%last_action = current_action

! Can we return to the main time continuum?
Do
  If (iLev == 0) Exit ! The main time continuum.

  ! We can rejoin the previous continuum when the date of
  ! the current continuum is greater than the date of the
  ! previous continuum.
  !
  ! pDate(iLev)%last_iv -- stardate of the current continuum.
  ! pDate(iLev-1)%last_iv -- stardate of the previous continuum.
  ! pDate(0)%last_iv -- stardate of the main time continuum.

  If (pDate(iLev-1)%last_iv > pDate(iLev)%last_iv) Exit
  pDate(iLev-1) = pDate(iLev)
  iLev = iLev - 1
End Do
End Do Look_for_Date

i__max = Ubound(Obs_ppt,1)
Do iv = 1, i__max

! Skip the 25th hour
If (Modulo(iv,25) == 0) Then
  Cycle
End If

```

```

! At this stage:
! %s == T_Unset, T_Missing, Zsource
! %f == yflag: { <blank> }
!
! * <A HREF="Utils1.f90#Allocate_SAMSON_arrays">
! Initialization: Obs_ppt = Val_and_Flag(T_Unset, Missing_Data, '')

If (Obs_ppt(iv)%s == T_Unset) Then
! At this stage "unset" implies Zero precipitation.
Obs_ppt(iv) = Val_and_Flag(Zsource, Zero, '')

Else If (Obs_ppt(iv)%s == T_Missing) Then
! Nothing to do.

Else If (Obs_ppt(iv)%s == Zsource) Then
If (Abs(Obs_ppt(iv)%v - Missing_Data) < Eps0) Then
Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
Write (ULog, 9190) '%s ok but missing %v:', &
iv, jyyyy, jmm, jdd, jhh, &
Obs_ppt(iv)%s, Obs_ppt(iv)%v, Trim(Obs_ppt(iv)%f)
End If
End If
End Do

9999 Continue
! All Data was Read. Close the file ...
If (file_was_open) Call IOClose(uin)

! On Output:
! Obs_ppt%s == T_Missing, Zsource
! Obs_ppt%f == yflag: { <blank> }

Have_ppt_Obs_hourly_data = ((nobs > 0) .And. (ierror == 0))
Okay = (ierrors == 0)

End Subroutine Read_Hourly_ppt

Subroutine Xupdate(ifrom, ito, The_action, Zsource)

! Auxiliary routine for Read_Hourly_ppt
Implicit None
Integer, Intent(In) :: ifrom, ito
Character(Len=*), Intent(In) :: The_action
Character(Len=*), Intent(In) :: Zsource

Integer :: iv

```

```

Do iv = ifrom, ito
  If (Modulo(iv,25) == 0) Then
    ! Skip 25-th hour of the day.
    Cycle

    !Else If (Obs_ppt(iv)%s == Zsource) Then
    !   ! Leave things the way they are.

  Else If (Obs_ppt(iv)%s == T_Unset) Then
    ! This entry has not been set, so we can overwrite it
    ! without a second thought.
    If (The_action == X_Zero) Then
      Obs_ppt(iv) = Val_and_Flag(Zsource, Zero, '')
    Else If (The_action == X_Missing) Then
      Obs_ppt(iv) = Val_and_Flag(T_Missing, Missing_Data, '')
    End If

  Else If (Obs_ppt(iv)%s == T_Missing) Then
    If (The_action == X_Zero) Then
      Obs_ppt(iv) = Val_and_Flag(Zsource, Zero, '')
    End If
  End If
End Do
End Subroutine Xupdate

Subroutine Yearly_Precip_Stats(Header, Jout)

  Implicit None
  Character(Len=*), Intent(In) :: Header
  Integer, Optional, Intent(In) :: Jout

  Integer :: uu
  Integer :: yyyy, mm, dd, hh, jv0, jv1, iv, hlen
  Logical :: okay
  Integer, Dimension(:), Pointer :: Days_in_Month
  Real :: sum_SH, sum_EIH, sum_EID

  ! ppt_SH - Sums computed using SAMSON hourly values (1h-24h)
  ! ppt_EIH - Sums computed using EarthInfo hourly values (1h-24h)
  ! ppt_EID - Sums computed using EarthInfo daily values (stored in 25h)
  !   The EarthInfo hourly and daily values came from different files.

  If (Present(Jout)) Then
    uu = Jout
  Else
    uu = ULog
  End If

  Write (uu, *)

```

```

hlen = Len_trim(Header)
Write (uu, 9130) '## Yearly_Precip_Stats: ', Header(1:hlen)
Write (uu, 9130) ' Station WBAN Number: ', Trim(pWBAN%WBAN), &
    ', ', Trim(pWBAN%Text)
9130 Format (1x, 2a, :, 2a)

ppt_SH = Year_Stats(0, Zero)
ppt_EIH = Year_Stats(0, Zero)
ppt_EID = Year_Stats(0, Zero)
HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

By_Year: Do yyyy = MinYear, MaxYear

    If (Year_Data(yyyy)%SAMSON_v10 == 0) Then
        ! Year is missing.
        Cycle By_Year
    End If

    Write (uu, 9150) yyyy, 'SAMSON(h)', 'EarthInfo(h)', 'EarthInfo(d)'
9150 Format (/ , 1x, i4, &
        T15, a13, &
        T30, a13, &
        T45, a12)

Days_in_Month => Number_of_Days_in_Month(yyyy)

sum_SH = Zero      ! Annual sum of SAMSON hours
sum_EIH = Zero     ! Annual sum of EarthInfo hours
sum_EID = Zero     ! Annual sum of EarthInfo summary of the day

By_Month: Do mm = 1, 12
    Call ymdh_to_iv(yyyy, mm, dd=01, hh=01, iv=jv0)
    Call ymdh_to_iv(yyyy, mm, dd=Days_in_Month(mm), hh=Nhours, iv=jv1)

    By_Hour: Do iv = jv0, jv1
        If (Modulo(iv,25) /= 0) Then

            Select Case(HP(iv)%s)
            Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
                ! Do nothing.
            Case Default
                ppt_SH(yyyy,mm)%k = ppt_SH(yyyy,mm)%k + 1
                ppt_SH(yyyy,mm)%Total = ppt_SH(yyyy,mm)%Total + HP(iv)%v
            End Select

            Select Case(Obs_ppt(iv)%s)
            Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
                ! Do nothing.
            Case Default
                ppt_EIH(yyyy,mm)%k = ppt_EIH(yyyy,mm)%k + 1

```

```

        ppt_EIH(yyyy,mm)%Total = ppt_EIH(yyyy,mm)%Total + Obs_ppt(iv)%v
    End Select

    Else
        ! 25th hour: Daily totals -- only for EarthInfo.
        Select Case(Obs_ppt(iv)%s)
        Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
            ! Do nothing.
        Case Default
            ppt_EID(yyyy,mm)%k = ppt_EID(yyyy,mm)%k + 1
            ppt_EID(yyyy,mm)%Total = ppt_EID(yyyy,mm)%Total + Obs_ppt(iv)%v
        End Select
    End If
End Do By_Hour

! Print this month's sums
Write (uu, 9170) mm, Month_Table(mm)(1:3), &
    ppt_SH(yyyy,mm)%Total, ppt_SH(yyyy,mm)%k, &
    ppt_EIH(yyyy,mm)%Total, ppt_EIH(yyyy,mm)%k, &
    ppt_EID(yyyy,mm)%Total, ppt_EID(yyyy,mm)%k
9170 Format (lx, 3x, i2, '-', a3, T15, f8.2, '(', i0, ')', &
    T30, f8.2, '(', i0, ')', &
    T45, f8.2, '(', i0, ')')

! Update yearly sums
sum_SH = sum_SH + ppt_SH(yyyy,mm)%Total
sum_EIH = sum_EIH + ppt_EIH(yyyy,mm)%Total
sum_EID = sum_EID + ppt_EID(yyyy,mm)%Total
End Do By_Month

! Because of roundoff errors and the initial precision of the
! precipitation data, differences less than 0.02 cm are not
! significant. Flag everything else.
If (Abs(sum_SH-sum_EID) <= ppt_Eps) Then
    ! Annual sum of SAMSON hours == Annual sum of EarthInfo summary of the day
    okay = .True.
Else
    ! Numbers are different. It is ok if the SAMSON
    ! number is the largest.
    !
    ! See <A HREF="global.f90#fuzzy comparisons">
    ! A > B :: Abs(A-B) >= Eps0 .And. Eps0 < (A-B)
    okay = (ppt_Eps < (sum_SH-sum_EID))
End If

If (okay) Then
    Write (uu, 9190) 'Annual', sum_SH, sum_EIH, sum_EID
Else
    Write (uu, 9190) 'Annual', sum_SH, sum_EIH, sum_EID, '??(SAMSON < EI)'
End If

```

```

9190     Format (1x, 3x, a, T15, f8.2, T30, f8.2, T45, f8.2, :, 7x, a)
      End Do By_Year

End Subroutine Yearly_Precip_Stats

Subroutine Standardize_ppt(Vdata, AccumList, Xok)

! Vdata -- loop only through 1-24 hours
! - Xparam(f_HP)%Samson_v10 ! Hourly Precipitation
! - Obs_ppt -- EarthInfo
!
! AccumList -- list of accumulation runs in Vdata
!
! This routine:
! #1. resolves runs of 'M' or 'D' into individual
!     points, each identified as T_Missing.
! #2. Identifies accumulation runs. Stores the beginning and
!     ending times, and the accumulated precipitation in
!     the allocatable array AccumList

! <A NAME="Standardize_ppt"> <A HREF="Precip.f90#Standardize_ppt">
! See <A HREF="0notes.txt#Note_27">
!
! Text from <A HREF="e:\5\3met\Docs\samson_format.txt#$1">
! DATA FORMAT--HOURLY PRECIPITATION
!
! It stands to reason that for most hours the non-occurrence of
! precipitation is prevalent. Therefore, in order to save space in
! the original digital file, there are entries only for:
!
! 1. The first day and hour of each month where observations were
!    taken even if no precipitation occurred during that month.
!
! 2. Hours with precipitation > zero.
!
! 3. Beginning and ending hours of missing periods.
!
! 4. Beginning and ending hours of accumulating periods.
!
! 5. Beginning and ending periods of deleted data.
!
! 6. First and last day of each month where the required charts
!    or forms never were received or processed at NCDC.
!
! The actual precipitation data value: The data value portion is a
! six-digit integer. Units are inches and hundredths. Range =
! 000000-099999. 000000 will be used only on the first hour of each

```

```

! month unless there is precipitation during that hour, in which case
! the measured value will be provided. On other days during the
! month without precipitation, no entry will be made. 099999
! indicates that the value is unknown.
!
! Hourly Precipitation Flag:
!
! A      Accumulated period and amount. An accumulated period
!        indicates that the precipitation amount is correct, but
!        the exact beginning and ending times are only known to
!        the extent that the precipitation occurred sometime
!        within the accumulation period. Begin accumulation data
!        value will always be 099999. *** [LSR] Not always so.
!                                     *** See example below.
!
! D      Deleted Flag. Beginning and ending of a deleted period.
!        A deleted value indicates that the original data were
!        received, but were unreadable or clearly recognized as
!        noise.
!
! M      Missing Flag. (Beginning and ending of a missing
!        Period.) A missing flag indicates that the data were
!        not received. This flag appears on the first and last
!        day of each month for which data were not received or
!        not processed by NCDC. Prior to 1984 a missing period
!        was recorded as "00000M" at the beginning and ending
!        hours. If precipitation occurred during the last hour
!        of the missing period, the second M appears with a non-
!        zero value. Beginning in 1984 the beginning and ending
!        hours of the missing period are recorded as "099999M".
!
! b      Blank. No Flag needed.
!
! Examples:
!
! The precipitation accumulation from 1st month day 02 to 2nd month
! day 04:
!
!      01      0002      0500      00030b
!                                     1000      099999A      Accumulation begins
!      02      0001      0100      099999A      Accumulation continues
!                                     0004      000390A      Accumulation ends
!
! Accumulated precipitation for 1 month only:
!
!      01      0002      1000      099999A      Accumulation begins
!                                     0031      2400      000320A      Accumulation ends
!
! Accumulated, deleted, and missing precipitation data through months
! 01 and 02:

```

```

!
!      01      0001      0100      000000b      First record of the
!                                     month
!      0002      1100      099999A      Accumulation begins
!      02      0001      0100      099999A      Accumulation continues
!                                     1400      000630A      Accumulation ends
!                                     1500      099999D      Deleted data begins
!      02      0028      1300      099999D      Deleted data ends
!                                     1400      099999M      Missing data
!                                     2400      099999M      Missing data
!

```

! Required precipitation charts or forms were never received at NCDC:

```

!
!      01      0001      0100      099999M      Missing data
!      0031      0100      099999M
!      02      0001      0100      099999M
!      0028      0100      099999M

```

! 19 Feb 2002 3:52 pm; it would appear (from 12842\_75.txt above)  
! that the starting entry is '0A', followed by 'nothing'. The cumulative  
! value is flagged with 'A'.

! That is not the complete story. Example: 94018: Boulder, CO

```

!
! yyyy mm dd hh  HP(i)%v  %f  %s
! -----
! 1982  6  1  1    0.0000  A  S  ! Begin
! 1982  6 30 24    5.5876  A  S  ! End
!
! 1984  4  6 10    999999  A  -  ! Begin
! 1984  5  1  1    999999  A  -  ! Continue
! 1984  5  1  9    6.0960  A  S  ! End
!
! 1988  1  1 16    999999  A  -  ! Begin
! 1988  1 31 24    1.0160  A  S  ! End

```

```

Implicit None
Type(Val_and_Flag), Dimension(:), Intent(InOut) :: Vdata
Type(Accum_type),   Dimension(:), Pointer      :: AccumList
Logical,            Intent(Out)               :: Xok

```

```

Integer :: col_beg, col_end, iv, i__max, ierr, jv
Integer :: jyyyy, jmm, jdd, jhh
Logical  :: in_accum_gap
Logical  :: in_delete_gap
Logical  :: in_missing_gap
Integer  :: ngaps, Gap_dim

```

```

If (Associated(AccumList)) Then
  Gap_dim = Ubound(AccumList,1)

```

```

Else
  Gap_dim = 30
  Allocate(AccumList(Gap_dim))
End If

! Observation Indicator 0 or 9 0 = Weather observation made.
!                               9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
!
!   Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
!   Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
!   Xparam(f_OI)%Samson_v10(iv)%s = data_source

ierr = 0
ngaps = 0
in_accum_gap = .False.
in_delete_gap = .False.
in_missing_gap = .False.

i__max = Ubound(Vdata,1)
By_Hours: Do iv = 1, i__max

  ! Do not check the 25th hour
  If (Modulo(iv,25) == 0) Then
    Cycle By_Hours
  End If

  If (Vdata(iv)%f == 'D') Then
    ! Delete gap (start or end).
    ! Delete this point.
    ! Toggle in_delete_gap flag.
    Vdata(iv)%v = Missing_Data
    Vdata(iv)%s = T_Missing ! T_Deleted
    in_delete_gap = (.Not. in_delete_gap)
    Cycle By_Hours
  Else If (in_delete_gap) Then
    Vdata(iv)%v = Missing_Data
    Vdata(iv)%s = T_Missing ! T_Deleted
    Vdata(iv)%f = 'D'
    Cycle By_Hours
  End If

  If (Vdata(iv)%f == 'M') Then
    ! Missing gap (start or end).
    ! Delete this point.
    ! Toggle in_missing_gap flag.
    If (in_missing_gap) Then
      ! From the SAMSON documentation above: If precipitation occurred
      ! during the last hour of the missing period, the second M appears
      ! with a non-zero value. Guess what? This is the "second M".

```

```

! (%v > 0) and (%v /= Missing_Data) ?
If ((Vdata(iv)%v > Zero) .And. (Abs(Vdata(iv)%v-Missing_Data) > Eps0)) Then
! Non zero value -- Keep it.
Vdata(iv)%s = T_Estimated
Vdata(iv)%f = ''
Else
Vdata(iv)%v = Missing_Data
Vdata(iv)%s = T_Missing
End If
Else
Vdata(iv)%v = Missing_Data
Vdata(iv)%s = T_Missing
End If
in_missing_gap = (.Not. in_missing_gap)
Cycle By_Hours
Else If (in_missing_gap) Then
Vdata(iv)%v = Missing_Data
Vdata(iv)%s = T_Missing
Vdata(iv)%f = 'M'
Cycle By_Hours
End If

If (Vdata(iv)%f == 'A') Then

! An accumulation point.
! Determine type: begin, continue, or end.

If (Abs(Vdata(iv)%v) < Eps0) Then
! If value is Zero, then this is the start of a new gap.
! If in_accum_gap == .T., then we have a problem.
If (in_accum_gap) Then
Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
Write(ULog,*) '?? Standardize_ppt: runaway gap at ', jyyyy, jmm, jdd, jhh
Write(6, *) '?? Standardize_ppt: runaway gap at ', jyyyy, jmm, jdd, jhh
Stop '?? Standardize_ppt: runaway gap'
End If
! Start of a new gap
in_accum_gap = .True.
col_beg = iv
Else If (Vdata(iv)%s == T_Missing) Then
! %s == T_Missing iff %v == 99999.
! This may be a "begin" or "continuation".
If (in_accum_gap) Then
! A continuation. Nothing to do.
Cycle By_Hours
Else
! A beginning. (A very delicate time.)
! Start of a new gap
in_accum_gap = .True.
col_beg = iv

```

```

End If
Else
! We better be within a gap.
If (.Not. in_accum_gap) Then
  Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
  Write(ULog,*) '?? Standardize_ppt: Begin-of-gap missing ', &
    jyyyy, jmm, jdd, jhh
  Write(6, *) '?? Standardize_ppt: Begin-of-gap missing ', &
    jyyyy, jmm, jdd, jhh
  Call FLushAll()
  Stop '?? Standardize_ppt: Begin-of-gap missing'
End If

! End of gap.
col_end = iv

Select Case(Vdata(col_end)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
! Not a valid number: This is a severe error.
  Call iv_to_ymdh(col_end, jyyyy, jmm, jdd, jhh)
  Write(ULog, 9130) jyyyy, jmm, jdd, jhh
9130  Format (lx, &
        '?? Standardize_ppt: ', &
        'Total amount missing for ', &
        i4, '-', i2.2, '-', i2.2, i3, 'h')
        ierr = ierr + 1
        in_accum_gap = .False.
        Cycle By_Hours
End Select

! The gap run just ended. Store it.
If (ngaps >= Gap_dim) Then
  Gap_dim = 2 * Gap_dim + 10
  AccumList => Reallocate_Accum_type(AccumList, Gap_dim)
End If
ngaps = ngaps + 1
AccumList(ngaps)%ibeg = col_beg
AccumList(ngaps)%iend = col_end
AccumList(ngaps)%Total = Vdata(col_end)%v

!Call iv_to_ymdh(col_beg, jyyyy, jmm, jdd, jhh)
!Call iv_to_ymdh(col_end, kyyyy, kmm, kdd, khh)
!Write (ULog, 9125) jyyyy, jmm, jdd, jhh, kyyyy, kmm, kdd, khh
9150  Format(/, lx, '## Standardize_ppt gap from ', &
        i4, '-', i2.2, '-', i2.2, i3, 'h to ', &
        i4, '-', i2.2, '-', i2.2, i3, 'h')

in_accum_gap = .False.

! Henceforth, for this array, accumulation data will be

```

```

! retrieved from AccumList.
Acc_Loop: Do jv = col_beg, col_end
  If (Modulo(jv,25) == 0) Then
    Cycle Acc_Loop
  End If
  Vdata(jv)%s = T_Missing
  Vdata(jv)%f = 'A'
End Do Acc_Loop
Vdata(col_end)%v = Zero
End If

Else If (in_accum_gap) Then
  ! We are in an accumulation run. Do not touch anything!
  Cycle By_Hours

End If
End Do By_Hours

If (in_accum_gap) Then
  ! This is a mistake. After an Accumulation gap
  ! we must have the amount measured.
  ierr = ierr + 1
  Call iv_to_ymdh(col_beg, jyyyy, jmm, jdd, jhh)
  Write (6, 9170) jyyyy, jmm, jdd, jhh
  Write (ULog, 9170) jyyyy, jmm, jdd, jhh
9170  Format(lx, '?? Standardize_ppt: (EOF) Unresolved Accumulation gap ', &
      'starting on ', i4, '-', i2.2, '-', i2.2, i3, 'h')
  Call FLushAll()
  Stop '?? Stopping in Standardize_ppt: (EOF) Unresolved Accumulation gap.'
End If

! Trim AccumList to exact size.
AccumList => Reallocate_Accum_type(AccumList, ngaps)
Xok = (ierr == 0)

End Subroutine Standardize_ppt

Function Reallocate_Accum_type(pOld, Nnew) Result(pNew)
  Implicit None
  Type(Accum_type), Dimension(:), Pointer :: pOld, pNew
  Integer, Intent(In) :: Nnew
  Integer :: nold, ierr

  Allocate(pNew(1:Nnew), STAT=ierr)
  If (ierr /= 0) Stop "?? Reallocate_Accum_type error"

```

```

If (.Not. Associated(pOld)) Return

nold = Min(Size(pOld), Nnew)
pNew(1:nold) = pOld(1:nold)
Deallocate(pOld)
End Function Reallocate_Accum_type

```

```

Subroutine Process_Precip_Records(Xok)

```

```

Implicit None
Logical, Intent(Out) :: Xok

```

```

Integer :: iv
Integer :: i__max, ierr
Logical :: are_ok, id_ok, v_ok, okay
Integer :: nsamson, igap, n0, n1, nx
Integer :: col_beg, col_end
Real    :: Total_ppt

```

```

! Observation Indicator 0 or 9 0 = Weather observation made.
!                               9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
!
!   Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
!   Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
!   Xparam(f_OI)%Samson_v10(iv)%s = data_source

```

```

Xok = .False.

```

```

ierr = 0
HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

```

```

! Pass 1: Process 'M', 'D', Zsource
! Needed only if we have Observed Hourly data.

```

```

If (Have_ppt_Obs_hourly_data) Then
  i__max = Ubound(HP,1)
  Pass_1: Do iv = 1, i__max

```

```

    ! Skip the 25th hour
    If (Modulo(iv,25) == 0) Then
      Cycle Pass_1
    End If

```

```

    !! Skip any 'accumulation' records. We will do them later.
    !If ((HP(iv)%f == 'A') .Or. (Obs_ppt(iv)%f == 'A')) Then
    !   Cycle Pass_1
    !End If

```

```

! SAMSON source: missing value
! EarthInfo source value >= 0
! EarthInfo replaces SAMSON value.
id_ok = ((HP(iv)%s == T_missing) .And. &
        (Obs_ppt(iv)%s == T_EarthInfo))

! SAMSON %v == missing and EarthInfo %v >= 0 ?
!      %f == "M"                %f == ""
v_ok = (Obs_ppt(iv)%v >= Zero) .And. &
        (Abs(HP(iv)%v-Missing_Data) < Eps0)
are_ok = (id_ok) .And. (v_ok)
If (are_ok) Then
    HP(iv) = Obs_ppt(iv)
End If
End Do Pass_1
End If

!If (Associated(Accum_Samson)) Nullify(Accum_Samson)
!If (Associated(Accum_EI)) Nullify(Accum_EI)

! Pass 2: process accumulation.
nsamson = 0
If (Associated(Accum_Samson)) nsamson = Ubound(Accum_Samson,1)

If (nsamson > 0) Then
    Do igap = 1, nsamson
        col_beg = Accum_Samson(igap)%ibeg
        col_end = Accum_Samson(igap)%iend
        Total_ppt = Accum_Samson(igap)%Total

        If (Have_ppt_Obs_daily_data) Then
            Call Resolve_Accum_24h(okay, col_beg, col_end, Total_ppt)
        Else
            Call Fill_Buckets(okay, col_beg, col_end, Total_ppt)
        End If

        If (.Not. okay) Then
            ierr = ierr + 1
        End If
    End Do
End If

Xok = (ierr == 0)

End Subroutine Process_Precip_Records

Subroutine Print_HP(Header, Jv0, Jv1, Jout)

```

```

Implicit None
Character(Len=*), Intent(In) :: Header
Integer, Optional, Intent(In) :: Jv0, Jv1
Integer, Optional, Intent(In) :: Jout

Integer :: uu, col_beg, col_end
Integer :: iv, jyyyy, jmm, jdd, jhh
Real    :: sum_samson_lday, sum_total

If (Present(Jout)) Then
    uu = Jout
Else
    uu = ULog
End If

HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

If (Present(Jv0) .And. Present(Jv1)) Then
    col_beg = Jv0
    col_end = Jv1
Else
    col_beg = 1
    col_end = Ubound(HP,1)
End If

Write(uu, 9130) Trim(Header)
9130 Format (1x, '## ', a, 1x, '##')

sum_samson_lday = Zero
sum_total = Zero

By_Hours: Do iv = col_beg, col_end
    Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)
    Write(uu, 9150) iv, jyyyy, jmm, jdd, jhh, &
        'HP', HP(iv)%s, HP(iv)%v, HP(iv)%f(1:2), &
        'Obs', Obs_ppt(iv)%s, Obs_ppt(iv)%v, Obs_ppt(iv)%f(1:2)
9150 Format (1x, ' ', i7, 1x, &
    i4, '-', i2.2, '-', i2.2, i3, 'h: ', &
    2(a, ': ', a, 1x, lpg14.6, ' ", a, "; '))

    If (Modulo(iv,25) /= 0) Then
        sum_samson_lday = sum_samson_lday + HP(iv)%v
        sum_total = sum_total + HP(iv)%v
    Else
        Write(uu, 9170) sum_samson_lday
9170 Format (1x, t26, 'Sum 1-24:', lpg14.6)
        sum_samson_lday = Zero
    End If
End Do By_Hours

```

```

Write(uu, 9190) sum_samson_1day
9190 Format (1x, t25, 'Sum Total:', 1pg14.6)

```

```
End Subroutine Print_HP
```

```
Subroutine Process_Precipitation(Xok)
```

```

! This routine collects most of the precipitation-related
! subroutine calls.
! <A NAME="Process_Precipitation">
! <A HREF="Precip.f90#Process_Precipitation">

```

```

Implicit None
Logical, Intent(Out) :: Xok

```

```

Logical :: Have_Precip_Data
Integer :: hh01, hh24, hh25, jday, jv0, jv1
Integer :: ierr
Type(Val_and_Flag), Dimension(:), Pointer :: HP

```

```

Xok = .False.
ierr = 0
HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

```

```
Call Station_With_Missing_Data(pWBAN%WBAN, Have_Precip_Data)
```

```

! Was this station identified by SAMSON as "Little/No Hourly ppt data" ?
! <A HREF="samson_format.txt#STATIONS WITH LITTLE OR NO HOURLY PRECIPITATION DATA">
! <A HREF="e:\5\3met\Docs\samson_format.txt#$1">

```

```

If (.Not. Have_Precip_Data) Then
! <A NAME="note_101">
! <A HREF="Precip.f90#note_101">
! * make all hourly precipitation missing, even when
! present. Fill daily value record (hh == 25) with
! the summary of the day.
L31: Do jday = jd0, jd1      ! step by day
    hh01 = (jday-jd0)*Nhours + 1 ! First hour of the day
    hh24 = hh01 + 23      ! Last hour of the day (24th)
    HP(hh01:hh24) = Val_and_Flag(T_Missing, Missing_Data, '')
End Do L31

```

```

! Do we have the EarthInfo summary of the day?
If (Have_ppt_Obs_daily_data) Then
! Yes.
L41: Do hh25 = 25, Ubound(HP,1), NHours
    Select Case(Obs_ppt(hh25)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)

```

```

        ! Missing value. Set to Zero.
        HP(hh25) = Val_and_Flag(T_Estimated, Zero, '')
    Case Default
        HP(hh25) = Obs_ppt(hh25)
    End Select
End Do L41

Else
    ! No. Bummer.
    Write (ULog, 9130) Trim(pWBAN%WBAN), Trim(pWBAN%Text)
9130   Format (///, &
        1x, 45('?'), /, &
        1x, '?? Missing EarthInfo summary of the day data for station ', a, ': ', a, /, &
        1x, '?? Daily values set to "missing"', /, &
        1x, 45('?'), /)
    L42: Do hh25 = 25, Ubound(HP,1), NHours
        HP(hh25) = Val_and_Flag(T_Missing, Missing_Data, '')
    End Do L42
    ierr = ierr + 1
End If

!! Compute the daily value.
!! See <A HREF="Precip.f90#note_102">
!!Call Daily_Values(f_HP, T_Cumulative, Xok)
!Call Ppt_Daily_Values(Xok)
!If (.Not. Xok) Then
!   ierr = ierr + 1
!   Errors_Detected = .True.
!End If
Xok = (ierr == 0)
Return
End If

! We have precipitation data, however the record may be incomplete.
! For example:
!   22516: Kahului HI --
!       Missing hourly data for 1961-Jan to 1962-Dec.
!       Complete hourly data for 1963-Jan to 1990-Dec.

! Precipitation: missing values not in an accumulation run
! will be set to Zero. Since one of the algorithms used to
! fill Accumulation runs references OSC, fill all gaps before
! resolving Accumulation runs.
!
! 5 Mar 2002 3:41 pm: Well, we have a problem here. '99999A'
! in an accumulation may mean one of two things:
!   1) this is the start of an accumulation run, or
!   2) continue the accumulation run.

```

```

! So, the 'missing' value has to be seen in the context of 'A';
! we cannot just set the missing value to zero.
!
! We need to fill the gaps of OSC before processing accumulation.
! Fill_Accumulation will also fill missing values.
!
! Note: Some of the files contain also flag=='E' (Estimated)

Call ToTTY('Process_Precipitation: Process_Precip_Records.')

Call Process_Precip_Records(Xok)
If (.Not. Xok) Then
    Errors_Detected = .True.
End If
Call FLushAll()

!Call Yearly_Precip_Stats('before Reconcile_HP_samson_earthinfo') !!--
Call Reconcile_HP_samson_earthinfo(Xok)
If (.Not. Xok) Then
    Errors_Detected = .True.
End If

! Compute the daily value.
! See <A HREF="Precip.f90#note_102">
!Call Daily_Values(f_HP, T_Cumulative, Xok)
Call Ppt_Daily_Values(Xok)
If (.Not. Xok) Then
    Errors_Detected = .True.
End If
Call Yearly_Precip_Stats('After all processing completed.')

End Subroutine Process_Precipitation

```

```

Subroutine Reconcile_HP_samson_earthinfo(Xok)

! <A NAME="Reconcile_HP_samson_earthinfo">
! <A HREF="Precip.f90#Reconcile_HP_samson_earthinfo">

! #4. Reconcile summary of the day with hourly data.
!   For each month:
!
!       31
!   A <-- Sum EI(d), i.e., sum the summary of the
!         days           day for the month.
!
!
!       744
!   B <-- Sum S(h), i.e., sum all the SAMSON hours

```

```

!           hours           of the month.
!
!   If A <= B, i.e., SAMSON has more precipitation,
!           Nothing to do. Exit.
!
! #5. A > B : For each day (i)
!
!           24
!   C(i) <-- Sum S(h),   i.e., sum the (24) SAMSON hours
!           hours           of the ith day.
!
!   If C(i) >= EI(d), i.e., the SAMSON hour data has more
!           rain than summary of the day.
!           Nothing to do. Exit.
!
! #6. C(i) < EI(d) (summary of the day has more rain).
!   Look at the EarthInfo hourly record for that day.
!
!           24
!   If Sum EI(h) == EI(d)
!       1
!
!       Then Replace: S(h) <-- EI(h)
!
!           24
!   Else If EI(d) > Sum EI(h)
!       1
!
!       Then: accumulation run:
!           delta = EI(d) - Sum S(h)
!           allocate delta over the 24-hour period
!
!   Else ! EI(d) < Sum EI(h)
!       Dump the month. Determine what is going on.

Implicit None
Logical, Intent(Out) :: Xok

Integer :: yyyy, mm, dd, hh, jv0, jv1, iv, hh01, hh24, hh25
Integer :: ierr, iday
Logical :: okay
Integer :: missing_sd, missing_SAMSON, missing_in_SAMSON_day
Integer, Dimension(:), Pointer :: Days_in_Month

! The following are used for fuzzy comparisons.
! See <A HREF="global.f90#fuzzy comparisons">
Logical :: pA_le_pB, pA_ge_pB, pA_gt_pB
Real :: pA, pB
Real :: sum_sd, sum_SAMSON_month, sum_SAMSON_day, Sum_EI_h, EI_d

```

```

ierr = 0
HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

! To reconcile records we need the summary of the day.
If (.Not. Have_ppt_Obs_daily_data) Then
  Xok = .True.
  Return
End If

By_Year: Do yyyy = MinYear, MaxYear

  If (Year_Data/yyyy)%SAMSON_v10 == 0) Then
    ! Year is missing.
    Cycle By_Year
  End If

  Days_in_Month => Number_of_Days_in_Month/yyyy)

  By_Month: Do mm = 1, 12

    Call ymdh_to_iv/yyyy, mm, dd=01,          hh=01, iv=jv0)
    Call ymdh_to_iv/yyyy, mm, dd=Days_in_Month(mm), hh=Nhours, iv=jv1)

    sum_sd = Zero ! sum the summary of the day for the complete month.
    missing_sd = 0 ! Number of summary of the day missing.
    sum_SAMSON_month = Zero ! sum all the SAMSON hours of the month.
    missing_SAMSON = 0 ! Number SAMSON hours missing.

    Do iv = jv0, jv1
      If (Modulo(iv,25) /= 0) Then
        Select Case(HP(iv)%s)
          Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
            missing_SAMSON = missing_SAMSON + 1
          Case Default
            sum_SAMSON_month = sum_SAMSON_month + HP(iv)%v
        End Select
      Else
        ! 25th hour: Daily totals -- only for EarthInfo.
        Select Case(Obs_ppt(iv)%s)
          Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
            missing_sd = missing_sd + 1
          Case Default
            sum_sd = sum_sd + Obs_ppt(iv)%v
        End Select
      End If
    End Do

    ! If A <= B, i.e., SAMSON has more precipitation,
    ! Nothing to do. Exit.

```

```

! See <A HREF="global.f90#fuzzy comparisons">
!   A <= B :: Abs(B-A) < Eps0 .Or. Eps0 < (B-A)
pA = sum_sd
pB = sum_SAMSON_month
pA_le_pB = (Abs(pB-pA) < Eps0) .Or. (Eps0 < (pB-pA))
If (pA_le_pB) Then ! Fuzzy sum_sd <= sum_SAMSON_month
  ! Done. Zero any missing entries.
  sum_SAMSON_day = Zero
  Do iv = jv0, jv1
    If (Modulo(iv,25) /= 0) Then
      ! 1h to 24h
      Select Case(HP(iv)%s)
      Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
        HP(iv) = Val_and_Flag(T_Estimated, Zero, '')
      Case Default
        sum_SAMSON_day = sum_SAMSON_day + HP(iv)%v
      End Select
    Else
      ! 25th hour: Daily totals -- only for EarthInfo.
      Select Case(Obs_ppt(iv)%s)
      Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
        HP(iv) = Val_and_Flag(T_Estimated, sum_SAMSON_day, '')
      Case Default
        HP(iv) = Obs_ppt(iv)
      End Select
      sum_SAMSON_day = Zero ! Initialize sum for next day.
    End If
  End Do
  Cycle By_Month
End If

! #5. A > B : For each day (i)
By_Day: Do iday = 1, Days_in_Month(mm)

  !           24
  !   C(i) <-- Sum S(h),   i.e., sum the (24) SAMSON hours
  !           1           of the ith day.

  sum_SAMSON_day = Zero
  Sum_EI_h = Zero
  missing_in_SAMSON_day = 0

  Call ymdh_to_iv(yyyy, mm, dd=iday, hh=01, iv=hh01)
  hh24 = hh01 + 23
  hh25 = hh24 + 1

  Do iv = hh01, hh24
    Select Case(HP(iv)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
      missing_in_SAMSON_day = missing_in_SAMSON_day + 1
    End Select
  End Do
End Do

```

```

Case Default
  sum_SAMSON_day = sum_SAMSON_day + HP(iv)%v
End Select

Select Case(Obs_ppt(iv)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
Case Default
  Sum_EI_h = Sum_EI_h + Obs_ppt(iv)%v
End Select
End Do

!       If C(i) >= EI(d), i.e., the SAMSON hour data has more
!       rain than summary of the day.
!       Nothing to do. Exit.
Select Case(Obs_ppt(hh25)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
! Summary of the day missing;
! Set the daily value to the sum of the SAMSON hourly values for this day.
HP(hh25) = Val_and_Flag(T_Estimated, sum_SAMSON_day, '')
Cycle By_Day
End Select
EI_d = Obs_ppt(hh25)%v

! The daily value is the maximum of the sum of the SAMSON hours
! of the day or the EarthInfo summary of the day.
! <A NAME="note_102"> <A HREF="Precip.f90#note_102">
HP(hh25) = Val_and_Flag(T_Estimated, Max(sum_SAMSON_day, EI_d), '')

! See <A HREF="global.f90#fuzzy comparisons">
! A >= B :: Abs(A-B) < Eps0 .Or. Eps0 < (A-B)
pA = sum_SAMSON_day
pB = EI_d
pA_ge_pB = (Abs(pA-pB) < Eps0) .Or. (Eps0 < (pA-pB))
If (pA_ge_pB) Then ! Fuzzy sum_SAMSON_day >= EI_d
! This day is ok. Set to Zero any missing entries.
Go To 99887 ! Jump to End-Do-By_Day <A HREF="Precip.f90#99887">
End If

! #6. C(i) < EI(d) (summary of the day has more rain).
! Look at the EarthInfo hourly record for that day.
!
!       24
!       If Sum EI(h) == EI(d)
!       1

If (Abs(Sum_EI_h-EI_d) < Eps0) Then

!       Then Replace: S(h) <-- EI(h)

```

```

    HP(hh01:hh24) = Obs_ppt(hh01:hh24)
    !Write(ULog,*) '(3) Sum_EI_h==EI_d: hh01, hh24:', hh01, hh24
    Go To 99887 ! Jump to End-Do-By_Day <A HREF="Precip.f90#99887">
End If

!
!           24
!   Else If EI(d) > Sum EI(h)
!           1

! See <A HREF="global.f90#fuzzy comparisons">
!   A > B :: Abs(A-B) >= Eps0 .And. Eps0 < (A-B)
pA = EI_d
pB = Sum_EI_h
pA_gt_pB = (Abs(pA-pB) >= Eps0) .And. (Eps0 < (pA-pB))
If (pA_gt_pB) Then ! Fuzzy EI_d > Sum_EI_h ...

!           Then: accumulation run:
!           delta = EI(d) - Sum S(h)
!           allocate delta over the 24-hour period

Call Fill_Buckets(okay, hh01, hh24, EI_d)
If (.Not. okay) ierr = ierr + 1
Cycle By_Day

Else
!           Else ! EI(d) < Sum EI(h)
!           Dump the month. Determine what is going on.
Write(ULog,*) '(5) Error: jv0, jv1, ierr:', jv0, jv1, ierr

!           ierr = ierr + 1
!           Write (ULog, 9130) yyyy, Trim(Month_Table(mm)), iday
9130      Format (/ , 1x, '?? Reconcile_HP_samson_earthinfo: ', &
!           i4, '-', a, '-', i2.2)

!           Write (ULog, 9150) 'EI(d) < Sum EI(h)', EI_d, Sum_EI_h
9150      Format (1x, ' ', a, /, &
!           1x, ' EI(d) == ', 1pg14.6, /, &
!           1x, ' Sum EI(h) == ', 1pg14.6)

9170      Format (1x, ' ', a, ': ', 1pg14.6)
!           Write (ULog, 9170) 'summary of the day for the complete month', sum_sd
!           Write (ULog, 9170) 'sum all the SAMSON hours of the month...', sum_SAMSON_month
!           Write (ULog, 9170) 'sum all the 24 SAMSON hours of the day...', sum_SAMSON_day

!           Call Print_HP('', jv0, jv1)
!           Cycle By_Day
End If

! Access this section only by Go To.
! Being here is impossible (at this time anyway).

```

```

Cycle By_Day

! *****
! ** Check for missing values
! *****
! <A NAME="99887">
! <A HREF="Precip.f90#99887">
!
! 29 Apr 2002  4:23 pm: I need the same code in three places.
! A subroutine would be too slow for production: adds about a
! minute of execution time per run times 250 runs == 4.17 hours.
99887 Continue
Do iv = hh01, hh24
  Select Case(HP(iv)%s)
  Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness, T_Unset)
    HP(iv) = Val_and_Flag(T_Estimated, Zero, '')
  End Select
End Do
Cycle By_Day

End Do By_Day
End Do By_Month
End Do By_Year

Xok = (ierr == 0)

End Subroutine Reconcile_HP_samson_earthinfo

Subroutine Ppt_Daily_Values(Xok)

! Make sure all ppt records are non-missing, if possible.

Implicit None
Logical, Intent(Out) :: Xok

Integer :: jday, hh01, hh24, hh25, iv, nn
Real    :: xsum
Type(Val_and_Flag), Dimension(:), Pointer :: HP

HP => Xparam(f_HP)%Samson_v10

Do jday = jd0, jd1
  hh01 = (jday-jd0)*Nhours + 1 ! step by day
  hh24 = hh01 + 23           ! First hour of the day
  hh25 = hh24 + 1           ! Last hour of the day (24th)
  hh25 = hh24 + 1           ! 25th hour of jday == (jday-jd0+1)*NHours

  xsum = Zero

```



```

!      25  1  2      a      24      1      A      a  25  1
!
!      ^
!
! Note a second accumulation run within the last day ----+
!
!
! ... +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ...
!      25  1  2      a      24      1      A      24  25
!      ^      ^      ^      ^
!      ^      ^      ^      ^
!      n0      Col_beg      Col_end      n1
!
! Accumulation run [Col_beg, Col_end]
! Total_amount -- amount measured during the accumulation run.
!
! n0 = Col_beg / Nhours
! n1 = Col_end / Nhours
!
! Define procedure Fill_Buckets(iv0, iv1, TotPPT) :
!   Resolve an accumulation run utilizing the old algorithms.
!
! ## -----
! ## If n0 == n1:
!   i.e., the accumulation interval fits in 24 hours:
!
! Procedure P(n0):
!   * hh01 = n0 * Nhours + 1
!   * hh24 = hh01 + 23
!
!   * amt_out = precipitation in the 24h period outside the
!               accumulation run.
!
!               hh24
!   = Sum S(h)
!     h=hh01
!     h not in [Col_beg, Col_end]
!
!
!   * amt_in = precipitation in the 24h period inside the
!               accumulation run.
!
!               hh24
!   = Sum S(h)
!     h=hh01
!     h in [Col_beg, Col_end]
!
!
!   * sd(n0) = summary of day n0
!
!   * amt(n0) = sd(n0) - amt_in - amt_out

```

```

!       If (amt(n0) < 0) Dump arrays.
!       Else
!           iv0 = Max(hh01, Col_beg)
!           iv1 = Min(hh24, Col_end)
!           Call Fill_Buckets(iv0, iv1, amt(n0))
!
!           hh24
!       * Verify: Sum S(h) == sd(n0)
!           h=hh01
!
!       * Done.
!
! ## -----
! ## n0 < n1:
!
! * For i = n0, n1-1 Do P(i)
!
!           n1-1
! * cumulative_ppt = Sum sd(i)
!           i=n0
!
! * Compute hh01, hh24, amt_in, amt_out @ n1
!
! * *** For complete details of the amount remaining
!     to be distributed "amt(n1)" see the Fortran code.
!     Documentation kept simple so that the basic
!     idea is not obscured.
!
!     If the last interval contains another
!     accumulation run
!         amt(n1) = Total_amount - cumulative_ppt - amt_in
!     Else
!         amt(n1) = sd(n1) - amt_in - amt_out
!     End If
!
! * iv0 = Max(hh01, Col_beg)
!     iv1 = Min(hh24, Col_end)
!     Call Fill_Buckets(iv0, iv1, amt(n1))
!
! ## -----
! ## Final Check:
!
!           Col_end
! * amt_allocated = Sum S(h)
!           h=Col_beg
!
! * Verify:
!     = If any sd(i) was missing,
!         Call Fill_Buckets(Col_beg, Col_end, Total_amount)
!     = If amt_allocated == Total_amount then Done.
!     = If amt_allocated < Total_amount and

```

```

!      If any of the P(i) worked, then
!      Call Fill_Buckets(Col_beg, Col_end, Total_amount)
!      = Else
!      Dump arrays.

Implicit None
Logical, Intent(Out) :: Xok
Integer, Intent(In)  :: Col_Beg, Col_End
Real,   Intent(In)  :: Total_amount

Integer :: ierr
Integer :: n0, n1, nx, hh01, hh24, hh25
Integer :: iv, iv0, iv1
Integer :: periods_resolved
Real    :: amt_in, amt_out, sd_nx, delta_nx
Real    :: cumulative_ppt, amt_allocated
Real    :: total_this_day
Logical :: another_accumulation, okay, found_missing
Logical :: current_sd_missing, some_sd_missing

Namelist/xx0/ amt_in, amt_out, sd_nx, delta_nx, total_this_day
Namelist/xx1/ another_accumulation, some_sd_missing, &
             current_sd_missing
Namelist/xx2/ Total_amount, &
             cumulative_ppt, amt_allocated

Xok = .False.
ierr = 0
HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

n0 = Col_beg / Nhours   ! First day of accumulation interval.
n1 = Col_end / Nhours   ! Last day of accumulation interval.
periods_resolved = 0
cumulative_ppt = Zero   ! ppt allocated so far.

! some_sd_missing -- true if any sd(n0:n1-1) were missing.
! if any sd was missing, then the sum "cumulative_ppt" is
! incomplete, because it is missing precipitation for at
! least one 24-hour period.
some_sd_missing = .False.
current_sd_missing = .False.

Loop_n0_n1: Do nx = n0, n1      ! For each day ...

    hh01 = nx*Nhours + 1      ! First hour of the day
    hh24 = hh01 + 23         ! Last hour of the day
    hh25 = hh24 + 1          ! Daily value
    amt_in = Zero
    amt_out = Zero
    another_accumulation = .False.

```

```

! At this state "current_sd_missing" contains the value
! of the previous iteration. By performing the test now,
! "some_sd_missing" tests whether any of the previous sd's
! (n0:nx-1) were missing.
If (current_sd_missing) Then
    some_sd_missing = .True.
End If

Do iv = hh01, hh24
    If ((Col_beg <= iv) .And. (iv <= Col_end)) Then
        ! iv falls inside the accumulation interval.
        If (HP(iv)%s /= T_Missing) Then
            amt_in = amt_in + HP(iv)%v
        Else
            HP(iv)%v = Zero    ! In preparation of resolution via "Fill_Buckets"
        End If
    Else
        ! iv falls outside accumulation interval.
        If (HP(iv)%s /= T_Missing) Then
            amt_out = amt_out + HP(iv)%v
        End If
        If (HP(iv)%f == 'A') Then
            another_accumulation = .True.
        End If
    End If
End Do

! sd_nx = daily precipitation (summary of the day) for day nx
If (Obs_ppt(hh25)%s /= T_Missing) Then
    sd_nx = Obs_ppt(hh25)%v
    current_sd_missing = .False.
Else
    ! The summary of the day is not available for this day.
    sd_nx = Zero
    current_sd_missing = .True.
End If

If (nx < n1) Then
    If (current_sd_missing) Cycle Loop_n0_n1
    delta_nx = sd_nx - (amt_in + amt_out)
Else
    ! Last Interval:  n0 == n1 == nx  -or-  nx == n1 (n0 < n1)
    If (another_accumulation) Then
        ! The last day contains another accumulation run.
        If (some_sd_missing) Then
            ! There is nothing more we can do at this time.
            ! A finishing pass will be applied later.
            Cycle Loop_n0_n1
        Else
            delta_nx = Total_amount - (cumulative_ppt + amt_in)
        End If
    End If
End If

```

```

    End If
Else If (.Not. current_sd_missing) Then
    ! We have summary of the day "sd_nx".
    delta_nx = sd_nx - (amt_in + amt_out)
Else If (.Not. some_sd_missing) Then
    ! If all sd(n0:n1-1) were present, then all previous intervals
    ! were allocated and cumulative_ppt is complete.
    delta_nx = Total_amount - (cumulative_ppt + amt_in)
Else
    ! Else at least one of the intervals was not allocated.
    ! If we use
    !     delta_nx = Total_amount - (cumulative_ppt + amt_in)
    ! we would be allocating all the remaining precipitation to the
    ! last interval, completely ignoring the unallocated intervals.
    !
    ! There is nothing more we can do at this time.
    ! A finishing pass will be applied later.
    Cycle Loop_n0_n1
End If
End If

iv0 = Max(hh01, Col_beg)
iv1 = Min(hh24, Col_end)

! delta_nx -- delta difference between the amount to
!     be allocated for the period nx and the amount
!     already in the period (amt_in).
! total_this_day -- amount to be allocated in this period.
total_this_day = delta_nx + amt_in

If (delta_nx > Eps0) Then
    Call Fill_Buckets(okay, iv0, iv1, total_this_day)
Else If (Abs(delta_nx) < Eps0) Then
    ! delta_nx == Zero (fuzzily). Make it Zero (exactly).
    ! (we still need to update the HP array).
    delta_nx = Zero
    Call Fill_Buckets(okay, iv0, iv1, total_this_day)
Else
    ! For example:
    ! sd == 0.0
    ! amt_out == 0.2032
    ! Total_amount == 0.127
    ! The day already has too much precipitation (according
    ! to the summary of the day) and we still want to add
    ! more precipitation.
    !
    ! If this is not the last day, allocate no precipitation
    ! (we still need to update the HP array).
    If (nx < n1) Then
        delta_nx = Zero

```

```

        Call Fill_Buckets(okay, iv0, iv1, total_this_day)
    Else
        ! Else, allocate
        delta_nx = Max(Total_amount - (cumulative_ppt+amt_in), Zero)
        total_this_day = delta_nx + amt_in
        Call Fill_Buckets(okay, iv0, iv1, total_this_day)
    End If
End If

If (okay) Then
    ! Do not update variables for the last interval.
    If (nx < n1) Then
        periods_resolved = periods_resolved + 1
        !cumulative_ppt = cumulative_ppt + sd_nx ! 2 May 2002 11:19 am
        ! 2 May 2002 11:32 am: Tested with 23129: Long Beach, CA
        ! 2 May 2002 3:23 pm: Tested with 23063: Eagle, CO
        cumulative_ppt = cumulative_ppt + total_this_day
    End If
Else
    ierr = ierr + 1
    Call Print_24('(2): okay == F', n0, n1, nx, &
        Col_Beg, Col_End, Total_amount, total_this_day)
End If
End Do Loop_n0_n1

! ## -----
! ## Final Check:

! It is conceivable that all precipitation has been
! allocated and that there are still
!   HP(Col_beg:Col_end)%s == missing
! present. For example, if some (non-last)
! sd was missing, that interval would not be
! called and ()%s would not be changed.

amt_allocated = Zero
found_missing = .False.
Do iv = Col_beg, Col_end
    If (HP(iv)%s /= T_Missing) Then
        amt_allocated = amt_allocated + HP(iv)%v
    Else
        found_missing = .True.
    End If
End Do

delta_nx = Total_amount - amt_allocated

If (Abs(delta_nx) < Eps0) Then
    ! Done

```

```

Else If ((Abs(delta_nx) >= Eps0) .And. (Eps0 < (-delta_nx))) Then
  ! delta_nx < 0 :: Believe EarthInfo over SAMSON,
  !           but note the discrepancy.
  !
  ! See <A HREF="global.f90#fuzzy comparisons">
  ! A < B :: Abs(B-A) >= Eps0 .And. Eps0 < (B-A)
  ! A < 0 :: Abs(A) >= Eps0 .And. Eps0 < (-A)
  Write (ULog, 9130) &
    Str_iv_to_ymdh(Col_Beg), &
    Str_iv_to_ymdh(Col_End), &
    Total_amount, amt_allocated, delta_nx
9130  Format(/, 1x, '## Accumulation run', &
    ' from ', a, ' to ', a, /, &
    1x, ' (a) Amount to be allocated: ', f6.2, ' cm', /, &
    1x, ' (b) Amount allocated based on EarthInfo summary of the day: ', &
    f6.2, ' cm', /, &
    1x, ' (a) - (b): ', f6.2, ' cm')

Else If (some_sd_missing .Or. found_missing .Or. (delta_nx >= Eps0)) Then
  ! The finishing pass will be applied now.
  Call Fill_Buckets(okay, Col_beg, Col_end, Total_amount)
  If (.Not. okay) Then
    ierr = ierr + 1
    Call Print_24('(3): okay == F', n0, n1, n1, &
      Col_Beg, Col_End, Total_amount, total_this_day)
  End If
End If

Xok = (ierr == 0)

End Subroutine Resolve_Accum_24h

Subroutine Print_24(Header, n0, n1, nx, Col_Beg, Col_End, &
  Total_amount, total_this_day, Jout)

  ! <A NAME="Print_24">
  ! <A HREF="Precip.f90#Print_24">

  ! The accumulation run failed for some period.
  ! Print data for the period in question.

  Implicit None
  Character(Len=*), Intent(In)  :: Header
  Integer,          Intent(In)  :: n0, n1, nx
  Integer,          Intent(In)  :: Col_Beg, Col_End
  Real,             Intent(In)  :: Total_amount, total_this_day
  Integer, Optional, Intent(In)  :: Jout

  Integer :: uu, iv, iv0, iv1, igap, ibeg, iend

```

```

Integer :: current_gap_size
Real    :: HPsum

If (Present(Jout)) Then
    uu = Jout
Else
    uu = ULog
End If

HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation

Write (uu, 9130) '?? Print_24: Resolve_Accum_24h -- ', Trim(Header)
9130 Format (/ , 1x, a, a)

9150 Format(1x, 2x, a, a)
9170 Format(1x, 2x, a, 7i7)

Write (uu, 9150) 'Col_Beg.....: ', Str_iv_to_ymdh(Col_Beg)
Write (uu, 9150) 'Col_End.....: ', Str_iv_to_ymdh(Col_End)

current_gap_size = Col_End - Col_Beg + 1 - Multiples_of(NHours, Col_Beg, Col_End)
Write (uu, 9170) 'Gap Size (hours).....: ', current_gap_size

Write (uu, 9190) 'Total_amount.....: ', Total_amount
Write (uu, 9190) 'total_this_day.....: ', total_this_day
9190 Format(1x, 2x, a, lpg14.6)

iv = nx*Nhours + 1
Write (uu, 9150) 'nx @ 1h.....: ', Str_iv_to_ymdh(iv)
Write (uu, 9170) 'n1-n0+1,n1,n0,nx.....: ', n1-n0+1, n1, n0, nx

Write (uu, 9210) 'Have_ppt_Obs_daily_data.: ', Have_ppt_Obs_daily_data
Write (uu, 9210) 'Have_ppt_Obs_hourly_data.: ', Have_ppt_Obs_hourly_data
9210 Format(1x, 2x, a, L1)

iv0 = n0*Nhours + 1
iv1 = (n1+1)*Nhours

Do igap = 1, Ubound(Accum_Samson,1)
    ibeg = Accum_Samson(igap)%ibeg
    iend = Accum_Samson(igap)%iend
    If (((iv0 <= ibeg) .And. (ibeg <= iv1)) .Or. &
        ((iv0 <= iend) .And. (iend <= iv1))) Then
        current_gap_size = iend - ibeg + 1 - Multiples_of(NHours, ibeg, iend)

        Write (ULog, 9230) igap, &
            Str_iv_to_ymdh(ibeg), &
            Str_iv_to_ymdh(iend), &
            Accum_Samson(igap)%Total
9230 Format(1x, 2x, i4, ': Accumulation gap from ', &

```

```

        a, ' to ', a, ', amt:', lpg14.6, ' cm')
    End If
End Do

HPsum = Zero
Do iv = iv0, iv1
    If (Modulo(iv,25) /= 0) Then
        Write(uu, 9250) Str_iv_to_ymdh(iv), &
            'HP...: ', HP(iv)%s, HP(iv)%v, Trim(HP(iv)%f)
        If (HP(iv)%s /= T_Missing) Then
            HPsum = HPsum + HP(iv)%v
        End If

    Else
        Write(uu, 9250) Str_iv_to_ymdh(iv), &
            'HPsum: ', ' ', HPsum
        Write(uu, 9250) Str_iv_to_ymdh(iv), &
            'EI...: ', Obs_ppt(iv)%s, Obs_ppt(iv)%v, Trim(Obs_ppt(iv)%f)
        HPsum = Zero
        Write(uu, *)
    End If
End Do

9250    Format (lx, 5x, a, ': ', a, a, lx, lpg14.6, ': ', a)
End Do

End Subroutine Print_24

```

```
Subroutine Fill_Buckets(Xok, Col_Beg, Col_End, Total_ppt, Release_Storage)
```

```

! <A NAME="Fill_Buckets">
! <A HREF="Precip.f90#Fill_Buckets">
! See <A HREF="0notes.txt#Note_27">
!
! 19 Feb 2002  3:46 pm -- this is the how things actually appear in the
!                      SAMSON files. See, e.g., 12842_75.txt, from *.z
!
! yy mm dd hh Hourly Precipitation in hundredths of inch;
! -- -- -- ----- R0 files report in cm;
! 75 10  4 14      0A
! 75 10  4 15
! 75 10  4 16
! 75 10  4 17
! 75 10  4 18
! 75 10  4 19      41A          41/100 * 2.54 cm/inch = 1.0414 cm
! 75 10  4 20      0A
! 75 10  4 21
! 75 10  4 22
! 75 10  4 23
! 75 10  4 24

```

```

! 75 10 5 1      9A
!
! Tested with 23063, Eagle, CO

Implicit None
Logical,          Intent(Out) :: Xok
Integer,          Intent(In)  :: Col_Beg, Col_End
Real,             Intent(In)  :: Total_ppt
Logical, Optional, Intent(In) :: Release_Storage

Character(Len(Flag_Done)) :: Status_Flag
Integer :: iv, k, current_gap_size, jf2, jf3, mf2, mf3
Integer :: jOSC, nv
Integer :: Observation_Indicator
Integer :: hours_with_ppt
Logical :: L_f2, L_f3, have_bounds, okay
Logical :: at_least_one
Real    :: HPsum, xresid, rdiv, delta_amt

! f2: Changed from 0.28 to 0.25 so we have no gaps in the range
Real, Dimension(0:2), Parameter :: f2l = (/ Zero, 0.250, 0.760  /)
Real, Dimension(0:2), Parameter :: f2u = (/ 0.250, 0.760, Huge(Zero) /)
Real, Dimension(0:2), Parameter :: f3l = (/ Zero, 0.025, 0.051  /)
Real, Dimension(0:2), Parameter :: f3u = (/ 0.025, 0.051, Huge(Zero) /)

Logical, Save :: First_Time = .True.
Integer, Save :: ddt_marker = 0

ddt_marker = ddt_marker + 1

! Observation Indicator  0 or 9  0 = Weather observation made.
!                          9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
!
!   Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
!   Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
!   Xparam(f_OI)%Samson_v10(iv)%s = data_source

HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation
OI => Xparam(f_OI)%Samson_v10      ! Observation Indicator
OSC => Xparam(f_OSC)%Samson_v10    ! Opaque Sky Cover

If (Present(Release_Storage)) Then
  If (Associated(Vmin)) Deallocate(Vmin, Vmax, Vval, &
    Vmissing_hours, Vzero_ppt, vSelect, &
    Vpw, vOSC_1_to_4, vOSC_5_to_10, &
    Vbounds, Vused, Vtempl, VpwRain, VpwMissing)
  First_Time = .True.
  Xok = .True.
  Return

```

```

End If

nv = Col_End - Col_Beg + 1
current_gap_size = nv

If (First_Time) Then
  First_Time = .False.
  dimV = Max(50, current_gap_size)
  Allocate(Vmin(dimV), Vmax(dimV), Vval(dimV), &
    Vmissing_hours(dimV), Vzero_ppt(dimV), &
    vSelect(dimV), Vpw(dimV), vOSC_1_to_4(dimV), &
    vOSC_5_to_10(dimV), Vbounds(dimV), Vused(dimV), &
    Vtempl(dimV), VpwRain(dimV), VpwMissing(dimV))
End If

If (current_gap_size > dimV) Then
  ! Not enough space. Increase it.
  dimV = Max(2*dimV, current_gap_size)
  Deallocate(Vmin, Vmax, Vval, &
    Vmissing_hours, Vzero_ppt, vSelect, &
    Vpw, vOSC_1_to_4, vOSC_5_to_10, &
    Vbounds, Vused, Vtempl, VpwRain, VpwMissing)
  Allocate(Vmin(dimV), Vmax(dimV), Vval(dimV), &
    Vmissing_hours(dimV), Vzero_ppt(dimV), &
    vSelect(dimV), Vpw(dimV), vOSC_1_to_4(dimV), &
    vOSC_5_to_10(dimV), Vbounds(dimV), Vused(dimV), &
    Vtempl(dimV), VpwRain(dimV), VpwMissing(dimV))
End If

hours_with_ppt = 0
HPsum = Zero
have_bounds = .False.

! Notes:
! #1. For passes 1 and 2, if an hour was used, then
!     it cannot be used again. Example:
!     Pass1: If a bucket was used for #1.2.1, then
!             it cannot be used for #1.2.2
!     Pass2: If a bucket was used for #2.2.2, then
!             it cannot be used for #2.2.3
!
! #2. Pass #3. Find all buckets with precipitation,
!     including those filled during passes 1 and 2.
!
!
! #1.0) Pass Number 1:
! #1.1) Collect missing hours
! #1.2) Allocate rain among those hours by
!       #1.2.1) Present Weather
!       #1.2.2) OSC

```

```

!
! ## If all rain has been allocated, exit.
!   Else, Pass #2.
!
! #2.0) Pass Number 2:
! #2.1) Collect hours with Zero precipitation.
! #2.2) Allocate rain among those hours by
!       #2.2.1) Present Weather == Rain
!       #2.2.2) OSC >= 50% and Present Weather == missing
!       #2.2.3) 40% >= OSC > 0% and Present Weather == missing
!
! ## If all rain has been allocated, exit.
!   Else, Pass #3.
!
! #3.0) Pass Number 3:
! #3.1) Collect hours with precipitation > 0.
! #3.2) Allocate the residual rain equally among
!       those hours.
!
! If it remains precipitation to be allocated ...

Do iv = Col_Beg, Col_End

    k = iv - Col_Beg + 1      ! Entry in V* arrays

    ! Collect all relevant information now.
    Vmin(k) = Zero
    Vmax(k) = Huge(Zero)
    Vmissing_hours(k) = .False.
    Vzzero_ppt(k) = .False.
    vSelect(k) = .False.
    Vbounds(k) = .False.
    Vused(k) = .False.

    ! Present weather flag: Q_PW_No_Rain, Q_PW_Yes_Rain, Q_PW_Missing
    Vpw(k) = Q_PW_No_Rain      ! Used for output of tables.
    VpwRain(k) = .False.
    VpwMissing(k) = .False.

    vOSC_1_to_4(k) = .False.
    vOSC_5_to_10(k) = .False.

    If (Modulo(iv,25) == 0) Then
        ! Skip 25-th hour of the day.
        Cycle
    End If

    If (HP(iv)%s /= T_Missing) Then
        Vval(k) = HP(iv)%v
        Vzzero_ppt(k) = (Abs(HP(iv)%v) < Eps0) ! Collect hours with precipitation==0.

```

```

Else
  Vval(k) = Zero
  Vmissing_hours(k) = .True.    ! Collect missing hours
End If

If (OSC(iv)%s /= T_Missing) Then
  jOSC = Nint(OSC(iv)%v)
  vOSC_1_to_4(k) = (0 < jOSC) .And. (jOSC <= 4)
  vOSC_5_to_10(k) = (jOSC >= 5)
End If

HPsum = HPsum + Vval(k)

Observation_Indicator = Nint(OI(iv)%v) ! 0 or 9
Select Case(Observation_Indicator)
Case(9) ! Weather observation not made or missing.
  Vpw(k) = Q_PW_Missing
  VpwMissing(k) = .True.

Case(0) ! Weather observation made.

  ! (1) Occurrence of Thunderstorm, Tornado, or Squall
  If (OI(iv)%f(1:1) /= '9') Then
    VpwRain(k) = .True.
    Vpw(k) = Q_PW_Yes_Rain
    !Write(ULog, 3120) 'Thunderstorm, Tornado, or Squall', OI(iv)%f(1:1)
  End If

  ! (2) Occurrence of Rain Showers, or Freezing Rain
  L_f2 = .False.

  ! OI(iv)%f(2:2) == '0' .. '9'
  ! jf2 == 0 1 2 3 4 5 6 7 8 9
  ! mf2 == 0 1 2 0 1 2 0 1 2 0 ; index for f2l, f2u

  jf2 = Iachar(OI(iv)%f(2:2)) - Iachar('0') ! jf2 = 0..9
  mf2 = Modulo(jf2, 3)
  Select Case(jf2)
  Case (9) ! Nothing occurred.
    ! Do nothing.

  Case (0, & ! Light rain
    3, & ! Light rain showers
    6) ! Light freezing rain
    ! ! Light = up to 0.25 cm per hour.
    Vmin(k) = f2l(mf2)
    Vmax(k) = f2u(mf2)
    L_f2 = .True.
    VpwRain(k) = .True.
    Vpw(k) = Q_PW_Yes_Rain

```

```

!Write(ULog, 3120) &
!      '0:Light rain, 3:Light rain showers, 6:Light freezing rain', &
!      OI(iv)%f(2:2)

Case (1, & ! Moderate rain
      4, & ! Moderate rain showers
      7)  ! Moderate freezing rain
!      ! Moderate = 0.28 to 0.76 cm per hour.
Vmin(k) = f2l(mf2)
Vmax(k) = f2u(mf2)
L_f2 = .True.
VpwRain(k) = .True.
Vpw(k) = Q_PW_Yes_Rain
!Write(ULog, 3120) &
!      '1:Moderate rain, 4:Moderate rain showers, 7:Moderate freezing rain', &
!      OI(iv)%f(2:2)

Case (2, & ! Heavy rain
      5, & ! Heavy rain showers
      8)  ! Heavy freezing rain
!      ! Heavy = greater than 0.76 cm per hour.
Vmin(k) = f2l(mf2)
Vmax(k) = f2u(mf2)
L_f2 = .True.
VpwRain(k) = .True.
Vpw(k) = Q_PW_Yes_Rain
!Write(ULog, 3120) &
!      '2:Heavy rain, 5:Heavy rain showers, 8:Heavy freezing rain', &
!      OI(iv)%f(2:2)

End Select
Format (6x, a, ';' flag = ', a)

! (3) Occurrence of Rain Squalls, Drizzle, or freezing Drizzle
L_f3 = .False.
jf3 = Iachar(OI(iv)%f(3:3)) - Iachar('0') ! jf3 = 0..9
mf3 = Modulo(jf3, 3)
Select Case(jf3)
Case (9)  ! Nothing occurred.
! Do nothing.

Case (0, & ! Light rain squalls
      3, & ! Light drizzle
      6)  ! Light freezing drizzle
!      ! Light = up to 0.025 cm per hour.
Vmin(k) = f3l(mf3)
Vmax(k) = f3u(mf3)
L_f3 = .True.
VpwRain(k) = .True.
Vpw(k) = Q_PW_Yes_Rain
!Write(ULog, 3120) &

```

9130

```

!      '0: Light rain squalls, 3: Light drizzle, 6: Light freezing drizzle', &
!      OI(iv)%f(3:3)

Case (1, & ! Moderate rain squalls
      4, & ! Moderate drizzle
      7) ! Moderate freezing drizzle
!      ! Moderate = 0.025 to 0.051 cm per hour.
Vmin(k) = f3l(mf3)
Vmax(k) = f3u(mf3)
L_f3 = .True.
VpwRain(k) = .True.
Vpw(k) = Q_PW_Yes_Rain
!Write(ULog, 3120) &
!      '1: Moderate rain squalls, 4: Moderate drizzle, &
!      &7: Moderate freezing drizzle', &
!      OI(iv)%f(3:3)

Case (2, & ! Heavy Rain Squalls
      5, & ! Heavy drizzle
      8) ! Heavy freezing drizzle
!      ! Heavy = greater than 0.051 cm per hour.
Vmin(k) = f3l(mf3)
Vmax(k) = f3u(mf3)
L_f3 = .True.
VpwRain(k) = .True.
Vpw(k) = Q_PW_Yes_Rain
!Write(ULog, 3120) &
!      '5: Heavy drizzle, 8: Heavy freezing drizzle', OI(iv)%f(3:3)
End Select

If (OI(iv)%f(4:4) /= '9') Then
  VpwRain(k) = .True.
  Vpw(k) = Q_PW_Yes_Rain
  !Write(ULog, 3120) 'Snow, Snow Pellets, or Ice Crystals', OI(iv)%f(4:4)
End If

If (OI(iv)%f(5:5) /= '9') Then
  VpwRain(k) = .True.
  Vpw(k) = Q_PW_Yes_Rain
  !Write(ULog, 3120) 'Snow Showers, or Snow Squalls', OI(iv)%f(5:5)
End If

If (OI(iv)%f(6:6) /= '9') Then
  VpwRain(k) = .True.
  Vpw(k) = Q_PW_Yes_Rain
  !Write(ULog, 3120) 'Sleet, Sleet Showers, or Hail', OI(iv)%f(6:6)
End If

Vbounds(k) = (L_f2 .Or. L_f3)

```

```

        If (L_f2 .And. L_f3) Then
            Vmin(k) = (f2l(mf2) + f3l(mf3)) / 2
            Vmax(k) = (f2u(mf2) + f3u(mf3)) / 2
        End If
        have_bounds = (have_bounds .Or. Vbounds(k))
    End Select
End Do

delta_amt = Max(Total_ppt-HPsum, Zero)
xresid = delta_amt
If (Abs(xresid) < Eps0) Then
    ! Success. Xresid == Zero (fuzzily). Make it Zero (exactly).
    xresid = Zero
    Status_Flag = Flag_Done
    Go To 99999
End If

! #1. For passes 1 and 2, if an hour was used, then
!     it cannot be used again. Example:
!     Pass1: If a bucket was used for #1.2.1, then
!             it cannot be used for #1.2.2
!     Pass2: If a bucket was used for #2.2.2, then
!             it cannot be used for #2.2.3
!
! #2. Pass #3. Find all buckets with precipitation,
!     including those filled during passes 1 and 2.
!
!
! #1.0) Pass Number 1:
! #1.1) Collect missing hours
! #1.2) Allocate rain among those hours by
!         #1.2.1) Present Weather == Rain
!         #1.2.2) OSC

Vused(1:nv) = .False.    ! All hours available
at_least_one = Any(VpwRain(1:nv))
If (at_least_one) Then
    ! missing hours and Present Weather == Rain
    Call Do_Phase(Col_Beg, Col_End, xresid, &
        Vmissing_hours, VpwRain, Status_Flag, Vused, Vtemp1)

    Select Case(Status_Flag)
    Case(Flag_Continue)
        ! Do next phase.
    Case(Flag_Done)
        ! All rain allocated. Jump to end of subroutine.
        Go To 99999
    Case(Flag_Error)
        ! Error. Jump to end of subroutine and return.
        Go To 99999
    End Select

```

```

    End Select
End If

! missing hours and OSC
vSelect(1:nv) = Vmissing_hours(1:nv) .And. vOSC_5_to_10(1:nv)
at_least_one = Any(vSelect(1:nv))
Call Accum_based_on_OSC(Status_Flag, Col_Beg, Col_End, &
    Xresid, vSelect, OSC(Col_Beg:Col_End)%v, Vused)

! ## If all rain has been allocated, exit.
! Else, Pass #2.

Select Case(Status_Flag)
Case(Flag_Continue)
    ! Do next phase.
Case(Flag_Done)
    ! All rain allocated. Jump to end of subroutine.
    Go To 99999
Case(Flag_Error)
    ! Error. Jump to end of subroutine and return.
    Go To 99999
End Select

! #2.0) Pass Number 2:
! #2.1) Collect hours with Zero precipitation.
! #2.2) Allocate rain among those hours by
!     #2.2.1) Present Weather == Rain
!     #2.2.2) OSC >= 50% and Present Weather == missing
!     #2.2.3) 40% >= OSC > 0% and Present Weather == missing

! ppt == 0 and Present Weather == Rain
Vused(1:nv) = .False.    ! All hours available.
at_least_one = Any(VpwRain(1:nv) .And. Vzero_ppt(1:nv))
Call Do_Phase(Col_Beg, Col_End, xresid, &
    Vzero_ppt, VpwRain, Status_Flag, Vused, Vtempl)

Select Case(Status_Flag)
Case(Flag_Continue)
    ! Do next phase.
Case(Flag_Done)
    ! All rain allocated. Jump to end of subroutine.
    Go To 99999
Case(Flag_Error)
    ! Error. Jump to end of subroutine and return.
    Go To 99999
End Select

! ppt == 0 and OSC >= 50% and Present Weather == missing
vtempl(1:nv) = Vzero_ppt(1:nv) .And. VpwMissing(1:nv) .And. vOSC_5_to_10(1:nv)
at_least_one = Any(vtempl(1:nv))

```

```

Call Accum_based_on_OSC(Status_Flag, Col_Beg, Col_End, &
    Xresid, vtemp1, OSC(Col_Beg:Col_End)%v, Vused)

Select Case(Status_Flag)
Case(Flag_Continue)
    ! Do next phase.
Case(Flag_Done)
    ! All rain allocated. Jump to end of subroutine.
    Go To 99999
Case(Flag_Error)
    ! Error. Jump to end of subroutine and return.
    Go To 99999
End Select

! ppt == 0 and 40% >= OSC > 0% and Present Weather == missing
vtemp1(1:nv) = Vzero_ppt(1:nv) .And. VpwMissing(1:nv) .And. vOSC_1_to_4(1:nv)
at_least_one = Any(vtemp1(1:nv))
Call Accum_based_on_OSC(Status_Flag, Col_Beg, Col_End, &
    Xresid, vtemp1, OSC(Col_Beg:Col_End)%v, Vused)

! ## If all rain has been allocated, exit.
!     Else, Pass #3.

Select Case(Status_Flag)
Case(Flag_Continue)
    ! Do next phase.
Case(Flag_Done)
    ! All rain allocated. Jump to end of subroutine.
    Go To 99999
Case(Flag_Error)
    ! Error. Jump to end of subroutine and return.
    Go To 99999
End Select

! #3.0) Pass Number 3:
! #3.1) Collect hours with precipitation > 0.
! #3.2) Allocate the residual rain equally among
!     those hours.
hours_with_ppt = 0

! In this context, Vused == Truth of "Use this hour"
Do iv = Col_Beg, Col_End
    k = iv - Col_Beg + 1      ! Entry in V* arrays
    Vused(k) = .False.

    If (Modulo(iv,25) == 0) Then
        ! Skip 25-th hour of the day.
        ! Do not select this point.
        Cycle
    End If
End Do

```

```

      If ((Abs(Vval(k)) >= Eps0) .And. &
          (Abs(Vval(k)-Missing_Data) >= Eps0)) Then
        ! Vval(k) > 0 and Vval(k) /= Missing_Data
        Vused(k) = .True.
        hours_with_ppt = hours_with_ppt + 1
      End If
    End Do

    If (hours_with_ppt > 0) Then
      rdiv = xresid / hours_with_ppt
      Do iv = Col_Beg, Col_End
        k = iv - Col_Beg + 1      ! Entry in V* arrays
        If (.Not. Vused(k)) Cycle
          Vval(k) = Vval(k) + rdiv
        End Do
        ! All rain allocated. Jump to end of subroutine.
        Status_Flag = Flag_Done
        Go To 99999
      End If

      ! If the sky is opaque, then it rained.

      ! All hours available except 25-th hour
      Do iv = Col_Beg, Col_End
        k = iv - Col_Beg + 1      ! Entry in V* arrays
        Vused(k) = (Modulo(iv,25) /= 0)
      End Do

      at_least_one = Any(vOSC_5_to_10(1:nv))
      Call Accum_based_on_OSC(Status_Flag, Col_Beg, Col_End, &
          Xresid, vOSC_5_to_10, OSC(Col_Beg:Col_End)%v, Vused)

      Select Case(Status_Flag)
      Case(Flag_Continue)
        ! Do next phase.
      Case(Flag_Done)
        ! All rain allocated. Jump to end of subroutine.
        Go To 99999
      Case(Flag_Error)
        ! Error. Jump to end of subroutine and return.
        Go To 99999
      End Select

      ! The sky was clear and it rained. Sigh.
      ! Last resort. Use all hours, regardless.
      ! All hours available except 25-th hour
      Do iv = Col_Beg, Col_End
        k = iv - Col_Beg + 1      ! Entry in V* arrays
        If (Modulo(iv,25) /= 0) Then

```

```

! 1h - 24h
Vused(k) = .False.
vSelect(k) = .True.
Else
! 25h
Vused(k) = .True.
vSelect(k) = .False.
End If
End Do

Call Do_Phase(Col_Beg, Col_End, Xresid, &
vSelect, vSelect, Status_Flag, Vused, Vtemp1)

Select Case(Status_Flag)
Case(Flag_Continue)
! Do next phase.
Go To 99999
Case(Flag_Done)
! All rain allocated. Jump to end of subroutine.
Go To 99999
Case(Flag_Error)
! Error. Jump to end of subroutine and return.
Go To 99999
End Select

```

```

99999 Continue
okay = (Status_Flag == Flag_Done)
Xok = okay
If (okay) Then

! Note:
! %v: value;
! %s: may contain "Missing" flag;
! %f: may contain T_Accumulation

! Note that
! HP(Col_Beg:Col_End)%v = Vval(1:nv)
! HP(Col_Beg:Col_End)%s = T_Estimated
! HP(Col_Beg:Col_End)%f = T_Accumulation
! would set the daily values, causing problems later.

Do iv = Col_Beg, Col_End
k = iv - Col_Beg + 1 ! Entry in V* arrays
If (Modulo(iv,25) == 0) Then
Cycle ! Skip 25th hour
End If

Select Case(HP(iv)%s)

```

```

Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  ! HP(iv) is missing. Replace it with the estimated value.
  HP(iv) = Val_and_Flag(T_Estimated, Vval(k), T_Accumulation)
Case Default
  If (Abs(HP(iv)%v-Vval(k)) >= Eps0) Then
    ! HP(iv) is different from the estimated value.
    ! Replace it with the estimated value.
    HP(iv) = Val_and_Flag(T_Estimated, Vval(k), T_Accumulation)
  End If
End Select
End Do

Else
  ! Algorithms failed.
  Write (ULog, 9150) str_iv_to_ymdh(col_beg), &
    str_iv_to_ymdh(col_end), ddt_marker

9150   Format(lx, '?? Fill_Buckets: ', &
        'Unresolved Accumulation gap from ', &
        a, ' to ', a, '; ddt_marker: ', i0)
  Call Print_Ph3('Fill_Buckets -- Final Failure', &
    Col_Beg, Col_End, &
    Total_ppt, Vused, Vtempl)
End If

End Subroutine Fill_Buckets

Subroutine Do_Phase(Col_Beg, Col_End, Xresid, &
  Xuse_me, Xhas_property, Status_Flag, Xused, Xtempl)

  ! Xuse_me(k) == .T. ==> This element is a candidate, subject to
  !   other restrictions.
  !
  ! Xhas_property(k) == .T. ==> This element possesses the property
  !   in question. Vague enough? Examples of properties are
  !   -- it rained during the kth hour
  !   -- OSC >= 5 during the hour
  !   -- 1 <= OSC <= 4 during the hour
  !
  ! Xused(k) :: Truth of element k was used.
  !   If element k was used before (e.g., previous call to
  !   Do_Phase), then element k will not be selected in this module.
  !
  ! Xtempl -- provides only storage space and carries no information
  !   neither into nor out of the routine. A local array would
  !   do, but I am trying to avoid allocating and deallocating
  !   continuously this array, which may be used up to 5 times
  !   per call to Fill_Buckets.

```

```

! Algorithm:
! 1. Generate the set of candidate hours. Stored the hours
!     in Xcandidate_hour (pointer to Xtemp1)
! 2. Determine if there are hours with bounds, e.g.,
!     if Present Weather(2:2) == "1" (i.e., Moderate rain),
!     then 0.25 cm <= capacity of the bucket <= 0.76 cm.
!     Fill these hours first.
!
! 3. Distribute the remaining rain equally among the buckets.
!
! 4. Distribute the remaining rain among the buckets.

Implicit None
Integer,          Intent(In)    :: Col_Beg, Col_End
Real,             Intent(InOut) :: Xresid
Logical, Dimension(:), Intent(In)  :: Xuse_me
Logical, Dimension(:), Intent(In)  :: Xhas_property
Character(Len(Flag_Done)), Intent(Out) :: Status_Flag
Logical, Dimension(:), Intent(InOut) :: Xused
Logical, Dimension(:), Target, Intent(InOut) :: Xtemp1

! Xcandidate_hour -- temporary internal array; points to Xtemp1.
Logical, Dimension(:), Pointer :: Xcandidate_hour
Real    :: amt, in_residual, mean_ppt
Integer :: iv, k, ncandidates, itimes, ierr
Logical :: at_least_one_bound
Integer, Save :: ddt_marker = 0

ddt_marker = ddt_marker + 1

ierr = 0
in_residual = Xresid
at_least_one_bound = .False.

! First, generate the set of candidate hours.
ncandidates = 0
Xcandidate_hour => Xtemp1
Xcandidate_hour = .False. ! Start with an empty set.
Do iv = Col_Beg, Col_End
  k = iv - Col_Beg + 1 ! Entry in V* arrays

  If (Modulo(iv,25) == 0) Then
    ! Skip 25-th hour of the day.
    ! Do not select this point.
    Cycle
  End If

  ! If this element is not to be used, cycle.
  If (.Not. Xuse_me(k)) Cycle

```

```

! If this element was used before, cycle.
If (Xused(k)) Cycle

! If this element does not have the property, cycle.
If (.Not. Xhas_property(k)) Cycle

Xcandidate_hour(k) = .True.
ncandidates = ncandidates + 1
at_least_one_bound = at_least_one_bound .Or. Vbounds(k)
End Do

If (ncandidates == 0) Then
! No candidates for the given conditions.
! Return with "Continue", since we may still solve
! this accumulation with a different set of conditions.
If (Abs(Xresid) < Eps0) Then
! Success. Xresid == Zero (fuzzily). Make it Zero (exactly).
Xresid = Zero
Status_Flag = Flag_Done
Else
Status_Flag = Flag_Continue
End If
Return
End If

! Loop three times:
!
! First time: fill cells with precipitation bounds with
! the minimum amount and see if that works.
!
! After the first pass all cells have the minimum amount.
!
! Second time: Distribute the remainder equally among the buckets.
!
! Third time: Allocate as much rain as each bucket can contain.
!
! In this way we avoid duplication of code.
! (I dislike duplication of code.)

Loop_Times: Do itimes = 1, 3

! mean_ppt will be used when itimes == 2.
! A <= B :: Abs(B-A) < Eps0 .Or. Eps0 < (B-A)
mean_ppt = Xresid / ncandidates
If ((Abs(minimum_allocation-mean_ppt) < Eps0) .Or. &
(Abs(minimum_allocation-mean_ppt) < Eps0)) Then
! mean_ppt <= minimum_allocation: mean_ppt is smaller than
! the initial precision of the data. Choose a larger value.
mean_ppt = Min(minimum_allocation, Xresid)
End If

```

```

Loop_k: Do iv = Col_Beg, Col_End
      k = iv - Col_Beg + 1      ! Entry in V* arrays

      ! If this element is not to be used, cycle.
      If (.Not. Xcandidate_hour(k)) Cycle Loop_k

      If (Abs(Xresid) < Eps0) Then
        ! Success. Xresid == Zero (fuzzily). Make it Zero (exactly).
        Xresid = Zero
        Exit Loop_k
      Else If (Xresid < -Eps0) Then
        ! Xresid < 0 -- error.
        ierr = ierr + 1
        Exit Loop_k
      End If

      Select Case(itimes)
      Case(1)

        ! Hours with bounds. Example:
        !   if Present Weather(2:2) == "1" (i.e., Moderate rain),
        !   then 0.25 cm <= capacity of the bucket <= 0.76 cm.
        !   Fill these hours first.
        ! If at_least_one_bound == .T., then at least one hour (k)
        ! has bounds. If no hours have bounds, then we can skip
        ! the iteration itimes==1 completely.
        If (.Not. at_least_one_bound) Cycle Loop_Times

        ! If this entry does not has bounds to be respected, get
        ! next hour. In this step we are only considering hours
        ! such that
        !   Vmin <= Vval <= Vmax
        ! where (Vmin, Vmax) /= (0, Huge(0))
        If (.Not. Vbounds(k)) Cycle Loop_k

        ! 9 May 2002 3:24 pm -- Initially, I was allocating
        !   Vval(k) = Vmin(k), i.e., as much rain as
        !   the minimum amount per Present Weather flags.
        !   The problem was that sometimes the amount
        !   of rain detected by an accumulation run
        !   would be Zero, but present weather says
        !   that on hour x, between 0.25 cm and 0.76 cm
        !   of rain fell (for example). This behaviour
        !   was observed with WBAN 23063, Eagle, CO,
        !   iv:189496 1981-10-02 21h.
        !
        ! We will allocate either Vmin or the residual, whichever
        ! is smaller. If the bucket amount Vval is already >= Vmin,
        ! no more precipitation will be added (in this iteration).

```

```

    amt = Min(Vmax(k)-Vval(k), Vmin(k)-Vval(k), Xresid)

Case(2)
    ! Try to allocate equally among the buckets.
    amt = Min(Vmax(k)-Vval(k), mean_ppt, Xresid)

Case(3)
    ! Allocate as much as the bucket will contain.
    amt = Min(Vmax(k)-Vval(k), Xresid)
End Select

amt = Max(amt, Zero)

! If the residue that would be left is less than the
! precision of the instrument, allocate the remainder
! in the current bucket.
If ((Xresid - amt) < minimum_allocation) Then
    amt = Xresid
End If

Vval(k) = Vval(k) + amt
Xresid = Xresid - amt
Xused(k) = .True.
End Do Loop_k

If (ierr /= 0) Then
    ! Some error.
    !If (print_now) Write (ULog, *) '## Do_Phase: Errors. ', &
    !     ' itimes='//Trim(Itoa(itimes)), ' Xresid: ', Xresid, &
    !     '; ddt_marker: ', ddt_marker
    Status_Flag = Flag_Error
    Return
Else If (Abs(Xresid) < Eps0) Then
    ! Success
    !If (print_now) Write (ULog, *) '## Do_Phase: Success itimes='//&
    !     Trim(Itoa(itimes)), '; ddt_marker: ', ddt_marker
    Status_Flag = Flag_Done
    Return
Else If (Xresid < -Eps0) Then
    ! Critical failure. After adding to each bucket the minimum
    ! amount, xresid < 0, i.e., not enough initial ppt was provided.
    !If (print_now) Then
    !     Write (ULog, *) '## Do_Phase: Failure itimes='//Trim(Itoa(itimes)), &
    !     ', Xresid: ', Xresid, ', ddt_marker: ', ddt_marker
    !End If
    Status_Flag = Flag_Error
    Return
Else
    ! We may continue. We may still solve this accumulation.
    !If (print_now) Write (ULog, *) '## Do_Phase: Status_Flag = ', &

```

```

        !      'Flag_Continue, itimes='//Trim(Itoa(itimes))
        Status_Flag = Flag_Continue
    End If
End Do Loop_Times

End Subroutine Do_Phase

Subroutine Print_Ph3(xHeader, Col_Beg, Col_End, Total_ppt, Xused, Xcandidate_hour)

! Print arrays associated with Fill_Buckets and Do_Phase.
Implicit None
Integer,          Intent(In) :: Col_Beg, Col_End
Character(Len=*), Intent(In) :: xHeader
Real,             Intent(In) :: Total_ppt
Logical, Dimension(:), Intent(In) :: Xused, Xcandidate_hour

Integer :: iv, k
Real    :: xs, qmin, qmax

! Observation Indicator  0 or 9  0 = Weather observation made.
!                       9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
!
!   Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
!   Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
!   Xparam(f_OI)%Samson_v10(iv)%s = data_source

Write (ULog, '(1x,4x,a)') Trim(xHeader)
Write (ULog, 9130) 'k  Vmin  Vval  Vmax  * OI PW      OSC Xcandidate_hour,Xused:'
9130 Format (1x, 2x, t28, a)

xs = Zero
Do iv = Col_Beg, Col_End

    If (Modulo(iv,25) == 0) Then
        ! Skip 25-th hour of the day.
        Cycle
    End If

    k = iv - Col_Beg + 1 ! Entry in V* arrays
    qmin = Vmin(k)
    qmax = Min(Vmax(k), 99.999)

    Write (ULog, 9150) &
        str_iv_to_ymdh(iv), &
        k, qmin, Vval(k), qmax, &
        Vpw(k), Nint(OI(iv)%v), OI(iv)%f, &
        Nint(OSC(iv)%v), &

```

```

          Xcandidate_hour(k), Xused(k)
9150      Format (1x, a, &
           i4, 1x, 3f7.3, 1x, &
           a, 1x, i1, 1x, a, &
           i4, &
           L2, L2)

          xs = xs + Vval(k)
      End Do

      Write (Ulog, 9170) Total_ppt, xs
9170      Format (1x, 2x, 'Total_ppt = ', f7.3, 2x, 'Sum(Vval) = ', f7.3)

      End Subroutine Print_Ph3

Subroutine Accum_based_on_OSC(Status_Flag, Col_Beg, Col_End, &
    Xresid, Xcandidate_hour, XOSC_val, Xused)

! <A NAME="Accum_based_on_OSC"> <A HREF="Precip.f90#Accum_based_on_OSC">
! See <A HREF="0notes.txt#Note_27">
!
! Fill Accumulation ranges in Hourly Precipitation data.
! Precipitation: missing values not in an accumulation run
! will be set to Zero.
!
! 19 Feb 2002  3:46 pm -- this is the how things actually appear in the
!                      SAMSON files. See, e.g., 12842_75.txt, from *.z
!
! yy mm dd hh Hourly Precipitation in hundredths of inch;
! -- -- -- -- ----- R0 files report in cm;
! 75 10  4 14      0A
! 75 10  4 15
! 75 10  4 16
! 75 10  4 17
! 75 10  4 18
! 75 10  4 19      41A          41/100 * 2.54 cm/inch = 1.0414 cm
! 75 10  4 20      0A
! 75 10  4 21
! 75 10  4 22
! 75 10  4 23
! 75 10  4 24
! 75 10  5  1      9A

Implicit None
Character(Len(Flag_Done)), Intent(Out) :: Status_Flag
Integer,                    Intent(In)  :: Col_Beg, Col_End
Real,                       Intent(InOut) :: Xresid
Logical, Dimension(:),      Intent(In)  :: Xcandidate_hour ! Entries to use.

```

```

Real,    Dimension(:),    Intent(In)  :: XOSC_val    ! OSC value
Logical, Dimension(:),    Intent(InOut) :: Xused

Integer :: iv, k, hours_with_osc
Real    :: frac_osc, amt, total_amount_osc

If (Abs(Xresid) < Eps0) Then
    ! Success. Xresid == Zero (fuzzily). Make it Zero (exactly).
    Xresid = Zero
    Status_Flag = Flag_Done
    Return
End If

total_amount_osc = Zero
hours_with_osc = 0

Do iv = Col_Beg, Col_End
    If (Modulo(iv,25) == 0) Then
        ! Skip 25-th hour of the day.
        Cycle
    End If
    k = iv - Col_Beg + 1    ! Entry in V* arrays

    If (.Not. Xcandidate_hour(k)) Cycle

    total_amount_osc = total_amount_osc + XOSC_val(k)
    hours_with_osc = hours_with_osc + 1
End Do

If (total_amount_osc > 0) Then
    ! If total_amount_osc > 0, then we will allocate
    ! all the precipitation.
    Status_Flag = Flag_Done
Else If (Abs(total_amount_osc) < Eps0) Then
    ! No OSC.
    Status_Flag = Flag_Continue
    Return
Else
    ! No candidates.
    Status_Flag = Flag_Continue
    Return
End If

frac_osc = Xresid / total_amount_osc
Do iv = Col_Beg, Col_End
    If (Modulo(iv,25) == 0) Then
        ! Skip 25-th hour of the day.
        Cycle
    End If

```

```

k = iv - Col_Beg + 1    ! Entry in V* arrays

! If this element is not to be used, cycle.
If (.Not. Xcandidate_hour(k)) Cycle

If (Abs(Xresid) < Eps0) Then
  ! Success. Xresid == Zero (fuzzily). Make it Zero (exactly).
  Xresid = Zero
  Status_Flag = Flag_Done
  Return
Else If (Xresid < -Eps0) Then
  ! Xresid < 0 -- error.
  Status_Flag = Flag_Error
  Return
End If

! If the residue that would be left is less than the
! precision of the instrument, allocate the remainder
! in the current bucket.
amt = Min(XOSC_val(k) * frac_osc, Xresid)
If ((Abs(minimum_allocation-amt) < Eps0) .Or. &
    (Eps0 < (minimum_allocation-amt))) Then
  ! amt <= minimum_allocation: amt is smaller than
  ! the initial precision of the data. Choose a larger value.
  amt = Min(minimum_allocation, Xresid)
End If

If ((Xresid - amt) < minimum_allocation) Then
  amt = Xresid
End If

Vval(k) = Vval(k) + amt
Xresid = Xresid - amt
Xused(k) = .True.
End Do

End Subroutine Accum_based_on_OSC

Subroutine Accum_Last_Resort(Status_Flag, Col_Beg, Col_End, Xresid)

! <A NAME="Accum_Last_Resort"> <A HREF="Precip.f90#Accum_Last_Resort">
! Fist and Faith! We will not fail! (Haruchai)
!
! #1. If the amount to be distributed is less than or equal to 0.1 inches,
!   allocate the whole amount to the reporting hour (Col_End).
!
! #2. Else, partition in chunks of 0.1 inches, starting with the last
!   hour and moving back. Any residue (i.e., a chunk of less than 0.1 inches)

```

```

!      will be dumped into one of the previously allocated buckets.

Implicit None
Character(Len(Flag_Done)), Intent(Out) :: Status_Flag
Integer,                    Intent(In)   :: Col_Beg, Col_End
Real,                      Intent(InOut) :: Xresid

Integer :: iv, k, nv
Real    :: amt

nv = Col_End - Col_Beg + 1

If ((Abs(minimum_allocation-Xresid) < Eps0) &
    .Or. (Eps0 < (minimum_allocation-Xresid))) Then
  ! A <= B :: Abs(B-A) < Eps0 .Or. Eps0 < (B-A)
  ! Xresid <= minimum_allocation
  ! Case #1. Dump the whole amount in the reporting hour.
  Vval(nv) = Vval(nv) + Xresid

Else

  ! Case #2. Allocate in chunks of minimum_allocation.

Do iv = Col_End, Col_Beg, -1
  If (Modulo(iv,25) == 0) Then
    ! Skip 25-th hour of the day.
    Cycle
  End If

  k = iv - Col_Beg + 1    ! Entry in V* arrays

  If (Abs(Xresid) < Eps0) Then
    ! Xresid <= 0
    ! No more precipitation to distribute. Terminate loop.
    Xresid = Zero
    Exit
  Else If ((Abs(minimum_allocation-Xresid) < Eps0) &
    .Or. (Eps0 < (minimum_allocation-Xresid))) Then
    ! 0 < Xresid <= minimum_allocation.
    ! We are done, one way or the other.
    ! Add the residue to the hour of the measurement.
    Vval(nv) = Vval(nv) + Xresid
    Xresid = Zero
    Exit
  End If

  ! Xresid >= minimum_allocation.
  amt = Min(minimum_allocation, Xresid)
  Vval(k) = Vval(k) + amt
  Xresid = Xresid - amt

```

```
End Do

If (Abs(Xresid) >= Eps0) Then
    ! Dump any remaining amount to the hour of the measurement.
    Vval(nv) = Vval(nv) + Xresid
End If
End If

Status_Flag = Flag_Done

End Subroutine Accum_Last_Resort
End Module Precipitation_module
```

## raw\_data

! Last change: LSR 6 Jun 2002 3:30 pm

Module Process\_Raw\_Data

```
Use Binary_Tree
Use Date_Module
Use Dump_MET
Use Dump_R0
Use ET0
Use Evaporation_module
Use Global_Variables
Use IoSubs
Use Read_Info
Use SAMSON
Use Strings
Use Utils1
Use Utils2
Use Utils5
Implicit None
```

Contains

Subroutine Internal\_Standard(Tests\_Passed)

```
Implicit None
Logical, Intent(Out) :: Tests_Passed

Real :: pan_Ep_day      ! Class A pan evaporation [mm/day]
Real :: pan_Rs, pan_up6d, pan-Ta, pan_RH, pan_P
Real :: pan_EpTest
Logical :: okay

Real :: FWS_Result, E_fws
Real :: FWS_Rs, FWS_Rdiff, FWS-Ta, FWS_Dew, FWS_u4d, FWS_RH, FWS_P
Integer :: nerrors

Write (ULog, *)

nerrors = 0

Call Stat_Test(ULog, okay)
If (okay) Then
  Write(6, *) '## Stats tests passed.'
  Write(ULog, *) '## Stats tests passed.'
Else
  Write(6, *) '## Stats tests failed.'
  Write(ULog, *) '## Stats tests failed.'
```

```

    nerrors = nerrors + 1
End If

Call Test_Accumulation(okay)
If (.Not. okay) nerrors = nerrors + 1

Call Test_Et0(okay)
If (okay) Then
    Write(6, 9130) 'passed'
    Write(ULog, 9130) 'passed'
9130    Format (1x, '## Tests of Et0: ', a, '.')
Else
    Write(ULog, 9130) 'failed'
    nerrors = nerrors + 1
End If

pan_Ta = 15.2      ! Ta = 15.2 °C
pan_up6d = 261    ! u_p = 261 km/day
pan_Rs = 5964     ! Rs = 5964 W h m^-2 day^-1
pan_RH = 43.7    ! RH = 43.7%
pan_P = 78.1     ! P = 78.1 kPa (at elevation = 2200 meters)
pan_EpTest = 7.32 ! Ep = 7.32 mm/day

Call Compute_Ep(pan_Ta, pan_up6d, pan_Rs, pan_RH, pan_P, pan_Ep_day, okay)
If (Abs(pan_EpTest-pan_Ep_day) < 0.01) Then
    Write (6, 9150) '## Test Ep passed: ', pan_EpTest, pan_Ep_day
    Write (ULog, 9150) '## Test Ep passed: ', pan_EpTest, pan_Ep_day
Else
    nerrors = nerrors + 1
    Write (6, 9150) '?? Test Ep failed (expected,computed): ', pan_EpTest, pan_Ep_day
    Write (ULog, 9150) '?? Test Ep failed (expected,computed): ', pan_EpTest, pan_Ep_day
End If
9150 Format (1x, a, lp2g17.6)

! Test case: 0.97Rdiff + 0.94(Rs-Rdiff) == 5964
FWS_Rs = 5964 / 0.94 ! Watt hour meter^-2 day^-1
FWS_Rdiff = 0       ! same units as Rs
FWS_Ta = 15.2      ! °C
FWS_Dew = 3.0      ! °C
FWS_P = 78.1       ! kPa
FWS_u4d = 261.0    ! km/day
FWS_RH = 43.7      ! %; unused by the current formulation
FWS_Result = 0.08  ! mm/day
Call Compute_E_fws(FWS_Rs, FWS_Rdiff, FWS_Ta, FWS_Dew, FWS_u4d, FWS_RH, FWS_P, E_fws)
If (Abs(FWS_Result-E_fws) < 0.01) Then
    Write (6, 9170) '## Test E_fws passed: ', FWS_Result, E_fws

```

```

        Write (ULog, 9170) '## Test E_fws passed: ', FWS_Result, E_fws
    Else
        nerrors = nerrors + 1
        Write (6, 9170) '?? Test E_fws failed (expected,computed): ', FWS_Result, E_fws
        Write (ULog, 9170) '?? Test E_fws failed (expected,computed): ', FWS_Result, E_fws
    End If
9170 Format (1x, a, 1p2g17.6)

```

```

Tests_Passed = (nerrors == 0)
If (Tests_Passed) Then
    Write (6, 9170) '## Internal_Standard tests passed.'
    Write (ULog, 9170) '## Internal_Standard tests passed.'
Else
    Write (6, 9170) '?? Internal_Standard tests failed.'
    Write (ULog, 9170) '?? Internal_Standard tests failed.'
End If
!Stop '@ Internal_Standard'

```

```
End Subroutine Internal_Standard
```

```
Subroutine Driver0()
```

```

    Implicit None

    Type(Site_Info), Pointer :: Xstations ! Pointer to root of tree
    Character(Len=80)       :: Filename
    Logical                 :: Tests_Passed

    ! Initialize the binary tree.
    Call Initialize_Binary_Tree(Xstations)

    ! Get basic info for all stations.
    Call Read_SAMSON_Station_Notes(Xstations)
    Call Read_Elevations(Xstations)
    Call Read_Info2(Xstations)
    Write(ULog, '(1x,a,i0)') 'Maximum_Text_Length = ', Maximum_Text_Length
    If (Errors_Detected) Return

    Call Display_Station_Info(Xstations)
    If (Errors_Detected) Return

    Call Internal_Standard(Tests_Passed)
    If (.Not. Tests_Passed) Then
        Errors_Detected = .True.
        Return
    End If

```

```

! Read and process the list of SAMSON WBAN stations.
Filename = '1.samson.list.txt' ! <A HREF="1.samson.list.txt#$1">
Call Read_R0_List(Filename, Xstations)

End Subroutine Driver0

Subroutine Read_R0_List(Filename, Xstations)

Implicit None
Character(Len=*), Intent(In) :: Filename
Type(Site_Info), Pointer    :: Xstations ! Pointer to root of tree

Character(Len=132) :: tbuf
Character(Len=10)  :: wban
Logical           :: xOK, Node_Was_New, loop_once
Integer          :: j, nbuf, jin, ios, nstations
Integer          :: nerrors
Type(Site_Info), Pointer :: xWBAN

Errors_Detected = .False.
nerrors = 0
Call IORead(jin, Filename, ok=xOK)
If (.Not. xOK) Then
    Errors_Detected = .True.
    Return
End If

loop_once = .False.
nstations = 0
ReadFile: Do
    Read (jin, '(a)', iostat = ios) tbuf
    If (ios /= 0) Exit ! End-Of-File or Error

    nbuf = Len_trim(tbuf)
    If (nbuf <= 0) Cycle ReadFile ! Empty line

    ! Remove comments from the line.
    j = Scan(tbuf(1:nbuf), '!#')
    If (j > 0) Then
        tbuf(j:nbuf) = ''
        nbuf = Len_trim(tbuf(1:j))
        If (nbuf <= 0) Cycle ReadFile ! Empty line
    End If

    ! Get the WBAN number or command
    j = 1
    Call GetQword(tbuf, j, wban)

    Select Case(LowerCase(Trim(wban)))

```

```

Case('exit', 'end', 'quit')
  Exit ReadFile

  !Case('senegal')
  !   loop_once = .True.
  !   Call Add_Senegal(Xstations, wban)
  !   Call Display_Station_Info(Xstations, .True.)

Case Default
End Select

Call Get_Node(Xstations, wban, Node_Was_New, xWBAN)
If (Node_Was_New) Then
  ! Error: all nodes should be defined by now.
  Errors_Detected = .True.
  Write (6, *)      '?? Read_R0_List: ', Trim(wban), ' not defined.'
  Write (ULog, *)  '?? Read_R0_List: ', Trim(wban), ' not defined.'
  Stop '?? Read_R0_List: all nodes should be defined by now.'
End If

Call FLushAll()

nstations = nstations + 1
Call Process_One_WBAN_Station(xWBAN, nstations)

Call FLushAll()

If (Errors_Detected) Then
  nerrors = nerrors + 1
  Errors_Detected = .False.
End If

If (loop_once) Exit ReadFile
End Do ReadFile

Errors_Detected = (nerrors /= 0)

Write(6, 9130) nstations
Write(ULog, 9130) nstations
9130 Format(1x, 'Number of stations processed: ', i0)

End Subroutine Read_R0_List

Subroutine Process_One_WBAN_Station(xWBAN, nstations)

! Read all WBAN files associated with the given xWBAN
! fill missing values et al.
! Generate r0 files for all years

```

```

Implicit None
Type(Site_Info), Pointer      :: xWBAN  ! Pointer to a WBAN
Integer,                Intent(In) :: nstations

Integer                :: yy, yyyy, nq0
Logical                :: Empty_File    ! Empty or non-existent file.
Character(Len=80)     :: tbuf, xq0
Logical                :: Have_Precip_Data
Type(Name_type)       :: Xnames
Type(FileInfo)        :: WF10, WF11
Type(Timing_Type)    :: T
Logical                :: Some_Years_Missing, All_Years_Missing

If (.Not. Associated(xWBAN)) Then
  Write (6,*)      '?? Process_One_WBAN_Station: xWBAN not associated'
  Write (ULog,*)  '?? Process_One_WBAN_Station: xWBAN not associated'
  Stop '?? Process_One_WBAN_Station: xWBAN not associated'
End If

Write (xq0, '(i0)') nstations
nq0 = Len_trim(xq0)

! Point global variable to the current WBAN station.
pWBAN => xWBAN
Have_ppt_Obs_hourly_data = .False.
Have_ppt_Obs_daily_data = .False.

Call xTiming(T, TBegin, Id='!!! '//Trim(pWBAN%WBAN)//': '//Trim(pWBAN%Text))

Write(ULog, 9130) nstations, &
  Trim(pWBAN%WBAN), & ! 14914
  Trim(pWBAN%Text)    ! Fargo, ND
9130 Format (///, &
  1x, 10('#===#'), /, &
  1x, '#===# ', i0, ': Processing ', a, ': ', a)

Errors_Detected = .False.

Call Station_With_Missing_Data(pWBAN%WBAN, Have_Precip_Data)
If (Have_Precip_Data) Then
  Write(ULog, 9150) 'Yes'
Else
  Write(ULog, 9150) 'No'
End If
9150 Format (1x, '#===# ', 'Have_Precip_Data: ', a)

Write(ULog, *)

Call ToTTY(xq0(1:nq0)//': Process_One_WBAN_Station '//&
  Trim(pWBAN%WBAN)//': '//Trim(pWBAN%Text))

```

```

Call Generate_File_names()

Call Initialize_Output_Directory()

Maximum_Horizontal_Visibility = Zero
Maximum_Ceiling_Height = Zero

! Algorithm:
!   = Find all WBAN*.z files
!   = Allocate storage
!   = Initialize storage
!   = For each wban_yy.z
!     + Collect data
!     + Issue error messages
!     + Store data & stats
!   = Process WBAN
!   = Deallocate storage

! Procedure:
! Do yyyy = MinYear, MaxYear
!   Decompress file wban_yy.z -> wban_yy.txt (in some temporary directory)
!   Open wban_yy.txt
!     check headers against the information in pWBAN
!     Read data, copy selected fields to pWBAN
!     Normalize windspeed to height = 10 meters
!   Open Samson_v1.1/wban_yy.txt
!     Read data, copy selected fields to pWBAN
!     Detect missing values and other problems
!   r0_file <- v:\r0\AK\wban_yy.txt
!   Write data to r0_file
!   Save r0_file
! End Do
! make sure all years were written.

! Allocate storage.
Call Allocate_SAMSON_arrays()

Year_Data%SAMSON_v10 = 0
Year_Data%SAMSON_v11 = 0

Write(ULog, *)
Do yyyy = MinYear, MaxYear
  yy = Modulo(yyyy,100)

  ! Initialize file information derived type.
  Call Initialize_FileInfo(WF10, yyyy)
  Call Initialize_FileInfo(WF11, yyyy)

  ! Get all file names associated with this WBAN and year.

```

```

Call Get_File_Names(Trim(pWBAN%WBAN), yy, Xnames)
Year_Names(yyyy) = Xnames

! Open and read SAMSON version 1.0 file
Call Read_SAMSON_v1x(Trim(Xnames%Samson_v10), WF10, &
    Is_v11=.False., Empty_File=Empty_File)
If (.Not. Empty_File) Then
    Year_Data(yyyy)%SAMSON_v10 = Year_Data(yyyy)%SAMSON_v10 + 1
End If

! Open and read SAMSON version 1.1 file
Call Read_SAMSON_v1x(Trim(Xnames%Samson_v11), WF11, &
    Is_v11=.True., Empty_File=Empty_File)
If (.Not. Empty_File) Then
    Year_Data(yyyy)%SAMSON_v11 = Year_Data(yyyy)%SAMSON_v11 + 1
End If
End Do

! Gather information on what years to issue.
Call Issue_Years(Some_Years_Missing, All_Years_Missing)

Call Str_years(tbuf)
Write (ULog, '(//,lx,a,a,/)') '## Years present:', Trim(tbuf)

If (All_Years_Missing) Then
    ! This year was missing; do not issue it.
    Write(ULog, 9170) &
        Trim(pWBAN%WBAN), & ! 24013
        Trim(pWBAN%Text) ! Minot, ND
9170    Format (///, &
        lx, 45('?'), /, &
        lx, '?? All years missing for station ', a, ': ', a, /, &
        lx, '?? No files will be issued.', /, &
        lx, 45('?'), /)
    Go To 99999
End If

! *** Generate functions for missing data et al.
! *** Fill missing data.
Call Process_Set()

Call Process_Evaporation_Data()

! Create R0 files.
Call ToTTY('Dumping R0 files')
Do yyyy = MinYear, MaxYear
    If (Issue_This_Year(yyyy)) Then
        Call Dump_R0_One_Year(yyyy)
    End If

```

```

End Do

! Create MET file.
Call Dump_MET_file()

Call Meta_Data_File()

! Dump ranges
!Write (ULog, 9430)
9190 Format (//, &
      1x, '*** WARNING: "hourly" header notwithstanding, the following ', /, &
      1x, '          ranges include the daily values', /)

!Do jpar = 1, f_end
!  ! Ranges et al. have no meaning for the Observation Indicator
!  If (jpar == f_OI) Cycle ! Observation Indicator
!
!  Call Stat_Output(ULog, Xp_ranges(jpar))
!End Do

! Release storage.

99999 Continue
Call Deallocate_SAMSON_arrays()

!* <A NAME="To Do: remember to deallocate storage in WF10 & WF11">
Call xTiming(T, TEnd)

If (Errors_Detected) Then
  tbuf = ' !!!~failed: ' // Trim(pWBAN%WBAN)//': '//Trim(pWBAN%Text)
Else
  tbuf = ' !!!~passed: ' // Trim(pWBAN%WBAN)//': '//Trim(pWBAN%Text)
End If
Call xTiming(T, TPrint, ULog, Id=Trim(tbuf))

End Subroutine Process_One_WBAN_Station

Subroutine Get_File_Names(WBAN, YY, Xnames)

! See Assumption[1]. All weather data is located in one directory,
! namely Raw_Data_dir.

Implicit None
Character(Len=*),          Intent(In)  :: WBAN
Integer,                  Intent(In)   :: YY
Type(Name_type), Target, Intent(Out) :: Xnames

Integer                    :: i

```

```

Character(10)      :: tversion
Character(MaxNamLen) :: tname, t0
Character(MaxNamLen), Pointer :: p
Character(MaxNamLen), Dimension(:), Pointer :: YList => Null()

Write (tname, 9130) Trim(WBAN), YY ! '13873_yy'
9130 Format(a, '_', i2.2)

Do i = 1, 2
  Select Case(i)
  Case(1)
    ! Look for SAMSON v1.0 file
    tversion = 'v1.0'
    t0 = Trim(tname) // '.z'
    p => Xnames%Samson_v10
  Case(2)
    ! Look for SAMSON v1.1 file
    tversion = 'v1.1'
    t0 = Trim(tname) // '.txt'
    p => Xnames%Samson_v11
  End Select

  p = ''
  Call Find_Files(Xname=t0, Xdir=Raw_Data_dir, YList=YList)
  If (.Not. Associated(YList)) Then
    Write (ULog, 9150) Trim(tversion), Trim(t0)
9150   Format(1x, '?? Get_File_Names: Could not find SAMSON ', a, ' file ', a)

    Else If (Ubound(YList,1) > 1) Then
      Write (ULog, 9170) Trim(tversion), Trim(t0)
9170   Format(1x, '?? Get_File_Names: Found more than one SAMSON ', a, ' file ', a)

    Else
      p = YList(1)
    End If
  End Do
  If (Associated(YList)) Deallocate(YList)

End Subroutine Get_File_Names

Subroutine Initialize_FileInfo(tFI, yyyy)

  ! Initialize file information derived type.

  Implicit None
  Type(FileInfo), Intent(InOut) :: tFI
  Integer,          Intent(In)   :: yyyy

  tFI%Head%WBAN = ''

```

```

tFI%Head%Text = ''
tFI%Head%Lat  = Coords('', 0, 0)
tFI%Head%Lon  = Coords('', 0, 0)
tFI%Head%Elev = 0
tFI%Head%TZ   = 0

tFI%yyyy = yyyy
tFI%Check_Header_Info = (yyyy == MinYear)

! Expected number of observations
If (IsLeapYear/yyyy) Then
    tFI%Expected_Ndays = 366
Else
    tFI%Expected_Ndays = 365
End If
Allocate(tFI%Every_Hour_Present(tFI%Expected_Ndays,Nhours))
tFI%Every_Hour_Present = 0

End Subroutine Initialize_FileInfo

Subroutine Deallocate_FileInfo(tFI)

! Deallocate file information derived type.
! Wed Nov 21 10:58:30 2001 -- at this time only one statement

Implicit None
Type(FileInfo), Intent(InOut) :: tFI

! Release space allocated to the list
!Call Add_Error(tFI%pHead, tFI%pTail, &

Deallocate(tFI%Every_Hour_Present)

End Subroutine Deallocate_FileInfo

End Module Process_Raw_Data

```

## read\_info

! Last change: LSR 6 Aug 2002 5:14 pm

Module Read\_Info

```
Use Binary_Tree
Use Floating_Point_Comparisons
Use GetNumbers
Use GetNumbers
Use Global_Variables
Use IoSubs
Use Strings
Use Utils0
Implicit None
```

Contains

Subroutine Read\_SAMSON\_Station\_Notes(Xstations)

```
Implicit None
Type(Site_Info), Pointer :: Xstations ! Pointer to root of tree

Character(Len=132) :: tname, tbuf
Character(Len=50)  :: wban, wtext, wstate, wtz
Integer           :: ierr, ios, jin, i, j, k, icol, npos, nstations
Integer           :: ksign
Logical           :: xok, Node_Was_New
Real              :: welev, rval, rlat, rlon
Type(Coords)     :: wlat, wlon, p
Type(Site_Info), Pointer :: xWBAN
```

```
Call ToTTY('Read_SAMSON_Station_Notes')
Errors_Detected = .False.
```

```
tname = 'v:\Docs\SAMSON Station Notes.txt'
Call IORead(jin, tname)
```

```
!SITES          WBAN      LAT      LONG      ELEV  TZ      FOOTNOTES
!
!123456789-123456789-
!AK Anchorage   26451     N 61 10  W 150 1     35   +9(V)   2,8
!  This station has little or no hourly precipitation data.
!
!CO Boulder     94018     N 40 01  W 105 15  1634  +7(T)   2
!  Denver      23062     N 39 46  W 104 52  1610
!
!MN Minneapolis/ 14922     N 44 52  W 93 13     255  +6(S)   2
!  St. Paul
```

```

!
!PI Guam          41415      N 13 33   W-144 49   110   -10(K)

! Skip lines until 'SITES' is found.
Call Skip_Until('SITES', Trim(tname), jin)

icol = 20 ! WBAN number starts at this column.
nstations = 0
xWBAN => Null()
Loop: Do
  Read (jin, '(a)', Iostat = ios) tbuf
  If (ios /= 0) Exit Loop
  ierr = 0

  ! Skip empty lines.
  If (Len_trim(tbuf) == 0) Cycle Loop

  ! These lines make processing more difficult. Ignore them.
  !           123456789-123456789-1234567
  If (tbuf(1:27) == ' This station has little ') Cycle Loop
  If (tbuf(1:05) == '') Cycle Loop
  !           123456789
  If (tbuf(1:09) == 'Footnotes') Exit Loop ! No more station information.
  ! Secondary station info may be ignored (e.g., see Denver above).
  If (tbuf(1:03) == '') Cycle Loop

  ! Remaining lines contain useful information.
  nstations = nstations + 1

  wstate = tbuf(1:2)
  wtext = tbuf(4:icol-1)
  npos = icol
  Call GetQword(tbuf, npos, wban)

  ! loop twice, first for the latitude, then the longitude.
  Do i = 1, 2
    npos = NextNb(tbuf, npos) ! Skip WhiteSpace

    p%Letter = tbuf(npos:npos) ! N
    npos = npos + 1

    Call Get_Numbers(tbuf, npos, xok, p%degrees) ! 61
    If (.Not. xok) Then
      Write (ULog, 9130) 'degrees', Trim(tbuf)
      Format (lx, '?? Read_SAMSON_Station_Notes(', a, '): Errors in >>', a, '<<')
      Errors_Detected = .True.
      Return
    End If

    Call Get_Numbers(tbuf, npos, xok, p%minutes) ! 10

```

9130

```

If (.Not. xok) Then
  Write (ULog, 9130) 'minutes', Trim(tbuf)
  Errors_Detected = .True.
  Return
End If

! Why ksign et al.? Consider Longitude for Guam (see above).
ksign = Sign(1.0, p%degrees)
rval = ksign * (Abs(p%degrees) + Abs(p%minutes)/60.0) * Degrees_to_Radians
If (i == 1) Then
  ! Latitude: N or S
  wlat = p
  If (p%Letter == 'N') Then
    rlat = rval
  Else
    rlat = -rval
  End If
Else
  ! Longitude: E or W
  wlon = p
  If (p%Letter == 'W') Then
    rlon = rval
  Else
    rlon = -rval
  End If
End If
End Do

! Get elevation (meters)
Call Get_Numbers(tbuf, npos, xok, welev)
If (.Not. xok) Then
  Write (ULog, 9150) Trim(tbuf(npos:))
9150   Format (1x, '?? Read_SAMSON_Station_Notes: Elevation errors in >>', a, '<<')
  Errors_Detected = .True.
  Return
End If

! Get time zone
Call GetQword(tbuf, npos, wtz)

! If the last character is a '/' then the next
! line contains the rest of the text.
j = Len_trim(wtext)
If (wtext(j:j) == '/') Then
  Read (jin, '(a)', Iostat = ios) tbuf
  If (ios /= 0) Then
    Write (ULog, 9170)
9170   Format (1x, '?? Read_SAMSON_Station_Notes: Expecting another line since ', &
    'site ends in "/"')
    Errors_Detected = .True.
  End If
End If

```

```

        Return
    End If

    wtext(j+1:) = Adjust1(tbuf(1:icol-1))
    j = Len_trim(wtext)
End If
Maximum_Text_Length = Max(Maximum_Text_Length, j)
wtext(j+1:) = ', ' // Trim(wstate)

!!!Write(ULog,*) '   wtext ...: ', Trim(wtext)
!!!Write(ULog,*) '   wban ...: ', Trim(wban)
!!!Write(ULog,*) '   wlat ...: ', wlat, rlat
!!!Write(ULog,*) '   wlon ...: ', wlon, rlon
!!!Write(ULog,*) '   welev ..: ', welev
!!!Write(ULog,*) '   wtz ....: ', Trim(wtz)

!! list of samson stations for 1.samson.list.txt
!Write(99, '(2x, a, " !!! ", a)') Trim(wban), Trim(wtext)

! All nodes should be new.
Call Get_Node(Xstations, wban, Node_Was_New, xWBAN)
If (Node_Was_New) Then
    ! New node initialization.
    xWBAN%WBAN = wban
    xWBAN%State = wstate
    xWBAN%Text = wtext
    xWBAN%Lat = wlat
    xWBAN%Lat_radians = rlat
    xWBAN%Lon = wlon
    xWBAN%Lon_radians = rlon
    xWBAN%Elev = welev
    xWBAN%TZ = wtz
    k = Index(wtz, '(') - 1
    Read(wtz(1:k), *) xWBAN%iTZ
Else
    Errors_Detected = .True.
    Write (ULog, 9190) Trim(xWBAN%WBAN), Trim(xWBAN%Text)
9190   Format (lx, '?? Read_SAMSON_Station_Notes: Duplicated station ', a, ': ', a)
    Return
End If
End Do Loop

Write(ULog, '(lx,a,i0)') '          nstations = ', nstations

Call IOClose(jin)
End Subroutine Read_SAMSON_Station_Notes

Subroutine Read_Elevations(Xstations)

```

```

! Get station Anemometer Elevation (in Feet) & Dates
!
! To generate the (text) elevation file:
! # Open WordPerfect file
!   ...\\3MET\\raw.data\\Elevation\\Anemometer heights.wpd
! # Make sure there are no "?"
! # Choose Print.
!   Select Printer: Generic Text only
!   Print Details: Print to file.
! # WP Menu:  Format/Page/Page Setup
!   Select Landscape
!   Page margins: set all to Zero.
! # Return to the document and manually increase
!   the width of the columns
! # Print the document.
! # Edit the *.prn document:
!   Align fields (use Ed4W)
!   Remove ^M^L, Ed4w regex \\013\\012 (decimal)
!   Add "+----" line -- see below.
! # Done.
!

! File fragment:
!
!14733          20          1959-08-24
!              33          1977-05-18
!
!14734          50          1941-12-10
!              20          1965-06-30

Implicit None
Type(Site_Info), Pointer :: Xstations ! Pointer to root of tree

Character(Len=50)  :: wban
Character(Len=132) :: tbuf, tname
Integer           :: ios, jin

Logical           :: xok, Node_Was_New
Integer           :: npos
Real              :: elevation_ft
Character(Len=15) :: edate
Type(Site_Info), Pointer :: xWBAN ! Pointer to the current WBAN station.

Call ToTTY('Read_Elevations')
Errors_Detected = .False.

! See <A HREF="z:\\Elevation\\Anemometer heights.prn#$1">
! 14914   86 feet = 26.213 meter   1953-11-01
!        28 feet = 8.5344 meter   1963-05-11
!tname = 'v:\\Docs\\Anemometer heights.prn'

```

```

tname = 'z:\Elevation\Anemometer heights.prn'
Call IORead(jin, tname)

xWBAN => Null()
Loop: Do
  Read (jin, '(a)', Iostat=ios) tbuf
  If (ios /= 0) Exit Loop

  ! Skip comments.
  If (tbuf(1:1) == '!') Cycle Loop
  ! Skip empty lines.
  If (Len_trim(tbuf) == 0) Cycle Loop

  ! Not an empty line. If field 1 (the Id) is non-blank,
  ! then the line contains a new station.
  npos = 5
  wban = tbuf(1:npos)
  If (Len_trim(wban) > 0) Then
    ! Start of elevations for a new station.
    Call Get_Node(Xstations, wban, Node_Was_New, xWBAN)
    If (Node_Was_New) Then
      ! Error: all nodes should be defined by now.
      Errors_Detected = .True.
      Write (6, *) '?? Read_Elevations: ', Trim(wban), ' not defined.'
      Write (ULog, *) '?? Read_Elevations: ', Trim(wban), ' not defined.'
      Stop '?? Read_Elevations: all nodes should be defined by now.'
    End If
  End If

  npos = npos + 1
  Call Get_Numbers(tbuf, npos, xok, elevation_ft)
  If (.Not. xok) Then
    Write (ULog, 9130) Trim(tbuf)
9130   Format (1x, '?? Read_Elevations: Elevation Errors in >>', a, '<<')
    Errors_Detected = .True.
    Return
  End If

  Call GetQword(tbuf, npos, edate)

  Call Store_Elevation(xWBAN, edate, elevation_ft)
End Do Loop
Call IOClose(jin)

End Subroutine Read_Elevations

Subroutine Store_Elevation(xWBAN, edate, elevation_ft)

  ! Store elevation data for the given station.

```

```

Implicit None
Type(Site_Info), Pointer      :: xWBAN ! Intent(InOut)
Character(Len=*), Intent(In) :: edate
Real,                Intent(In) :: elevation_ft

Integer, Pointer :: Nelev
Integer :: yyyy, mm, dd, ios
Logical :: ok
Type(ElevationBlock), Dimension(:), Pointer :: Elev_Directives

Errors_Detected = .False.
If (.Not. Associated(xWBAN)) Then
  Write (6,*) '?? Store_Elevation: xWBAN not associated'
  Write (ULog,*) '?? Store_Elevation: xWBAN not associated'
  Stop '?? Store_Elevation: xWBAN not associated'
End If

Nelev => xWBAN%Nelev
Elev_Directives => xWBAN%Elev_Directives

If (Nelev >= Ubound(Elev_Directives,1)) Then
  Write (ULog, *) '?? Increase Dimension of Elev_Directives.'
  Errors_Detected = .True.
  Return
End If

Nelev = Nelev + 1

Read (edate , '(i4,lx,i2,lx,i2)', iostat = ios) yyyy, mm, dd
If (ios /= 0) Then
  Errors_Detected = .True.
  Write (6,*) '?? Store_Elevation: Problems With edate "', Trim(edate), '"'
  Write (ULog,*) '?? Store_Elevation: Problems With edate "', Trim(edate), '"'
  !Stop '?? Store_Elevation: Problems With edate'
End If

Elev_Directives(Nelev)%Julian_Day = Jd(yyyy, mm, dd)

! The input elevation is expressed in feet.
! The elevation is stored in meters.
! 1 foot == 0.30480 meter == 304.80 mm
!
! <A HREF="e:\5\Larry\Make-MET-files\readf.f90#$1">
Elev_Directives(Nelev)%Elevation_meter = elevation_ft * feet_to_meter

! Make sure entries in chronological order
If (Nelev > 1) Then
  ok = Elev_Directives(Nelev-1)%Julian_Day < &
      Elev_Directives(Nelev)%Julian_Day
  If (.Not. Ok) Then

```

```

        Errors_Detected = .True.
        Write (ULog, 9130) Trim(xWBAN%WBAN), Trim(xWBAN%Text)
        Write (ULog, 9150)
9130     Format(1x, '?? Store_Elevation: ', a, ': ', ' ', a)
9150     Format(1x, '   Elevation Data not in chronological order.')
        Return
    End If
End If
End Subroutine Store_Elevation

```

```
Subroutine Read_Info2(Xstations)
```

```

    Implicit None
    Type(Site_Info), Pointer :: Xstations ! Pointer to root of tree

    Character(Len=50) :: wban
    Character(Len=132) :: tbuf, tname
    Integer           :: ios, jin
    Logical           :: xok, Node_Was_New
    Type(Site_Info), Pointer :: xWBAN ! Pointer to the current WBAN station.

    ! This is a comma-delimited file.
    Character(Len=*) , Parameter :: Xdelim = ','

    Integer, Parameter :: dimz = 13
    Integer, Parameter :: max_len = 50
    Character(Len=max_len), Dimension(dimz) :: zfields
    Character(Len=max_len) :: q0buf
    Integer           :: nargs, npos, nb
    Integer           :: nlines, tlen, q0len

    Real :: rlat, rlon, delta_sec

    Call ToTTY('Read_Info2')
    Errors_Detected = .False.

    ! See <A HREF="v:\Docs\WBAN.txt#$1">
    tname = 'v:\Docs\WBAN.txt'
    Call IORead(jin, tname)

    nlines = 0
    xWBAN => Null()
    Loop: Do
        Read (jin, '(a)', Iostat=ios) tbuf
        If (ios /= 0) Exit Loop

        ! Skip comments.
        If (tbuf(1:1) == '!') Cycle Loop
        ! Skip empty lines.

```

```

If (Len_trim(tbuf) == 0) Cycle Loop
tlen = Len_trim(tbuf)
nlines = nlines + 1

! Split the line.
nargs = 0
zfields = ''
npos = 1

!Write (uout1, 9230) nlines, tbuf(1:tlen)
9130 Format (/ , lx, 'Line ', i3, ': ', a)

!WBAN,Station,State,LRR,MLRA,Lat.,Long.,Elev.,T,R.H.,Wind,Precip,AvailableData
! Line 152: 94728, NYC (Central Park/LGA), NY, S, 149B, 40.77, -73.90, 3, 12.4, 63, 5.5, 1070, 1961-1990
!   1: 94728
!   2: NYC (Central Park/LGA)
!   3: NY
!   4: S
!   5: 149B
!   6: 40.77
!   7: -73.90
!   8: 3
!   9: 12.4
!  10: 63
!  11: 5.5
!  12: 1070
!  13: 1961-1990

Split_the_line: Do
! Terminate if End-of-Line
If (npos > tlen) Exit Split_the_line

! Find next delimiter
nb = Scan(tbuf(npos:tlen), Set=Xdelim)
If (nb > 0) Then
    nb = nb + npos - 1 - 1
Else
    ! Last field has no trailing delimiter.
    nb = tlen
End If

If (nargs >= dimz) Then
    Errors_Detected = .True.
    Write (*, *) ' ?? Increase the dimension of zfields.'
    Exit Loop
End If

If ((nb-npos+1) > max_len) Then
    Errors_Detected = .True.
    Write (*, *) ' ?? Increase the character length of zfields.'

```

```

        Exit Loop
    End If

    nargs = nargs + 1
    zfields(nargs) = tbuf(npos:nb)
    !Write (uout1, 9330) nargs, Trim(zfields(nargs))
9150   Format (1x, 1x, i3, ': ', a)

    ! Note that tbuf(nb + 1) == Xdelim
    npos = nb + 2

End Do Split_the_line

wban = zfields(1)
If (Len_trim(wban) > 0) Then
    Call Get_Node(Xstations, wban, Node_Was_New, xWBAN)
    If (Node_Was_New) Then
        ! Error: all nodes should be defined by now.
        Errors_Detected = .True.
        Write (6, *)    '?? Read_Info2: ', Trim(wban), ' not defined.'
        Write (ULog, *) '?? Read_Info2: ', Trim(wban), ' not defined.'
        Stop '?? Read_Info2: all nodes should be defined by now.'
    End If
End If

! Replace the station text.
q0buf = Trim(zfields(2)) // ', ' // Trim(zfields(3))
q0len = Len_trim(q0buf)

If (xWBAN%Text /= q0buf(1:q0len)) Then
    !Write (ULog, 9430) Trim(xWBAN%Text), q0buf(1:q0len)
9170   Format (1x, 'Replacing "', a, '" with "', a, '"')
End If

Maximum_Text_Length = Max(Maximum_Text_Length, q0len)

xWBAN%Text = q0buf(1:q0len)

!Read (zfields(6), *) rlat
!rlat = Sign(rlat,xWBAN%Lat_radians) * Degrees_to_Radians
!
!Read (zfields(7), *) rlon
!rlon = Sign(rlon,xWBAN%Lon_radians) * Degrees_to_Radians
!
!delta_sec = (xWBAN%Lat_radians - rlat) * Radians_to_Degrees * 60
!
!! A preview of the differences show all Abs(deltas) < 0.20 minutes
!!If (Abs(xWBAN%Lat_radians - rlat) > 0.20) Then
!   Write (6, *)    '?? Read_Info2: Lats different (N,r,delta): ', xWBAN%Lat_radians, rlat, xWBAN%Lat_radians - rlat, delta_sec
!   Write (ULog, *) '?? Read_Info2: Lats different (N,r,delta): ', xWBAN%Lat_radians, rlat, xWBAN%Lat_radians - rlat, delta_sec

```

```
!End If

!delta_sec = (xWBAN%Lon_radians - rlon) * Radians_to_Degrees * 60
!If (Abs(xWBAN%Lon_radians - rlon) > 0.20) Then
!  Write (6, *) '?? Read_Info2: lons different (N,r,delta): ', xWBAN%Lon_radians, rlon, xWBAN%Lon_radians - rlon, delta_sec
!  Write (ULog, *) '?? Read_Info2: lons different (W,r,delta): ', xWBAN%Lon_radians, rlon, xWBAN%Lon_radians - rlon, delta_sec
!End If

End Do Loop
Call IOClose(jin)

End Subroutine Read_Info2

End Module Read_Info
```

## Red\_Black

```
!      Last change:  LSR  16 May 2002   6:01 pm

! [] Robin A. Vowels. 1998. Algorithms and data
!   structures in F and Fortran. Pages 98-130.
!   ISBN: 0-9640135-4-1
!
! Example: See
!   ...\\0.examples\\Red.Black.BinaryTree\\Red_Black.f90
! for an example.
!
!   Figure 3.14 Algorithm for manipulating a
!               Red-Black binary tree
!
! History:
! * Last change:  LSR  23 Oct 2001   7:32 am
```

Module Red\_Black\_Binary\_Tree

```
Use Global_Variables
Implicit None
Private
```

```
Logical, Parameter, Public :: Red = .True.
Logical, Parameter, Public :: Black = .False.
Integer, Parameter, Public :: Right = 1
Integer, Parameter, Public :: Left = 0
```

```
Public :: Insert_In_Tree
```

```
Character(Len=1), Parameter :: ESC = '!char(27) ! The ASCII ESCape character.
Character(Len=1), Parameter :: Rosso = '!ESC // "[31m"
Character(Len=1), Parameter :: Verde = '!ESC // "[36m"
Character(Len=1), Parameter :: White = '!ESC // "[37m"
```

Contains

```
Subroutine Insert_In_Tree(Xroot, Xnew, Node_Was_New)
! This procedure receives an isolated node Xnew, in which all pointers are
! null, and which contains a value, ready for insertion in the tree. The
! procedure searches the tree from the root along a path determined by
! the node Xnew's value, until the item of found or the path terminates
! in a null pointer.
!
! If the value was found, Xnew will point to that node and the subroutine
! returns. Otherwise the node is inserted in that position, and is
```

```

! colored red. The pointer to its parent is initialized. The
! pointer V indicates the position of the current node in the tree as
! the search progresses along a branch. Pointer W gives the current
! parent of V.
!
! In: Xnew -- A pointer to a node to be inserted into the tree.
! Out: Xnew -- points to the node containing the value (old or new node).
!       Node_Was_New -- If Xnew was inserted, .True.
!       -- If the value was in the tree, .False.

Implicit None
Type(Site_Info), Pointer :: Xroot
Type(Site_Info), Pointer :: Xnew
Logical,      Intent(Out) :: Node_Was_New

Type(Site_Info), Pointer :: V, W

Node_Was_New = .True.
V => Xroot ! The initial position of the search.
Nullify(W) ! A pointer to the parent of V.

! Find the place in the tree to graft the new node - travel from
! the root to a node containing a null pointer, unless the item
! is already there.
Do
  If (.Not. Associated(V)) Exit
  W => V

  If (Xnew%WBAN == V%WBAN) Then
    ! The item is already in the tree. No new node to insert.
    ! Give the user a pointer to the existing node and return.
    Deallocate(Xnew) ! Release storage.
    Xnew => V
    Node_Was_New = .False.
    Return
  Else If (Xnew%WBAN < V%WBAN) Then
    V => V%pLeft
  Else
    V => V%pRight
  End If
End Do

! We have found a node W whose left or right pointer field is null.

Xnew%Parent => W ! Make the new node point at its parent.
If (.Not. Associated(W)) Then ! There's only one node in the tree.
  Xroot => Xnew
  Nullify(Xroot%Parent) ! The root has no parent.
  Xroot%Color = Black ! The root is always Black.

```

```

Else If (Xnew%WBAN < W%WBAN) Then
    ! Make the parent point at the new node.
    W%pLeft => Xnew          ! The new node is in the left branch.

Else
    W%pRight => Xnew        ! The new node is in the right branch.
End If

! Rebalance and re-color the nodes of Xroot (a Red-Black
! sorted binary tree) following an insertion.
Call Rebalance_Tree(Xroot, Xnew)

End Subroutine Insert_In_Tree

Subroutine ROTATE_LEFT(Root, Pivot)
! This procedure performs a left rotation of a branch of a
! Red-Black tree, with node <Pivot> as the pivot.
!
! In: Pivot = a pointer to the node about which a branch of
!           the tree is to be rotated.

Implicit None
Type(Site_Info), Pointer :: Root
Type(Site_Info), Pointer :: Pivot

Type(Site_Info), Pointer :: Y, X

X => Pivot          ! Can't use Pivot directly. Must use a temporary.

Y => X%pRight       ! Y is the right child of X.
X%pRight => Y%pLeft ! Change the left subtree of Y into X's right subtree.
If (Associated(Y%pLeft)) Then
    Y%pLeft%Parent => X ! The left sub-node of Y points back at X --
    !               ! that is, X is now the parent of Y's left subtree.
End If
Y%Parent => X%Parent ! X's parent now becomes the parent of Y.
If (.Not. Associated(X%Parent)) Then ! We are at the root node.
    Root => Y
Else If (Associated(X, X%Parent%pLeft)) Then ! X is on the left of its parent.
    X%Parent%pLeft => Y          ! The left pointer of X's parent points at Y.
Else
    ! X is on the right subtree of its parent.
    X%Parent%pRight => Y        ! The right pointer of X's parent points at Y.
End If
Y%pLeft => X    ! Make X a left child of node Y. (X's left subtree comes with X).
X%Parent => Y    ! X's parent is now Y.

End Subroutine ROTATE_LEFT

```

```

Subroutine ROTATE_RIGHT(Root, Pivot)
! This procedure performs a right rotation of a branch of a Red-Black tree, with node <Pivot>
! as the pivot. This procedure is the mirror image of ROTATE_LEFT, with the words Left
! and Right interchanged.
!
! In: Pivot = a pointer to the node about which a branch of the tree is to be rotated.

Implicit None
Type(Site_Info), Pointer :: Root
Type(Site_Info), Pointer :: Pivot

Type(Site_Info), Pointer :: Y, X

X => Pivot           ! Can't use Pivot directly. Must use a temporary.

Y => X%pLeft         ! Y is the left child of X.
X%pLeft => Y%pRight  ! Change the right subtree of Y into X's left subtree.
If (Associated(Y%pRight)) Then
  Y%pRight%Parent => X ! The right sub-node of Y points back at X --
  !                 ! that is, X is now the parent of Y's right subtree
End If
Y%Parent => X%Parent ! X's parent now becomes the parent of Y.
If (.Not. Associated(X%Parent)) Then ! We are at the root node.
  Root => Y
Else If (Associated(X, X%Parent%pRight)) Then ! X is on the right of its parent.
  X%Parent%pRight => Y ! The right pointer of X's parent points at Y.
Else ! X is on the left of its parent.
  X%Parent%pLeft => Y ! The left pointer of X's parent points at Y.
End If
Y%pRight => X       ! Put X on the left of Y.
X%Parent => Y       ! X's parent is now Y.

End Subroutine ROTATE_RIGHT

```

```

Subroutine DELETE(Root, Vanish)
! This subroutine deletes a node from a Red-Black binary tree.
! There are four main sections:
!
! Section 1: The tree is checked to see if it is empty, and if
!           it is, the subroutine quits.
!
! Section 2: Deals with the case where the node to be deleted has two children.
!
! Section 3: Deals with the straight-forward case where the node to be deleted
!           has no children.
!
! Section 4: Deals with the case where the node to be deleted has one child.

```

```

!
! In: Vanish points at the node to be deleted.

Implicit None

Type(Site_Info), Pointer :: Root
Type(Site_Info), Pointer :: Vanish

Type(Site_Info), Pointer :: Child, X

X => Vanish
! Case 1: the tree is empty.
If (.Not. Associated(X)) Then ! The tree is empty.
    Return
End If

! Case 2: the node has 2 children.
! We find the successor node, move the content of that node to the current node X,
! and then delete the successor node.
If (Associated(X%pLeft) .And. Associated(X%pRight)) Then ! Find the next-largest node.
    ! The next-largest node is found in the right subtree.
    Child => X%pRight
    Do
        If (.Not. Associated(Child%pLeft)) Exit ! Seek the left-most node of the subtree.
        Child => Child%pLeft
    End Do

    ! Replace the Item to be deleted by its successor.
    ! Child is a node to be deleted.
    X%Item = Child%Item

    ! The successor is now due for deletion. This will be handled by Case 3
    ! (the node is a leaf) or by Case 4 (the node has one child).
    X => Child
End If

! Now delete node X.
! Case 3: the node has no children.
If ((.Not. Associated(X%pRight)) .And. (.Not. Associated(X%pLeft))) Then
    ! The node is a leaf.
    If (.Not. Associated(X%Parent)) Then ! Node X is the root.
        Deallocate(X)
        Nullify(Root)
        Return
    End If
    If (X%Color .Eqv. Black) Then ! If the color of the node to be deleted is black ...
        Call RESTRUCTURE_AFTER_DELETION(Root, X)
    End If
    If (Associated(X, X%Parent%pRight)) Then ! Chop off the leaf X.
        Nullify(X%Parent%pRight) ! Remove X which is a right child..

```

```

    Else
        Nullify(X%Parent%pLeft)          ! Remove X which is a left child.
    End If
    Deallocate(X)
    Return
End If

! Case 4: the node has one child. First find out which it is, and then splice it out.
If (.Not. Associated(X%pRight)) Then
    Child => X%pLeft ! It's a left child of X.
Else
    Child => X%pRight ! It's a right child of X.
End If

If (.Not. Associated(X%Parent)) Then ! Node X is the root.
    Root => Child ! X's only child becomes the root.
    Deallocate(X) ! Get rid of X.
    Nullify(Root%Parent) ! The root can't have a parent.
    Root%Color = Black ! Color it black ...
    Return ! ... and quit.
Else ! Node X is not the root. Splice out the node.
    ! Make the grandparent point at the child.
    If (Associated(X, X%Parent%pRight)) Then ! Node X is a right child.
        X%Parent%pRight => Child
    Else ! Node X is a left child.
        X%Parent%pLeft => Child
    End If

    Child%Parent => X%Parent ! Make the child point up at the grandparent.
End If

! The next segment will restructure the tree when the deleted node is
! black. However, there is one particular case when the deleted node is
! black and its child is red. In this situation, its child is merely
! re-colored black, thus restoring the deficiency of one black node. The
! following code indirectly achieves this: RESTRUCTURE_AFTER_DELETION is
! called, its main loop is not executed, and the subroutine terminates after
! re-coloring the child black.
If (X%Color .Eqv. Black) Then ! If the color of the node to be deleted is black ...
    Call RESTRUCTURE_AFTER_DELETION(Root, Child)
End If

Deallocate(X)

End Subroutine DELETE

Subroutine RESTRUCTURE_AFTER_DELETION(Root, Current)
    ! This procedure re-colors nodes and/or restructures a Red-Black tree
    ! following a deletion. This routine is entered when the deleted node

```

```

! was black. (Node X is the child of the deleted node). The crux is to
! balance the two subtrees whose roots are node X and its brother.
! General strategy:
! (1) Simple case: If the child of the deleted node is red, change it
!     to black and finish.
! (2) General case: Either:
!     (2a) Perform a rotation about the parent, so as to bring a node
!           into the subtree, to compensate for the deleted node. In
!           this case, color the new node black and finish.
!     (2b) Change a node of the sibling's subtree from black to red,
!           and move up the tree. In this event, repeat step (2).
!
! In: Current = points at the child of the deleted node.
Implicit None

Type(Site_Info), Pointer :: Root
Type(Site_Info), Pointer :: Current

Type(Site_Info), Pointer :: X ! the child of the deleted node
Type(Site_Info), Pointer :: Brother

X => Current

TREE_CLIMBING_LOOP: Do
  ! Quit when we are at the root, or if the node is red.
  If (Associated(X, Root) .Or. (X%Color .Eqv. Red)) Exit

  ! The crux balances the two subtrees whose roots are at node X and its brother.
  If (Associated(X, X%Parent%pLeft)) Then ! Node X is a left child.
    Brother => X%Parent%pRight
    If (Brother%Color .Eqv. Red) Then
      ! Case 1: Rotate left about X's parent, and re-color.
      Brother%Color = Black
      X%Parent%Color = Red
      Call ROTATE_Left(Root, X%Parent)
      Brother => X%Parent%pRight
    End If
    ! ... and move on to apply case 2.

  If (COLOR_OF(Brother%pLeft) .Eqv. Black .And. & ! If both children of the
    COLOR_OF(Brother%pRight) .Eqv. Black) Then ! brother are black ...
    ! Case 2: Color the brother red, and move up the tree.
    Brother%Color = Red
    X => X%Parent ! Move up the tree.
  Else
    If (COLOR_OF(Brother%pRight) .Eqv. Black) Then
      ! Case 3: Rotate right about the brother, and re-color
      If (Associated(Brother%pLeft)) Then
        Brother%pLeft%Color = Black
      End If
      Brother%Color = Red
    End If
  End If
End Do

```

```

        Call ROTATE_RIGHT(Root, Brother)
        Brother => X%Parent%pRight
    End If
        ! ... and go on to Case 4.

    ! Case 4: Rotate left about X's parent, which brings an extra node to
    ! the subtree through X, and which is then colored black.
    Brother%Color = X%Parent%Color
    X%Parent%Color = Black
    Brother%pRight%Color = Black
    Call ROTATE_Left(Root, X%Parent)    ! Brings a black node to the left
    !                                     ! subtree; Left & Right subtrees are balanced.
    Exit TREE_CLIMBING_LOOP            ! Quit, rebalancing is complete.
End If
Else
    ! Node X is a right child.
    ! Same as the THEN clause, with "Right" and "Left" interchanged.
    Brother => X%Parent%pLeft
    If (Brother%Color .Eqv. Red) Then
        ! Case 1: Rotate left about X's parent, and re-color
        Brother%Color = Black
        X%Parent%Color = Red
        Call ROTATE_RIGHT(Root, X%Parent)
        Brother => X%Parent%pLeft
    End If
        ! ... and move on to apply case 2.

    If (COLOR_OF(Brother%pLeft) .Eqv. Black .And.    & ! If both children of the
        COLOR_OF(Brother%pRight) .Eqv. Black) Then    ! brother are black ...
        ! Case 2: Color the brother red, and move up the tree.
        Brother%Color = Red
        X => X%Parent
        ! Move up the tree.
    Else
        If (COLOR_OF(Brother%pLeft) .Eqv. Black) Then
            ! Case 3: Rotate left about the brother, and re-color
            If (Associated(Brother%pRight)) Then
                Brother%pRight%Color = Black
            End If
            Brother%Color = Red
            Call ROTATE_Left(Root, Brother)
            Brother => X%Parent%pLeft
        End If
        ! ... and go on to Case 4.

        ! Case 4: Rotate right about X's parent, which brings an extra node to
        ! the subtree through X, and which is then colored black.
        Brother%Color = X%Parent%Color
        X%Parent%Color = Black
        Brother%pLeft%Color = Black
        Call ROTATE_RIGHT(Root, X%Parent) ! Brings a black node to the right
        ! subtree; Left & Right subtrees are balanced.
        Exit TREE_CLIMBING_LOOP            ! Quit, rebalancing is complete.
    End If
End If

```

```

End Do TREE_CLIMBING_LOOP

X%Color = Black ! Having encountered a red node, color it black and quit.

End Subroutine RESTRUCTURE_AFTER_DELETION

Function COLOR_OF(Node_Ptr) Result(Node_Color)
! This function returns the color of the specified node, or black if
! the node does not exist.
!
! In: Node_Ptr = a pointer to the node whose color is to be obtained.
Implicit None

Type(Site_Info), Pointer :: Node_Ptr

Logical :: Node_Color

If (Associated(Node_Ptr)) Then
    Node_Color = Node_Ptr%Color
Else
    Node_Color = Black
End If

End Function COLOR_OF

Subroutine Rebalance_Tree(Root, Dummy_Current)
! This procedure rebalances and re-colors the nodes of a Red-Black sorted binary tree,
! following an insertion. The node pointed at by X has just been inserted into the tree,
! and is the node where one of the 4 properties of Red-Black trees may have been violated.
! After a breach of Rule 1 has been rectified, X is adjusted to point at its grandparent,
! where tests for a breach of the rules is again carried out, and so on for each grandparent
! in turn. If Rules 2 or 3 are breached, one or two rotations of a branch of the tree are
! carried out (about the current node X), and the procedure terminates.
! When this procedure terminates, the root node is black, and the tree is quasi-balanced,
! and the four properties of Red-Black trees are satisfied:
!
! Rule 1. Every node is either red or black;
! Rule 2. Every null pointer is considered to be pointing to an imaginary black node;
! Rule 3. If a node is red, then both its children are black; and
! Rule 4. Every direct path from the root to a leaf contains the same number of black nodes.
!
! In: Dummy_Current = a pointer to a (red) node that has been inserted in the tree.
Implicit None
Type(Site_Info), Pointer :: Root
Type(Site_Info), Pointer :: Dummy_Current

Type(Site_Info), Pointer :: X, Y
Logical :: Red_Uncle

```

```

Logical :: Iterating

X => Dummy_Current

Do
  ! when X and its parent are both red.
  Iterating = .Not. Associated(X, Root) ! Cannot iterate when we are at the root.
  If (Iterating) Then
    ! There must be a parent ...
    Iterating = X%Parent%Color .Eqv. Red
  End If
  If (Iterating) Then
    ! ... and there must be a grandparent.
    Iterating = Associated(X%Parent%Parent)
  End If
  If (.Not. Iterating) Exit

  ! We enter this loop when X and its parent are both red.
  If (Associated(X%Parent, X%Parent%Parent%pLeft)) Then
    ! The parent is a left node of X's grandparent.
    Y => X%Parent%Parent%pRight ! Get the address of the uncle.
    Red_Uncle = Associated(Y)
    If (Red_Uncle) Then
      Red_Uncle = Y%Color .Eqv. Red
    End If
    If (Red_Uncle) Then
      ! CASE 1. There is an uncle. X and its parent and
      ! uncle are all red. Fix violation of Rule 3.
      X%Parent%Color = Black ! The parent must be black.
      Y%Color = Black ! The uncle must be black.
      X%Parent%Parent%Color = Red ! The grandparent must be red.
      X => X%Parent%Parent ! Move 2 places up the tree, to the grandparent.
    Else
      ! The uncle is black, or is non-existent.
      If (Associated(X, X%Parent%pRight)) Then ! CASE 2.
        X => X%Parent ! Move up the tree.
        Call ROTATE_Left(Root, X)
      End If
      ! CASE 3.
      X%Parent%Color = Black
      X%Parent%Parent%Color = Red
      Call ROTATE_RIGHT(Root, X%Parent%Parent)
    End If
    ! This segment is the mirror image of the code for the "then" part,
    ! with the words Right and Left interchanged.
  Else
    ! The parent is a right node of X's grandparent.
    Y => X%Parent%Parent%pLeft ! Get the address of the uncle.
    Red_Uncle = Associated(Y)
    If (Red_Uncle) Then
      Red_Uncle = Y%Color .Eqv. Red
    End If
    If (Red_Uncle) Then
      ! CASE 1.
      X%Parent%Color = Black ! The parent must be black.
    End If
  End If
End Do

```

```

        Y%Color = Black           ! The uncle must be black.
        X%Parent%Parent%Color = Red ! The grandparent must be red.
        X => X%Parent%Parent      ! Move 2 places up the tree, to the grandparent.
Else                               ! X and its parent are red, but its uncle is black
    !                               ! or is missing. Fix violation of Rule 3.
    If (Associated(X, X%Parent%pLeft)) Then ! CASE 2.
        X => X%Parent             ! Move up the tree.
        Call ROTATE_RIGHT(Root, X)
    End If
    ! CASE 3.
    X%Parent%Color = Black
    X%Parent%Parent%Color = Red
    Call ROTATE_Left(Root, X%Parent%Parent)
End If
End If
End Do

Root%Color = Black ! Ensure that the root is black.

End Subroutine Rebalance_Tree

Recursive Subroutine TRAVERSE(Current)
    ! This recursive subroutine retrieves and prints the values from the tree in ascending order.
    Implicit None
    Type(Site_Info), Pointer :: Current

    If (Associated(Current%pLeft)) Then ! Take the left subtree.
        Call TRAVERSE(Current%pLeft)
    End If

    Print *, Current%Item ! Retrieve value from tree and print it.

    If (Associated(Current%pRight)) Then ! Take the right subtree.
        Call TRAVERSE(Current%pRight)
    End If

End Subroutine TRAVERSE

Subroutine DISPLAY_SIDEWAYS(Root)
    ! This procedure prints/displays a binary tree sideways.
    !
    ! Method: An in-order traversal of the tree is first performed, in order
    ! to determine the longest branch (that is, the greatest depth of the
    ! tree). This value is used to determine the number of nodes as if all
    ! branches were of this length. For each node, an excursion is taken from
    ! the root to the node to be printed. The route is determined by the
    ! number of the current node (nodes number from 1 to 2**Max_Depth-1), and

```

```

! is decoded by Mask, initially set to 2**Max_Depth/2. Each bit in the
! binary representation of the node's number corresponds to a fork in the
! tree. A zero bit causes a right path to be taken, while a one-bit
! causes a left path to be taken. When a node is absent from the tree,
! an empty line is written out so as to maintain the geometry of the tree.

Implicit None
Type(Site_Info), Pointer :: Root

Type(Site_Info), Pointer :: Ptr
Integer          :: Max_Nodes
Integer          :: Node_No

! Used to indicate the current level of a node during an excursion from
! the root to a node to be printed.
Integer :: Level
Integer :: Mask      ! is Ariadne's thread (the key to decode the
Integer :: Dmask    ! path to be taken at each fork).
Integer :: Max_Depth ! The maximum depth of the tree (levels of nodes).
Integer :: Nlevels
Logical  :: Printed

Character(Len=10) :: Layout ! For preparing a dynamic format specification.

Max_Depth = Longest_Branch(Root) ! To count the nodes in the tree.
Max_Nodes = 2**Max_Depth ! Determine the depth of the tree.

! Section to visit each node of the tree.
! Nodes are numbered starting from the right-most leaf, as if doing an in-order traversal.
Do Node_No = 1, Max_Nodes - 1
  Ptr => Root ! Begin each journey from the root.
  Mask = Max_Nodes / 2
  Printed = .False.

  ! Section to travel from the root, to the node to be printed.
  Do Level = 1, 1000
    If (Iand(Node_No, Mask) /= 0) Then
      ! Take the right path.
      Ptr => Ptr%pRight
    Else ! Take the left path.
      If ((Modulo(Node_No, Mask) == 0) .Or. (Mask == 1)) Then
        Write (unit=*, fmt=*)
        Dmask = 2 * Mask
        If (Node_No /= Max_Nodes/2) Then
          Write (unit=Layout, fmt="(a, i3, a)" "(tr", 10*(Level-1)+1, ", a)"
          If (Iand(Node_No, Dmask) /= 0) Then
            Write (unit=*, fmt=Layout, advance="no") "\"
          Else
            Write (unit=*, fmt=Layout, advance="no") "/"
          End If
        End If
      End If
    End Do
  End Do

```

```

        End If
        If (Ptr%Color .Eqv. Red) Then
            Write (unit=*, fmt="(a)") Rosso
        Else
            Write (unit=*, fmt="(a)") Verde
        End If
        Write (unit=*, fmt="('(',i0,')')", advance="no") Ptr%Item
        Printed = .True.
    End If
    Ptr => Ptr%pLeft
End If
Mask = Mask / 2
If ((Level >= Max_Depth) .Or. &
    (.Not. Associated(Ptr)) .Or. Printed) Then
    Exit
End If
End Do

If (.Not. Printed) Then          ! The node did not exist.
    Do Nlevels = Level, Max_Depth
        Print *                  ! Leave a blank line so as to preserve the
    End Do                       ! structure of the tree.
End If
End Do
Print *, Verde
End Subroutine DISPLAY_SIDEWAYS

```

```

Subroutine DISPLAY_TREE(Root)
! This subroutine displays a tree in graphic form, showing the nodes in their actual
! positions. It is intended for a screen display, but the screen parameters can be altered
! readily enough for a wide screen or printer. For each node, this subroutine displays the
! value in its appropriate color (red or black), along with the letters R or B (R=red,
! B = black), in case you don't have a color screen. This routine displays 5 levels;
! at the fifth level, 16 elements are displayed on one screen line of 80 columns.
!
! Method: An in-order traversal of the tree is first performed, in order to determine the
! longest branch (that is, the greatest depth of the tree). This value is used to
! determine the number of nodes as if all branches were of this length.

Implicit None
Type(Site_Info), Pointer :: Root
Type(Site_Info), Pointer :: Ptr

! Used to indicate the current level of a node during an excursion from
! the root to a node to be printed.
Integer :: Level
Integer :: Nodes ! The maximum number of nodes at level LEVEL.
Integer :: Depth ! The number of nodes from here to the root.
Integer :: Spaces, Gap, J, Node_No, Trip, Mask

```

```

Integer :: Max_Depth ! The maximum depth of the tree (levels of nodes).

Character(Len=1), Dimension(0:1), Parameter :: Palette = (/ "B", "R" /)
Character(Len=10) :: Layout ! For preparing a dynamic format specification.

Logical, Save :: Can_Color = .False.

Max_Depth = Longest_Branch(Root) ! To count the nodes in the tree.
Max_Depth = Min(Max_Depth, 5)

Level = 0
Nodes = 1
Ptr => Root
Call SET_COLOR(Ptr)
Write (unit=*, fmt="( / tr35, i3)", advance="no") Ptr%Item
If (Ptr%Color .Eqv. Red) Then
    J = 1
Else
    J = 0
End If
Write (unit=*, fmt="(a)") Palette(J)
Write (unit=*, fmt="(tr37, a)") "/" "\"
Write (unit=*, fmt="(tr36, a)") "/" "\"

Depth = 1
Spaces = 80

Do
    ! Visit all the nodes to the specified depth.
    If (Depth >= Max_Depth) Exit

    Spaces = Spaces/2 ! Adjust the spacing between nodes.
    Gap = Spaces/2 ! The initial spacing is half that between nodes.
    Nodes = Nodes * 2
    Depth = Depth + 1 ! The level of the nodes being displayed.
    Print *

    Do Node_No = 0, Nodes-1 ! Visit all the nodes at a given level.
        Ptr => Root ! Start from the root.
        Mask = 1
        Do J = 1, Depth-2
            Mask = Mask * 2
        End Do

        Do Trip = 1, Depth-1
            If (.Not. Associated(Ptr)) Exit
            ! Travel from the root to a node at the given depth
            If (Iand(Node_No, Mask) /= 0) Then ! Take the right branch.
                Ptr => Ptr%pRight
            Else ! Take the left branch.

```

```

        Ptr => Ptr%pLeft
    End If
    Mask = Mask/2
End Do

If (Associated(Ptr)) Then
    Call SET_COLOR(Ptr)
    Write (unit=Layout, fmt="(a, i3.3, a)" "(tr", Gap-4, ",i3)"
    If (Gap < 5) Then
        Layout = "(i3)"
    End If
    Write (unit=*, fmt=Layout, advance="no") Ptr%Item
    If (Can_Color) Then
        ! Comment the next line if your PC or terminal lacks color.
        Write (unit=*, fmt="(a)", advance="no") " "
    Else

        ! Un-comment the next 6 lines if your PC or terminal lacks color.
        If (Ptr%Color .Eqv. Red) Then
            J = 1
        Else
            J = 0
        End If
        Write (unit=*, fmt="(a)", advance="no") Palette(J)
    End If
Else
    Write (unit=Layout, fmt="(a, i3.3, a)" "(tr", Gap-4, ",2a)"
    If (Gap < 5) Then
        Layout = "(2a)"
    End If
    If (Can_Color) Then
        ! Comment the next line if your PC or terminal lacks color.
        Write (unit=*, fmt=Layout, advance="no") White, " - "
    Else
        ! Un-comment the next line if your PC or terminal lacks color.
        Write (unit=*, fmt=Layout, advance="no") " - "
    End If
End If
Gap = Spaces          ! Assume normal spacing between nodes.
Mask = Mask/2
End Do

If (Can_Color) Then
    ! Comment the next line if your PC or terminal lacks color.
    Write (unit=*, fmt="(a)") White
Else
    ! Un-comment the next line if your PC or terminal lacks color.
    Write (unit=*, fmt=*)
End If
Gap = Spaces/2

```

```

If (Depth < Max_Depth) Then
  Do Node_No = 0, Nodes-1
    Write (unit=Layout, fmt="(a, i3.3, a)" "(tr", Gap-2, ", a)"
    If (Gap < 3) Then
      Layout = "(a)"
    End If
    Write (unit=*, fmt=Layout, advance="no") "/ \"
    Gap = Spaces
  End Do
End If

Write (unit=*, fmt="(a)")
Write (unit=*, fmt="(a)", advance="no") " "
Gap = Spaces/2
If (Depth < Max_Depth) Then
  Do Node_No = 0, Nodes-1
    Write (unit=Layout, fmt="(a, i3.3, a)" "(tr", Gap-4, ", a)"
    If (Gap < 5) Then
      Layout = "(a)"
    End If
    Write (unit=*, fmt=Layout, advance="no") "/ \"
    Gap = Spaces
  End Do
End If
End Do

```

Contains

```

! This procedure changes the display color to either red or green.
Subroutine SET_COLOR(Ptr)
  Implicit None
  Type(Site_Info), Pointer :: Ptr

  If (Ptr%Color .Eqv. Red) Then
    Write (unit=*, fmt="(a)", advance="no") Rosso
  Else
    Write (unit=*, fmt="(a)", advance="no") Verde
  End If
End Subroutine SET_COLOR
End Subroutine DISPLAY_TREE

```

```

Function Longest_Branch(xRoot) Result(Max_Depth)
  ! Determines the longest branch of a binary tree
  Implicit None
  Type(Site_Info), Pointer :: xRoot
  Integer :: Max_Depth

  Integer :: Current_Depth

```

```

Max_Depth = 0
Current_Depth = 0
Call LB0(xRoot)

Contains
Recursive Subroutine LB0(Current) ! i.e., Longest_Branch_0
! Auxiliary subroutine for Longest_Branch
Implicit None
Type(Site_Info), Pointer :: Current

If (.Not. Associated(Current)) Return

Current_Depth = Current_Depth + 1

If (Current_Depth > Max_Depth) Then
  Max_Depth = Current_Depth
End If
If (Associated(Current%pLeft)) Then
  Call LB0(Current%pLeft)
  Current_Depth = Current_Depth - 1
End If
If (Associated(Current%pRight)) Then
  Call LB0(Current%pRight)
  Current_Depth = Current_Depth - 1
End If

End Subroutine LB0
End Function Longest_Branch
End Module Red_Black_Binary_Tree

```

## samson

! Last change: LSR 17 May 2002 1:47 pm  
Module SAMSON

! For a description of the SAMSON file format see  
! <A HREF="e:\5\3met\Docs\samson\_format.txt#\$1">

Use Floating\_Point\_Comparisons  
Use Global\_Variables  
Use IoSubs  
Use Utils1  
Implicit None

Contains

Subroutine Read\_SAMSON\_v1x(InFile, WFlx, Is\_v11, Empty\_File, Jout)

! This subroutine reads a SAMSON v1.x data file.  
!  
! Is\_v11 -- Truth of "InFile" represents a SAMSON v1.1 file.  
!  
! Version 1.1 files contain only the first five parameters  
! described in the parameter section below. everything else  
! is equal.  
!  
! Version 1.0 : InFile = '..\12345\_61.z', a compressed file.  
! Version 1.1 : InFile = '..\12345\_61.txt', a text file.

Implicit None  
Logical, Intent(In) :: Is\_v11  
Character(Len=\*), Intent(In) :: InFile  
Type(FileInfo), Target, Intent(InOut) :: WFlx  
Logical, Intent(Out) :: Empty\_File  
Integer, Optional, Intent(In) :: Jout

Integer :: jin  
Integer :: ios  
Character(Len=05) :: WBAN\_Number  
Character(Len=22) :: City  
Character(Len=02) :: State  
Integer :: Time\_Zone  
Character(Len=01) :: Latitude\_Let ! North/South  
Integer :: Latitude\_Degrees, Latitude\_Minutes  
Character(Len=01) :: Longitude\_Let ! East/West  
Integer :: Longitude\_Degrees, Longitude\_Minutes  
Integer :: Elevation\_meters  
Integer :: Observation\_Indicator

```

Integer :: yy, mm, dd, hh ! 2-digit year, month, day, hour
Integer :: yyyy          ! 4-digit year
Integer :: EHR           ! Extraterrestrial Horizontal Radiation
Integer :: EDNR         ! Extraterrestrial Direct Normal Radiation
Integer :: GHR_v, GHR_u ! Global Horizontal Radiation
Integer :: DNR_v, DNR_u ! Direct Normal Radiation
Integer :: DHR_v, DHR_u ! Diffuse Horizontal Radiation
Character(Len=01) :: GHR_s, DNR_s, DHR_s
Integer :: Total_Sky_Cover, Opaque_Sky_Cover
Real    :: Dry_Bulb_Temperature, Dew_Point_Temperature
Integer :: Relative_Humidity
Integer :: Station_Pressure
Integer :: Wind_Direction
Real    :: Wind_Speed ! in m/s
Real    :: Horizontal_Visibility
Integer :: Ceiling_height ! in meters
Character(Len=09) :: Present_weather
Integer :: Precipitable_water
Real    :: baod ! Broadband aerosol optical_depth
Integer :: Snow_Depth
Integer :: Days_Since_Last_Snowfall
Integer :: Hourly_Precipitation
Character(Len=01) :: Hourly_Precipitation_Flag

Integer :: Fyear, nlines
Logical :: ok, is_daytime
Integer :: doy, jyyyy, jmm, jdd, jdoy, jhh, jv
Integer :: jd_today
Integer :: nbase, iv
Character(Len=200) :: tbuf, qline
Real    :: ul0, Station_elevation_in_meters
Logical :: Xfound

Integer, Dimension(:, :), Pointer :: p_every_hour
Type(Val_and_Flag) :: d
Character(Len(d%s)) :: data_source

!Logical :: Print_now
!Integer :: jv0, jv1

Integer, Parameter :: ia0 = Iachar('0')

! PWT -- Present Weather Table
! Used to validate Present_weather's 9 flags
! Numbers with "*" denote additions to the SAMSON
! document. See <A HREF="0notes.txt#Note_4">.
Character(Len=*) , Parameter :: PWT = '0123456789'
Integer :: i

nlines = 0

```

```

Empty_File = .True.

! Open the input file
If (Len_trim(InFile) == 0) Then
    ! No file name.
    Return
End If

If (Is_v11) Then
    Call IORead(jin, InFile)
    data_source = T_SAMSONv11
Else
    Call Decompress_and_Open_File(Jin, InFile)
    data_source = T_SAMSONv10
End If
Fyear = WFlx%yyyy
Write (ULog, *) '## File: ', Trim(InFile)
Call ToTTY('Read_SAMSON_vlx '//Trim(InFile))

! nbase + iv
nbase = Hours_since_Jd0(Fyear)

! Assign pointers
p_every_hour => WFlx%Every_Hour_Present

!Call ymdh_to_iv(1978, 07, 20, 15, jv0)
!Call ymdh_to_iv(1978, 07, 21, 03, jv1)

! <A NAME="Missing years">
! See <A HREF="0notes.txt#Note_25">
! See <A HREF="4.missing.years.bat#$1">
!
! * [lsr] 25 Mar 2002  8:50 am
!   + We will copy the missing years from existing years.
!     Say 1990 is missing. We will copy 1961 or 1964 (if
!     leap year). This will allow make_r0 to complete the
!     run. We will then eliminate the 1990.hvf file and the
!     MET file records associated with 1990.

! Read first record (example below) -- Valid for both SAMSON versions 1.0 and 1.1.
!
! Fld Position      Element          Definition
!
! 001 - 001         Indicator       Indicates the header record.
!
! 002 - 006         WBAN Number     Station's Weather Bureau Army Navy number.
!
! 008 - 029         City             City where the station is located (maximum
!                                     of 22 characters).
!
!

```

```

! 031 - 032      State      State where the station is located (abbreviated
!                to two letters).
!
! 034 - 036      Time Zone   Time zone is the number of hours by which the
!                local standard time lags or leads Universal
!                Time. For example, Mountain Standard Time is
!                designated -7 because it lags Universal Time by
!                7 hours.
!
! 039 - 044      Latitude    Latitude of the station.
! 039            N = North of equator
! 040 - 041      Degrees
! 043 - 044      Minutes
!
! 047 - 053      Longitude   Longitude of the station.
! 047            W = West, E = East
! 048 - 050      Degrees
! 052 - 053      Minutes
!
! 056 - 059      Elevation   Elevation of station in meters above sea level.
!
!                10        20        30        40        50        60
! 123456789=123456789=123456789=123456789=123456789=123456789=123456789=
! 13873 ATHENS                GA -5 N33 57 W083 19 244

```

```

Read (Jin, '(a)', Iostat = ios) qline
If (ios /= 0) Goto 9999
nlines = nlines + 1

```

```

Read (qline, 9130, Iostat = ios) WBAN_Number, City, State, Time_Zone, &
Latitude_Let, Latitude_Degrees, Latitude_Minutes, &
Longitude_Let, Longitude_Degrees, Longitude_Minutes, &
Elevation_meters

```

```

9130 Format (          1x, a5, 1x, a22, 1x, a2,          1x, i3, &
          2x, a1,          i2,          1x, i2, &
          2x, a1,          i3,          1x, i2, &
          2x, i4)

```

```

If (ios /= 0) Goto 9999

```

```

If (WFlx%Check_Header_Info) Then
WFlx%Head%WBAN = WBAN_Number
WFlx%Head%Text = Trim(City) // ', ' // Trim(State)
WFlx%Head%Lat = Coords(Latitude_Let, Latitude_Degrees, Latitude_Minutes)
WFlx%Head%Lon = Coords(Longitude_Let, Longitude_Degrees, Longitude_Minutes)
WFlx%Head%Elev = Elevation_meters
WFlx%Head%TZ = Time_Zone
End If

```

```

If (Present(Jout)) Then
  Write (Jout, 9150) WBAN_Number, City, State, Time_Zone, &
    Latitude_Let, Latitude_Degrees, Latitude_Minutes, &
    Longitude_Let, Longitude_Degrees, Longitude_Minutes, &
    Elevation_meters

  ! Same format as '9350'
9150  Format ( '~', a5, lx, a22, lx, a2, lx, i3, &
        2x, a1, i2.2, lx, i2.2, &
        2x, a1, i3.3, lx, i2.2, &
        2x, i4)

  ! Add the output header
  Write (Jout, 9170)
9170  Format ('~YR MO DA HR I 1 2 3 4 5 6 7', &
        ' 8 9 10 11 12 13 14 15 16', &
        ' 17 18 19 20 21')
End If

```

! From: [http://rredc.nrel.gov/solar/old\\_data/nsrdb/hourly/tab2.html](http://rredc.nrel.gov/solar/old_data/nsrdb/hourly/tab2.html) (SAMSON)

!

! Table 2. Data Elements in the NSRDB Hourly Data Files

! (For all except the first record of each file)

!

! IP - position in input file

! OP - field number in output file

!

OP	IP	Element	Values	Definition
!	---	-----	-----	-----
!	002-012	Local Standard Time		
!	002-003	Year	61-90	Year of observation
!	005-006	Month	1-12	Month of observation
!	008-009	Day	1-31	Day of month
!	011-012	Hour	1-24	Hour of day in local standard time
!				
!	I 096-096	Observation Indicator	0 or 9	0 = Weather observation made. 9 = Weather observation not made or missing.
!				
!	[lsr] Thu 19 Oct 2000 11:58:19	This field appears to be column 96 of the input file. The manual provides no data to support this assertion.		If this field = 9 OR if field 13 (wind speed) = missing (9999. or 99.0), then fields 6, 7, 8, 10, 11, 17, and 18 were all modeled and not actually observed.
!	[lsr] Wed Nov 21 09:33:59 2001	Observation Indicator appears only in SAMSON v 1.0 files.		
!				
!	01 014-017	Extraterrestrial Horizontal Radiation	0-1415	Amount of solar radiation in Wh/m2 received on a horizontal surface at the top of the atmosphere during the 60 minutes preceding the hour indicated.
!				
!				

```

!
! 02 019-022 Extraterrestrial Direct Normal Radiation 0-1415 Amount of solar radiation in Wh/m2
! received on a surface normal to
! the sun at the top of the atmosphere
! during the 60 minutes preceding the
! hour indicated.
!
! 03 024-030 Global Horizontal Radiation 0-1415 Total amount of direct and diffuse
! 024-027 Data Value 0-1415 solar radiation in Wh/m2 received on a
! 029 Flag for Data Source A-H, ? horizontal surface during the 60
! 030 Flag for Data Uncertainty 0-9 minutes preceding the hour indicated.
! 9999 = missing data.
!
! 04 032-038 Direct Normal Radiation 0-1415 Amount of solar radiation in Wh/m2
! 032-035 Data Value 0-1415 received within a 5.7o field of view
! 037 Flag for Data Source A-H, ? centered on the sun, during the 60
! 038 Flag for Data Uncertainty 0-9 minutes preceding the hour indicated.
! 9999 = missing data.
!
! 05 040-046 Diffuse Horizontal Radiation 0-1415 Amount of solar radiation in Wh/m2
! 040-043 Data Value 0-1415 received from the sky (excluding the
! 045 Flag for Data Source A-H, ? solar disk) on a horizontal surface,
! 046 Flag for Data Uncertainty 0-9 during the 60 minutes preceding the
! hour indicated. 9999 = missing data.
!
! =====
! ===== End of SAMSON version 1.1 parameters =====
! =====
!
! 06 048-049 Total Sky_Cover 0-10 Amount of sky dome (in tenths)
! covered by clouds. 99 = missing data.
!
! 07 051-052 Opaque Sky_Cover 0-10 Amount of sky dome (in tenths)
! covered by clouds that prevent
! observing the sky or higher cloud
! layers. 99 = missing data.
!
! 08 054-058 Dry Bulb Temperature -70.0 to 60.0 Dry bulb temperature in degrees C.
! 9999. = missing data.
!
! 09 060-064 Dew_Point Temperature -70.0 to 60.0 Dew_point temperature in degrees C.
! 9999. = missing data.
!
! 10 066-068 Relative Humidity 0-100 Relative humidity in percent.
! 999 = missing data.
!
! 11 070-073 Station Pressure 700-1100 Station pressure in millibars.
! 9999 = missing data.
!
!

```

!	12	075-077	Wind Direction	0-360	Wind direction in degrees. (N = 0 or 360, E = 90, S = 180, W = 270) 999 = missing data.
!					
!	13	078-082	Wind Speed	0.0-99.0	Wind speed in m/s. 9999. or 99.0 = missing data.
!					
!	14	083-088	Visibility	0.0-160.9	Horizontal visibility in kilometers. 777.7 = unlimited visibility. 99999. = missing data.
!					
!	15	089-094	Ceiling Height	0-30450	Ceiling height in meters. 77777 = unlimited ceiling height. 88888 = cirroform. 999999 = missing data.
!					
!	**	096-096	See 'Observation Indicator' above.		*****
!					
!	16	097-105	Present_weather	See Table Below	Present_weather conditions denoted by 9 indicators. See present weather table below.
!					
!	17	106-109	Precipitable Water	0-100	Precipitable_water in millimeters. 9999 = missing data.
!					
!	18	110-115	Broadband Aerosol Optical_Depth	0.0-0.900	Broadband aerosol optical_depth (broadband turbidity) on the day indicated. 99999. = missing data.
!					
!	19	116-119	Snow_Depth	0-100	Snow_depth in centimeters on the day indicated. 9999 = missing data.
!					
!	20	120-122	Days Since Last Snowfall	0-88	Number of days since last snowfall. 88 = 88 or greater days. 999 = missing data.
!					
!	21	124-129	Hourly Precipitation	000000- 099999	In inches and hundredths (See information below).
!					
!		130-130	Hourly Precipitation Flag		See explanation below.

```

Loop: Do
  Read (Jin, '(a)', Iostat = ios) qline
  If (ios /= 0) Goto 9999
  nlines = nlines + 1

```

```

! Print the Hourly Precipitation part of the record.
! This record is available only for v1.0 files.
!If (.Not. Is_v11) Then
!   Write(1001, '(1x,a,a,1x,a)') 'SAMSON: ', qline(1:13), Trim(qline(123:))
!End If

Read (qline, 9190, Iostat = ios) yy, mm, dd, hh, EHR, EDNR, &
    GHR_v, GHR_s, GHR_u, DNR_v, DNR_s, DNR_u, DHR_v, DHR_s, DHR_u, &
    Total_Sky_Cover, Opaque_Sky_Cover, &
    Dry_Bulb_Temperature, Dew_Point_Temperature, &
    Relative_Humidity, Station_Pressure, &
    Wind_Direction, Wind_Speed, &
    Horizontal_Visibility, Ceiling_height, Observation_Indicator, &
    Present_weather, Precipitable_water, &
    baod, Snow_Depth, Days_Since_Last_Snowfall, &
    Hourly_Precipitation, Hourly_Precipitation_Flag

9190   Format ( 4(1x,i2), 2(1x,i4), &
            3(1x,i4,1x,a1,i1), &
            2(1x,i2), 2(1x,f5.1), 1x, i3, 1x, i4, &
            1x, i3, f5.1, f6.1, i6, 1x, i1, &
            a9, i4, &
            f6.3, i4, i3, &
            1x, i6, a1 )

If (ios /= 0) Exit Loop

yyyy = 1900 + yy

!!! ***** Convert to appropriate units
!!! ***** Handle missing data
!!! ***** Identify data source

!!! ***** Calendar and "iv" tests commented out on Wed Dec 12 12:29:37 2001, because:
!!! ***** #1. All tests were passed without problems.
!!! ***** #2. With the tests active, it took 3m 21s to read all files;
!!! ***** With the tests commented out, it took 1m 36s to read all files;

    jd_today = jd(yyyy, mm, dd)
!!!   Call Jd_to_ymd(jd_today, jyyyy, jmm, jdd)
!!!   ok = (yyyy == jyyyy) .And. (mm == jmm) .And. (dd == jdd)
!!!   If (.Not. ok) Then
!!!       Write(ULog,0102) '?? jd /= Jd_to_ymd ', yyyy, mm, dd
!!!       Write(6,0102) '?? jd /= Jd_to_ymd ', yyyy, mm, dd
!!!       Stop '?? jd /= Jd_to_ymd'
!!!   End If

9210   Format(///, a, 1x, i4, '-', i2.2, '-', i2.2, ':', ' ', i2, 'h')

```

```

doy = iDoY(yyyy, mm, dd)
!!! Call Calend(yyyy, doy, jmm, jdd)
!!! ok = (mm == jmm) .And. (dd == jdd)
!!! If (.Not. ok) Then
!!!   Write(ULog,0102) '?? iDoY /= Calend ', yyyy, mm, dd
!!!   Write(6,0102)   '?? iDoY /= Calend ', yyyy, mm, dd
!!!   Stop '?? iDoY /= Calend'
!!! End If

!Write(tbuf,0105) yyyy, mm, dd, hh
9230 Format ('Read_SAMSON_v1x: (', i4, 2('-',i2.2), 1x, i2.2, 'h)')
!!!!!! Call Prev_Node(pPrev, pHead, pTail, Trim(tbuf))

! Store data in a vector array so that we can find gaps easier.
! Assume a virtual array: vData(ndays, Nhours)
!   Ndays = 365 or 366 == WFlx%Expected_Ndays
!
!   Ndays| 1      2 3 4 ...      24      25      Nhours = 25
!   -----+-----
!   1 | 1      2 3 4 ...      24      25
!   2 | 26     27 28 29 ...     49      50
!   : | :      : : : ...     :      :
!   365| 9101  : : : ...     9124    9125
!   366| 9126  . . . ...     9149    9150
!
! Samson_v1x(iv) <==> vData(doy,hh)
!   iv = (doy-1)*Nhours + hh
!
!   doy = (iv+Nhours-1) / Nhours
!   hh  = iv - (doy-1)*Nhours
!
! nbase = Hours_since_Jd0(yyyy)
! iv = (doy-1)*Nhours + hh + nbase
! Xparam(f_EHR)%Samson_v10(iv)%v = EHR

iv = (doy-1)*Nhours + hh

!!!   ! Test inverse.
!!!   jdoy = (iv+Nhours-1) / Nhours
!!!   jhh  = iv - (jdoy-1)*Nhours
!!!   ok = (hh == jhh) .And. (doy == jdoy)
!!!   If (.Not. ok) Then
!!!     Write(ULog,0102) '?? iv /= (doy,hh) ', yyyy, mm, dd, hh
!!!     Write(6,0102)   '?? iv /= (doy,hh) ', yyyy, mm, dd, hh
!!!     Stop '?? iv /= (doy,hh)'
!!!   End If

! Add nbase so that the index references the correct year.
iv = iv + nbase

```

```

!Print_now = ((jv0 <= iv) .And. (iv <= jv1))
!If (Print_now) Then
!   Write (ULog, *) yyyy, mm, dd, hh, &
!       ': Days_Since_Last_Snowfall = ', Days_Since_Last_Snowfall
!End If

!!!
! Test iv_to_ymdh and inverse. -- tests ok: Wed Dec 12 12:21:43 2001
!!!
Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh, jdoy)
!!!
ok = (yyyy == jyyyy) .And. (mm == jmm) .And. (dd == jdd) .And. (hh == jhh) .And. (doy == jdoy)
!!!
If (.Not. ok) Then
!!!
Write(ULog,0202) '?? iv_to_ymdh: ', iv, doy, yyyy, mm, dd, hh, ' /= ', jdoy, jyyyy, jmm, jdd, jhh
!!!
Write(6, 0202) '?? iv_to_ymdh: ', iv, doy, yyyy, mm, dd, hh, ' /= ', jdoy, jyyyy, jmm, jdd, jhh
!!!0202
Format(///, 1x, a, 'iv=', i0, ', ', ' ', 2('doy=', i0, i5, '- ', i2.2, '- ', i2.2, i3, 'h)', :, a)
!!!
Stop '?? iv_to_ymdh'
!!!
End If
!!!
Call ymdh_to_iv(yyyy, mm, dd, hh, jv, jdoy)
!!!
ok = (iv == jv) .And. (doy == jdoy)
!!!
If (.Not. ok) Then
!!!
Write(ULog,0212) '?? ymdh_to_iv: ', iv, jv, doy, jdoy, yyyy, mm, dd, hh
!!!
Write(6, 0212) '?? ymdh_to_iv: ', iv, jv, doy, jdoy, yyyy, mm, dd, hh
!!!0212
Format(///, 1x, a, 'iv=', i0, '/=', i0, '; doy=', i0, '/=', i0, ' (' , i4, '- ', i2.2, '- ', i2.2, i3, 'h)')
!!!
Stop '?? ymdh_to_iv'
!!!
End If

! Tally this day and hour.
p_every_hour(doy,hh) = p_every_hour(doy,hh) + 1
!WFlx%Every_Hour_Present(doy,hh) = 1 + &
!     WFlx%Every_Hour_Present(doy,hh)

! It is daylight if Rs>0 (and Rs is not missing). (Rs is an alias for GHR_v.)
! "is_daytime" is used to determine if other parameters are truly
! missing, e.g., if a parameter X is to be output only for daylight hours,
! then parameter X is (effectively) non-missing if Rs>0 for the same record (hour).
! The default for "is_daytime" is .True., i.e., any parameter will be missing
! unless we can determine it is nighttime. This will take care of the (rare) case
! when Rs is missing.
is_daytime = .True.

! 03 024-030      Global Horizontal Radiation      Total amount of direct and diffuse
! 024-027      Data Value      0-1415      solar radiation in Wh/m2 received on
! 029          Flag for Data Source      A-H, ?      a horizontal surface during the 60
! 030          Flag for Data Uncertainty      0-9      minutes preceding the hour indicated.
!                                     9999 = missing data.
! Ref[FAO] Rs == Global Horizontal Radiation == samson GHR_v == huswo iGRAD
! 1 Watt hour == 3.6e-3 MJoule
! 1 Watt hour m^-2 == 8.59845E-02 Langley
If ((0 <= GHR_v) .And. (GHR_v <= 1415)) Then
is_daytime = (GHR_v > 0)
If (Is_v11) Then
Xparam(f_GHR)%Samson_v11(iv)%v = GHR_v

```

```

        Xparam(f_GHR)%Samson_v11(iv)%s = data_source
    Else
        Xparam(f_GHR)%Samson_v10(iv)%v = GHR_v
        Xparam(f_GHR)%Samson_v10(iv)%s = data_source
    Endif
If (Verify(GHR_s, 'abcdefghABCDEFGH?') == 0) Then
    If (Is_v11) Then
        Xparam(f_GHR)%Samson_v11(iv)%f(1:1) = GHR_s
    Else
        Xparam(f_GHR)%Samson_v10(iv)%f(1:1) = GHR_s
    Endif
Else
    Write (ULog, 9290) iv, yyyy, mm, dd, hh, &
        'GHR_s', GHR_s, '{abcdefghABCDEFGH?}'
End If
If ((0 <= GHR_u) .And. (GHR_u <= 9)) Then
    If (Is_v11) Then
        Xparam(f_GHR)%Samson_v11(iv)%f(2:2) = Achar(GHR_u+ia0)
    Else
        Xparam(f_GHR)%Samson_v10(iv)%f(2:2) = Achar(GHR_u+ia0)
    Endif
Else
    Write (ULog, 9270) iv, yyyy, mm, dd, hh, &
        'GHR_u', GHR_u, '{0..9}'
End If
End If

! 01 014-017      Extraterrestrial          0-1415  Amount of solar radiation in Wh/m2
!                Horizontal Radiation        received on a horizontal surface at
!                                                        the top of the atmosphere during the
!                                                        60 minutes preceding the hour indicated.
! Ref[FAO] Ra == Extraterrestrial Horizontal Radiation
! 1 Watt hour == 3.6e-3 MJoule
If ((0 <= EHR) .And. (EHR <= 1415)) Then
    If (Is_v11) Then
        Xparam(f_EHR)%Samson_v11(iv)%v = EHR
        Xparam(f_EHR)%Samson_v11(iv)%s = data_source
    Else
        Xparam(f_EHR)%Samson_v10(iv)%v = EHR
        Xparam(f_EHR)%Samson_v10(iv)%s = data_source
    Endif
End If

! Extraterrestrial Direct Normal Radiation
If ((0 <= EDNR) .And. (EDNR <= 1415)) Then
    If (Is_v11) Then
        Xparam(f_EDNR)%Samson_v11(iv)%v = EDNR
        Xparam(f_EDNR)%Samson_v11(iv)%s = data_source
    Else
        Xparam(f_EDNR)%Samson_v10(iv)%v = EDNR

```

```

        Xparam(f_EDNR)%Samson_v10(iv)%s = data_source
    Endif
End If

! Direct Normal Radiation
If ((0 <= DNR_v) .And. (DNR_v <= 1415)) Then
    If (Is_v11) Then
        Xparam(f_DNR)%Samson_v11(iv)%v = DNR_v
        Xparam(f_DNR)%Samson_v11(iv)%s = data_source
    Else
        Xparam(f_DNR)%Samson_v10(iv)%v = DNR_v
        Xparam(f_DNR)%Samson_v10(iv)%s = data_source
    Endif
    If (Verify(DNR_s, 'abcdefghABCDEFGH?') == 0) Then
        If (Is_v11) Then
            Xparam(f_DNR)%Samson_v11(iv)%f(1:1) = DNR_s
        Else
            Xparam(f_DNR)%Samson_v10(iv)%f(1:1) = DNR_s
        Endif
    Else
        Write (ULog, 9290) iv, yyyy, mm, dd, hh, &
            'DNR_s', DNR_s, '{abcdefghABCDEFGH?}'
    End If
    If ((0 <= DNR_u) .And. (DNR_u <= 9)) Then
        If (Is_v11) Then
            Xparam(f_DNR)%Samson_v11(iv)%f(2:2) = Achar(DNR_u+ia0)
        Else
            Xparam(f_DNR)%Samson_v10(iv)%f(2:2) = Achar(DNR_u+ia0)
        Endif
    Else
        Write (ULog, 9270) iv, yyyy, mm, dd, hh, &
            'DNR_u', DNR_u, '{0..9}'
    End If
End If

! Diffuse Horizontal Radiation
If ((0 <= DHR_v) .And. (DHR_v <= 1415)) Then
    If (Is_v11) Then
        Xparam(f_DHR)%Samson_v11(iv)%v = DHR_v
        Xparam(f_DHR)%Samson_v11(iv)%s = data_source
    Else
        Xparam(f_DHR)%Samson_v10(iv)%v = DHR_v
        Xparam(f_DHR)%Samson_v10(iv)%s = data_source
    Endif
    If (Verify(DHR_s, 'abcdefghABCDEFGH?') == 0) Then
        If (Is_v11) Then
            Xparam(f_DHR)%Samson_v11(iv)%f(1:1) = DHR_s
        Else
            Xparam(f_DHR)%Samson_v10(iv)%f(1:1) = DHR_s
        Endif
    Endif

```

```

Else
  Write (ULog, 9290) iv, yyyy, mm, dd, hh, &
    'DHR_s', DHR_s, '{abcdefghABCDEFGH?}'
End If
If ((0 <= DHR_u) .And. (DHR_u <= 9)) Then
  If (Is_v11) Then
    Xparam(f_DHR)%Samson_v11(iv)%f(2:2) = Achar(DHR_u+ia0)
  Else
    Xparam(f_DHR)%Samson_v10(iv)%f(2:2) = Achar(DHR_u+ia0)
  Endif
Else
  Write (ULog, 9270) iv, yyyy, mm, dd, hh, &
    'DHR_u', DHR_u, '{0..9}'
End If
End If

! =====
! SAMSON version 1.1 files have no more parameters
If (Is_v11) Cycle Loop
! =====
Call GetElevation(pWBAN, jd_today, Xfound, Station_elevation_in_meters)

! Total Sky_Cover
If ((0 <= Total_Sky_Cover) .And. (Total_Sky_Cover <= 10)) Then
  Xparam(f_TSC)%Samson_v10(iv)%v = Total_Sky_Cover
  Xparam(f_TSC)%Samson_v10(iv)%s = data_source
End If

! 07 051-052      Opaque Sky_Cover          0-10      Amount of sky dome (in tenths)
!                                                         covered by clouds that prevent
!                                                         observing the sky or higher cloud
!                                                         layers. 99 = missing data.
If ((0 <= Opaque_Sky_Cover) .And. (Opaque_Sky_Cover <= 10)) Then
  Xparam(f_OSC)%Samson_v10(iv)%v = Opaque_Sky_Cover
  Xparam(f_OSC)%Samson_v10(iv)%s = data_source
End If

! 08 054-058      Dry Bulb Temperature      -70.0 to   Dry bulb temperature in degrees C.
!                                                         60.0      9999. = missing data.
If ((-70.0 <= Dry_Bulb_Temperature) .And. &
  (Dry_Bulb_Temperature <= +60.0)) Then
  Xparam(f_DBT)%Samson_v10(iv)%v = Dry_Bulb_Temperature
  Xparam(f_DBT)%Samson_v10(iv)%s = data_source
End If

! Dew Point Temperature
If ((-70.0 <= Dew_Point_Temperature) .And. &
  (Dew_Point_Temperature <= +60.0)) Then
  Xparam(f_DPT)%Samson_v10(iv)%v = Dew_Point_Temperature
  Xparam(f_DPT)%Samson_v10(iv)%s = data_source

```

```

End If

! 10 066-068      Relative Humidity          0-100      Relative humidity in percent.
!
! 999 = missing data.
If ((0 <= Relative_Humidity) .And. (Relative_Humidity <= 100)) Then
  Xparam(f_RH)%Samson_v10(iv)%v = Relative_Humidity
  Xparam(f_RH)%Samson_v10(iv)%s = data_source
End If

! 11 070-073      Station Pressure           700-1100     File: Station pressure in millibars.
! 1 millibar = 0.10000 kilopascal          Internal: kilopascal
If ((700 <= Station_Pressure) .And. (Station_Pressure <= 1100)) Then
  Xparam(f_SP)%Samson_v10(iv)%v = Station_Pressure * millibar_to_kilopascal
  Xparam(f_SP)%Samson_v10(iv)%s = data_source
End If

! 12 075-077      Wind Direction             0-360        Wind direction in degrees.
!
! (N = 0 or 360, E = 90, S = 180,
! W = 270) 999 = missing data.
If ((0 <= Wind_Direction) .And. (Wind_Direction <= 360)) Then
  Xparam(f_WD)%Samson_v10(iv)%v = Wind_Direction
  Xparam(f_WD)%Samson_v10(iv)%s = data_source
End If

! 13 078-082      Wind Speed                 0.0-99.0     Wind speed in m/s.
!
! 9999. or 99.0 = missing data.
!
! Wind Speed will be normalized to z=10 meters. See "Data Base Project Documentation".
If ((0.0 <= Wind_Speed) .And. (Wind_Speed < +99.0)) Then
  If (Wind_Speed .NotEqual. 0.0) Then
    u10 = Wind_Speed * 5.81 / Log(Station_elevation_in_meters/0.03)
    Xparam(f_WS)%Samson_v10(iv)%v = u10
  Else
    Xparam(f_WS)%Samson_v10(iv)%v = Wind_Speed
  End If
  Xparam(f_WS)%Samson_v10(iv)%s = data_source
End If

!! too many missing.
!! 14 083-088      Visibility                 0.0-160.9    Horizontal visibility in
!!
!! kilometers. 777.7 = unlimited
!!
!! visibility. 99999. = missing data.
If ((0.0 <= Horizontal_Visibility) .And. &
  (Horizontal_Visibility < 99999)) Then
  Xparam(f_HV)%Samson_v10(iv)%v = Horizontal_Visibility
  Xparam(f_HV)%Samson_v10(iv)%s = data_source
  If ((0.0 <= Horizontal_Visibility) .And. &
    (Horizontal_Visibility <= 160.9)) Then
    Maximum_Horizontal_Visibility = Max(Maximum_Horizontal_Visibility, Horizontal_Visibility)
  Else If (Horizontal_Visibility .Equals. 777.7) Then

```

```

        Xparam(f_HV)%Samson_v10(iv)%s = T_Unlimited
    End If
End If

! 15 089-094      Ceiling Height          0-30450  Ceiling height in meters.
!
!              77777 = unlimited ceiling height.
!              88888 = cirroform.
!              999999 = missing data.
If ((0 <= Ceiling_height) .And. (Ceiling_height < 999999)) Then
    Xparam(f_CH)%Samson_v10(iv)%v = Ceiling_height
    Xparam(f_CH)%Samson_v10(iv)%s = data_source
    If ((0 <= Ceiling_height) .And. (Ceiling_height <= 30450)) Then
        Maximum_Ceiling_Height = Max(Maximum_Ceiling_Height, Ceiling_height)
    Else If (Ceiling_height == 77777) Then
        Xparam(f_CH)%Samson_v10(iv)%s = T_Unlimited
    Else If (Ceiling_height == 88888) Then
        Xparam(f_CH)%Samson_v10(iv)%s = T_Cirroform
    End If
End If

! Observation Indicator  0 or 9  0 = Weather observation made.
!                          9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
Select Case(Observation_Indicator)
Case(0, 9)
    ! Obs made. Save the present weather table.
    Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
    Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
    Xparam(f_OI)%Samson_v10(iv)%s = data_source

    ! Check the present weather table.
    Do i = 1, 9
        If (Verify(Present_weather(i:i), PWT) == 0) Cycle    ! Ok.
        Write (ULog, 9250) iv, yyyy, mm, dd, hh, &
            'Present_weather(', i, ')', Present_weather(i:i), PWT
9250      Format (/ , 1x, '?? Read_SAMSON_File_v1x: ', &
            i7, 1x, i4, 2('-', i2.2), i3, 'h, ', a, i0, a, &
            ': invalid value: ', a, '"', expecting: '"', a, '" ' //)
    End Do

Case Default
    Write (ULog, 9270) iv, yyyy, mm, dd, hh, &
        'Observation_Indicator', Observation_Indicator, '{0 9}'
9270      Format (/ , 1x, '?? Read_SAMSON_File_v1x: ', &
            i7, 1x, i4, 2('-', i2.2), i3, 'h, ', a, &
            ': invalid value: ', i0, ', expecting: ', a, /)
End Select

! Precipitable water
If ((0 <= Precipitable_water) .And. (Precipitable_water <= 100)) Then

```

```

        Xparam(f_pH20)%Samson_v10(iv)%v = Precipitable_water
        Xparam(f_pH20)%Samson_v10(iv)%s = data_source
    End If

    ! Broadband aerosol optical_depth (broadband turbidity) on the day indicated.
    If ((0.0 <= baod) .And. (baod <= 0.900)) Then
        Xparam(f_baod)%Samson_v10(iv)%v = baod
        Xparam(f_baod)%Samson_v10(iv)%s = data_source
    End If

    ! Snow Depth
    If ((0 <= Snow_Depth) .And. (Snow_Depth <= 100)) Then
        Xparam(f_SD)%Samson_v10(iv)%v = Snow_Depth
        Xparam(f_SD)%Samson_v10(iv)%s = data_source
    End If

    ! Days Since Last Snowfall
    If ((0 <= Days_Since_Last_Snowfall) .And. (Days_Since_Last_Snowfall <= 88)) Then
        Xparam(f_DSLS)%Samson_v10(iv)%v = Days_Since_Last_Snowfall
        Xparam(f_DSLS)%Samson_v10(iv)%s = data_source
    End If
    !If (Print_now) Then
    !   Write (ULog, *) yyyy, mm, dd, hh, &
    !       ': Xparam(f_DSLS)%Samson_v10(iv) = ', Xparam(f_DSLS)%Samson_v10(iv)
    !End If

! 21 124-129      Hourly Precipitation      000000-      In inches and hundredths
!                                     099999      (See information below).
!
!   130-130      Hourly Precipitation Flag      See explanation below.
!
! DATA FORMAT--HOURLY PRECIPITATION
!
! It stands to reason that for most hours the non-occurrence of
! precipitation is prevalent. Therefore, in order to save space in
! the original digital file, there are entries only for:
!
!   1. The first day and hour of each month where observations were
!       taken even if no precipitation occurred during that month.
!
!   2. Hours with precipitation > zero.
!
!   3. Beginning and ending hours of missing periods.
!
!   4. Beginning and ending hours of accumulating periods.
!
!   5. Beginning and ending periods of deleted data.
!
!   6. First and last day of each month where the required charts
!       or forms never were received or processed at NCDC.

```

```

!
! The actual precipitation data value: The data value portion is a
! six-digit integer. Units are inches and hundredths. Range =
! 000000-099999. 000000 will be used only on the first hour of each
! month unless there is precipitation during that hour, in which case
! the measured value will be provided. On other days during the
! month without precipitation, no entry will be made. 099999
! indicates that the value is unknown.
!
! Hourly Precipitation Flag:
!
! A      Accumulated period and amount. An accumulated period
!        indicates that the precipitation amount is correct, but
!        the exact beginning and ending times are only known to
!        the extent that the precipitation occurred sometime
!        within the accumulation period. Begin accumulation data
!        value will always be 099999.
!
! D      Deleted Flag. Beginning and ending of a deleted period.
!        A deleted value indicates that the original data were
!        received, but were unreadable or clearly recognized as
!        noise.
!
! M      Missing Flag. (Beginning and ending of a missing
!        Period.) A missing flag indicates that the data were
!        not received. This flag appears on the first and last
!        day of each month for which data were not received or
!        not processed by NCDC. Prior to 1984 a missing period
!        was recorded as " 00000M" at the beginning and ending
!        hours. If precipitation occurred during the last hour
!        of the missing period, the second M appears with a non-
!        zero value. Beginning in 1984 the beginning and ending
!        hours of the missing period are recorded as "099999M".
!
! b      Blank. No Flag needed.
!
! Examples:
!
! The precipitation accumulation from 1st month day 02 to 2nd month
! day 04:
!
!      01      0002      0500      000030b
!      1000      099999A      Accumulation begins
!      02      0001      0100      099999A      Accumulation continues
!      0004      1400      000390A      Accumulation ends
!
! Accumulated precipitation for 1 month only:
!
!      01      0002      1000      099999A      Accumulation begins
!      0031      2400      000320A      Accumulation ends

```

```

!
! Accumulated, deleted, and missing precipitation data through months
! 01 and 02:
!
!      01      0001      0100      000000b      First record of the
!                                     month
!      02      0002      1100      099999A      Accumulation begins
!      02      0001      0100      099999A      Accumulation continues
!                                     1400      000630A      Accumulation ends
!      02      0028      1500      099999D      Deleted data begins
!      02      0028      1300      099999D      Deleted data ends
!                                     1400      099999M      Missing data
!                                     2400      099999M      Missing data
!
! Required precipitation charts or forms were never received at NCDC:
!
!      01      0001      0100      099999M      Missing data
!      02      0031      0100      099999M
!      02      0001      0100      099999M
!      02      0028      0100      099999M
!
! Again Hourly Precipitation In inches and hundredths
! Example: WBAN 14914, 1961 range: 0 - 218; i.e., 218 represents 2.18 inches.

If ((0 <= Hourly_Precipitation) .And. (Hourly_Precipitation < 099999)) Then
  Xparam(f_HP)%Samson_v10(iv)%v = inches__to__cm * Hourly_Precipitation / 100.0
  Xparam(f_HP)%Samson_v10(iv)%s = data_source
End If

! Valid range: A D M E(undocumented) Blank
Xparam(f_HP)%Samson_v10(iv)%f = Hourly_Precipitation_Flag
Select Case(Hourly_Precipitation_Flag)
Case ('A', 'D', 'M', 'E', ' ')
  ! Ok.
Case Default
  ! Invalid value.
  Write (ULog, 9290) iv, yyyy, mm, dd, hh, &
    'Hourly_Precipitation flag', &
    Hourly_Precipitation_Flag, &
    '{A D M E Blank}'
9290   Format (/ , 1x, '?? Read_SAMSON_File_v1x: ', &
    i7, 1x, i4, 2('-',i2.2), i3, 'h, ', a, &
    ': invalid value: "', a, '"; Expecting: ', a, /)
End Select
End Do Loop
9999 Continue
Call IOClose(jin)
Empty_File = (nlines == 0)
End Subroutine Read_SAMSON_v1x
End Module SAMSON

```

## setup

! Last change: LSR 17 May 2002 6:11 pm

### Module Setup

```
Use Date_Module
Use F2kCLI
Use FileStuff
Use Global_Variables
Use Stats
Use Utils1
Use IoSubs
Implicit None
```

### Contains

#### Subroutine InitialSetUp()

```
Implicit None
Character(Len=30) :: ydate = ''
Character(Len=80) :: xexe = ''
Character(Len=80) :: xversion = ''
Integer :: nn

! Get the directory delimiter
DirDelim = DirectoryDelimiter()

ydate = Unix_Date()
Call Get_Command_Argument(0, xexe) ! Get application name.
! Remove possible path.
nn = 1 !+ Index(xexe, DirDelim, Back=.True.)
Call FileTimeStamp(xexe, xversion)

Call STD_dir(Log_dir)
Call STD_dir(R0_root)
Call STD_dir(Raw_Data_dir)

Call Check_Drives_and_Files()

! Open the log file.
Call Get_Log_File_Name(Log_file, Log_dir, ULog)

! Dump data for Mathematica here
Call IOwrite(Umath, 'Omath.del')

! Time stamp for generated files.
TimeStamp = ISO_Date()
```

```

! Header for the log file: date and files used.
Write (ULog, *) 'Executable: ', Trim(xexe(nn:))
Write (ULog, *) 'Version ..: ', Trim(xversion)
Write (ULog, *) 'run of ...: ', Trim(ydate)
Write (ULog, *) 'TimeStamp : ', Trim(TimeStamp)

jd0 = Jd(MinYear, 01, 01)
jd1 = Jd(MaxYear, 12, 31)

Write (ULog, 9130)
9130 Format (/ &
      1x, 'Notes:', /, &
      1x, '* None as of 11 Feb 2002  2:27 pm', /)

! To Do List: <A NAME="setup.f90#To Do List:">
Write (ULog, 9150)
9150 Format ( &
      1x, 'To Do List: <A HREF="setup.f90#To Do List:">', /, &
      1x, '* Look for """" "!!" "!!!"', /)

Call SetFieldInfo()
Call Set_Hours_since_Jd0()

!Call Test_Winteracter()
!Call Test_Multiples_of()
!Call Test_Azimuth()
!Call Test_NaN()
!Call Test_iv_to_ymdh()

Call FLushAll()

End Subroutine InitialSetUp

Subroutine Check_Drives_and_Files()

Implicit None
Character(Len=10), Dimension(2), Parameter :: xDrive = (/ 'v:\.', 'z:\.' /)
Logical :: have_drive
Logical :: have_file
Integer :: nerr, j, n

nerr = 0

! Verify that the substituted drives are present.
Do j = 1, Ubound(xDrive,1)
  Inquire(File=xDrive(j), Exist=have_drive)
  If (.Not. have_drive) Then
    n = Len_trim(xDrive(j)) - 2  ! Do not display trailing "\"
    Write(ULog, 9130) xDrive(j)(1:n)
  End If
End Do

```

```

9130      Format (1x, '?? Check_Drives_and_Files: Could not find drive ', a)
          nerr = nerr + 1
        End If
      End Do

      ! make sure "zcat" exists.
      Inquire(File=zcat, Exist=have_file)
      If (.Not. have_file) Then
        Write (ULog, *) '?? Check_Drives_and_Files: Could not find file zcat: ', Trim(zcat)
        nerr = nerr + 1
      End If

      ! make sure metadata file exists.
      Inquire(File=metadata_coda, Exist=have_file)
      If (have_file) Then
        Call IORead(Umetadata, metadata_coda)
      Else
        Write (ULog, *) '?? Check_Drives_and_Files: Could not find file: ', Trim(metadata_coda)
        nerr = nerr + 1
      End If

      If (nerr > 0) Then
        Stop '?? Missing drives/files'
      End If

End Subroutine Check_Drives_and_Files

Subroutine Get_Log_File_Name(Log_Name, Log_Path, ULog)

! ** insert subroutine in
! <A HREF="e:\5\Fortran\f90Lib\genutils.f90#$1">

! Generate log file name, e.g., 'Logs\2001-10-03_090547.log';
! If Log_Path is present, it must terminate with the directory delimiter, 'Logs\';
! If ULog is present, Log_Name will be opened with write access and attached to ULog,

Implicit None
Character(Len=*),          Intent(Out) :: Log_Name
Character(Len=*), Optional, Intent(In)  :: Log_Path
Integer,                  Optional, Intent(Out) :: ULog

Character(Len=MaxNamLen) :: tbuf, tdir
Character(Len=08) :: xdate ! yyyymmdd
Character(Len=10) :: xtime ! hhmmss.sss

Call Date_and_time(Date = xdate, Time = xtime)

!           1           2
!       123456789-123456789-1

```

```

! tbuf 2001-10-03_090547.log
! xdate yyyymmdd
! xtime hhmmss.sss
!      123456789-123456789-1
tbuf = 'yyyy-mm-dd_hhmmss.log'

tbuf(01:04) = xdate(1:4) ! year
tbuf(06:07) = xdate(5:6) ! Month
tbuf(09:10) = xdate(7:8) ! day of Month
tbuf(12:17) = xtime(1:6) ! hhmmss

If (Present(Log_Path)) Then
    tdir = Log_Path
    Call STD_dir(tdir)
    Log_Name = Trim(tdir) // tbuf
Else
    Log_Name = tbuf
End If

If (Present(ULog)) Then
    Call IOWrite(ULog, Log_Name)
End If
End Subroutine Get_Log_File_Name

```

```
Subroutine SetFieldInfo()
```

```
Implicit None
```

```
! Parameter Zero: Other counts
```

```
FieldInfo(0)%Name = 'Year counts'
```

```
FieldInfo(0)%Minimum_obs_per_day = 0 ! Unused for this field.
```

```
! *** 12 hours below: To check the range of the (25th) daily value,
! multiply the maximum hourly value by 12 hours.
```

```
! 11 Feb 2002 5:12 pm: Well, in Alaska daylight can last 24 hours
```

```
! FAO Ra == Extraterrestrial Horizontal Radiation
```

```
! Extraterrestrial Horizontal Radiation - Amount of solar radiation in Wh/m2
```

```
! received on a horizontal surface at the top of the atmosphere during the
! 60 minutes preceding the hour indicated.
```

```
FieldInfo(f_EHR)%Name = 'Extraterrestrial Horizontal Radiation [Wh/m2] (Ra)'
```

```
FieldInfo(f_EHR)%Minimum_obs_per_day = 3
```

```
FieldInfo(f_EHR)%minimum_value = Zero
```

```
FieldInfo(f_EHR)%maximum_value = 1415 * 24 ! 24 hours
```

```
FieldInfo(f_EDNR)%Name = 'Extraterrestrial Direct Normal Radiation [Wh/m2]'
```

```
FieldInfo(f_EDNR)%Minimum_obs_per_day = 3
```

```
FieldInfo(f_EDNR)%minimum_value = Zero
```

```
FieldInfo(f_EDNR)%maximum_value = 1415 * 24 ! 24 hours
```

```

! MET Solar Radiation == FAO Rs == Global Horizontal Radiation
!
! == samson GHRv == huswo iGRAD
! Global Horizontal Radiation - Total amount of direct and diffuse solar
! radiation in Wh/m2 received on a horizontal surface during the
! 60 minutes preceding the hour indicated.
FieldInfo(f_GHR)%Name = 'Global Horizontal Radiation [Wh/m2] (Rs)'
FieldInfo(f_GHR)%Minimum_obs_per_day = 3
FieldInfo(f_GHR)%minimum_value = Zero
FieldInfo(f_GHR)%maximum_value = 1415 * 24 ! 24 hours

FieldInfo(f_DNR)%Name = 'Direct Normal Radiation [Wh/m2]'
FieldInfo(f_DNR)%Minimum_obs_per_day = 3
FieldInfo(f_DNR)%minimum_value = Zero
FieldInfo(f_DNR)%maximum_value = 1415 * 24 ! 24 hours

FieldInfo(f_DHR)%Name = 'Diffuse Horizontal Radiation [Wh/m2]'
FieldInfo(f_DHR)%Minimum_obs_per_day = 3
FieldInfo(f_DHR)%minimum_value = Zero
FieldInfo(f_DHR)%maximum_value = 1415 * 24 ! 24 hours

! Total Sky Cover
FieldInfo(f_TSC)%Name = 'Total Sky Cover [tenths]'
FieldInfo(f_TSC)%Minimum_obs_per_day = 3
FieldInfo(f_TSC)%minimum_value = Zero
FieldInfo(f_TSC)%maximum_value = 10

! Opaque Sky_Cover - Amount of sky dome (in tenths) covered by clouds that prevent
! observing the sky or higher cloud layers.
FieldInfo(f_OSC)%Name = 'Opaque_Sky_Cover [tenths]'
FieldInfo(f_OSC)%Minimum_obs_per_day = 3
FieldInfo(f_OSC)%minimum_value = Zero
FieldInfo(f_OSC)%maximum_value = 10

! Dry bulb temperature in degrees C.
FieldInfo(f_DBT)%Name = 'Dry Bulb Temperature [°C]'
FieldInfo(f_DBT)%Minimum_obs_per_day = 3
FieldInfo(f_DBT)%minimum_value = -70.0
FieldInfo(f_DBT)%maximum_value = +60.0

! Dew Point Temperature in degrees C.
FieldInfo(f_DPT)%Name = 'Dew Point Temperature [°C]'
FieldInfo(f_DPT)%Minimum_obs_per_day = 3
FieldInfo(f_DPT)%minimum_value = -80.0
FieldInfo(f_DPT)%maximum_value = +60.0

! Relative humidity in percent.
FieldInfo(f_RH)%Name = 'Relative_Humidity [Percent]'
FieldInfo(f_RH)%Minimum_obs_per_day = 3
FieldInfo(f_RH)%minimum_value = Zero
FieldInfo(f_RH)%maximum_value = 100

```

```

! Station pressure in kPa.
FieldInfo(f_SP)%Name = 'Station_Pressure [kPa]'
FieldInfo(f_SP)%Minimum_obs_per_day = 1
FieldInfo(f_SP)%minimum_value = 70.0
FieldInfo(f_SP)%maximum_value = 110.0

! Wind direction in degrees.
! (N = 0 or 360, E = 90, S = 180, W = 270)
FieldInfo(f_WD)%Name = 'Wind_Direction [Degrees]'
FieldInfo(f_WD)%Minimum_obs_per_day = 3
FieldInfo(f_WD)%minimum_value = Zero
FieldInfo(f_WD)%maximum_value = 360

! Wind Speed in meters/sec at height = 10 meters
FieldInfo(f_WS)%Name = 'Wind_Speed @z=10m [m/s]'
FieldInfo(f_WS)%Minimum_obs_per_day = FieldInfo(f_WD)%Minimum_obs_per_day
FieldInfo(f_WS)%minimum_value = Zero
FieldInfo(f_WS)%maximum_value = 98.9

! Horizontal Visibility
FieldInfo(f_HV)%Name = 'Horizontal_Visibility [km]'
FieldInfo(f_HV)%Minimum_obs_per_day = 3
FieldInfo(f_HV)%minimum_value = Zero
FieldInfo(f_HV)%maximum_value = Zero ! See Process_Set. Maximum value is station-dependent

! Ceiling Height
FieldInfo(f_CH)%Name = 'Ceiling_Height [m]'
FieldInfo(f_CH)%Minimum_obs_per_day = 3
FieldInfo(f_CH)%minimum_value = Zero
FieldInfo(f_CH)%maximum_value = Zero ! See Process_Set. Maximum value is station-dependent

! Observation Indicator 0 or 9 0 = Weather observation made.
! 9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
FieldInfo(f_OI)%Name = 'Observation_Indicator/Present_weather'
FieldInfo(f_OI)%Minimum_obs_per_day = 1
FieldInfo(f_OI)%minimum_value = Zero ! N/A for this parameter
FieldInfo(f_OI)%maximum_value = Zero ! N/A for this parameter

! Precipitable Water
FieldInfo(f_pH2O)%Name = 'Precipitable_Water [mm]'
FieldInfo(f_pH2O)%Minimum_obs_per_day = 3
FieldInfo(f_pH2O)%minimum_value = Zero
FieldInfo(f_pH2O)%maximum_value = 100

! Broadband Aerosol Optical_Depth (0.0-0.90) - Broadband aerosol optical_depth
! (broadband turbidity) on the day indicated.
FieldInfo(f_baod)%Name = 'Broadband_Aerosol_Optical_Depth []'
FieldInfo(f_baod)%Minimum_obs_per_day = 1
FieldInfo(f_baod)%minimum_value = Zero

```

```

FieldInfo(f_baod)%maximum_value = 0.900

FieldInfo(f_SD)%Name = 'Snow Depth [cm]'
FieldInfo(f_SD)%Minimum_obs_per_day = 3
FieldInfo(f_SD)%minimum_value = Zero
FieldInfo(f_SD)%maximum_value = 100

FieldInfo(f_DSLS)%Name = 'Days since last Snowfall [days]'
FieldInfo(f_DSLS)%Minimum_obs_per_day = 3
FieldInfo(f_DSLS)%minimum_value = Zero
FieldInfo(f_DSLS)%maximum_value = 88

! Hourly Precipitation
FieldInfo(f_HP)%Name = 'Hourly Precipitation [cm]'
FieldInfo(f_HP)%Minimum_obs_per_day = 3
FieldInfo(f_HP)%minimum_value = Zero
FieldInfo(f_HP)%maximum_value = 400

FieldInfo(f_FAO_SG_PET)%Name = 'Eto, FAO Short Grass [mm/day]'
FieldInfo(f_FAO_SG_PET)%Minimum_obs_per_day = 0
FieldInfo(f_FAO_SG_PET)%minimum_value = -1
FieldInfo(f_FAO_SG_PET)%maximum_value = 35

FieldInfo(f_KP_FWS_Evaporation)%Name = 'K-P FWS Evaporation [mm/day]'
FieldInfo(f_KP_FWS_Evaporation)%Minimum_obs_per_day = 0
FieldInfo(f_KP_FWS_Evaporation)%minimum_value = -10
FieldInfo(f_KP_FWS_Evaporation)%maximum_value = +20

FieldInfo(f_Ep)%Name = 'Ep, Class A pan Evaporation [mm/day]'
FieldInfo(f_Ep)%Minimum_obs_per_day = 0
FieldInfo(f_Ep)%minimum_value = -1
FieldInfo(f_Ep)%maximum_value = 180

! MET field information.
MET_field(g_Precipitation)%Name = 'Precipitation [cm/day]'
MET_field(g_Pan_Evaporation)%Name = 'Pan Evaporation [cm/day]'
MET_field(g_Temperature_mean)%Name = 'Temperature mean [°C]'
MET_field(g_Wind_Speed)%Name = 'Wind Speed @10 meter [cm/s]'
MET_field(g_Solar_Radiation)%Name = 'Solar Radiation [Langleys/day]'
MET_field(g_FAO_Short_Grass)%Name = FieldInfo(f_FAO_SG_PET)%Name
MET_field(g_Daylight_Station_Pressure)%Name = 'Daylight Station Pressure [kiloPascal]'
MET_field(g_Daylight_Relative_Humidity)%Name = 'Daylight Relative Humidity [%]'
MET_field(g_Daylight_Opaque_Sky_Cover)%Name = 'Daylight Opaque Sky Cover [tenths]'
MET_field(g_Daylight_Temperature)%Name = 'Daylight Temperature [°C]'
MET_field(g_Daylight_Broadband_Aerosol)%Name = 'Daylight Broadband Aerosol [optical depth]'
MET_field(g_Daylight_Mean_Wind_Speed)%Name = 'Daylight Mean Wind Speed @ 10 meters [m/s]'
MET_field(d_Daylight_max_wind_speed)%Name = 'Maximum Daylight Mean Wind Speed @10m [m/s]'
MET_field(d_Daylight_direction_of_max_wind_speed)%Name = 'Direction of Maximum Daylight Wind [degrees]'
MET_field(d_PWS)%Name = 'Daylight Prevailing Wind_Speed @z=10m [m/s]'
MET_field(d_PWD)%Name = 'Daylight Prevailing Wind_Direction [Degrees]'

```

```
End Subroutine SetFieldInfo
```

```
Subroutine Test_Winteracter()
```

```
Use Winteracter
Implicit None
Integer :: uu = 6
Character(Len=200) :: tbuf, wbuf, tdir
Character(Len=200), Dimension(50) :: vfiles
Integer :: nn, j, ierr
Logical :: have_dir
```

```
Write (uu, *) ' *** Winteracter *** '
```

```
Call IOsArgument(0, tbuf)
Write (uu, *) 'Exe name: ', Trim(tbuf)
```

```
tdir = 's:\ '
have_dir = IOsDirExists(tdir)
Write (uu, *) 'Have directory ', Trim(tdir), ' ? ', have_dir
```

```
! Check create directory; Get rid of the path and create
! the new directory in the current directory.
! Warning: cannot create C:\a\b\c where \a\b does not exist.
```

```
Call Temp_File_Name(tdir)
j = Index(tdir, '\', Back=.True.) + 1
tdir = '000.' // tdir(j:)
If (.Not. IOsDirExists(tdir)) Then
  Write (uu, *) 'Directory ', Trim(tdir), ' did not exist; creating ....'
  Call IOsDirMake(tdir)
  have_dir = IOsDirExists(tdir)
  Write (uu, *) 'Directory ', Trim(tdir), ' exists ? ', have_dir
```

```
Else
  Write (uu, *) 'Bummer. Directory ', Trim(tdir), ' exists; will not create.'
End If
```

```
! %temp% == "e:\TEMP"
! Note: no trailing slash.
tbuf = 'temp'
Call IOsVariable(tbuf, wbuf) ! wbuf == e:\TEMP
Write (uu, 9130) Trim(tbuf), Trim(wbuf)
9130 Format (1x, '%', a, '% == "', a, '"')
```

```
! does not look into subdirs.
Vfiles(1) = '**None**'
tdir = 'z:\'
tbuf = '14914_*.*'
nn = 1
```

```

Do j = 1, 2
  Call IOsDirEntryType('FD')
  Call IOsDirInfo(tdir, tbuf, vfiles, nn)
  ierr = InfoError(1)
  Write (uu, *) 'IOsDirInfo: nn = ', nn, '; ierr = ', ierr
  Write (uu, *) 'Vfiles(1) == ', Vfiles(1), ''
  nn = Ubound(vfiles,1)
End Do

  Stop 'Stopping at Test_Winteracter'

End Subroutine Test_Winteracter

Subroutine Test_Multiples_of()

  Implicit None

  Integer :: iNum
  Integer :: uu = 9021

  iNum = 25
  Write (uu, *) '!! iNum = ', iNum
  Call tt0(25, 50)
  Call tt0(23, 24)
  Call tt0( 1, 24)
  Call tt0(24, 25)
  Call tt0(25, 25)
  Call tt0(24, 26)
  Call tt0(48, 48)
  Call tt0(48, 52)
  Call tt0( 2, 55)

  iNum = 3
  Write (uu, *) '!! iNum = ', iNum
  Call tt0(1, 2) ! 0
  Call tt0(2, 4) ! 1
  Call tt0(2, 8) ! 2
  Call tt0(27, 29) ! 1
  Call tt0(26, 33) ! 3
  Call tt0(26, 34) ! 3
  Call tt0(33, 33) ! 1
  Call tt0(33, 39) ! 3

  Stop 'Stopping at Test_Multiples_of'

Contains

  Subroutine tt0(k0, k1)

```

```

Implicit None
Integer, Intent(In) :: k0, k1

Integer :: n

n = Multiples_of(iNum, k0, k1)
Write (uu, 9130) k0, k1, n
9130 Format (1x, '!!!!', i6, 3x, i6, 3x, i4)

End Subroutine tt0

End Subroutine Test_Multiples_of

Subroutine Test_Azimuth()

! Wind Direction          0-360      Wind direction in degrees.
!                          (N = 0 or 360, E = 90, S = 180,
!                          W = 270)
!
! Wind Speed              0.0-99.0   Wind speed in m/s.
!
!           0                90
!           ^                ^
!           |                |
!           |                |
! 270 <-----+-----> 90   180 <-----+-----> 0
!           |                |
!           |                |
!           v                v
!           180              270
!
!           Azimuth          Theta
!
! Azimuth: Wind direction in degrees.
!           (N = 0 or 360, E = 90, S = 180, W = 270)
!
! Theta: regular angle measured counterclockwise
!         from the +x axis.
!
! Interconversion Formulae:
!   Theta = Modulo(90-Azimuth, 360)
!   Azimuth = Modulo(90-Theta, 360)
!
! From Mathematica, for the test case: (ArcTan(x/y) + Pi) == Azimuth
! Note that Theta = ArcTan(y/x) + Pi

```

Azimuth	Theta	Azimuth from Theta
0.0	90.0	0.0
15.0	75.0	15.0
30.0	60.0	30.0
45.0	45.0	45.0
60.0	30.0	60.0
75.0	15.0	75.0
90.0	0.0	90.0
105.0	345.0	105.0
120.0	330.0	120.0
135.0	315.0	135.0
150.0	300.0	150.0
165.0	285.0	165.0
180.0	270.0	180.0
195.0	255.0	195.0
210.0	240.0	210.0
225.0	225.0	225.0
240.0	210.0	240.0
255.0	195.0	255.0
270.0	180.0	270.0
285.0	165.0	285.0
300.0	150.0	300.0
315.0	135.0	315.0
330.0	120.0	330.0
345.0	105.0	345.0
360.0	90.0	0.0

?? ! 0 is equivalent to 360. Result is ok.

```

Implicit None
Real :: x0, x1, xinc
Real :: aj ! Azimuth
Real :: tj ! Theta
Real :: bj ! Azimuth computed from Theta
Integer :: npts, j
Integer :: Jout = 200
Character(Len=2) :: tcoda

```

```

x0 = 0
x1 = 360
xinc = 15
npts = Ceiling((x1-x0)/xinc)
Write (Jout, 9130) 'Azimuth', 'Theta', 'Azimuth from Theta'
9130 Format (1x, a10, 3x, a10, 3x, a18)

```

```

Do j = 0, npts
  aj = x0 + j*xinc
  tj = Modulo(90-aj, 360)
  bj = Modulo(90-tj, 360)
  If (Abs(aj-bj) <= 1e-6) Then
    tcoda = ' '

```

```

        Else
            tcoda = '??'
        End If
        Write (Jout, 9150) aj, tj, bj, tcoda
9150    Format (1x, f10.1, 3x, f10.1, 3x, f18.1, 4x, a)
        End Do

        Stop 'Stopping at Test_Azimuth'
End Subroutine Test_Azimuth

```

```
Subroutine Test_NaN()
```

```

! Zero/Zero generates a NAN
Implicit None

Real :: r0, r1, r2

r0 = Zero
r1 = Zero
r2 = r1 / r0

If (IsNaN(r2)) Then
    Write (6,*) 'Zero/Zero == NAN'
Else
    Write (6,*) 'Zero/Zero /= NAN'
End If

```

```
End Subroutine Test_NaN
```

```
Subroutine Test_iv_to_ymdh()
```

```

! <A NAME="Test_iv_to_ymdh">
! <A HREF="setup.f90#Test_iv_to_ymdh">

!      iv  Date          nperiods  ymax
!  -----  -----
!      0   1960-12-31 25h      0      1961
!      1   1961-01-01 1h       1      1962
!  273924  1990-12-31 24h     30      1991
!  999999  2070-07-07 24h    110      2071
!  9999999 3056-02-29 24h   1096      3057

! Given iv, this subroutine computes the corresponding yyyy-mm-dd hh
! and, optionally, doy (day of the year).
!
! Store data in a vector array so that we can find gaps easier.
! Assume a virtual array: vData(ndays, Nhours)

```

```

!      Ndays = 365 or 366 == WFlx%Expected_Ndays
!
!      Ndays|  1    2  3  4  ...    24    25    Nhours = 25
!      -----+-----
!      1  |  1    2  3  4  ...    24    25
!      2  | 26   27 28 29 ...    49    50
!      :  |  :    :  :  :  ...    :    :
!      365| 9101 :  :  :  ...    9124  9125
!      366| 9126 .  .  .  ...    9149  9150
!
! Samson_vlx(iv) <=> vData(doy,hh)
!      iv = (doy-1)*Nhours + hh
!
!      doy = (iv+Nhours-1) / Nhours
!      hh  = iv - (doy-1)*Nhours
!
! nbase = Hours_since_Jd0(yyyy)
! iv = (doy-1)*Nhours + hh + nbase
! Xparam(f_EHR)%Samson_v10(iv)%v = EHR

Implicit None
Integer :: yyyy, mm, dd, hh
Integer :: jv, jdoy, nbase, ymax, i, nperiods

Integer, Parameter :: MaxN = 5
Integer, Dimension(MaxN) :: iv = (/ 0, 1, 273924, 999999, 9999999 /)

Do i = 1, MaxN
  jv = iv(i)

  ! nperiods: years since MinYear
  nperiods = Ceiling(jv/(365.2425*Nhours))
  ymax = nperiods + MinYear

  Write (ULog, *)
  Write (ULog, *) 'jv.....: ', jv
  Write (ULog, *) 'nperiods...: ', nperiods
  Write (ULog, *) 'ymax.....: ', ymax

  ! from the table above, it would appear that yyyy(output) == ymax-1
  Do yyyy = ymax+10, 0, -1
    !! Hours_since_Jd0(yyyy) = (Jd(yyyy,01,01) - Jd0) * Nhours
    !!nbase = Hours_since_Jd0(yyyy)
    nbase = (Jd(yyyy,01,01) - Jd0) * Nhours
    If (jv > nbase) Exit
  End Do
  Write (ULog, *) 'nperiods*Nhours.....: ', nperiods*Nhours
  Write (ULog, *) '(Jd(yyyy,01,01) - Jd0) .....: ', (Jd(yyyy,01,01) - Jd0)
  Write (ULog, *) '(Jd(yyyy,01,01) - Jd0) * Nhours...: ', (Jd(yyyy,01,01) - Jd0) * Nhours

```

```
    jv = jv - nbase
    jdoy = (jv+Nhours-1) / Nhours

    hh = jv - (jdoy-1)*Nhours
    Call Calend(yyyy, jdoy, mm, dd)

    Write(ULog, 9130) iv(i), yyyy, mm, dd, hh
9130    Format (1x, '## Test_iv_to_ymdh: ', i7, 1x, i4, 2('-',i2.2), i3, 'h')

    End Do
End Subroutine Test_iv_to_ymdh

End Module Setup
```

## Stats

! Last change: LSR 12 Mar 2002 9:05 am

Module Stats

Use Global\_Variables

```
! Numerically stable computation of the variance.
!
!
!           1      n
!   Sample variance = ----- Sum (x_i-x_mean)^2
!                   n - 1  i=1
! Reference:
! [2] Nicholas J. Higham. 1996. Accuracy and Stability of Numerical
!     Algorithms. SIAM (Society for Industrial & Applied Mathematics).
!     ISBN 0-89871-355-2. Page 13.
!
! Accumulate:
!
!           1      k
!   M_k = - * Sum x_i
!           k     i=1
!
!           k           k           1      k
!   Q_k = Sum (x_i - M_k)^2 = Sum (x_i)^2 - - (Sum x_i)^2
!           i=1           i=1           k  i=1
!
! Updating formulae:
!
!   M_1 = x_1
!
!   M_k = M_{k-1} + -----,      k = 2..n
!                   x_k - M_{k-1}
!                   k
!
!   Q_1 = 0
!
!   Q_k = Q_{k-1} + -----,      k = 2..n
!                   (k-1) (x_k - M_{k-1})^2
!                   k
!
! After which:
!
!   Sample Mean = M_n
!
!
!           Q_n
!   Sample Variance = -----
!                   n - 1
!
! Note that the updating formulae can be written:
!
!   M_0 = 0
!
!           x_k - M_{k-1}
```

```

!   M_k = M_{k-1} + -----,          k = 1..n
!                   k
!   Q_0 = 0
!                   (k-1) (x_k - M_{k-1})^2
!   Q_k = Q_{k-1} + -----, k = 1..n
!                   k

```

```

Implicit None
Public
Real, Private, Parameter :: rZero = 0.0

```

Contains

```

Subroutine Stat_Test(Uout, xOk)

```

```

! Simple test of module "Stats"

```

```

Implicit None
Integer, Intent(In)  :: Uout
Logical, Intent(Out) :: xOk

```

```

Integer :: k
Real      :: Eps
Type(Stat_Block) :: Xblock

```

```

! Test set #1
Real, Dimension(3) :: Set1 = (/ 10000.0, 10001.0, 10002.0 /)
Real                :: Mset1 = 10001.0 ! Mean of Set1
Real                :: Vset1 = 1.0 ! Variance of Set1

```

```

! Test set #2
Real, Dimension(7) :: Set2 = (/ 6.5, 3.8, 6.6, 5.7, 6.0, 6.4, 5.3 /)
Real                :: Mset2 = 5.75714 ! Mean of Set2
Real                :: Vset2 = 0.962857 ! Variance of Set2

```

```

9130 Format (1x, '## Test of ', a, ' passed.')

```

```

9150 Format (1x, '?? Test of ', a, ' failed.')

```

```

9170 Format (1x, ' Mean: ', f12.5, ':, ' (True=', f12.5, ')')

```

```

9190 Format (1x, ' Var : ', f12.5, ':, ' (True=', f12.5, ')')

```

```

! ===== Test set #1
! Initialize accumulator block
Call Stat_Initialize(Xblock)

```

```

! Add points to accumulator
Do k = 1, Ubound(Set1,1)
  Call Stat_Add_Point(Xblock, Set1(k))
End Do

```

```

! Compute results
Call Stat_Results(Xblock)

Eps = Epsilon(Vset1)
xOk = (Abs(Xblock%xMean-Mset1) < Eps) .And. &
      (Abs(Xblock%xVariance-Vset1) < Eps)

If (xOk) Then
  Write(Uout, 9130) 'Set1'
  Write(Uout, 9170) Xblock%xMean
  Write(Uout, 9190) Xblock%xVariance
Else
  Write(Uout, 9150) 'Set1'
  Write(Uout, 9170) Xblock%xMean, Mset1
  Write(Uout, 9190) Xblock%xVariance, Vset1
End If

! ===== Test set #2
! Initialize accumulator block
Call Stat_Initialize(Xblock)

! Add points to accumulator
Do k = 1, Ubound(Set2,1)
  Call Stat_Add_Point(Xblock, Set2(k))
End Do

! Compute results
Call Stat_Results(Xblock)

! Epsilon(Vset2) = 1.19209290E-07
! Abs(Xblock%xMean-Mset2) = 2.38418579E-06
! Abs(Xblock%xVariance-Vset2) = 5.96046448E-08
Eps = 1.0e-5 ! Kludge
xOk = (Abs(Xblock%xMean-Mset2) < Eps) .And. &
      (Abs(Xblock%xVariance-Vset2) < Eps)

!Write(Uout,*) 'Eps = ', Eps
!Write(Uout,*) 'Epsilon(Vset2) = ', Epsilon(Vset2)
!Write(Uout,*) 'Abs(Xblock%xMean-Mset2) = ', Abs(Xblock%xMean-Mset2)
!Write(Uout,*) 'Abs(Xblock%xVariance-Vset2) = ', Abs(Xblock%xVariance-Vset2)

If (xOk) Then
  Write(Uout, 9130) 'Set2'
  Write(Uout, 9170) Xblock%xMean
  Write(Uout, 9190) Xblock%xVariance
Else
  Write(Uout, 9150) 'Set2'
  Write(Uout, 9170) Xblock%xMean, Mset2
  Write(Uout, 9190) Xblock%xVariance, Vset2

```

```

End If

End Subroutine Stat_Test

Subroutine Stat_Initialize(Xblock, Header)

  Implicit None
  Type(Stat_Block),          Intent(Out) :: Xblock
  Character(Len=*), Optional, Intent(In)  :: Header

  Xblock%k = 0
  Xblock%M_k = rZero
  Xblock%Q_k = rZero
  Xblock%xMean = -Huge(rZero)
  Xblock%xVariance = -Huge(rZero)
  Xblock%xmin = +Huge(rZero)
  Xblock%xmax = -Huge(rZero)

  If (Present(Header)) Then
    Xblock%Header = Header
  Else
    Xblock%Header = ''
  End If

End Subroutine Stat_Initialize

Subroutine Stat_Add_Point(Xblock, x_k)

  Implicit None
  Type(Stat_Block), Target, Intent(InOut) :: Xblock
  Real,          Intent(In)      :: x_k

  Real,  Pointer :: M_k, Q_k
  Real   :: M_kml  ! M_{k-1}
  Integer, Pointer :: k

  k => Xblock%k
  M_k => Xblock%M_k
  Q_k => Xblock%Q_k

  k = k + 1
  !If (k >= 2) Then
  !  M_kml = M_k
  !  M_k = M_kml + (x_k-M_kml)/Real(k)
  !  Q_k = Q_k + (Real(k-1)/Real(k))*(x_k-M_kml)**2
  !Else

```

```

!   ! k = 1
!   M_k = x_k
!   Q_k = rZero
!End If
M_kml = M_k
M_k = M_kml + (x_k-M_kml)/Real(k)
Q_k = Q_k + (Real(k-1)/Real(k))*(x_k-M_kml)**2
Xblock%xmin = Min(x_k, Xblock%xmin)
Xblock%xmax = Max(x_k, Xblock%xmax)

End Subroutine Stat_Add_Point

Subroutine Stat_Results(Xblock)

  Implicit None
  Type(Stat_Block), Target, Intent(InOut) :: Xblock

  Xblock%xMean = Xblock%M_k

  If (Xblock%k >= 2) Then
    Xblock%xVariance = Xblock%Q_k/Real(Xblock%k-1)
  Else
    ! Variance not defined for k<=1
    Xblock%xVariance = -Huge(rZero)
  End If

End Subroutine Stat_Results

Subroutine Stat_Output(Uout, Xblock, Header)

  ! Output data descriptive statistics.
  ! Calls: Stat_Results(Xblock)

  Implicit None
  Integer,          Intent(In)    :: Uout
  Type(Stat_Block), Intent(InOut) :: Xblock
  Character(Len=*), Optional, Intent(In) :: Header

  Real :: std_dev ! Sample standard deviation

  ! Compute results
  Call Stat_Results(Xblock)

  Write(Uout, *)

  If (Present(Header)) Then

```

```

        Write(Uout, 9130) Trim(Header)
    Else If (Len_trim(Xblock%Header) > 0) Then
        Write(Uout, 9130) Trim(Xblock%Header)
    End If
9130 Format (1x, a)

    Write(Uout, 9150) Xblock%k, Xblock%xMean, Xblock%xVariance
9150 Format ( &
        1x, '  n ....: ', i0, /, &
        1x, '  Mean .: ', lpg14.6, /, &
        1x, '  Var ..: ', lpg14.6)

    If (Xblock%xVariance > rZero) Then
        std_dev = Sqrt(Xblock%xVariance)
        Write(Uout, 9170) std_dev
9170 Format (1x, '  StdDev: ', lpg14.6)
    End If

    Write(Uout, 9190) Xblock%xmin, Xblock%xmax
9190 Format (1x, '  Range : ', 2(1x,lpg14.6))

    End Subroutine Stat_Output

End Module Stats

```

## Utils0

! Last change: LSR 16 May 2002 3:34 pm

Module Utils0

```
Use Date_Module
Use Global_Variables
Use IoSubs
Use Strings
Implicit None
```

Contains

Subroutine Skip\_Until(Xstring, FileName, Jin)

```
! Skip lines until we find a line starting with Xstring;
! All errors are fatal.
```

```
Implicit None
Character(Len=*), Intent(In) :: FileName
Character(Len=*), Intent(In) :: Xstring
Integer, Intent(In) :: Jin
```

```
Character(Len=132) :: tbuf
Integer :: ios, Ilen
Logical :: Found_header_line
```

```
Errors_Detected = .False.
Found_header_line = .False.
Ilen = Len(Xstring)
```

```
GetHeader: Do
  Read (jin, '(a)', Iostat = ios) tbuf
  If (ios /= 0) Exit GetHeader
```

```
  If (tbuf(1:Ilen) == Xstring) Then
    ! Found the header line.
    Found_header_line = .True.
    Exit
```

```
  End If
End Do GetHeader
```

```
If (.Not. Found_header_line) Then
  Errors_Detected = .True.
```

```
  Write (6, 9130) Xstring, Trim(FileName)
  Write (ULog, 9130) Xstring, Trim(FileName)
```

```
9130 Format (1x, '?? Skip_Until: Could not find "', a, '" in ', a)
  Stop '?? Skip_Until: Could not find header line'
!Return
```

```

End If
End Subroutine Skip_Until

```

```

Subroutine RoundOff(Xval, Nval, Wformat, &
    Target_total, Xeps, Yval, Delta_Sum)

```

```

! <A NAME="RoundOff"> <A HREF="Utils0.f90#RoundOff">
!
! Statement of the problem: The problem when we distribute
! hourly precipitation among hours. The total amount to be
! distributed is given (Target_total, 2.3622). When the
! partial amounts are printed (.e.g, '(f10.2)'), the total
! sum of the printed values (xtotal, 2.35) is different.
! This subroutine attempts to
!
!
!      Xval      tval
! i  OSC  HP(full)  HP(f0.2)
! -  ---  -
! 1  8    0.402077    0.40
! 2  7    0.351817    0.35
! 3  7    0.351817    0.35
! 4  8    0.402077    0.40
! 5  8    0.402077    0.40
! 6  9    0.452336    0.45
! ---  -
!      47    2.3622      2.35      Delta_Sum = 0.0122
!      Target_total  xtotal

```

```

Implicit None
Real, Dimension(:), Intent(In) :: Xval
Integer, Intent(In) :: Nval
Character(Len=*), Intent(In) :: Wformat
Real, Intent(In) :: Target_total
Real, Intent(In) :: Xeps
Real, Dimension(:), Pointer :: Yval
Real, Intent(Out) :: Delta_Sum

```

```

Character(Len=50), Dimension(Nval) :: tbuf
Integer :: i, n
Real :: xtotal

```

```

If (Associated(Yval)) Deallocate(Yval)
Allocate(Yval(Nval))

```

```

xtotal = Zero
n = Len_trim(Wformat)

```

```

Do i = 1, Nval

```

```
Write(tbuf(i), Wformat(1:n)) Xval(i)
Read(tbuf(i), *) Yval(i)
xtotal = xtotal + Yval(i)
End Do
```

```
Delta_Sum = Target_total - xtotal
If (Abs(Delta_Sum) > Xeps) Then
  Yval(1) = Yval(1) + Delta_Sum
End If
```

```
End Subroutine RoundOff
```

```
End Module Utils0
```

## Utils1

! Last change: LSR 6 Jun 2002 4:10 pm

Module Utils1

```
Use Date_Module
Use FileStuff
Use Global_Variables
Use IoSubs
Use Strings
Use Winteracter
Use Utils5
Implicit None
```

```
Real, Private, Parameter :: es_c1 = 0.6108
```

```
Real, Private, Parameter :: es_c2 = 17.27
```

```
Real, Private, Parameter :: es_c3 = 237.3
```

```
Logical, Private, Save :: HsJD0_unset = .True.
```

```
Integer, Private, Dimension(MinYear:MaxYear), Save :: HsJD0 = 0
```

Contains

Recursive Subroutine Display\_Station\_Info(Xstations, Xfull)

```
Implicit None
Type(Site_Info), Pointer :: Xstations ! Pointer to root of tree
Logical, Optional, Intent(In) :: Xfull

! Check for empty tree/subtree.
If (Associated(Xstations)) Then ! Tree/subtree is not empty.
  ! Visit smaller values first.
  Call Display_Station_Info(Xstations%pLeft, Xfull)

  ! Process and write the current node.
  Call Display_This_Station(Xstations, Xfull)

  ! Then visit larger values.
  Call Display_Station_Info(Xstations%pRight, Xfull)
End If
End Subroutine Display_Station_Info
```

Subroutine Display\_This\_Station(xWBAN, Xfull)

```
Implicit None
Type(Site_Info), Intent(In) :: xWBAN
Logical, Optional, Intent(In) :: Xfull
```

```

Integer :: i, yyyy, mm, dd
Logical  :: all_info

If (Present(Xfull)) Then
    all_info = Xfull
Else
    all_info = .False.
End If

If (all_info) Then
9130   Write (ULog, 9130)
        Format (//)

        Write (ULog, 9150) Trim(xWBAN%WBAN)
9150   Format (1x, '@@@ Station Name: ', a)

        Write (ULog, 9170) Trim(xWBAN%Text)
9170   Format (1x, 'Station Text: "', a, '"')

        Write (ULog, *) 'Latitude : ', xWBAN%Lat
        Write (ULog, *) 'Longitude: ', xWBAN%Lon

        Write (ULog, 9190) xWBAN%Elev
9190   Format (1x, 'Elev: ', lpg14.6)

        Write (ULog, 9210) Trim(xWBAN%TZ)
9210   Format (1x, 'TZ: ', a)

        Write (ULog, 9230) xWBAN%Nelev
9230   Format (1x, 'Nelev: ', i0)

        Do i = 1, xWBAN%Nelev
            Call Jd_to_ymd(xWBAN%Elev_Directives(i)%Julian_Day, yyyy, mm, dd)
            Write (ULog, 9250) i, yyyy, mm, dd, xWBAN%Elev_Directives(i)%Elevation_meter
9250   Format (1x, '          [' , i0, ' ] ', i4, '-', i2.2, '-', i2.2, ': ', lpg14.6)
        End Do
    End If

    If (xWBAN%Nelev == 0) Then
        Errors_Detected = .True.
        Write (ULog, 9270) Trim(xWBAN%WBAN), Trim(xWBAN%Text)
9270   Format ('?? Station without elevation data: ', 3x, a, 3x, a)
    End If

End Subroutine Display_This_Station

Subroutine Allocate_SAMSON_arrays(Nelements)

    ! Allocate SAMSON arrays.

```

```

! Propagate modifications to Allocate_SAMSON_arrays and Deallocate_SAMSON_arrays
! * <A NAME="Allocate_SAMSON_arrays">
! * <A HREF="Utils1.f90#Allocate_SAMSON_arrays">

! 0.0/0.0 generates a NaN. I am using the NaN so that the program will
! generate a message if the entry is used without being set first.
! Good idea, but what about missing entries in the SAMSON file?
! Missing entries are not read ==> value is NaN.

Implicit None

! Nelements: used for testing Senegal. We need an array of an
! specific size
Integer, Optional, Intent(In) :: Nelements

Integer :: jpar, M_elems

If (Present(Nelements)) Then
    M_elems = Nelements
Else
    M_elems = (jdl-jd0+1) * Nhours
End If

! Parameter zero is special:
Xparam(0)%Samson_v10 => Null() ! Unused for param zero
Xparam(0)%Samson_v11 => Null() ! Unused for param zero

! Allocate and initialize arrays for each parameter.
Do jpar = 1, f_end

    ! Samson_v10
    Allocate(Xparam(jpar)%Samson_v10(M_elems))
    Xparam(jpar)%Samson_v10%s = T_Missing
    Xparam(jpar)%Samson_v10%v = Missing_Data
    !Xparam(jpar)%Samson_v10%v = Zero/Zero ! NaN
    Xparam(jpar)%Samson_v10%f = ''

    ! Samson_v11
    If ((f_EHR <= jpar) .And. (jpar <= f_DHR)) Then
        Allocate(Xparam(jpar)%Samson_v11(M_elems))
        Xparam(jpar)%Samson_v11%s = T_Missing
        Xparam(jpar)%Samson_v11%v = Missing_Data
        !Xparam(jpar)%Samson_v11%v = Zero/Zero ! NaN
        Xparam(jpar)%Samson_v11%f = ''
    End If
End Do

Allocate(Obs_Ppt(M_elems))
Obs_Ppt = Val_and_Flag(T_Unset, Missing_Data, '')
!Obs_Ppt = Val_and_Flag(T_Unset, Zero/Zero, '')

```

```

End Subroutine Allocate_SAMSON_arrays

Subroutine Deallocate_SAMSON_arrays()

! Deallocate SAMSON arrays.
! Propagate modifications to Allocate_SAMSON_arrays and Deallocate_SAMSON_arrays

Implicit None
Integer :: jpar

! Parameter zero is special:
Xparam(0)%Samson_v10 => Null() ! Unused for param zero
Xparam(0)%Samson_v11 => Null() ! Unused for param zero

! Deallocate arrays for each parameter.
Do jpar = 1, f_end

    Deallocate(Xparam(jpar)%Samson_v10)

    If ((f_EHR <= jpar) .And. (jpar <= f_DHR)) Then
        Deallocate(Xparam(jpar)%Samson_v11)
    End If

End Do

Deallocate(Obs_Ppt)
End Subroutine Deallocate_SAMSON_arrays

Subroutine Decompress_and_Open_File(Jin, Zfile)

! Decompress Zfile. Attach the decompressed file to unit "Jin".

Implicit None
Integer,          Intent(Out) :: Jin
Character(Len=*), Intent(In)  :: Zfile ! e.g., d:\AllFiles\13873_91.z

Character(MaxNamLen)  :: tname, ypath, yname, ytype
Character(4*MaxNamLen) :: xcmd

! Decompress data file and open it. See Assumption[5].

Call FileNameParts(Zfile, ypath, yname, ytype)
Call Temp_File_Name(tname, NamePrologue=yname)

Write (xcmd, 9130) Trim(zcat), Trim(zfile), Trim(tname)
9130 Format (a, 1x, a, ' > ', a)
Call System(Trim(xcmd))

```

```

Call IORead(Jin, tname)
End Subroutine Decompress_and_Open_File

```

```

Subroutine Compare_SAMSON_Headers(WF10, WF11)

```

```

! Compare SAMSON v1.0 and v1.1 headers.

```

```

Implicit None
Type(FileInfo), Intent(In) :: WF10, WF11
Logical :: ok

```

```

If (WF10%Head%WBAN /= WF11%Head%WBAN) Then
Write (ULog, '(1x,4a)') '?? WBANs do not match: ', &
Trim(WF10%Head%WBAN), '; ', Trim(WF11%Head%WBAN)
End If

```

```

If (.Not. String_Eq(WF10%Head%Text, WF11%Head%Text)) Then
Write (ULog, '(1x,4a)') '?? WBANs do not match: ', &
Trim(WF10%Head%Text), '; ', Trim(WF11%Head%Text)
End If

```

```

ok = (String_Eq(WF10%Head%Lat%Letter, WF11%Head%Lat%Letter)) .And. &
(WF10%Head%Lat%degrees == WF11%Head%Lat%degrees) .And. &
(WF10%Head%Lat%minutes == WF11%Head%Lat%degrees)

```

```

If (.Not. ok) Then
Write (ULog, '(a,2(2x,a2,i0,1x,i0,:",")') '?? Latitudes do not match: ', &
WF10%Head%Lat%Letter, WF10%Head%Lat%degrees, WF10%Head%Lat%minutes, &
WF11%Head%Lat%Letter, WF11%Head%Lat%degrees, WF11%Head%Lat%minutes
End If

```

```

ok = (String_Eq(WF10%Head%Lon%Letter, WF11%Head%Lon%Letter)) .And. &
(WF10%Head%Lon%degrees == WF11%Head%Lon%degrees) .And. &
(WF10%Head%Lon%minutes == WF11%Head%Lon%degrees)

```

```

If (.Not. ok) Then
Write (ULog, '(a,2(2x,a2,i0,1x,i0,:",")') '?? Longitudes do not match: ', &
WF10%Head%Lon%Letter, WF10%Head%Lon%degrees, WF10%Head%Lon%minutes, &
WF11%Head%Lon%Letter, WF11%Head%Lon%degrees, WF11%Head%Lon%minutes
End If

```

```

If (Abs(WF10%Head%Elev - WF11%Head%Elev) > Eps0) Then
! WF10%Head%Elev /= WF11%Head%Elev
Write (ULog, '(a,1pg14.6,3x,1pg14.6)') '?? Elevations do not match: ', &
WF10%Head%Elev , WF10%Head%Elev
End If

```

```

If (WF10%Head%TZ /= WF11%Head%TZ) Then
Write (ULog, '(a,i0,3x,i0)') '?? TimeZones do not match: ', &
WF10%Head%TZ , WF10%Head%TZ

```

```

End If

End Subroutine Compare_SAMSON_Headers

Subroutine GetElevation(xWBAN, Julian_Day, Xfound, Elevation)

! Returns elevation at a particular date. The default elevation is 30 feet.

Implicit None
Type(Site_Info), Target, Intent(In)  :: xWBAN
Integer,          Intent(In)         :: Julian_Day
Logical,          Intent(Out)        :: Xfound
Real,             Intent(Out)        :: Elevation

Integer          :: ielev
Integer, Pointer :: Nelev
Type(ElevationBlock), Dimension(:), Pointer :: Elev_Directives

Nelev => xWBAN%Nelev
Elev_Directives => xWBAN%Elev_Directives

Xfound = .False.

! Nominal elevation: 30 feet from 1800-01-01 to present
! 30 feet = 9.1440 meter
Elevation = 30.0 * feet__to__meter

If (Nelev == 0) Return

! The Julian Day limits in Elev_Directives are such that
!
! If    Jd(i) <= Julian_Day < Jd(i+1),  i = 1 .. Nelev
! Then  Elevation on Julian_Day is Elev_Directives(i)%Elevation_meter
!
! Note
! * Jd(i) = Elev_Directives(i)%Julian_Day
! * Jd(Nelev+1) is defined to be +Infinity, i.e., present day
!
! Since the entries in Elev_Directives are arranged in chronological
! order, the above "If" may be simplified:
!
! If    Jd(i) <= Julian_Day,  i = Nelev .. 1
! Then  Elevation on Julian_Day is Elev_Directives(i)%Elevation_meter

Do ielev = Nelev, 1, -1
  If (Elev_Directives(ieleiv)%Julian_Day <= Julian_Day) Then
    Xfound = .True.
    Elevation = Elev_Directives(ieleiv)%Elevation_meter
  Return

```

```

        End If
    End Do

End Subroutine GetElevation

Subroutine Set_Hours_since_Jd0()

    ! Make sure this routine is called before any of
    ! Hours_since_Jd0, iv_to_ymdh, ymdh_to_iv are called.
    !
    ! See <A HREF="setup.f90#Test_iv_to_ymdh">

    Implicit None
    Integer :: y4

    If (HsJD0_unset) Then
        HsJD0_unset = .False.
        Do y4 = MinYear, MaxYear
            HsJD0(y4) = (Jd(y4,01,01) - Jd0) * Nhours
        End Do
    End If

End Subroutine Set_Hours_since_Jd0

Function Hours_since_Jd0(yyyy)

    ! Hours_since_Jd0: number of hours between Jd0 (e.g., 1961-01-01)
    !                   and the beginning of the year yyyy.
    !
    ! Examples: Assume Jd0 = Julian_Day 1961-01-01
    !  yyyy  Hours_since_Jd0
    !  1961   0
    !  1962  365 * Nhours
    !  1963  365*2 * Nhours
    !  1964  365*3 * Nhours
    !  1965  (365*3 + 366) * Nhours
    !  :      :
    !  1989 10227 * Nhours
    !  1990 10592 * Nhours
    !
    ! See <A HREF="setup.f90#Test_iv_to_ymdh">

    Implicit None
    Integer, Intent(In) :: yyyy
    Integer               :: Hours_since_Jd0

    !Hours_since_Jd0 = (Jd(yyyy,01,01) - Jd0) * Nhours
    Hours_since_Jd0 = HsJD0(yyyy)

```

```
End Function Hours_since_Jd0
```

```
Function str_Jd_to_ymd(jday)
```

```
!   Jd           Date
! -----
! 2451545       2000 Jan  1
! 2436116       1957 Oct  4
```

```
Implicit None
```

```
Integer, Intent(In) :: jday
```

```
Character(Len=18) :: str_Jd_to_ymd
```

```
Character(Len(str_Jd_to_ymd)) :: xstr
```

```
Integer :: jyyyy, jmm, jdd
```

```
!
!                               1
!                               123456789012345678
!                               2451545 2000-01-01
str_Jd_to_ymd = 'iiiiiii yyyy-mm-dd'
Call Jd_to_ymd(jday, jyyyy, jmm, jdd)
```

```
xstr = itoa(jday)
str_Jd_to_ymd(01:07) = Adjustr(xstr(1:7))
```

```
xstr = itoa(jyyyy)
str_Jd_to_ymd(09:12) = Adjustr(xstr(1:4))
```

```
xstr = itoa(jmm)
str_Jd_to_ymd(14:15) = Adjustr(xstr(1:2))
If (str_Jd_to_ymd(14:14) == '') Then
    str_Jd_to_ymd(14:14) = '0'
End If
```

```
xstr = itoa(jdd)
str_Jd_to_ymd(17:18) = Adjustr(xstr(1:2))
If (str_Jd_to_ymd(17:17) == '') Then
    str_Jd_to_ymd(17:17) = '0'
End If
```

```
End Function str_Jd_to_ymd
```

```
Function str_iv_to_ymdh(iv)
```

```
! Returns: '223676 1985-07-01 1h'
!           123456789-123456789-1
! i.e., iv and its date.
```

```

Implicit None
Integer, Intent(In) :: iv
Character(Len=21) :: str_iv_to_ymdh

Character(Len(len(str_iv_to_ymdh))) :: xstr
Integer :: jyyyy, jmm, jdd, jhh

!           1           2
!           123456789012345678901
!           223676 1985-07-01 1h
str_iv_to_ymdh = 'iiiiii yyyy-mm-dd HHh'
Call iv_to_ymdh(iv, jyyyy, jmm, jdd, jhh)

xstr = itoa(iv)
str_iv_to_ymdh(01:06) = Adjustr(xstr(1:6))

xstr = itoa(jyyyy)
str_iv_to_ymdh(08:11) = Adjustr(xstr(1:4))

xstr = itoa(jmm)
str_iv_to_ymdh(13:14) = Adjustr(xstr(1:2))
If (str_iv_to_ymdh(13:13) == '') Then
    str_iv_to_ymdh(13:13) = '0'
End If

xstr = itoa(jdd)
str_iv_to_ymdh(16:17) = Adjustr(xstr(1:2))
If (str_iv_to_ymdh(16:16) == '') Then
    str_iv_to_ymdh(16:16) = '0'
End If

xstr = itoa(jhh)
str_iv_to_ymdh(19:20) = Adjustr(xstr(1:2))

!           Write (str_iv_to_ymdh, 9150) iv, jyyyy, jmm, jdd, jhh
!9150 Format(i6, 1x, i4, '-', i2.2, '-', i2.2, i3, 'h')

End Function str_iv_to_ymdh

Subroutine iv_to_ymdh(iv, yyyy, mm, dd, hh, doy)

! Given iv, this subroutine computes the corresponding yyyy-mm-dd hh
! and, optionally, doy (day of the year).
!
! Store data in a vector array so that we can find gaps easier.
! Assume a virtual array: vData(ndays, Nhours)
!     Ndays = 365 or 366 == WFlx%Expected_Ndays
!

```

```

!   Ndays| 1     2 3 4 ...     24     25   Nhours = 25
!   -----+-----
!   1 | 1     2 3 4 ...     24     25
!   2 | 26    27 28 29 ...     49     50
!   : | :     : : : ...     :     :
!   365| 9101 : : : ...     9124   9125
!   366| 9126 . . . ...     9149   9150
!
! Samson_vlx(iv) <=> vData(doy, hh)
!   iv = (doy-1)*Nhours + hh
!
!   doy = (iv+Nhours-1) / Nhours
!   hh  = iv - (doy-1)*Nhours
!
! nbase = Hours_since_Jd0(yyyy)
! iv = (doy-1)*Nhours + hh + nbase
! Xparam(f_EHR)%Samson_v10(iv)%v = EHR

! See <A HREF="setup.f90#Test_iv_to_ymdh">
!   iv  Date           nperiods  ymax
!   -----
!       0 1960-12-31 25h     0       1961
!       1 1961-01-01 1h      1       1962
! 273924 1990-12-31 24h     30       1991
! 999999 2070-07-07 24h    110       2071
! 9999999 3056-02-29 24h  1096       3057

Implicit None
Integer,          Intent(In)  :: iv
Integer,          Intent(Out) :: yyyy, mm, dd, hh
Integer, Optional, Intent(Out) :: doy

Integer :: jv, jdoy, nbase

! Get the year
jv = iv
Do yyyy = MaxYear, MinYear, -1
  !nbase = Hours_since_Jd0(yyyy)
  nbase = HsJD0(yyyy)
  If (jv > nbase) Exit
End Do
If (yyyy < MinYear) Then
  Stop '?? Internal error in iv_to_ymdh: yyyy < MinYear'
End If

jv = jv - nbase
jdoy = (jv+Nhours-1) / Nhours
If (Present(doy)) doy = jdoy

```

```

    hh = jv - (jdoy-1)*Nhours
    Call Calend(yyyy, jdoy, mm, dd)
End Subroutine iv_to_ymdh

```

```

Subroutine ymdh_to_iv(yyyy, mm, dd, hh, iv, doy)

```

```

! Given yyyy-mm-dd hh, this subroutine computes the corresponding iv
! and, optionally, doy (day of the year).
!

```

```

! Store data in a vector array so that we can find gaps easier.
! Assume a virtual array: vData(ndays, Nhours)
! Ndays = 365 or 366 == WFlx%Expected_Ndays
!

```

```

! Ndays| 1    2  3  4  ...    24    25    Nhours = 25
! -----+-----
! 1 | 1    2  3  4  ...    24    25
! 2 | 26   27 28 29 ...    49    50
! : | :    :  :  :  ...    :    :
! 365| 9101 :  :  :  ...    9124  9125
! 366| 9126 .  .  .  ...    9149  9150
!

```

```

! Samson_vlx(iv) <=> vData(doy,hh)
!   iv = (doy-1)*Nhours + hh
!
!   doy = (iv+Nhours-1) / Nhours
!   hh  = iv - (doy-1)*Nhours
!
! nbase = Hours_since_Jd0(yyyy)
! iv = (doy-1)*Nhours + hh + nbase
! Xparam(f_EHR)%Samson_vl0(iv)%v = EHR

```

```

Implicit None
Integer,          Intent(In)  :: yyyy, mm, dd, hh
Integer,          Intent(Out) :: iv
Integer, Optional, Intent(Out) :: doy

```

```

Integer :: jdoy, nbase

```

```

jdoy = iDoY(yyyy, mm, dd)
If (Present(doy)) doy = jdoy

```

```

!nbase = Hours_since_Jd0(yyyy)
nbase = HsJD0(yyyy)
iv = (jdoy-1)*Nhours + hh + nbase

```

```

End Subroutine ymdh_to_iv

```

```

Function Multiples_of(iNum, Kmin, Kmax) Result(Nmultiples)

```

```

! Compute the number of multiples of iNum that appear
! in the interval [Kmin, Kmax], 0 < Kmin <= Kmax.
!
! Examples:
! iNum = 25
!   Kmin      Kmax   Nmultiples
!   25        50     2
!   23        24     0
!   1         24     0
!   24        25     1
!   25        25     1
!   24        26     1
!   48        48     0
!   48        52     1
!   2         55     2
!
! iNum = 3
!   Kmin      Kmax   Nmultiples
!   1         2     0
!   2         4     1
!   2         8     2
!   27        29     1
!   26        33     3
!   26        34     3
!   33        33     1
!   33        39     3

Implicit None
Integer, Intent(In) :: iNum, Kmin, Kmax
Integer               :: Nmultiples

Integer :: i, j
Logical :: ok
Real    :: rnum

Nmultiples = 0

! Enforce the contract.
ok = ((0 < Kmin) .And. (Kmin <= Kmax))
If (.Not. ok) Return

! If iNum == 0 we should not be here in the first place.
If (iNum == 0) Return

! If Kmax < iNum then there are no multiples.
If (Kmax < iNum) Return

rnum = Real(iNum)
i = Ceiling(Real(Kmin)/rnum)
j = i * iNum

```

```

! The numbers in [Kmin,j) are not multiples of iNum.
! We have multiples of iNum every iNum numbers.
Nmultiples = Ceiling(Real(Kmax-j+1)/rnum)

End Function Multiples_of

Function Pythag(a, b) Result(Rval)

! Computes Sqrt(a**2 + b**2) without destructive overflow or underflow.
!
! History:
! = [lsr] Tue Dec 18 16:37:05 2001
! . processed by "to_f90" on Tue Dec 18 16:37:05 2001
! . Adapted from SLATEC/Pythag; ported to f95
!
!***LIBRARY    SLATEC
!***TYPE       SINGLE PRECISION (Pythag-S)
!***AUTHOR     (UNKNOWN)
!***REVISION HISTORY (YMMDD)
!   811101 DATE WRITTEN
!   890531 Changed all specific intrinsics to generic. (WRB)
!   891214 Prologue converted to Version 4.0 format. (BAB)
!   900402 Added TYPE section. (WRB)

Implicit None
Real, Intent(In) :: a
Real, Intent(In) :: b
Real              :: Rval

Real :: p , q , r , s , t
Real, Parameter :: Two = 2.0e0
Real, Parameter :: Four = 4.0e0
Real, Parameter :: eps = Tiny(Two)

p = Max(Abs(a), Abs(b))
q = Min(Abs(a), Abs(b))

! Reference:
! [] Nicholas J. Higham. 1996. Accuracy and Stability of
!   Numerical Algorithms. SIAM. See pages 511-512.
!
! The algorithm converges in at most three iterations,
! assuming unit roundoff >= 1e-20.

If (q <= eps) Then ! (q == Zero) ?
  Rval = p
  Return
End If

```

```

Do
  r = (q/p) ** 2
  t = Four + r
  If (Abs(t-Four) <= eps) Then ! (t == Four) ?
    Rval = p
    Exit
  End If

  s = r / t
  p = p + Two*p*s
  q = q * s
End Do

End Function Pythag

Function TimeZone_to_Central_Meridian(TimeZone) Result(Central_Meridian)

  ! Longitude of the center of the local time zone [degrees west of Greenwich]

  Implicit None
  Integer, Intent(In) :: TimeZone
  Integer              :: Central_Meridian

  Select Case(TimeZone)
  Case(4)      ! Letter Q, Atlantic
    Central_Meridian = 60 ! 60W

  Case(5)      ! Letter R, Eastern
    Central_Meridian = 75

  Case(6)      ! Letter S, Central
    Central_Meridian = 90

  Case(7)      ! Letter T, Mountain
    Central_Meridian = 105

  Case(8)      ! Letter U, Pacific
    Central_Meridian = 120

  Case(9)      ! Letter V, Alaska
    Central_Meridian = 135

  Case(10)     ! Letter W, Hawaii-Aleutian
    Central_Meridian = 150

  Case(11)     ! Letter X, Samoa
    Central_Meridian = 165

  Case(-10)   ! Letter K, Chamorro (proposed)

```

```

        Central_Meridian = -150      ! -150W is equivalent to 150E

Case Default
    Central_Meridian = TimeZone * 15
End Select

End Function TimeZone_to_Central_Meridian

Subroutine Es_and_Delta(Tc, e_s, Delta)

    ! Tc      Temperature [°C]
    ! e_s     saturation water vapor pressure [kPa] at temperature Tc; Ref[1:36]
    ! Delta   slope of saturation water vapor pressure [kPa/°C]
    Implicit None
    Real, Intent(In)  :: Tc
    Real, Intent(Out) :: e_s
    Real, Intent(Out) :: Delta

    Real :: tmp0

    ! Propagate changes in the parameters
    ! to Es_and_Delta and DewPointF
    ! [FAO, Page 36, eq. 11]
    !
    !      e0(T) = es_c1 * Exp(es_c2 * T / (es_c3 + T))
    tmp0 = Exp(es_c2 * Tc / (es_c3+Tc))
    e_s = es_c1 * tmp0
    Delta = es_c1 * es_c2 * es_c3 * tmp0 / (es_c3+Tc)**2

End Subroutine Es_and_Delta

Function GammaF(xP, Tdew) Result(GammaV)

    ! <A NAME="GammaF"> <A HREF="Utils1.f90#GammaF">
    !
    ! xP      Pressure [kPa]
    ! Tdew    Dew point temperature [°C]
    ! GammaV  psychrometric coefficient [kPa/°C]
    !
    !
    !          Cp * P
    !      = -----
    !          eps * lambda

    Implicit None
    Real, Intent(In) :: xP, Tdew
    Real              :: GammaV

```

```

Real, Parameter :: Cp = 1.013      ! KJ Kg^-1 K^-1
Real, Parameter :: Eps = 0.622    ! dimensionless
Real :: LambdaF

!GammaV = Cp * xP / (Eps * LambdaF(Tdew))

LambdaF = 2501 - 2.361*Tdew
GammaV = Cp * xP / (Eps * LambdaF)

End Function GammaF

!Function LambdaF(Tdew) Result(LambdaV)
!
!   ! <A NAME="LambdaF"> <A HREF="Utils1.f90#LambdaF">
!   !
!   ! Tdew      Dew point temperature [°C]
!   ! LambdaV   Latent heat (enthalpy) of vaporazation [KiloJoules/Kg]
!   Implicit None
!   Real, Intent(In) :: Tdew
!   Real           :: LambdaV
!
!   LambdaV = 2501 - 2.361*Tdew
!
!End Function LambdaF

Function Wind_Speed_F(u10, T_Height) Result(ux)

! u10      Wind Speed in [m/s] at z=10 meters
! T_Height Height indicator
! ux       Wind Speed in [m/s] at z indicated by T_Height
Implicit None
Real, Intent(In) :: u10
Integer, Intent(In) :: T_Height
Real           :: ux

Real :: tu

! <A NAME="Wind Conversion Table">
! On input, Wind Speed in meters/sec was normalized to z=10 meters.
!
!   u_10   u_x
!   ---- = --- ==> u_x = (u10/h_10) * h_x
!   h_10   h_x
!
!
! Height   Height indicator   h_x

```

```

! -----
! 10 m    T_u10      5.81 == Ln((10-0)/0.03)
!  2 m    T_u2      4.87
!  4 m    T_u4      4.89
! 0.6 m   T_up6     3.56
! 0.1 m   T_up1     1.05
!
!
!           u_c           u_1
! ----- = -----
! Ln ((z_c-d_c)/z_0c)   Ln ((z_1-d_1)/z_01)
!
! u_c : wind speed at height z_c (m)
! d_c : zero plane displacement (m)
! z_0c: surface roughness length or roughness height (m)
!
! For Open Flat Terrain (used for Metereological Stations):
!   z0 = 0.03 meters
!   d0 = 0
!   z  = 10 meters (reference height)
!
! h_10 = Ln((10-0)/0.03) = 5.81
! h_x  = Ln((x-0)/0.03) = Ln(x/0.03), x measured in meters
!
! u10 = Wind_Speed * 5.81 / Log(Station_elevation_in_meters/0.03)

tu = u10 / 5.81
Select Case(T_Height)
Case(T_u10)
  ! Height = 10 m
  ux = u10

Case(T_u2)
  ! Height = 2 m
  ux = tu * 4.87

Case(T_u4)
  ! Height = 4 m
  ux = tu * 4.89

Case(T_up6)
  ! Height = 0.6 m
  ux = tu * 3.56

Case(T_up1)
  ! Height = 0.1 m
  ux = tu * 1.05

Case Default
  Write (6, *) '?? Wind_Speed_F: unknown wind indicator == ', T_Height

```

```

        Write (ULog, *) '?? Wind_Speed_F: unknown wind indicator == ', T_Height
        Stop '?? Wind_Speed_F: unknown wind indicator'
    End Select

End Function Wind_Speed_F

Function DewPointF(Ta, RH) Result(Tdew)

    ! <A NAME="DewPoint"> <A HREF="Utils1.f90#DewPoint">
    ! See <A HREF="0notes.txt#Note_26">
    !
    ! Estimate dewpoint from inversion of eq. 14, FAO, Page 37.
    !
    ! Ta      Air temperature [°C]
    ! RH      Relative humidity [%]
    ! Tdew    Dew point temperature [°C]
    Implicit None
    Real, Intent(In) :: Ta
    Real, Intent(In) :: RH
    Real              :: Tdew

    ! e_s    saturation water vapor pressure [kPa] at temperature Tc; Ref[1:36]
    ! e_a    Atmospheric water vapor pressure; actual water vapor pressure, [kPa]
    ! Delta  slope of saturation water vapor pressure [kPa/°C]
    Real :: e_s, Delta, e_a

    ! e_s = e0(Ta)
    Call Es_and_Delta(Ta, e_s, Delta)    ! Delta unused.

    ! RH = 100 * e_a / e_s
    e_a = 0.01 * e_s * RH

    ! Propagate changes in the parameters
    ! to Es_and_Delta and DewPointF
    ! [FAO, Page 36, eq. 11]
    !
    !      e0(T) = es_c1 * Exp(es_c2 * T / (es_c3 + T))
    !
    ! e0(Tdew) = e_a          ! FAO, Page 37, eq. 14
    ! Tdew = e0^-1(e_a)
    !      = c3 * Ln(e_a/c1) / (c2 - Ln((e_a/c1)))

    Tdew = es_c3 * Log(e_a/es_c1) / (es_c2 - Log(e_a/es_c1))

End Function DewPointF

```

```

Subroutine Generate_File_names()

! Generate the names of all files associated with the current wban number.
! <A NAME="Generate_File_names">
! <A HREF="Utils1.f90#Generate_File_names">
Implicit None

Character(MaxNamLen) :: tdir, tname
Integer :: yyyy, n
Character(Len(pWBAN%WBAN)+1) :: W_Wban
Character(Len=5):: W_State

! Output directory of the form: v:\r0\<STATE>\14914\
!     where <STATE> is the two letter abbreviation of
!     state where the station is located, .e.g,
!     v:\r0\AK\25501\
W_State = pWBAN%State
R0_full = Trim(R0_root) // Trim(W_State) // DirDelim // Trim(pWBAN%WBAN)
Call STD_dir(R0_full)
tdir = R0_full

W_Wban = 'w' // Trim(pWBAN%WBAN)
n = Len_trim(W_Wban)

! Names of r0 files
Do yyyy = MinYear, MaxYear

! r0_file -- Hourly Values File
!           == v:\r0\AK\25501\w255011.h##
!           == v:\r0\AK\25501\w255011.h67

Write (name_r0(yyyy), 9130) &
Trim(tdir), W_Wban(1:n), Modulo(yyyy,100)
9130 Format (2a, ".h", i2.2)
End Do

! name_met == v:\r0\ND\14914\w14914.dvf
! name_txt == v:\r0\ND\14914\w14914.txt
tname = Trim(tdir) // W_Wban(1:n)
name_met = Trim(tname) // '.dvf'
name_txt = Trim(tname) // '.txt'

! Generate filenames:
! -----
! Daily Precipitation files:
!     v:\precip\03940.d
!     v:\precip\03940g.d           ! with gaps in the yearly record.
! Hourly Precipitation files:
!     v:\precip\03940.h
!     v:\precip\03940g.h

```

```

tname = Trim(Raw_Data_dir) // 'precip' // DirDelim // Trim(pWBAN%WBAN)
name_Daily_ppt = Trim(tname) // '.d'
name_Hourly_ppt = Trim(tname) // '.h'

! Daily evaporation file:
! v:\evaporation\T_14914.evp
name_Daily_Evap = Trim(Raw_Data_dir) // 'evaporation' // &
DirDelim // 'T_' // Trim(pWBAN%WBAN) // '.evp'

End Subroutine Generate_File_names

Subroutine Initialize_Output_Directory()

! #1. Creates output directory (if needed).
! #2. Deletes old output files in that directory.
Implicit None

Character(Len(pWBAN%WBAN)) :: Wban
Character(MaxNamLen) :: tdir
Integer :: yyyy, k0, k1, tlen
Logical :: have_file

Wban = pWBAN%WBAN

! If the output directory does not exist, exit.
! Output directory of the form: v:\r0.by.State\AK\25501\
! IOsDirMake will not create directory path x:\a\b\c\
! if, for example, directories a or b do not exist.
! We have to check and create each component individually -- bummer.

tdir = R0_full
Call STD_dir(tdir) ! Make sure tdir has a trailing delimiter
tlen = Len_Trim(tdir)
k1 = 1

do
  If (k1 > tlen) Exit

  k0 = Index(tdir(k1:tlen), DirDelim)
  If (k0 > 0) k0 = k0 + k1 - 1
  If (k0 == 0) Exit

  If (.Not. IOsDirExists(tdir(1:k0))) Then
    Call IOsDirMake(tdir(1:k0))
    If (.Not. IOsDirExists(tdir(1:k0))) Then
      Write(ULog, *) '?? Could not create Directory ', Trim(tdir(1:k0))
      Stop '?? Stopping -- Could not create Directory.'
    !Return
  
```

```

        End If
    End If
    k1 = k0 + 1 ! Skip over previous delimiter
End Do

! Hourly Values Files
Do yyyy = MinYear, MaxYear
    Inquire(File=name_r0(yyyy), Exist=have_file)
    If (have_file) Then
        Call IOsDeleteFile(name_r0(yyyy))
    End If
End Do

! Later vintage: Winteracter does not complain if
!     the file does not exist.
Call IOsDeleteFile(name_met)
Call IOsDeleteFile(name_txt)

End Subroutine Initialize_Output_Directory

```

```

Subroutine Beaufort_Wind_Scale(Wind_Speed, Vforce, Vtext)

```

```

! http://www.spc.noaa.gov/faq/tornado/beaufort.html
! gyre.umeoce.maine.edu/data/gomoos/php/variable_description.php?variable=wind_speed
! -- Warning: the m/s quantities in gyre.umeoce.maine.edu are wrong!
! http://whale.wheelock.edu/whalenet-stuff/beaufort.html
Implicit None
Real,          Intent(In)  :: Wind_Speed ! in meters/second
Integer,       Intent(Out) :: Vforce     ! Beaufort Force
Character(Len=*), Intent(Out) :: Vtext    ! explanation

Type :: BF
    Real :: Wind_Speed ! Lower limit of range, in meters/second
    Character(Len=80) :: WMO_Classification = ''
    Character(Len=80) :: SeaText = ''
    Character(Len=80) :: LandText = ''
End Type BF

Integer :: i
Type(BF), Dimension(0:12), Parameter :: VInfo = (/ &
    BF( 0.0, 'Calm', 'Sea surface smooth and mirror-like', 'Smoke rises vertically'), &
    BF( 0.3, 'Light air', 'Scaly ripples, no foam crests', &
        'Smoke drift indicates wind direction, still wind vanes'), &
    BF( 1.6, 'Light Breeze', 'Small wavelets, crests glassy, no breaking', &
        'Wind felt on face, leaves rustle, vanes begin to move'), &
    BF( 3.4, 'Gentle Breeze', 'Large wavelets, crests begin to break, scattered whitecaps', &
        'Leaves and small twigs constantly moving, light flags extended '), &

```

```

BF( 5.5, 'Moderate Breeze', 'Small waves 1-4 ft. becoming longer, numerous whitecaps', &
'Dust, leaves, and loose paper lifted, small tree branches move'), &
BF( 8.0, 'Fresh Breeze', 'Moderate waves 4-8 ft taking longer form, many whitecaps, some spray', &
'Small trees in leaf begin to sway'), &
BF(10.8, 'Strong Breeze', 'Larger waves 8-13 ft, whitecaps common, more spray', &
'Larger tree branches moving, whistling in wires'), &
BF(13.9, 'Moderate Gale', 'Sea heaps up, waves 13-20 ft, white foam streaks off breakers', &
'Whole trees moving, resistance felt walking against wind'), &
BF(17.2, 'Fresh Gale', 'Moderately high (13-20 ft) waves of greater length, edges&
& of crests begin to break into spindrift, foam blown in streaks', &
'Twigs broken off trees, walking against wind very difficult'), &
BF(20.8, 'Strong Gale', 'High waves (20 ft), sea begins to roll, dense&
& streaks of foam, spray may reduce visibility', &
'Slight structural damage occurs, slate blows off roofs'), &
BF(24.5, 'Storm', 'Very high waves (20-30 ft) with overhanging crests,&
& sea white with densely blown foam, heavy rolling, lowered visibility', &
'Seldom experienced on land, trees broken or uprooted, "considerable structural damage"'), &
BF(28.5, 'Violent Storm', 'Exceptionally high (30-45 ft) waves, foam patches&
& cover sea, visibility more reduced', &
'Widespread damage, very rare occurrence'), &
BF(32.7, 'Hurricane', 'Air filled with foam, waves over 45 ft, sea completely&
& white with driving spray, visibility greatly reduced', &
'Violent destruction') /)

Do i = Ubound(VInfo,1), Lbound(VInfo,1), -1
  If (VInfo(i)%Wind_Speed <= Wind_Speed) Then
    Vforce = i
    Vtext = VInfo(i)%LandText
    Return
  End If
End Do

Vforce = -1
Write (Vtext, 9130) Wind_Speed
9130 Format ('?? Beaufort_Wind_Scale: incorrect WindSpeed = ', lpg14.6)
End Subroutine Beaufort_Wind_Scale

```

```
Subroutine Meta_Data_File()
```

```
  Implicit None
```

```
  Integer :: j, k0, k1, uu, f0, f1
```

```
  Logical :: xok
```

```
  Character(Len=80) :: tbuf
```

```
  Call IOwrite(uu, name_txt, Ok=xok)
```

```
  If (.Not. xok) Then
```

```
    Write (ULog, *) '?? Could not open TXT file ', Trim(name_txt)
```

```

        Errors_Detected = .True.
        Return
    End If

    j = Index(pWBAN%Text, ',')
    k0 = j - 1      ! City
    k1 = j + 2      ! State
    j = Index(pWBAN%TZ, '(') - 1

    Write (uu, 9130) 'Station WBAN Number: ', Trim(pWBAN%WBAN)
    Write (uu, 9130) 'Station Name: ', pWBAN%Text(1:k0)
    Write (uu, 9130) 'Station Location (State): ', pWBAN%Text(k1:k1+1)
    Write (uu, 9130) 'Station Time Zone: ', Trim(pWBAN%TZ)
9130 Format (a, a)

    Write (uu, 9150) 'Station Latitude: ', &
        pWBAN%Lat%Letter, pWBAN%Lat%degrees, pWBAN%Lat%minutes
    Write (uu, 9150) 'Station Longitude: ', &
        pWBAN%Lon%Letter, pWBAN%Lon%degrees, pWBAN%Lon%minutes
9150 Format (a, a1, i5, ' degrees ', i2, ' minutes')

    Write (uu, 9170) 'Station elevation above sea level [meters]: ', Nint(pWBAN%Elev)
9170 Format (a, i4)

    Write (uu, 9130) 'File generation date: ', Trim(TimeStamp)

    Call Str_years(tbuf)
    Write (uu, '(a,a)') 'Years present:', Trim(tbuf)

    f0 = 1 + Index(name_met, DirDelim, Back=.True.)
    f1 = Len_trim(name_met)
    Write (uu, 9190) name_met(f0:f1)
9190 Format (&
        '!', /, &
        '! The meteorological Daily Values File "', a, &
        '" has the following format:')

    ! Copy the metadata coda.
    Call Copy_From_To(Umetadata, uu)

    Call IOClose(uu)

End Subroutine Meta_Data_File

Subroutine Copy_From_To(Uin, Uout)

```

```
! Copy contents of Uin to Uout.
Implicit None
Integer, Intent(In) :: Uin, Uout

Integer          :: iostatus
Character(Len=132) :: tbuf

Rewind (Uin)

Read_a_Line: Do
  Read (Uin, '(a)', iostat=iostatus) tbuf
  If (iostatus /= 0) Exit Read_a_Line
  Write (Uout, '(a)') Trim(tbuf)
End Do Read_a_Line

End Subroutine Copy_From_To

End Module Utils1
```

## Utils2

! Last change: LSR 16 Jul 2002 3:07 pm

Module Utils2

```
!Use Binary_Tree
!Use Date_Module
!Use FileStuff
!Use Floating_Point_Comparisons
!Use GetNumbers
!Use IoSubs
!Use Linked_List
!Use Read_Info
!Use Reallocate_Module
!Use SAMSON
!Use Strings
!Use Utils0
!Use Utils1
Use ET0
Use Fix_Data_Records
Use Global_Variables
Use Precipitation_module
Use Process_Gaps
Use Stats
Use Utils4
Implicit None
```

```
Type :: XQuadrant
  Integer :: n = 0
  Integer, Dimension(1:24) :: h = 0
  Real    :: ws_max = Zero
  Integer :: ws_d12 = 0
  Real    :: ws_mean = Zero
End Type XQuadrant
```

```
Type :: XPerSeg
  Type(XQuadrant), Dimension(:), Pointer :: Quads
  Integer, Dimension(:), Pointer :: Tied_List
  Integer :: Tied_N
  Integer :: isub, imod
  Real    :: rdiv
End Type XPerSeg
```

```
!!!      ! For each SAMSON parameter ...
!!!      Do k = 1, f_SAMSON
!!!        Select Case(k)
!!!          Case(f_EHR)      ! Extraterrestrial Horizontal Radiation
!!!          Case(f_EDNR)    ! Extraterrestrial Direct Normal Radiation
!!!          Case(f_GHR)     ! Global Horizontal Radiation
```

```

!!!          Case(f_DNR)      ! Direct Normal Radiation
!!!          Case(f_DHR)      ! Diffuse Horizontal Radiation
!!!          Case(f_TSC)      ! Total Sky Cover
!!!          Case(f_OSC)      ! Opaque Sky Cover
!!!          Case(f_DBT)      ! Dry Bulb Temperature
!!!          Case(f_DPT)      ! Dew Point Temperature
!!!          Case(f_RH)       ! Relative Humidity
!!!          Case(f_SP)       ! Station Pressure
!!!          Case(f_WD)       ! Wind Direction
!!!          Case(f_WS)       ! Wind Speed
!!!          Case(f_HV)       ! Horizontal Visibility
!!!          Case(f_CH)       ! Ceiling Height
!!!          Case(f_pH2O)     ! Precipitable Water
!!!          Case(f_baod)     ! Broadband Aerosol Optical Depth
!!!          Case(f_SD)       ! Snow Depth
!!!          Case(f_DSLS)     ! Days since last Snowfall
!!!          Case(f_HP)       ! Hourly Precipitation (value + flags)
!!!          Case(f_OI)       ! Observation_Indicator / Present Weather
!!!          Case Default
!!!          Write(6,*) 'Process_Set: Internal error: no "CASE(f_k)" for k=', k
!!!          Stop            '?? Stopping in Process_Set: Internal error: no "CASE(f_x)"'
!!!          End Select
!!!          End Do

```

Contains

Subroutine Process\_Set()

```

! <A NAME="Process_Set">

Implicit None

Integer :: yyyy, ybase, from_beg, from_end, to_beg, to_end
Logical  :: okay
Integer  :: k, k1, iv, jday, hh25, j, vdim
Integer  :: ierr
Type(Val_and_Flag), Dimension(:), Pointer :: HV, CH, HP, DSLS, vf

okay = .True.
ierr = 0

HP => Xparam(f_HP)%Samson_v10      ! Hourly Precipitation
DSLS => Xparam(f_DSLS)%Samson_v10  ! Days since last snowfall

! Transfer data SAMSON v 1.1 to SAMSON v 1.0
Xparam(f_GHR )%Samson_v10 = Xparam(f_GHR )%Samson_v11
Xparam(f_EHR )%Samson_v10 = Xparam(f_EHR )%Samson_v11
Xparam(f_EDNR)%Samson_v10 = Xparam(f_EDNR)%Samson_v11
Xparam(f_DNR )%Samson_v10 = Xparam(f_DNR )%Samson_v11

```

```

Xparam(f_DHR )%Samson_v10 = Xparam(f_DHR )%Samson_v11

! Read observed ppt data
Call Read_Hourly_Ppt(okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

Call Read_Daily_Ppt(okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

! <A NAME="Missing years"> <A HREF="Utils2.f90#Missing years">
! See also <A HREF="0notes.txt#Note_25">
! if some years are missing, fill the missing years with data
! from an existing year. We will delete the added data when
! generating the r0 files and the MET records.

Do yyyy = MinYear, MaxYear
    ! If the SAMSON version 1.0 file is missing, then we will
    ! say that everything is missing, even if SAMSON version 1.1
    ! is present.
    If (Year_Data/yyyy)%SAMSON_v10 == 0) Then
        ! For non-leap years, copy year = 1963
        ! For leap years, copy year = 1964
        ! These years are always present.
        If (IsLeapYear/yyyy) Then
            ybase = 1964
        Else
            ybase = 1963
        End If

        ! My previous comment notwithstanding, verify that the
        ! data for the stated base year exists.
        If (Year_Data/ybase)%SAMSON_v10 == 0) Then
            Write(*, *) '?? Missing base year ', yyyy, '. Stopping ...'
            Write(ULog, *) '?? Missing base year ', yyyy, '. Stopping ...'
            Stop '?? Missing base year '
        End If

        Call ymdh_to_iv(ybase, mm=01, dd=01, hh=01, iv=from_beg)
        Call ymdh_to_iv(ybase, mm=12, dd=31, hh=Nhours, iv=from_end)
        Call ymdh_to_iv/yyyy, mm=01, dd=01, hh=01, iv=to_beg)
        Call ymdh_to_iv/yyyy, mm=12, dd=31, hh=Nhours, iv=to_end)

        Write (ULog, 9130) yyyy, ybase, yyyy
        Format (/, &
            1x, '## Data for ', i4, &
            ' is missing and was filled with data from ', i4, &

```

9130

```

        ' for analysis purposes.', /, &
        lx, ' The year ', i4, &
        ' will not be present in the final suite of files.')

    ! Copy all records.
    Do k = 1, f_SAMSON
        Xparam(k)%Samson_v10(to_beg:to_end) = &
            Xparam(k)%Samson_v10(from_beg:from_end)
    End Do
    Obs_ppt(to_beg:to_end) = Obs_ppt(from_beg:from_end)
End If
End Do

Call Standardize_ppt(HP, Accum_Samson, okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

If (Have_ppt_Obs_hourly_data) Then
    Call Standardize_ppt(Obs_ppt, Accum_EI, okay)
    If (.Not. okay) Then
        Errors_Detected = .True.
    End If
End If

! Fix_SAMSON performs manual correction of the data records.
! Be careful when fixing precipitation records, in particular,
! with runs of missing, deleted, or accumulated values.
! Make sure the beginning or end of such a run is not lost
! on transfer.
Call Fix_SAMSON(okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

! Ojo! The order of operations is VERY important.
!
! #1.1 Process BAOD first. For some sites, BAOD is
!       missing for nighttime (in particular, hh = 24).
!
! #2. N/A.
!
! #3. Interpolate, fill gaps, and otherwise process the
!     hourly data as needed.
!
! #4. THEN fill the 25th hour with the daily values.
!     Compute other parameters as needed.
!     Note that if we compute the daily values first
!     and then fill gaps (invert steps #3 and #4), the
!     interpolation scheme would have to determine if

```

```

!      the interpolation boundary or the interpolation
!      range contains a "25th" hour. Do not use nor
!      fill a 25th hour during interpolation.

! Missing values for BAOD require special treatment.
Call Process_BAOD(okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

!! Use Fill-Gaps algorithms. 12 Mar 2002  2:08 pm
!! Wind Speed: Fill missing hourly values with the monthly mean
!!Call Set_Hourly_Values(f_WS, okay)
!!If (.Not. okay) Then
!!  Errors_Detected = .True.
!!End If

! Set the 25th hour to NaN. We will see these on output,
! unless set first.
! 6 May 2002  3:24 pm: Warning: according to lf95,
!      Zero/Zero == NaN
!      and Nint(Xparam(1)%Samson_v10(2*NHours)%v) == 0,
!      i.e, Nint(NaN) == 0 !?

Do k = 1, f_SAMSON
    vf => Xparam(k)%Samson_v10
    vdim = Ubound(vf,1)
    vf(NHours:vdim:NHours)%v = Zero / Zero
End Do

Call ToTTY('Process_Set: Process_Days_since_last_Snowfall')
Call FLushAll()
Call Process_Days_since_last_Snowfall(okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

! Convert "flag" values to numbers
! See <A NAME="Flag_values_to_numbers">
! See <A HREF="0notes.txt#Note_15">
! See <A HREF="0notes.txt#Note_16">
Write(ULog, *)
Write(ULog, 9150) '## Maximum_Horizontal_Visibility: ', Maximum_Horizontal_Visibility
Write(ULog, 9150) '## Maximum_Ceiling_Height .....: ', Maximum_Ceiling_Height
9150 Format (lx, a, lpg14.6)

HV => Xparam(f_HV)%Samson_v10
CH => Xparam(f_CH)%Samson_v10
k1 = Ubound(HV,1)

```

```

! Change the flag value of Unlimited visibility (777.7)
!   to 110% of the maximum value attained.
! Change the flag values of Unlimited ceiling height (77777),
!   Cirroform (88888) to 110% of the maximum attained.
FieldInfo(f_HV)%maximum_value = Maximum_Horizontal_Visibility * 1.10
FieldInfo(f_CH)%maximum_value = Maximum_Ceiling_Height * 1.10
Do iv = 1, k1
  If (HV(iv)%s == T_Unlimited) Then
    HV(iv)%v = FieldInfo(f_HV)%maximum_value
  End If
  Select Case(CH(iv)%s)
  Case(T_Unlimited)
    CH(iv)%v = FieldInfo(f_CH)%maximum_value
  Case(T_Cirroform)
    CH(iv)%v = FieldInfo(f_CH)%maximum_value
  End Select
End Do

Call ToTTY('Process_Set: Fill gaps et al.')
Call FLushAll()
! Fill gaps et al.
Do k = 1, f_SAMSON

  ! Gaps anywhere.
  Select Case(k)
  Case(f_HP)
    ! Hourly precipitation must be processed after
    ! everybody else. We will process HP manually.

  Case(f_DPT)
    ! Dew Point must be processed after everybody else.
    ! We will process HP manually.

  Case(f_OI)
    ! Observation Indicator, Present_weather
    ! It makes no sense to fill gaps in these parameters

    !Case(f_HV) ! Horizontal Visibility
    !Case(f_CH) ! Ceiling Height
    ! Also, be careful filling gaps when
    ! #1. Visibility == 777.7 = unlimited visibility.
    ! #2. Ceiling Height == 77777 = unlimited ceiling height, or
    !     == 88888 = cirroform.
    ! We have this problem for 14914 (Fargo), 1965-01-01 12h-15h

  Case Default
    Call Find_Gaps(k)
  End Select
End Do

```

```

! Process precipitation.
Call Process_Precipitation(okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

! Estimate missing Dew Point observations.
Call Fill_Dew_Point(okay)
If (.Not. okay) Then
    Errors_Detected = .True.
End If

!Call Find_Gaps(0, Release_Storage=.True.) ! Release storage.

! <A NAME="Daily values">
! Compute daily values.
! For each SAMSON parameter ...

Call ToTTY('Process_Set: Compute daily values.')
```

Do k = 1, f\_SAMSON  
okay = .True.

```

Select Case(k)
Case( & ! *** Daily value is the sum of hourly values
    f_EHR, & ! Extraterrestrial Horizontal Radiation
    f_EDNR, & ! Extraterrestrial Direct Normal Radiation
    f_GHR, & ! Global Horizontal Radiation
    f_DNR, & ! Direct Normal Radiation
    f_DHR) ! Diffuse Horizontal Radiation
    Call Daily_Values(k, T_Cumulative, okay)

Case( & ! *** Daily value is the average of hourly values
    f_TSC, & ! Total Sky Cover
    f_OSC, & ! Opaque Sky Cover
    f_DBT, & ! Dry Bulb Temperature
    f_DPT, & ! Dew Point Temperature
    f_RH, & ! Relative Humidity
    f_SP, & ! Station Pressure
    f_HV, & ! Horizontal Visibility
    f_baod, & ! Broadband Aerosol Optical Depth
    f_CH, & ! Ceiling Height
    f_pH2O, & ! Precipitable Water
    f_SD) ! Snow Depth
    Call Daily_Values(k, T_Average, okay)

Case(f_HP) ! Hourly Precipitation (value + flags)
! Do nothing. The daily value was computed by
! <A HREF="Precip.f90#Process_Precipitation">
!Call Daily_Values(k, T_Cumulative, okay)

```

```

Case(f_WD)      ! Wind Direction
! Do nothing. Wind Direction and Wind Speed must
! be done simultaneously. See f_WS below.

! = 29 Jan 2002  4:11 pm
! * since we are computing mean wind speed,
! we need to fill the 25-th hour with Zero.
Call Daily_Values(k, T_Not_Applicable, okay)

Case(f_WS)      ! Wind Speed
! *** <A NAME="To Do: describe daily value Wind Speed; Needs work">
! If a particle were transported ... **** Needs work
! *** Decompose into x- and y- components ...
! The daily value is the vector sum of the hourly components.
! -- not anymore 29 Jan 2002  4:08 pm
! Call Daily_Wind_Speed(okay) ! unused, 29 Jan 2002  4:08 pm

! = 29 Jan 2002  4:20 pm;
! * daily value WS is a simple average
Call Daily_Values(k, T_Average, okay)

Case(f_DSLS)    ! Days since last Snowfall
! Copy the 24h value to the 25h.
hh25 = 0
Do jday = jd0, jd1      ! step by day
  hh25 = hh25 + NHours ! Index of the 25th hour of jday == (jday-jd0+1)*NHours
  j = hh25 - 1        ! the 24th hour of the current day (jday)
  Select Case(DSLS(j)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
      ! Not a valid number.
      DSLS(hh25) = Val_and_Flag(T_Estimated, Zero, '')
    Case Default
      DSLS(hh25) = Val_and_Flag(T_Estimated, DSLS(j)%v, '')
  End Select
End Do
okay = .True.

Case(f_OI)      ! Observation_Indicator / Present Weather
Call Daily_Values(k, T_Not_Applicable, okay)

Case Default
  Write(6,*) 'Process_Set: Internal error: no "CASE(f_k)" for k=', k
  Stop      '?? Stopping in Process_Set: Internal error: no "CASE(f_x)"'
End Select

If (.Not. okay) Then
  Errors_Detected = .True.
End If
Call FLushAll()
End Do

```

```

! Rs = (a + b[opaque_sky_cover]) * Ra, for daylight (Rs > 0)
! Rso = a * Ra
! Regression: y = a*Ra + b*[opaque_sky_cover]*Ra
!
! Rs      (a + b[opaque_sky_cover]) * Ra
! --- = -----
! Rso      a * Ra
!
!      (a + b[opaque_sky_cover])
! = -----
!      a
!
!      b
! = 1 + - [opaque_sky_cover]
!      a

```

```

Call ToTTY('ET0_et_al')
Call ET0_et_al(okay)
Call FLushAll()

```

```

! Must be the last statement of the file.
Call ToTTY('Process_Set: Set_Missing_Flags')
Call Set_Missing_Flags()

```

```
End Subroutine Process_Set
```

```
Subroutine Process_Param(k_id)
```

```

Implicit None
Integer, Intent(In) :: k_id      ! Parameter number

Integer, Pointer :: min_obs_per_day
Type(Val_and_Flag), Dimension(:), Pointer :: xsd    ! X-Data

xsd => Xparam(k_id)%Samson_v10
min_obs_per_day => FieldInfo(k_id)%Minimum_obs_per_day

! Gaps anywhere.
Select Case(k_id)
Case(f_OI)
    ! Observation Indicator, Present_weather
    ! It makes no sense to fill gaps in these parameters

    !Case(f_HV) ! Horizontal Visibility
    !Case(f_CH) ! Ceiling Height
    ! Also, be careful filling gaps when
    ! #1. Visibility == 777.7 = unlimited visibility.
    ! #2. Ceiling Height == 77777 = unlimited ceiling height, or

```

```

!                               == 88888 = cirroform.
! We have this problem for 14914 (Fargo), 1965 1 1 12h-15h

Case Default
  Call Find_Gaps(k_id)
End Select
End Subroutine Process_Param

Subroutine Set_Hourly_Values(k_id, Xok)

! <A NAME="Set_Hourly_Values">
! Fill missing hourly values with the monthly mean

Implicit None
Integer, Intent(In)  :: k_id
Logical, Intent(Out) :: Xok

Integer :: yyyy, mm, dd, hh, iv, jv0, jv1
Integer :: ierr, npoints, minN
Real    :: xsum
Type(Val_and_Flag), Dimension(:), Pointer :: vf, WD
Integer, Dimension(:), Pointer :: Days_in_Month

WD => Xparam(f_WD)%Samson_v10    ! Wind Direction
vf => Xparam(k_id)%Samson_v10
minN = FieldInfo(k_id)%Minimum_obs_per_day

ierr = 0
Do yyyy = MinYear, MaxYear

  Days_in_Month => Number_of_Days_in_Month(yyyy)

  MonthLoop: Do mm = 1, 12
    Call ymdh_to_iv(yyyy, mm, dd=01,          hh=01,      iv=jv0)
    Call ymdh_to_iv(yyyy, mm, dd=Days_in_Month(mm), hh=Nhours, iv=jv1)

    xsum = Zero
    npoints = 0

    CheckThisMonth: Do iv = jv0, jv1
      If (Modulo(iv,25) == 0) Then
        ! Skip 25-th hour of the day.
        Cycle CheckThisMonth
      End If

      Select Case(vf(iv)%s)
        Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)

```

```

! Not a valid number: Do nothing.
Case Default
  npoints = npoints + 1
  xsum = xsum + vf(iv)%v
End Select
End Do CheckThisMonth

! Done with the month. Do we have points?
If (npoints < minN) Then
  ierr = ierr + 1
  Write(ULog, 9130) Trim(FieldInfo(k_id)%Name), yyyy, mm, npoints, minN
  Format(lx, '?? Set_Hourly_Values: ', a, lx, &
    i4, '-', i2.2, ': found ', i0, ' points, need ', i0)
  Cycle MonthLoop
End If

! Compute monthly mean.
xsum = xsum / npoints

! Set Values
SetThisMonth: Do iv = jv0, jv1
  If (Modulo(iv,25) == 0) Then
    ! Skip 25-th hour of the day.
    Cycle SetThisMonth
  End If

  Select Case(vf(iv)%s)
  Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
    vf(iv)%v = xsum
    vf(iv)%s = T_Estimated
    vf(iv)%f = ''
    ! If we are dealing with Wind Speed,
    ! Then zero Wind Direction also.
    If (k_id == f_WS) Then
      WD(iv)%v = Zero
      WD(iv)%s = T_Estimated
      WD(iv)%f = ''
    End If
  End Select
End Do SetThisMonth

End Do MonthLoop
End Do

Xok = (ierr == 0)

End Subroutine Set_Hourly_Values

Subroutine Daily_Wind_Speed(Xok)

```

```

! = 29 Jan 2002  4:05 pm
!   * module not used. Wind Speed daily value will be
!     a simple average.

! Compute Mean Wind Speed over 24-hour periods.
!* <A NAME="Daily_Wind_Speed">
Implicit None
Logical, Intent(Out) :: Xok

Integer :: jday, hh01, hh24, hh25, hh
Integer :: minN, nn, ierr, ntt, jyyyy, jmm, jdd
Real     :: xsum, ysum, resultant, Azimuth, Theta
Type(Val_and_Flag), Dimension(:), Pointer :: WD
Type(Val_and_Flag), Dimension(:), Pointer :: WS
Character(Len(FieldInfo(1)%Name)), Pointer :: txt
Type(Stat_Block) :: daily_ws
Call Stat_Initialize(daily_ws, 'Wind Speed, daily values')

ierr = 0
WD => Xparam(f_WD)%Samson_v10 ! Wind Direction in degrees
WS => Xparam(f_WS)%Samson_v10 ! Wind Speed in m/s
minN = FieldInfo(f_WD)%Minimum_obs_per_day
txt => FieldInfo(f_WD)%Name
ntt = Len_trim(txt)

! Wind Direction          0-360          Wind direction in degrees.
!                          (N = 0 or 360, E = 90, S = 180,
!                          W = 270)
!
! Wind Speed              0.0-99.0      Wind speed in m/s.
!
!           0                90
!           ^                ^
!           |                |
!           |                |
! 270 <-----+-----> 90    180 <-----+-----> 0
!           |                |
!           |                |
!           v                v
!           180              270
!
!           Azimuth          Theta
!
! Azimuth: Wind direction in degrees.
!           (N = 0 or 360, E = 90, S = 180, W = 270)
!
! Theta: regular angle measured counterclockwise

```

```

!           from the +x axis.
!
!   Interconversion Formulae:
!       Theta = Modulo(90-Azimuth, 360)
!       Azimuth = Modulo(90-Theta, 360)
!
! From Mathematica, for the test case: (ArcTan(x/y) + Pi) == Azimuth
! Note that Theta = ArcTan(y/x) + Pi
!
! Subroutine Test_Azimuth tests these formulae.

Do jday = jd0, jd1                ! step by day
  hh01 = (jday-jd0)*Nhours + 1    ! First hour of the day
  hh24 = hh01 + 23               ! Last hour of the day (24th)
  hh25 = hh24 + 1               ! 25th hour of jday == (jday-jd0+1)*NHours

  nn = 0
  xsum = Zero
  ysum = Zero

  OneDay: Do hh = hh01, hh24
    Select Case(WD(hh)%s)
      Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        Cycle OneDay
      End Select
    Select Case(WS(hh)%s)
      Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        Cycle OneDay
      End Select

    nn = nn + 1

    ! Convert Azimuth (measured in degrees) to
    ! Theta (measured in radians)
    Theta = Modulo(90-WD(hh)%v, 360) * Degrees_to_Radians
    xsum = xsum + WS(hh)%v * Cos(Theta)
    ysum = ysum + WS(hh)%v * Sin(Theta)
  End Do OneDay

If ((nn > 0) .And. (nn >= minN)) Then
  ! The range of Atan2(x,y) is: -Pi < Atan2(y,x) <= Pi
  ! Convert the range to 0 < Atan2(y,x) <= 360 (degrees)

  resultant = Pythag(xsum, ysum)
  Theta = (Atan2(ysum,xsum)+Pi) * Radians_to_Degrees ! Theta = ArcTan(y/x) in degrees
  Azimuth = Modulo(90-Theta, 360)

  WD(hh25)%v = Azimuth
  WD(hh25)%s = T_Estimated
  WD(hh25)%f = ' '

```

```

        WS(hh25)%v = resultant
        WS(hh25)%s = T_Estimated
        WS(hh25)%f = ''
        Call Stat_Add_Point(daily_ws, resultant)
    Else
        ierr = ierr + 1
        WD(hh25)%v = Missing_Data
        WD(hh25)%s = T_Missing
        WD(hh25)%f = ''

        WS(hh25)%v = Missing_Data
        WS(hh25)%s = T_Missing
        WS(hh25)%f = ''
        Call Jd_to_ymd(jday, jyyyy, jmm, jdd)
        Write(ULog, 9130) jyyyy, jmm, jdd, txt(1:ntt), nn, minN
9130    Format(1x, '?? Daily Values: ', i4, 2('-', i2.2), 1x, a, ' found ', i0, ', need ', i0)
        End If

    End Do
    Xok = (ierr == 0)
    Call Stat_Output(ULog, daily_ws)

End Subroutine Daily_Wind_Speed

```

```
Subroutine Process_Days_since_last_Snowfall(Xok)
```

```

!* <A NAME="Process_Days_since_last_Snowfall">
! Correct the field Days_since_last_Snowfall.
!
! =====
! Statement of the problem,  8 Feb 2002 10:19 am
!
! For 03937: Lake Charles, LA
!   Days since last Snowfall [days]: gap of 7008 hours starting on 1964-04-26  1h
!
! The abbreviated SAMSON record (03937_64.txt) is:
!
!           Observation Indicator (OI)
!           | Present_weather (PW)
!           | | Days Since Last Snowfall (DSL)
! yy mm dd hh | |
! -- -- -- -- - ---45--- ---
! 64  4 22  1  0 999999999 ... 61
! 64  4 23  1  0 999999999 ... 62
! 64  4 24  1  0 999999999 ... 63
!           :
!           63      ! hours 2-23
! 64  4 24 24  0 999999999 ... 63
! 64  4 25  1  0 999999999 ... 64      ! Counter increments

```

```

!           :                64      ! hours 2-23
! 64 4 25 24 0 999999999 ... 64
! 64 4 26 1 0 999999999 ... 999
! 64 4 26 2 0 999999999 ... 999
! 64 4 26 3 0 999999999 ... 999
!
! Days Since Last Snowfall counter increments on hour == 1.
!
! =====
! SAMSON information. For a description of the SAMSON file format see
!   <A HREF="e:\5\3met\Docs\samson_format.txt#$1">
!
! I 096-096      Observation Indicator      0 or 9      0 = Weather observation made.
!                                     9 = Weather observation not
! [lsr] Thu 19 Oct 2000 11:58:19          made or missing.
! This field appears to be                If this field = 9 OR if field
! column 96 of the input file.            13 (wind speed) = missing
! The manual provides no data             (9999. or 99.0), then
! to support this assertion.              fields 6, 7, 8, 10, 11, 17,
! [lsr] Wed Nov 21 09:33:59 2001          and 18 were all modeled and
! Observation Indicator appears            not actually observed.
! only in SAMSON v 1.0 files.
!
! 16 097-105     Present_weather           See          Present_weather conditions
!                                     Table          denoted by 9 indicators.
!                                     Below          See present weather table below.
!
! 20 120-122     Days Since Last Snowfall  0-88         Number of days since last snowfall.
!                                     88 = 88 or greater days.
!                                     999 = missing data.
!
! Snow-related fields of Present_weather are 4 and 5.
! Field Number
! |           Contents
! |           |           Values
! |           |           |           Description
! ---|-----|-----|-----
! (4) Occurrence of Snow, Snow Pellets, or Ice Crystals 0 - 9
!                                     0 = Light snow
!                                     1 = Moderate snow
!                                     2 = Heavy snow
!                                     3 = Light snow pellets
!                                     4 = Moderate snow pellets
!                                     5 = Heavy snow pellets
!                                     6 = Light ice crystals
!                                     7 = Moderate ice crystals
!                                     8 = Heavy ice crystals
!                                     9 = None if Observation
!                                     Indicator element equals
!                                     0, else unknown or
!                                     missing if Observation

```

```

!                               Indicator element
!                               equals 9.
!
!                               Notes:
!                               Beginning in April 1963, any
!                               occurrence of ice crystals
!                               is recorded as a 7.
!
!
! (5)      Occurrence of 0-5, 9  0 = Light snow showers
!          Snow Showers,        1 = Moderate snow showers
!          or Snow Squalls      2 = Heavy snow showers
!                               3 = Light snow squall
!                               4 = Moderate snow squall
!                               5 = Heavy snow squall
!                               9 = None if Observation
!                               Indicator element equals 0,
!                               else unknown or
!                               missing if Observation
!                               Indicator element equals 9.

```

```

Implicit None
Logical, Intent(Out) :: Xok

```

```

! Relevant field numbers
Integer, Parameter :: f4 = 4
Integer, Parameter :: f5 = 5

```

```

Integer :: jday, hh01, hh24, hh25, hh
Integer :: ierr, jyyyy, jmm, jdd, jhh
Integer :: Observation_Indicator
Logical :: DSLS_unknown, observation_is_missing
Integer :: nDSLS, hours_with_snow
Real    :: old_DSLS
Type(Val_and_Flag), Dimension(:), Pointer :: OI, DSLS
Logical :: Print_now
Integer :: jv0, jv1

```

```

Integer :: pDSLS ! See explanation below

```

```

! Observation Indicator 0 or 9  0 = Weather observation made.
!                               9 = Weather observation not made or missing.
! Present_weather - Present_weather conditions denoted by 9 indicators.
!
!   Xparam(f_OI)%Samson_v10(iv)%v = Observation_Indicator
!   Xparam(f_OI)%Samson_v10(iv)%f = Present_weather
!   Xparam(f_OI)%Samson_v10(iv)%s = data_source

```

```

OI => Xparam(f_OI)%Samson_v10    ! Observation Indicator
DSLS => Xparam(f_DSLS)%Samson_v10 ! Days since last Snowfall

```

```

Call ymdh_to_iv(1988, 12, 31, 01, jv0)
!Call ymdh_to_iv(1978, 07, 21, 01, jv1)

ierr = 0

! Start with Days Since Last Snowfall (DSLS) == Missing,
! because we do not know when the last snow occurred.
!
! This is what SAMSON did for San Juan, P.R. (WBAN=11641),
! where it has not snow in recorded history.
!
! 8 Feb 2002 4:01 pm; However, this is not what SAMSON
! did for 03937: Lake Charles, LA.
! Days Since Last Snowfall for 1961-01-01 1h is 18 days.
!
! New algorithm. If DSLS is not missing, accept it.
! Otherwise try to do something sensible with Present_weather.

nDSLS = -1                ! Non sensical value.
Do jday = jd0, jd1        ! step by day
  hh01 = (jday-jd0)*Nhours + 1 ! First hour of the day
  hh24 = hh01 + 23          ! Last hour of the day (24th)
  hh25 = hh24 + 1          ! 25th hour of jday == (jday-jd0+1)*NHours

  !Print_now = (hh01 == jv0)
  !If (print_now) Then
  !  Write(6,*) 'in dsls'
  !  Write(6,*) 'in dsls'
  !End If

! If at any time during the day DSLS(hh)%v holds
! a non-missing value, we want pDSLS to point to it.
! We will use this value for the 25th hour (the daily value).
pDSLS = Tbogus

hours_with_snow = 0
observation_is_missing = .False.

OneDay: Do hh = hh01, hh24
  ! Save the old value to test cognitive dissonances.
  old_DSLS = DSLS(hh)%v
  If (DSLS(hh)%s /= T_Missing) Then
    nDSLS = Nint(old_DSLS)
    pDSLS = hh
    DSLS_unknown = .False.
  Else
    DSLS_unknown = .True.
  End If

!Print_now = ((jv0 <= hh) .And. (hh <= jv1))

```

```

!If (Print_now) Then
!   Write (6, *) 'in Process_Days_since_last_Snowfall'
!End If

Observation_Indicator = Nint(OI(hh)%v) ! 0 or 9
Select Case(Observation_Indicator)
Case(9) ! Weather observation not made or missing.
    observation_is_missing = .True.

Case(0) ! Weather observation made.
    ! It snow if
    ! a) (Present_weather, field #4 /= 9) ! snow
    !     OR
    ! b) (Present_weather, field #5 /= 9) ! snow showers
    observation_is_missing = .False.
    If ((OI(hh)%f(f4:f4) /= '9') .Or. (OI(hh)%f(f5:f5) /= '9')) Then
        hours_with_snow = hours_with_snow + 1
        !Call iv_to_ymdh(hh, jyyyy, jmm, jdd, jhh)
        !Write (ULog, 9310) hh, jyyyy, jmm, jdd, jhh, OI(hh)%f(f4:f5)
9130      Format ('## Observation Indicator == 0: ', &
                ' for ', i7, 1x, i4, '-', i2.2, '-', i2.2, i3, 'h', &
                '; OI(hh)%f(f4:f5) == ', a, '')
    End If

Case Default
    Call iv_to_ymdh(hh, jyyyy, jmm, jdd, jhh)
9150      Write (ULog, 9150) Observation_Indicator, hh, jyyyy, jmm, jdd, jhh
    Format ('?? Observation Indicator not 0 nor 9: ', i0, &
            ' for ', i7, 1x, i4, '-', i2.2, '-', i2.2, i3, 'h')
    observation_is_missing = .True.
End Select

! If Days since last Snowfall is unknown, try to do something sensible.
If (DSLS_unknown) Then
    ! If it snow during this fraction of a day, then Days since last Snowfall == 0.
    ! This inference is valid regardless of the value of observation_is_missing.
    If (hours_with_snow > 0) Then
        nDSLS = 0
        DSLS(hh)%v = nDSLS
        DSLS(hh)%s = T_Estimated
        DSLS(hh)%f = ''
        pDSLS = hh
        DSLS_unknown = .False.
    Else If (observation_is_missing) Then
        ! Nothing we can do. It may have snow (or not).
        nDSLS = -1 ! Non sensical value.
    Else
        ! It has not snow.
        If (nDSLS >= 0) Then
            ! We have an estimate.

```

```

        DSLS(hh)%v = nDSLS
        DSLS(hh)%s = T_Estimated
        DSLS(hh)%f = ''
        pDSLS = hh
        DSLS_unknown = .False.
    Else
        ! We still do not know when the last snowfall occurred.
        DSLS(hh)%v = Missing_Data
        DSLS(hh)%s = T_Missing
        DSLS(hh)%f = ''
    End If
End If
Else
    ! Old value not unknown. Keep it. Nothing to do.
End If
!If (print_now) Then
    ! Write (ULog, *) str_iv_to_ymdh(hh), DSLS(hh)
!End If
End Do OneDay

! Now assign a daily value. If we had a non-missing value
! during the day, use it.
If (pDSLS /= Tbogus) Then
    ! We had a good hour. Use it for the daily value
    DSLS(hh25)%v = DSLS(pDSLS)%v
    DSLS(hh25)%s = T_Estimated
    DSLS(hh25)%f = ''
Else
    ! Else no good hours during this day. Missing.
    DSLS(hh25)%v = Missing_Data
    DSLS(hh25)%s = T_Missing
    DSLS(hh25)%f = ''
End If
!If (print_now) Then
! Write (ULog, *) str_iv_to_ymdh(hh25), DSLS(hh25)
!End If

! If the value of nDSLS is known, increment it.
! The maximum value of this field is 88 days.
! See SAMSON description above.
If (nDSLS >= 0) nDSLS = Min(nDSLS + 1, 88)

End Do
Xok = (ierr == 0)

End Subroutine Process_Days_since_last_Snowfall

```

```

Subroutine Process_BAOD(Xok)

```

```

!* <A NAME="Process_BAOD">
! Broadband aerosol optical_depth (broadband turbidity) on the day indicated.
! Range: 0.0-0.900
!
! SAMSON. BAOD is not present during nighttime. One value (measured or
! estimated) was produced and replicated for the day. See fragment
! file below (13893_61.txt):
!   yy mm dd hh BAOD
!   61  1  1  1 99999.
!   61  1  1  : 99999.  ! hours 2-6
!   61  1  1  7 99999.
!   61  1  1  8  .034
!   61  1  1  :  .034  ! hours 9-16
!   61  1  1 17  .034
!   61  1  1 18 99999.
!   61  1  1  : 99999.  ! hours 19-6
!   61  1  2  7 99999.
!   61  1  2  8  .104
!   61  1  2  9  .104  ! etc.
!
! Algorithm:
! For every day: Starting at 1 h, find the first non-missing
! value, call it "rv". Replace all missing values during that
! day with "rv".
Implicit None
Logical, Intent(Out) :: Xok

Integer :: jday, hh01, hh24, hh
Integer :: ierr, jyyyy, jmm, jdd
Real    :: rv
Logical :: xfound
Type(Val_and_Flag), Dimension(:), Pointer :: BAOD

ierr = 0
BAOD => Xparam(f_baod)%Samson_v10

Do jday = jd0, jd1                ! step by day
  hh01 = (jday-jd0)*Nhours + 1  ! First hour of the day
  hh24 = hh01 + 23              ! Last hour of the day (24th)

  xfound = .False.
  FindRv: Do hh = hh01, hh24
    Select Case(BAOD(hh)%s)
      Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        ! Do nothing.
      Case Default
        rv = BAOD(hh)%v
        xfound = .True.
        Exit FindRv
    End Select
  End Do
End Do

```

```

End Do FindRv

If (.Not. xfound) Then
  ierr = ierr + 1
  Call Jd_to_ymd(jday, jyyyy, jmm, jdd)
  Write(ULog, 9130) jyyyy, jmm, jdd
9130  Format(lx, '?? Process_BAOD: Values all missing for ', i4, 2('-',i2.2))
  Cycle
End If

OneDay: Do hh = hh01, hh24
  If (BAOD(hh)%s == T_Missing) Then
    BAOD(hh)%v = rv
    BAOD(hh)%s = T_Estimated
    BAOD(hh)%f = ''
  End If
End Do OneDay
End Do
Xok = (ierr == 0)

End Subroutine Process_BAOD

```

```

Subroutine Daylight_Prevailing_Wind(Xok, MET_pwd, MET_pws, OnlyDaylight)

```

```

! Compute Daylight Prevailing Wind
! <A NAME="Daylight_Prevailing_Wind">

! Xok -- truth of "all entries in MET_* not missing"
! MET_pwd -- prevailing wind direction
! MET_pws -- prevailing wind speed
! OnlyDaylight -- Truth of "compute prevailing stuff using only daylight hours"

! From http://www.doc.mmu.ac.uk/aric/eae/Climate/Older/Prevailing\_Winds.html
!
! The direction of wind is measured in terms of where the air
! is coming from. A northerly wind blows air from north to south.
! A southwesterly wind blows air from the southwest to the northeast.
!
! The prevailing wind is the wind that blows most frequently across a
! particularly region. Different regions on Earth have different
! prevailing wind directions which are dependent upon the nature of
! the general circulation of the atmosphere and the latitudinal wind
! zones.
!
! Prevailing wind - The wind direction most frequently observed during a
! given period. The periods most frequently used are the observational day,
! month, season, and year. Methods of determination vary from a single count

```

```

! of periodic observations to the computation of a wind rose.

Implicit None
Logical,                               Intent(Out) :: Xok
Type(Val_and_Flag), Dimension(jd0:jd1), Intent(Out) :: MET_pwd, MET_pws
Logical,                               Intent(In)  :: OnlyDaylight

Character(Len=80) :: Selection_Criterion = ''
Integer :: i, j, n, n0, j0, j1, ierr, jb, points_in_day
Integer :: iq, idim, jt, ik, nmax, ndel, min_d12
Integer :: jday, hh01, hh24, hh
Integer :: jyyyy, jmm, jdd, jhh
Integer :: prevailing_quadrant, Prevailing_CoordSys, ncount
Logical :: is_daytime, N_is_even
Real    :: xws, xwd, median_wd, mean_ws, vmax
Type(Val_and_Flag), Dimension(:), Pointer :: WD, WS, Rs

Integer, Parameter :: CoordAx_dim = 6
Type(XPerSeg), Dimension(CoordAx_dim), Target, Save :: CoordAx ! Coordinate Axes
Type(XQuadrant), Pointer :: qik
Logical :: have_quadrant
Integer, Dimension(:), Pointer :: p2Hour
Integer, Pointer :: p2N
Logical, Save :: first_time = .True.

Logical :: Print_now = .False.
Integer :: jul_day0, jul_day1

ierr = 0
WD => Xparam(f_WD)%Samson_v10   ! Wind Direction in degrees
WS => Xparam(f_WS)%Samson_v10   ! Wind Speed in m/s
Rs => Xparam(f_Rs)%Samson_v10   ! Global Horizontal Radiation

MET_pwd = Val_and_Flag(T_Missing, Missing_Data, '')
MET_pws = Val_and_Flag(T_Missing, Missing_Data, '')

jul_day0 = Jd(1976, 01, 05)
jul_day1 = Jd(1976, 01, 09)

If (first_time) Then

    ! Allocate only once.
    first_time = .False.

    ! Allocate Quadrant/HemiPlane arrays.
    ! isub, rdiv, imod define the function
    !     i = Modulo(Floor((x-isub)/rdiv),imod) + 1
    ! imod = number of quadrants
    ! rdiv * imod = 360

```

```

! Define 2 sets of coordinate axes (q00, q45), each with 4 quadrants
! q00 -- standard coordinate axes.
!     i = Modulo(Floor((x)/90.0),4) + 1

iq = 1
idim = 4
Allocate(CoordAx(iq)%Quads(idim), CoordAx(iq)%Tied_List(idim))
CoordAx(iq)%isub = 0
CoordAx(iq)%imod = idim
CoordAx(iq)%rdiv = 360.0 / idim

! q45 -- axes rotated 45°
!     i = Modulo(Floor((x-45)/90.0),4) + 1
iq = iq + 1
idim = 4
Allocate(CoordAx(iq)%Quads(idim), CoordAx(iq)%Tied_List(idim))
CoordAx(iq)%isub = 45
CoordAx(iq)%imod = idim
CoordAx(iq)%rdiv = 360.0 / idim

! hemiplanes: 2 "quadrants"
Do i = 1, 4
    iq = iq + 1
    idim = 2
    Allocate(CoordAx(iq)%Quads(idim), CoordAx(iq)%Tied_List(idim))
    jb = (i-1) * 45 ! jb == 0, 45, 90, 135
    !i = Modulo(Floor((x-jb)/180.0),2) + 1
    CoordAx(iq)%isub = jb
    CoordAx(iq)%imod = idim
    CoordAx(iq)%rdiv = 360.0 / idim
End Do

!! http://www.windpower.dk/tour/wres/rose.htm
!! Wind rose. Divide the compass into 12 sectors (European Wind
!! Atlas standard), one for each 30 degrees of the horizon.
!! A wind rose may also be drawn for 8 or 16 sectors.
!iq = iq + 1
!idim = 12
!Allocate(CoordAx(iq)%Quads(idim), CoordAx(iq)%Tied_List(idim))
!CoordAx(iq)%isub = 0
!CoordAx(iq)%imod = idim
!CoordAx(iq)%rdiv = 360.0 / idim
End If

ByJD: Do jday = jd0, jd1 ! step by day
    hh01 = (jday-jd0)*Nhours + 1 ! First hour of the day
    hh24 = hh01 + 23 ! Last hour of the day (24th)

! #1. Find the quadrant with the most observations.

```





```

Select Case(WS(hh)%s)
Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  Cycle OneDay
End Select

points_in_day = points_in_day + 1
xwd = WD(hh)%v
xws = WS(hh)%v

Do iq = 1, CoordAx_dim
  ! (1): ik = Modulo(Floor((xwd-0)/90.0),4) + 1
  ! (2): ik = Modulo(Floor((xwd-45)/90.0),4) + 1
  ! (3-6): jb = (j-1) * 45, j = 1..4, jb == 0, 45, 90, 135
  !       ik = Modulo(Floor((xwd-jb)/180.0),2) + 1
  ! (7): ik = Modulo(Floor((xwd-0)/30.0),12) + 1

  ik = Modulo(Floor((xwd-CoordAx(iq)%isub)/CoordAx(iq)%rdiv), &
    CoordAx(iq)%imod) + 1
  qik => CoordAx(iq)%Quads(ik) ! The target quadrant

  n = qik%n + 1
  qik%n = n
  qik%h(n) = hh
  qik%ws_mean = qik%ws_mean + xws

  ! ndel -- the absolute delta time from 12 noon
  ndel = Abs(hh - hh01 + 1 - 12)

  If ((xws-qik%ws_max) > Eps0) Then
    ! xws > maximum so far : Found a new maximum.
    qik%ws_max = xws
    qik%ws_d12 = ndel
  Else If (Abs(xws-qik%ws_max) < Eps0) Then
    ! xws == maximum wind speed
    ! We have a wind speed equal to the recorded maximum WS.
    ! If this wind speed falls in the same quadrant of the
    ! recorded maximum, ok. Else we cannot determine the
    ! most frequent quadrant uniquely.
    If (ndel < qik%ws_d12) Then
      qik%ws_d12 = ndel
    End If
  End If
End Do
End Do OneDay

If (points_in_day == 0) Then
  ! No data.
  ierr = ierr + 1
  Cycle ByJD
End If

```

```

! Compute mean wind speeds for each quadrant.
Do iq = 1, CoordAx_dim
  Do ik = 1, CoordAx(iq)%imod
    n = CoordAx(iq)%Quads(ik)%n
    If (n == 0) Cycle
    CoordAx(iq)%Quads(ik)%ws_mean = CoordAx(iq)%Quads(ik)%ws_mean / n
  End Do
End Do

have_quadrant = .False.
Do iq = 1, CoordAx_dim
  ! prevailing_quadrant -- the quadrant with the most observations.
  ! This is the prevailing wind quadrant.
  ! CoordAx(iq)%Quads(prevaling_quadrant)%n --
  ! The number of elements the fell
  ! in prevailing_quadrant.
  ! ncount > 1 ==> ties.
  prevailing_quadrant = Maxloc(CoordAx(iq)%Quads(:)%n, Dim=1)
  nmax = CoordAx(iq)%Quads(prevaling_quadrant)%n

  ! Tied_List is a list of the tied quadrants.
  ncount = 0
  Do ik = 1, CoordAx(iq)%imod
    If (CoordAx(iq)%Quads(ik)%n == nmax) Then
      ncount = ncount + 1
      CoordAx(iq)%Tied_List(ncount) = ik
    End If
  End Do
  CoordAx(iq)%Tied_N = ncount

  If (ncount == 1) Then
    have_quadrant = .True.
    Prevailing_CoordSys = iq
    ik = prevailing_quadrant
    p2N => CoordAx(iq)%Quads(ik)%n
    p2Hour => CoordAx(iq)%Quads(ik)%h
    Selection_Criterion = 'quadrant with the most observations'
    Exit
  End If
End Do

If (.Not. have_quadrant) Then
  ! Among the tied quadrants, find the one with the largest mean wind speed.
  L41: Do iq = 1, CoordAx_dim
    ncount = 0
    vmax = -Huge(Zero)

    L42: Do jt = 1, CoordAx(iq)%Tied_N
      ik = CoordAx(iq)%Tied_List(jt)

```

```

    If (CoordAx(iq)%Quads(ik)%ws_mean > vmax) Then
        ncount = 1
        vmax = CoordAx(iq)%Quads(ik)%ws_mean
        prevailing_quadrant = ik
    Else If (Abs(CoordAx(iq)%Quads(ik)%ws_mean - vmax) < Eps0) Then
        ! CoordAx(iq)%Quads(ik)%ws_mean == vmax
        ! A tie. Note that we cannot give up yet because vmax
        ! may still change.
        ncount = ncount + 1
    End If
End Do L42

If (ncount == 1) Then
    have_quadrant = .True.
    Prevailing_CoordSys = iq
    ik = prevailing_quadrant
    p2N => CoordAx(iq)%Quads(ik)%n
    p2Hour => CoordAx(iq)%Quads(ik)%h
    Selection_Criterion = 'tied quadrant with the largest mean wind speed'
    Exit
End If

End Do L41
End If

If (.Not. have_quadrant) Then
    ! Among the tied quadrants, select the quadrant where the
    ! maximum wind speed is closest to noon.
L51: Do iq = 1, CoordAx_dim
    ncount = 0
    min_d12 = Huge(0)

    L52: Do jt = 1, CoordAx(iq)%Tied_N
        ik = CoordAx(iq)%Tied_List(jt)
        If (CoordAx(iq)%Quads(ik)%ws_d12 < min_d12) Then
            ncount = 1
            min_d12 = CoordAx(iq)%Quads(ik)%ws_d12
            prevailing_quadrant = ik
        Else If (CoordAx(iq)%Quads(ik)%ws_d12 == min_d12) Then
            ! A tie. Note that we cannot give up yet because min_d12
            ! may still change.
            ncount = ncount + 1
        End If
    End Do L52

    If (ncount == 1) Then
        have_quadrant = .True.
        Prevailing_CoordSys = iq
    End If
End If

```

```

        ik = prevailing_quadrant
        p2N => CoordAx(iq)%Quads(ik)%n
        p2Hour => CoordAx(iq)%Quads(ik)%h
        Selection_Criterion = 'tied quadrant with the maximum wind speed closest to noon'
        Exit
    End If

End Do L51
End If

! 25 Apr 2002 11:21 am: Debugging test.
!   For 26451: Anchorage, AK,
!   hh01 == 66701, 1968-04-22  1h
!   Selection_Criterion: tied quadrant with the largest mean wind speed

!If (.Not. have_quadrant) Then
If ((.Not. have_quadrant) .Or. (Print_now)) Then
    Call iv_to_ymdh(hh01, jyyyy, jmm, jdd, jhh)
    If (Print_now) Then
        Write(ULog, 9130) &
            '?? Daylight_Prevailing_Wind: "Print_now" at ', &
            hh01, jyyyy, jmm, jdd, jhh, 'h'
    Else
        Write(ULog, 9130) &
            '?? Daylight_Prevailing_Wind: No resolution at ', &
            hh01, jyyyy, jmm, jdd, jhh, 'h'
    End If
9130 Format (/, 1x, a, i7, 1x, i4, 2('-',i2.2), i3, a)

Qerr: Do hh = hh01, hh24
    Select Case(Rs(hh)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        Cycle Qerr
    End Select

    If (OnlyDaylight) Then
        ! Skip non-daylight hours.
        is_daytime = (Rs(hh)%v > Zero)
        If (.Not. is_daytime) Then
            Cycle Qerr
        End If
    End If

    Select Case(WD(hh)%s)
    Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        Cycle Qerr
    End Select
    Select Case(WS(hh)%s)

```

```

Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
  Cycle Qerr
End Select

i = hh - hh01 + 1
Write(ULog, 9150) &
  ' ', hh, jyyyy, jmm, jdd, i, 'h WD, WS: ', WD(hh)%v, WS(hh)%v
9150 Format (1x, a, i7, 1x, i4, 2('-',i2.2), i3, a, 1p2g14.6)
End Do Qerr

Write (ULog, *) '      have_quadrant: ', have_quadrant
If (have_quadrant) Then
  Write (ULog, *) 'Selection_Criterion: ', Trim(Selection_Criterion)
  Write (ULog, *) 'Prevailing_CoordSys: ', Prevailing_CoordSys
  Write (ULog, *) 'prevailing_quadrant: ', prevailing_quadrant
  Write (ULog, *) '      p2N: ', p2N
  n0 = Min(p2N, Ubound(CoordAx(1)%Quads(1)%h,1), 24)
  Write (ULog, *) '      p2Hour: ', p2Hour(1:n0)
End If
Do iq = 1, CoordAx_dim
  Write(ULog, *)
  Write(ULog, *) '====='
  Write(ULog, *) 'CoordAx(', iq, ')'
  n = CoordAx(iq)%Tied_N
  n0 = Min(n, 4)
  Write(ULog, *) '      %Tied_List:', CoordAx(iq)%Tied_List(1:n0)
  Do ik = 1, CoordAx(iq)%imod
    n = CoordAx(iq)%Quads(ik)%n
    n0 = Min(n, Ubound(CoordAx(1)%Quads(1)%h,1), 24)
    Write(ULog, *)
    Write(ULog, *) '      %Quads(', ik, '): n == ', n
    Write(ULog, *) '      h:', CoordAx(iq)%Quads(ik)%h(1:n0)
    Write(ULog, *) '      %ws_max:', CoordAx(iq)%Quads(ik)%ws_max
    Write(ULog, *) '      %ws_d12:', CoordAx(iq)%Quads(ik)%ws_d12
    Write(ULog, *) '      %ws_mean:', CoordAx(iq)%Quads(ik)%ws_mean
  End Do
End Do
Call FLushAll()
End If

If (.Not. have_quadrant) Then
  ierr = ierr + 1
  Cycle ByJD
End If

! The prevailing wind direction is the median of the
! wind directions that fell in the most frequent quadrant.
!
! First, sort the wind directions in the most frequent quadrant.
! Insertion sort is used because the number of entries is small

```

```

! (at most 24 entries).
!
! Note that something like
!   Call Insertion_Sort_Real(XList=WD(hh01:hh24)%v, &
!     pindex=p2Hour, low=1, high=p2N, sortup=.True.)
!
! is problematic: The entries of WD (hh01:hh24) are (say)
! 137058:137067. On entry to Insertion_Sort_Real the addresses
! of WD are 1:24. However, the values in pindex are still in
! the range 137058:137067. So, references to WD(p2Hour(j)) will
! be out of bounds *within* the subroutine. We need to pass the
! address bounds of WD to the subroutine. This is more trouble
! than it is worth, since the insertion sort is used only once
! and the code is short. Therefore ( 2 Jul 2002 1:29 pm) code in line.

! Insertion sort is best if the number of entries is less than
! approximately 20 entries.
Do i = 1, p2N - 1
  Do j = i+1, p2N
    If (WD(p2Hour(j))%v < WD(p2Hour(i))%v) Then
      n0 = p2Hour(i)
      p2Hour(i) = p2Hour(j)
      p2Hour(j) = n0
    End If
  End Do
End Do

If (Print_now) Then
  Write(ULog, *)
  Write(ULog, *) '=====
  n0 = Min(p2N, Ubound(CoordAx(1)%Quads(1)%h,1), 24)
  Write(ULog, *) 'Sorted p2Hour: ', p2Hour(1:n0)
  Write(ULog, *) 'Sorted WD(p2Hour(1:n0))%v: ', WD(p2Hour(1:n0))%v
  Write(ULog, *)
End If

! Compute median
N_is_even = (Modulo(p2N,2) == 0)
If (N_is_even) Then
  i = p2N / 2
  j0 = p2Hour(i)
  j1 = p2Hour(i+1)
  median_wd = (WD(j0)%v + WD(j1)%v) / 2
  If (Print_now) Then
    Write(ULog, *) 'N is even: i, j0, j1: ', i, j0, j1
  End If
Else
  i = (p2N + 1) / 2
  j0 = p2Hour(i)
  median_wd = WD(j0)%v

```

```

        If (Print_now) Then
            Write(ULog, *) 'N is odd: i, j0: ', i, j0
        End If
    End If

    !! The prevailing wind speed is the mean of the
    !! wind speeds that fell in the prevailing quadrant.
    !mean_ws = Zero
    !Do i = 1, p2N
    !    j0 = p2Hour(i)
    !    mean_ws = mean_ws + WS(j0)%v
    !End Do
    !mean_ws = mean_ws / p2N
    mean_ws = CoordAx(Prevailing_CoordSys)%Quads(prevailing_quadrant)%ws_mean

    If (Print_now) Then
        Write(ULog, *) 'median_wd: ', median_wd
        Write(ULog, *) 'mean_ws..: ', mean_ws
    End If

    MET_pwd(jday) = Val_and_Flag(T_Estimated, median_wd, '')
    MET_pws(jday) = Val_and_Flag(T_Estimated, mean_ws, '')

End Do ByJD
Xok = (ierr == 0)

End Subroutine Daylight_Prevailing_Wind

```

```

Function a_eq_b(A, B)

    Implicit None
    Type(XQuadrant), Dimension(:), Intent(In) :: A, B
    Logical :: a_eq_b

    Integer :: sa, sb, i

    a_eq_b = .False.
    sa = Size(A)
    sb = Size(B)

    If (sa /= sb) Return

    Do i = 1, sa
        If (A(i)%n /= B(i)%n) Then
            Return
        End If
    End Do

    If (Any(A(i)%h /= B(i)%h)) Then

```

```
        Return
    End If
End Do
a_eq_b = .True.

End Function a_eq_b

End Module Utils2
```

## Utils3

! Last change: LSR 22 Aug 2002 4:04 pm

Module Utils3

```
! List generated by Print_Cols
!!! 1 16- 22: Extraterrestrial Horizontal Radiation
!!! 2 24- 30: Extraterrestrial Direct Normal Radiation
!!! 3 32- 40: Global Horizontal Radiation
!!! 4 42- 50: Direct Normal Radiation
!!! 5 52- 60: Diffuse Horizontal Radiation
!!! 6 62- 65: Total Sky_Cover
!!! 7 67- 70: Opaque Sky_Cover
!!! 8 72- 78: Dry Bulb Temperature
!!! 9 80- 86: Dew_Point Temperature
!!! 10 88- 92: Relative Humidity
!!! 11 94-100: Station Pressure
!!! 12 102-106: Wind Direction
!!! 13 108-114: Wind Speed
!!! 14 116-123: Visibility
!!! 15 125-132: Ceiling Height
!!! 16 134-136: Observation Indicator
!!! 17 138-148: Present_weather
!!! 18 150-154: Precipitable Water
!!! 19 156-163: Broadband Aerosol Optical_Depth
!!! 20 165-170: Snow_Depth
!!! 21 172-176: Days Since Last Snowfall
!!! 22 178-186: Hourly Precipitation
!!! 23 188-197: FAO Short Grass PET
!!! 24 199-208: K-P FWS Evaporation
!!! 25 210- : Pan Evaporation
```

```
!Use Binary_Tree
!Use FileStuff
!Use GetNumbers
!Use Linked_List
!Use Read_Info
!Use SAMSON
Use Strings
!Use Utils0
!Use Utils1
!Use Date_Module
Use Global_Variables
!Use IoSubs
!Use Utils2
!Use Winteracter
Implicit None
```

Contains

```

Subroutine Print_Cols(Col_Text, k0, k1, Last, Icount, Initialize)

  ! Call Print_Cols('Extraterrestrial Horizontal Radiation', k0, k1, Icount=1)
  Implicit None
  Character(Len=*), Intent(In) :: Col_Text
  Integer,          Intent(In) :: k0, k1

  ! The value of "Last" is unimportant.
  ! We check only for the presence/absence of the variable.
  Logical, Optional, Intent(In) :: Last
  Integer, Optional, Intent(In) :: Icount
  Logical, Optional, Intent(In) :: Initialize

  Integer, Save :: uu = 9010
  Integer, Save :: Kcount = 0
  Logical, Save :: do_return = .False.

  If (Present(Initialize)) Then
    ! By increasing 'uu' we will be able to store
    ! different sequences in different files.
    uu = uu + 1
    do_return = .False.
    Return
  End If

  If (do_return) Then
    Return
  End If

  If (Present(Icount)) Then
    Kcount = Icount - 1
  End If

  If (Present(Last)) Then
    do_return = .True.
    Return
  End If

  Kcount = Kcount + 1
  Write (uu, 9130) Kcount, k0, k1, Trim(Col_Text)
9130 Format ('!!! ', 3x, i4, 3x, i3, '-', i3, ': ', a)

End Subroutine Print_Cols

```

```

Subroutine dvf_MkFMT(k0, xFMT, Last)

```

```

  ! Call dvf_MkFMT(k0, '(f10.1)')

```

```

Implicit None
Integer,          Intent(In) :: k0
Character(Len=*), Intent(In) :: xFMT

! The value of "Last" is unimportant.
! We check only for the presence/absence of the variable.
Logical, Optional, Intent(In) :: Last

Integer, Save :: ip = 1
Logical, Save :: do_return = .False.

Integer :: j0, j1

If (do_return) Then
  Return
End If

If (Present(Last)) Then
  do_return = .True.
  Write (ULog, '(/, 2a, /)') '## Daily values file format: ', Trim(FMT_dvf)
  Return
  ! Stop '## Stopping in "dvf_MkFMT" as requested'
End If

! We assume xFMT with leading and trailing parenthesis.
! We will not check.

j0 = 2
j1 = Len_trim(xFMT) - 1
If (ip > 1) Then
  FMT_dvf(ip:ip+1) = ', '
  ip = ip + 2
End If

Write (FMT_dvf(ip:), 9130) k0, Trim(xFMT(j0:j1))
ip = Len_trim(FMT_dvf) + 1

9130 Format ('t', i0, ', ', a)

End Subroutine dvf_MkFMT

Subroutine dvf_Cols(Col_Text, k0, k1, Last, Icount, Initialize)

! Call dvf_Cols('Extraterrestrial Horizontal Radiation', k0, k1, Icount=1)
Implicit None
Character(Len=*), Intent(In) :: Col_Text
Integer,          Intent(In) :: k0, k1

```

```

! The value of "Last" is unimportant.
! We check only for the presence/absence of the variable.
Logical, Optional, Intent(In) :: Last
Integer, Optional, Intent(In) :: Icount
Logical, Optional, Intent(In) :: Initialize

Character(132) :: qbuf
Integer, Save :: uu = 9010
Integer, Save :: Kcount = 0
Logical, Save :: do_return = .False.

If (Present(Initialize)) Then
  ! By increasing 'uu' we will be able to store
  ! different sequences in different files.
  uu = ULog
  do_return = .False.
  Return
End If

If (do_return) Then
  Return
End If

If (Present(Icount)) Then
  Kcount = Icount - 1
End If

If (Present(Last)) Then
  do_return = .True.
  Return
  ! Stop '## Stopping in "dvf_Cols" as requested'
End If

Kcount = Kcount + 1
Write (qbuf, 9130) Kcount, k0, k1, Trim(Col_Text)
9130 Format (3x, i4, 3x, i3, '-', i3, ': ', a)

Write (ULog, '(1x,a)') Trim(qbuf)
End Subroutine dvf_Cols

Subroutine hvf_Cols(Col_Text, k0, k1, Xfmt, Last, Icount, Initialize)

! Call hvf_Cols('Extraterrestrial Horizontal Radiation', k0, k1, Icount=1)
Implicit None
Character(Len=*), Intent(In) :: Col_Text
Integer, Intent(In) :: k0, k1
Character(Len=*), Intent(In) :: Xfmt

```

```

! The value of "Last" is unimportant.
! We check only for the presence/absence of the variable.
Logical, Optional, Intent(In) :: Last
Integer, Optional, Intent(In) :: Icount
Logical, Optional, Intent(In) :: Initialize

Integer, Save :: uu = 9010
Integer, Save :: Kcount = 0
Logical, Save :: do_return = .False.

Character(Len=132) :: qbuf
Character(Len=50)  :: wfmt, wtype
Integer :: j0, j1
Integer, Save :: ip = 1

uu = ULog
If (Present(Initialize)) Then
    do_return = .False.
    Return
End If

If (do_return) Then
    Return
End If

If (Present(Icount)) Then
    Kcount = Icount - 1
End If

If (Present(Last)) Then
    do_return = .True.
    Write (ULog, '(/, 2a, /)') '## Hourly values file format: ', Trim(FMT_hvf)
    Return
    ! Stop '## Stopping in "hvf_MkFMT" as requested'
End If

Kcount = Kcount + 1

! We assume xFMT with leading and trailing parenthesis.
! We will not check.
wfmt = Xfmt
Call Collapse(wfmt, NewLen=j1)
j0 = 2
j1 = j1 - 1
Select Case(wfmt(j0:j0))
Case('f')
    wtype = 'Real'
Case('i')
    wtype = 'Integer'
Case Default

```

```

        wtype = ''
    End Select

    If (ip > 1) Then
        FMT_hvf(ip:ip+1) = ', '
        ip = ip + 2
    End If

    Write (FMT_hvf(ip:), 9130) k0, wfmt(j0:j1)
    ip = Len_trim(FMT_hvf) + 1
    Write (6,*) '--', FMT_hvf(1:ip)

9130 Format ('t', i0, ', ', a)

    Write (qbuf, 9150) Kcount, k0, k1, Trim(Col_Text), wtype(1:10), wfmt(j0:j1)
9150 Format (3x, i4, 3x, i3.3, ' - ', i3.3, 2x, a, 5x, a, 3x, a)

    Write (ULog, '(lx,a)') Trim(qbuf)

End Subroutine hvf_Cols

End Module Utils3

```

## Utils4

! Last change: LSR 16 May 2002 3:21 pm

Module Utils4

```
Use Global_Variables
Use Date_Module
Use Utils1
Use Utils5
Implicit None
```

Contains

Subroutine Daily\_Values(k\_id, T\_code, Xok)

```
Implicit None
Integer,      Intent(In)  :: k_id
Character(Len=*), Intent(In) :: T_code ! Daily value code.
Logical,      Intent(Out) :: Xok

Integer :: jday, hh01, hh24, hh25, iv
Integer :: nn, ierr
Real     :: xsum
Type(Val_and_Flag), Dimension(:), Pointer :: VF
Character(Len(FieldInfo(1)%Name)), Pointer :: txt

ierr = 0
VF => Xparam(k_id)%Samson_v10
txt => FieldInfo(k_id)%Name

Do jday = jd0, jd1                ! step by day
  hh01 = (jday-jd0)*Nhours + 1    ! First hour of the day
  hh24 = hh01 + 23                ! Last hour of the day (24th)
  hh25 = hh24 + 1                ! 25th hour of jday == (jday-jd0+1)*NHours

  If (T_code == T_Not_Applicable) Then
    VF(hh25) = Val_and_Flag(T_Not_Applicable, Zero, '')
    Cycle
  End If

  ! xsum = Sum(VF(hh01:hh24)%v, Mask=(VF(hh01:hh24)%s /= T_Missing))
  ! nn   = Count(Mask=(VF(hh01:hh24)%s /= T_Missing))
  xsum = Zero
  nn = 0
  Do iv = hh01, hh24
    Select Case(VF(iv)%s)
      Case(T_Missing, T_Not_Applicable, T_Undefined, T_Perpetual_Darkness)
        ! Do nothing.
```

```

    Case Default
      xsum = xsum + VF(iv)%v
      nn = nn + 1
    End Select
  End Do

  If (nn == 0) Then
    ! All values missing
    VF(hh25) = Val_and_Flag(T_Missing, Missing_Data, '')

  Else If (T_code == T_Average) Then
    ! Daily value is an average.
    VF(hh25) = Val_and_Flag(T_Estimated, xsum/nn, '')

  Else If (T_code == T_Cumulative) Then
    ! Daily value is cumulative.
    VF(hh25) = Val_and_Flag(T_Estimated, xsum, '')
  End If

End Do
Xok = (ierr == 0)

End Subroutine Daily_Values

Subroutine Set_Missing_Flags()

! <A NAME="Set_Missing_Flags">
! <A HREF="Utils4.f90#Set_Missing_Flags">

! Numerically stable computation of the variance.
!
!
!           1      n
!   Sample variance = ---- Sum (x_i-x_mean)^2
!                   n - 1  i=1
! Reference:
! [2] Nicholas J. Higham. 1996. Accuracy and Stability of Numerical
!     Algorithms. SIAM (Society for Industrial & Applied Mathematics).
!     ISBN 0-89871-355-2. Page 13.
!
! Accumulate:
!
!           1      k
!   M_k = - * Sum x_i
!          k     i=1
!
!           k           k           1      k
!   Q_k = Sum (x_i - M_k)^2 = Sum (x_i)^2 - - (Sum x_i)^2
!           i=1           i=1           k  i=1

```

```

!
! Updating formulae:
!
!   M_1 = x_1
!   M_k = M_{k-1} +  $\frac{x_k - M_{k-1}}{k}$ ,          k = 2..n
!
!   Q_1 = 0
!   Q_k = Q_{k-1} +  $\frac{(k-1)(x_k - M_{k-1})^2}{k}$ ,    k = 2..n
!
! After which:
!
!   Sample Mean = M_n
!
!   Sample Variance =  $\frac{Q_n}{n - 1}$ 
!
! Note that the updating formulae can be written:
!
!   M_0 = 0
!   M_k = M_{k-1} +  $\frac{x_k - M_{k-1}}{k}$ ,          k = 1..n
!
!   Q_0 = 0
!   Q_k = Q_{k-1} +  $\frac{(k-1)(x_k - M_{k-1})^2}{k}$ ,    k = 1..n
!

Implicit None

Integer :: iv, jpar, yyyy, mm, dd, hh
Integer :: npts, k
Integer :: n_missing, n_nan, n_out_of_range, n_undefined
Integer :: n_Perpetual_Darkness
Integer :: n_1_24, n_25
Integer :: loop_min, loop_max, loop_inc
Logical  :: erase_value, Have_Precip_Data
Real     :: minV, maxV
Real     :: M_k, Q_k, M_kml, x_k
Real     :: v_mean, v_variance, v_std_dev, v_min, v_max
Type(Val_and_Flag), Dimension(:), Pointer :: vf

Call Station_With_Missing_Data(pWBAN%WBAN, Have_Precip_Data)

Write (ULog, '(//)')

Loop_Jpar: Do jpar = 1, f_end

```

```

vf => Xparam(jpar)%Samson_v10

Select Case(jpar)
Case Default
!!!      Case(f_EHR, f_EDNR, f_BAOD, f_GHR, f_DNR, f_TSC, f_OSC, &
!!!          f_DBT, f_DPT, f_RH, f_SP, f_WD, f_WS, f_HV, f_CH, &
!!!          f_pH2O, f_SD, f_DSLS)
          loop_min = 1
          loop_max = Ubound(vf,1)
          loop_inc = 1

Case( &
      f_FAO_SG_PET,          & ! FAO Short Grass PET, mm/day
      f_Ep)                  ! Class A pan Evaporation, mm/day
      ! Loop only through daily values.
      loop_min = NHours
      loop_max = Ubound(vf,1)
      loop_inc = NHours

Case(f_OI) ! Observation Indicator
      ! Do nothing.
      Cycle Loop_Jpar

Case( f_KP_FWS_Evaporation) ! K-P FWS Evaporation, mm/day
      ! 8 Feb 2002 5:41 pm: until we develop a better formulation,
      ! Kill KP_FWS_Evaporation.
      ! Do nothing.
      Cycle Loop_Jpar

End Select

Write(ULog, 9130) Trim(FieldInfo(jpar)%Name)
9130 Format (/ , 1x, a)

npts = 0
n_missing = 0
n_nan = 0
npts = 0
n_1_24 = 0
n_25 = 0
n_out_of_range = 0
n_undefined = 0
n_Perpetual_Darkness = 0
minV = FieldInfo(jpar)%minimum_value
maxV = FieldInfo(jpar)%maximum_value

k = 0          ! Number of points.
M_k = Zero
Q_k = Zero
v_min = +Huge(Zero)

```

```

v_max = -Huge(Zero)

Loop_iv: Do iv = loop_min, loop_max, loop_inc

    npts = npts + 1
    erase_value = .False.

!!!! 19 Mar 2002 12:12 pm: This code is slower. I wonder if "Select Case"
!!!! is (in general) slower than IFs.
!!!!
!!!!     If (IsNaN(vf(iv)%v)) Then
!!!!         erase_value = .True.
!!!!         n_nan = n_nan + 1
!!!!         Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
!!!!         Write (ULog, 9150) Trim(FieldInfo(jpar)%Name), yyyy, mm, dd, hh
!!!!
!!!!     Else
!!!!         Select Case(vf(iv)%s)
!!!!         Case(T_Missing)
!!!!             erase_value = .True.
!!!!             n_missing = n_missing + 1
!!!!         Case(T_Not_Applicable)
!!!!             ! Do nothing.
!!!!         Case(T_Undefined)
!!!!             n_undefined = n_undefined + 1
!!!!         Case(T_Perpetual_Darkness)
!!!!             n_Perpetual_Darkness = n_Perpetual_Darkness + 1
!!!!
!!!!         Case Default
!!!!             in_range = ((minV <= vf(iv)%v) .And. (vf(iv)%v <= maxV))
!!!!             If (in_range) Then
!!!!                 ! We have a value
!!!!                 k = k + 1
!!!!                 x_k = VF(iv)%v
!!!!                 M_kml = M_k
!!!!                 M_k = M_kml + (x_k-M_kml)/Real(k)
!!!!                 Q_k = Q_k + (Real(k-1)/Real(k))*(x_k-M_kml)**2
!!!!                 v_min = Min(x_k, v_min)
!!!!                 v_max = Max(x_k, v_max)
!!!!                 !Call Stat_Add_Point(Xp_ranges(jpar), vf(iv)%v)
!!!!             Else
!!!!                 n_out_of_range = n_out_of_range + 1
!!!!                 erase_value = .True.
!!!!                 Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
!!!!                 Write (ULog, 9170) Trim(FieldInfo(jpar)%Name), yyyy, mm, dd, hh, vf(iv)%v
!!!!             End If
!!!!         End Select
!!!!     End If
!!!! End If

! 19 Mar 2002 12:10 pm: It is faster (by about 5 seconds)

```

```

! to perform all the checks with IFs statements, rather
! than breaking the if-sequence by using "Case Select(vf(iv)%s)"
! and then returning to IFs.

If (IsNaN(vf(iv)%v)) Then
  ! Produced, by example, by the operation Zero/Zero
  erase_value = .True.
  n_nan = n_nan + 1
  Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
  Write (ULog, 9150) Trim(FieldInfo(jpar)%Name), &
    iv, yyyy, mm, dd, hh
9150  Format(1x, 6x, a, ' NaN value for ', &
    i7, 1x, i4, '-', i2.2, '-', i2.2, i3, 'h')

Else If (vf(iv)%s == T_Missing) Then
  erase_value = .True.
  n_missing = n_missing + 1
  If (Modulo(iv,25) /= 0) Then
    n_1_24 = n_1_24 + 1
  Else
    n_25 = n_25 + 1
    Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
    vf(iv) = Val_and_Flag(T_Estimated, Zero, '')
    Write (ULog, 9160) Trim(FieldInfo(jpar)%Name), &
      iv, yyyy, mm, dd, hh
    !!9160  Format(1x, 6x, a, ' h25 missing for ', &
    !!      i7, 1x, i4, '-', i2.2, '-', i2.2, i3, 'h; value zeroed.')
  End If

Else If (vf(iv)%s == T_Not_Applicable) Then
  ! Do nothing.

Else If (vf(iv)%s == T_Undefined) Then
  n_undefined = n_undefined + 1

Else If (vf(iv)%s == T_Perpetual_Darkness) Then
  n_Perpetual_Darkness = n_Perpetual_Darkness + 1

Else If ((minV <= vf(iv)%v) .And. (vf(iv)%v <= maxV)) Then
  ! Value in range; we have a value.
  k = k + 1
  x_k = VF(iv)%v
  M_kml = M_k
  M_k = M_kml + (x_k-M_kml)/Real(k)
  Q_k = Q_k + (Real(k-1)/Real(k))*(x_k-M_kml)**2
  v_min = Min(x_k, v_min)
  v_max = Max(x_k, v_max)
  !Call Stat_Add_Point(Xp_ranges(jpar), vf(iv)%v)

Else

```

```

        n_out_of_range = n_out_of_range + 1
        erase_value = .True.
        Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
        Write (ULog, 9170) Trim(FieldInfo(jpar)%Name), &
            iv, yyyy, mm, dd, hh, vf(iv)%v
9170      Format(1x, 6x, a, ' Out-of-range value for ', &
            i7, 1x, i4, '-', i2.2, '-', i2.2, i3, 'h', 3x, lpg14.6)
    End If

    If (erase_value) Then
        vf(iv)%v = 0
        vf(iv)%s = T_Missing
        vf(iv)%f = ''
!!!      If (jpar == f_HP) Then
!!!          Call iv_to_ymdh(iv, yyyy, mm, dd, hh)
!!!          Write (ULog, 9130) Trim(FieldInfo(jpar)%Name), iv, yyyy, mm, dd, hh
!!!9130      Format(1x, 6x, a, ' missing value for ', &
!!!          i7, 1x, i4, '-', i2.2, '-', i2.2, i3, 'h')
!!!      End If
    End If
End Do Loop_iv

! Compute mean and variance
v_mean = M_k
If (k >= 2) Then
    v_variance = Q_k / Real(k-1)
    v_std_dev = Sqrt(v_variance)
Else
    ! Variance not defined for k<=1
    v_variance = Huge(Zero)
    v_std_dev = Huge(Zero)
End If

Write(ULog, 9190) k, v_mean, v_variance
9190      Format ( &
            1x, '   n ....: ', i0, '/', &
            1x, '   Mean .: ', lpg14.6, '/', &
            1x, '   Var ..: ', lpg14.6)
Write(ULog, 9210) v_std_dev
9210      Format (1x, '   StdDev: ', lpg14.6)

Write(ULog, 9230) v_min, v_max
9230      Format (1x, '   Range : ', 2(1x,lpg14.6))

If (n_missing > 0) Then
    Write (ULog, 9250) Trim(FieldInfo(jpar)%Name), &
        n_missing, npts
9250      Format (1x, 3x, '?? Missing values for ', a, &
            ' = ', i0, '/', i0)

```

```

        If (n_1_24 > 0) Write (ULog, 9270) n_1_24
        If (n_25 > 0) Write (ULog, 9290) n_25
9270      Format (1x, 3x, ' * 1h - 24h hours missing: ', i0)
9290      Format (1x, 3x, ' * Summary of the day (h25) missing: ', i0)
    End If

    If (n_nan > 0) Then
        Write (ULog, 9310) Trim(FieldInfo(jpar)%Name), &
            n_nan, npts
9310      Format (1x, 3x, '?? NaN values for ', a, ' = ', i0, '/', i0)
    End If

    If (n_out_of_range > 0) Then
        Write (ULog, 9330) Trim(FieldInfo(jpar)%Name), &
            n_out_of_range, npts
9330      Format (1x, 3x, '?? Out-of-range values for ', a, ' = ', i0, '/', i0)
    End If

    If (n_undefined > 0) Then
        Write (ULog, 9350) Trim(FieldInfo(jpar)%Name), &
            n_undefined, npts
9350      Format (1x, 3x, '?? Undefined values for ', a, ' = ', i0, '/', i0)
    End If

    If (n_Perpetual_Darkness > 0) Then
        Write (ULog, 9370) Trim(FieldInfo(jpar)%Name), &
            n_Perpetual_Darkness, npts
9370      Format (1x, 3x, '?? Perpetual Darkness values for ', a, ' = ', i0, '/', i0)
    End If

    End Do Loop_Jpar
End Subroutine Set_Missing_Flags

End Module Utils4

```

## Utils5

! Last change: LSR 4 Jun 2002 10:35 am

Module Utils5

Use Global\_Variables  
Implicit None

Contains

Subroutine Station\_With\_Missing\_Data(WBAN, Have\_Precip\_Data)

! See:

! <A HREF="e:\5\3met\Docs\samson\_format.txt#\$1">

Implicit None

Character(Len=\*), Intent(In) :: WBAN

Logical, Intent(Out) :: Have\_Precip\_Data

Select Case(WBAN)

Case Default

! Default:

! \* All stations have precipitation data

Have\_Precip\_Data = .True.

! STATIONS WITH LITTLE OR NO HOURLY PRECIPITATION DATA:

! See:

! <A HREF="samson\_format.txt#STATIONS WITH LITTLE OR NO HOURLY PRECIPITATION DATA">

```
Case( '04751', & ! Bradford PA
      '14850', & ! Traverse City MI
      '14940', & ! Mason City IA
      '14991', & ! Eau Claire WI
      '23048', & ! Tucumcari NM
      '23161', & ! Daggett CA
      '24013', & ! Minot ND
      '24025', & ! Pierre SD
      '24230', & ! Redmond OR
      '24283', & ! Arcata CA
      '24284', & ! North Bend OR
      '26415', & ! Big Delta AK
      '26425', & ! Gulkana AK
      '26528', & ! Talkeetna AK
      '26533', & ! Bettles AK
      '26615', & ! Bethel AK
      '26616', & ! Kotzebue AK
      '26617', & ! Nome, AK
      '27502', & ! Barrow AK
```

```

        '93129', &      ! Cedar City UT
        '93738', &      ! Washington-Dulles VA
        '93987', &      ! Lufkin TX
        '94725')        ! Massena NY
! = 26 Apr 2002  2:40 pm
! * make all hourly precipitation missing, even when
!   present. Fill daily value record (hh == 25) with
!   the summary of the day.
Have_Precip_Data = .False.

Case( '22516')        ! Kahului HI
! Special case. SAMSON identified this station
! "with little or no hourly ppt data" but we
! augmented the precipitation data with EarthInfo
! hourly and summary of the day data.
Have_Precip_Data = .True.

End Select

End Subroutine Station_With_Missing_Data

Subroutine Issue_Years(Some_Years_Missing, All_Years_Missing)

! The following stations have unfixable gaps in
! the precipitation record. We will not issue a year
! if the year's precipitation record is incomplete.
! <A HREF="e:\5\3met\Docs\samson_format.txt#$1">

! This routine should be called after all the data files
! have been read and before data processing starts. This
! routine consolidates the information gotten from the
! files read variable Year_Data(yyyy)%SAMSON_v10) with
! its internal information, e.g.,
!   Issue_This_Year(1965:1990) = .True.

! Outputs:
! -- See argument list
! -- Logical array "Issue_This_Year" (Module Global_Variables)

Implicit None
Logical, Intent(Out) :: Some_Years_Missing, All_Years_Missing

Integer :: yyyy

Select Case(pWBAN%WBAN)
Case Default
    Issue_This_Year = .True. ! Keep all years

```

```

Case( '23184')      ! Prescott, AZ
! Only the years 1961-1968 are complete.
! Too few years.
Issue_This_Year = .False.    ! Kill all years.

Case( '94814')      ! Houghton Lake, MI
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1965:1990) = .True.

Case( '03945')      ! Columbia, MO
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1970:1990) = .True.

Case( '03947')      ! Kansas City, MO
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1973:1990) = .True.

Case( '03812')      ! Asheville, NC
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1965:1990) = .True.

Case( '24013')      ! Minot, ND
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1961:1988) = .True.

!Case( '94918')      ! Omaha N WSFO, NE

Case( '23048')      ! Tucumcari, NM
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1963:1981) = .True.

Case( '94185')      ! Burns, OR
! Only the years 1983-1988 are complete.
! Too few years.
Issue_This_Year = .False.    ! Kill all years.

Case( '03870')      ! Greenville/Spartanburg, SC
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1963:1990) = .True.

!!! 4 Jun 2002 10:34 am -- precipitation files
!!! corrected. We have all years.
!Case( '03927')      ! Fort Worth, TX
! ! Only the years 1975-1990 are complete.
! ! Too few years.
! Issue_This_Year = .False.    ! Kill all years.

Case( '12960')      ! Houston, TX
Issue_This_Year = .False.    ! Kill all years.
Issue_This_Year(1970:1990) = .True.

```

```

Case( '94240')      ! Quillayute, WA
  Issue_This_Year = .False.    ! Kill all years.
  Issue_This_Year(1967:1990) = .True.

End Select

Do yyyy = MinYear, MaxYear
  If (Year_Data(yyyy)%SAMSON_v10 == 0) Then
    ! SAMSON input routines determined that this year
    ! is missing. Do not keep this year.
    Issue_This_Year(yyyy) = .False.
  End If
End Do

Some_Years_Missing = Any(.Not. Issue_This_Year)
All_Years_Missing = All(.Not. Issue_This_Year)

End Subroutine Issue_Years

Subroutine Str_years(Xstr)

! Output: a string suitable for:
! ## Years present: 1961-1990
! ## Years present: ** none **
! ## Years present: 1965-1988

Implicit None
Character(Len=*), Intent(Out) :: Xstr

Integer :: ymax, yyyy, tlen
Logical :: entry_ok, in_gap, n_issued

ymax = Ubound(Issue_This_Year,1) + 1
in_gap = .False.
n_issued = .False.
Xstr = ''
tlen = 1

Do yyyy = MinYear, ymax
  If (yyyy < ymax) Then
    entry_ok = Issue_This_Year(yyyy)
  Else
    ! yyyy == ymax
    ! The point is not missing -- it does not exist.
    entry_ok = .False.
  End If

  If (in_gap) Then

```

```

    If (entry_ok) Then
        ! Nothing to do.
    Else
        ! The gap run just ended.
        Write (Xstr(tlen:), '(i4)') yyyy-1
        tlen = Len_trim(Xstr) + 1
        in_gap = .False.
        n_issued = .True.
    End If
Else
    If (entry_ok) Then
        ! Start of a new gap
        Write (Xstr(tlen:), '(lx,i4,"-")') yyyy
        tlen = Len_trim(Xstr) + 1
        in_gap = .True.
    Else
        ! No missing value and no gap.
        ! Do nothing.
    End If
End If
End Do

    If (.Not. n_issued) Then
        Write (Xstr(tlen:), '(lx,a)') '** none **'
    End If
End Subroutine Str_years

End Module Utils5

```

Test Data for Laramie, Wyoming: average values for August 1987 (Burman and Pochop 1994:221)

Element	S.I.	English
Wind movement	261 km/day	163 mi/day
Solar radiation	5964 Wh/m <sup>2</sup> /day	514 Ly/day
Relative Humidity	43.7%	43.7%
Air temperature	15.2C	59.4F
Dew-point temperature	3.0C	37.4F
Elevation	2200m	7218 feet
Station pressure	78.1 kPa	23.04 inches Hg
Vapor pressure deficit	0.972 kPa	0.2869 inches Hg
Saturation vapor pressure	1.727 kPa	0.5095 inches of Hg
Actual vapor pressure	0.755 kPa	0.2227 inches of Hg
$\ddot{a}$	0.1110 kPa/°C	0.0182 inches Hg/°F
$\tilde{a}_p$ (for class A evap pan)	0.1225 kPa/°C	0.02007 inches Hg/°F
$\tilde{a}$	0.0519 kPa/°C	0.00846 inches Hg/°F
Daily pan evaporation	7.32 mm	0.29 inches
Daily FWS evaporation	0.06 mm	0.0025 inches

## SAMSON Station Locations

Weather Bureau Army Navy (WBAN) number, station location (City and State), geographic coordinates (latitude and longitude) and elevation (m M.S.L.) of the 234 stations available for coordinated climatological dataset for AgDisp, PRZM, and EXAMS (see ?). Primary stations (those with measured solar radiation data for at least one year) are in bold type; stations lacking hourly precipitation data are italicized.

WBAN	Station	State	Latitude	Longitude	Elevation (m)
03103	Flagstaff	AZ	35.1	-111.7	2135
03812	Asheville	NC	35.4	-82.5	661
03813	Macon	GA	32.7	-83.7	110
03820	Augusta	GA	33.4	-82.0	45
03822	<b>Savannah</b>	GA	32.1	-81.2	16
03856	Huntsville	AL	34.7	-86.8	190
03860	Huntington	WV	38.4	-82.6	255
03870	Greenville	SC	34.9	-82.2	296
03927	<b>Fort Worth</b>	TX	32.8	-97.1	164
03928	Wichita	KS	37.7	-97.4	408
03937	<b>Lake Charles</b>	LA	30.1	-93.2	3
03940	Jackson	MS	32.3	-90.1	101
03945	<b>Columbia</b>	MO	38.8	-92.2	270
03947	Kansas City	MO	39.3	-94.7	315
04725	Binghamton	NY	42.2	-76.0	499
04751	Bradford	PA	41.8	-78.6	600
11641	<b>San Juan</b>	PR	18.4	-66.0	19
12834	<b>Daytona Beach</b>	FL	29.2	-81.1	12
12836	Key West	FL	24.6	-81.8	1
12839	<b>Miami</b>	FL	25.8	-80.3	2
12842	Tampa	FL	28.0	-82.5	3

WBAN	Station	State	Latitude	Longitude	Elevation (m)
12844	West Palm Beach	FL	26.7	-80.1	6
12912	Victoria	TX	28.9	-96.9	32
12916	New Orleans	LA	30.0	-90.3	3
12917	Port Arthur	TX	30.0	-94.0	7
12919	<b>Brownsville</b>	TX	25.9	-97.4	6
12921	<b>San Antonio</b>	TX	29.5	-98.5	242
12924	Corpus Christi	TX	27.8	-97.5	13
12960	Houston	TX	30.0	-95.4	33
13722	<b>Raleigh</b>	NC	35.9	-78.8	134
13723	Greensboro	NC	36.1	-80.0	270
13729	Elkins	WV	38.9	-79.9	594
13733	Lynchburg	VA	37.3	-79.2	279
13737	Norfolk	VA	36.9	-76.2	9
13739	Philadelphia	PA	39.9	-75.3	9
13740	Richmond	VA	37.5	-77.3	50
13741	Roanoke	VA	37.3	-80.0	358
13748	Wilmington	NC	34.3	-77.9	9
13781	Wilmington	DE	39.7	-75.6	24
13865	Meridian	MS	32.3	-88.8	94
13866	Charleston	WV	38.4	-81.6	290
13873	Athens	GA	34.0	-83.3	244
13874	<b>Atlanta</b>	GA	33.7	-84.4	315
13876	Birmingham	AL	33.6	-86.8	192
13877	Bristol	TN	36.5	-82.4	459
13880	<b>Charleston</b>	SC	32.9	-80.0	12
13881	Charlotte	NC	35.2	-80.9	234
13882	Chattanooga	TN	35.0	-85.2	210
13883	Columbia	SC	34.0	-81.1	69
13889	Jacksonville	FL	30.5	-81.7	9
13891	Knoxville	TN	35.8	-84.0	299

WBAN	Station	State	Latitude	Longitude	Elevation (m)
13893	Memphis	TN	35.1	-90.0	87
13894	Mobile	AL	30.7	-88.3	67
13895	<b>Montgomery</b>	AL	32.3	-86.4	62
13897	<b>Nashville</b>	TN	36.1	-86.7	180
13957	Shreveport	LA	32.5	-93.8	79
13958	Austin	TX	30.3	-97.7	189
13959	Waco	TX	31.6	-97.2	155
13962	Abilene	TX	32.4	-99.7	534
13963	Little Rock	AR	34.7	-92.2	81
13964	Fort Smith	AR	35.3	-94.4	141
13966	Wichita Falls	TX	34.0	-98.5	314
13967	Oklahoma City	OK	35.4	-97.6	397
13968	Tulsa	OK	36.2	-95.9	206
13970	Baton Rouge	LA	30.5	-91.2	23
13985	<b>Dodge City</b>	KS	37.8	-100.0	787
13994	St. Louis	MO	38.8	-90.4	172
13995	Springfield	MO	37.2	-93.4	387
13996	Topeka	KS	39.1	-95.6	270
14607	<b>Caribou</b>	ME	46.9	-68.0	190
14733	Buffalo	NY	42.9	-78.7	215
14734	Newark	NJ	40.7	-74.2	9
14735	<b>Albany</b>	NY	42.8	-73.8	89
14737	Allentown	PA	40.7	-75.4	117
14739	<b>Boston</b>	MA	42.4	-71.0	5
14740	Hartford	CT	41.9	-72.7	55
14742	<b>Burlington</b>	VT	44.5	-73.2	104
14745	Concord	NH	43.2	-71.5	105
14751	Harrisburg	PA	40.2	-76.9	106
14764	Portland	ME	43.7	-70.3	19
14765	Providence	RI	41.7	-71.4	19

WBAN	Station	State	Latitude	Longitude	Elevation (m)
14768	Rochester	NY	43.1	-77.7	169
14771	Syracuse	NY	43.1	-76.1	124
14777	Wilkes-Barre	PA	41.3	-75.7	289
14778	<i>Williamsport</i>	PA	41.3	-77.1	243
14820	Cleveland	OH	41.4	-81.9	245
14821	Columbus	OH	40.0	-82.9	254
14826	Flint	MI	43.0	-83.7	233
14827	Fort Wayne	IN	41.0	-85.2	252
14836	Lansing	MI	42.8	-84.6	256
14837	<b>Madison</b>	WI	43.1	-89.3	262
14839	Milwaukee	WI	43.0	-87.9	211
14840	Muskegon	MI	43.2	-86.3	191
14842	Peoria	IL	40.7	-89.7	199
14847	Sault Ste. Marie	MI	46.5	-84.4	221
14848	South Bend	IN	41.7	-86.3	236
14850	<i>Traverse City</i>	MI	44.7	-85.6	192
14852	Youngstown	OH	41.3	-80.7	361
14860	Erie	PA	42.1	-80.2	225
14891	Mansfield	OH	40.8	-82.5	395
14895	Akron	OH	40.9	-81.4	377
14898	Green Bay	WI	44.5	-88.1	214
14913	Duluth	MN	46.8	-92.2	432
14914	Fargo	ND	46.9	-96.8	274
14918	International Falls	MN	48.6	-93.4	361
14920	La Crosse	WI	43.9	-91.3	205
14922	Minneapolis	MN	44.9	-93.2	255
14923	Moline	IL	41.5	-90.5	181
14925	Rochester	MN	43.9	-92.5	402
14926	Saint Cloud	MN	45.6	-94.1	313
14933	Des Moines	IA	41.5	-93.7	294

WBAN	Station	State	Latitude	Longitude	Elevation (m)
14935	Grand Island	NE	41.0	-98.3	566
14936	Huron	SD	44.4	-98.2	393
14940	<i>Mason City</i>	IA	43.2	-93.3	373
14941	Norfolk	NE	42.0	-97.4	471
14943	Sioux City	IA	42.4	-96.4	336
14944	Sioux Falls	SD	43.6	-96.7	435
14991	<i>Eau Claire</i>	WI	44.9	-91.5	273
21504	Hilo	HI	19.7	-155.1	11
22516	Kahului	HI	20.9	-156.4	15
22521	<b>Honolulu</b>	HI	21.3	-157.9	5
22536	Lihue	HI	22.0	-159.4	45
23023	<b>Midland/Odessa</b>	TX	31.9	-102.2	871
23034	San Angelo	TX	31.4	-100.5	582
23042	Lubbock	TX	33.7	-101.8	988
23044	<b>El Paso</b>	TX	31.8	-106.4	1194
23047	Amarillo	TX	35.2	-101.7	1098
23050	<b>Albuquerque</b>	NM	35.1	-106.6	1619
23061	<b>Alamosa</b>	CO	37.5	-105.9	2297
23065	Goodland	KS	39.4	-101.7	1124
23066	<b>Grand Junction</b>	CO	39.1	-108.5	1475
23129	Long Beach	CA	33.8	-118.2	17
23153	<i>Tonopah</i>	NV	38.1	-117.1	1653
23154	<b>Ely</b>	NV	39.3	-114.9	1906
23155	Bakersfield	CA	35.4	-119.1	150
23160	<b>Tucson</b>	AZ	32.1	-110.9	779
23161	<b>Daggett</b>	CA	34.9	-116.8	588
23169	<b>Las Vegas</b>	NV	36.1	-115.2	664
23174	<b>Los Angeles</b>	CA	33.9	-118.4	32
23183	<b>Phoenix</b>	AZ	33.4	-112.0	339
23184	<i>Prescott</i>	AZ	34.7	-112.4	1531

WBAN	Station	State	Latitude	Longitude	Elevation (m)
23185	Reno	NV	39.5	-119.8	1341
23188	<b>San Diego</b>	CA	32.7	-117.2	9
23232	Sacramento	CA	38.5	-121.5	8
23234	San Francisco	CA	37.6	-122.4	5
23273	<b>Santa Maria</b>	CA	34.9	-120.5	72
24011	<b>Bismarck</b>	ND	46.8	-100.8	502
24013	Minot	ND	48.3	-101.3	522
24018	Cheyenne	WY	41.2	-104.8	1872
24021	<b>Lander</b>	WY	42.8	-108.7	1696
24023	North Platte	NE	41.1	-100.7	849
24025	<i>Pierre</i>	SD	44.4	-100.3	526
24027	Rock Springs	WY	41.6	-109.1	2056
24028	Scottsbluff	NE	41.9	-103.6	1206
24029	Sheridan	WY	44.8	-107.0	1209
24033	Billings	MT	45.8	-108.5	1088
24089	Casper	WY	42.9	-106.5	1612
24090	Rapid City	SD	44.1	-103.1	966
24121	Elko	NV	40.8	-115.8	1547
24127	<b>Salt Lake City</b>	UT	40.8	-112.0	1288
24128	Winnemucca	NV	40.9	-117.8	1323
24131	<b>Boise</b>	ID	43.6	-116.2	874
24143	<b>Great Falls</b>	MT	47.5	-111.4	1116
24144	Helena	MT	46.6	-112.0	1188
24146	Kalispell	MT	48.3	-114.3	904
24153	Missoula	MT	46.9	-114.1	972
24155	Pendleton	OR	45.7	-118.9	456
24156	Pocatello	ID	42.9	-112.6	1365
24157	Spokane	WA	47.6	-117.5	721
24221	<b>Eugene</b>	OR	44.1	-123.2	109
24225	<b>Medford</b>	OR	42.4	-122.9	396

WBAN	Station	State	Latitude	Longitude	Elevation (m)
24227	Olympia	WA	47.0	-122.9	61
24229	<b>Portland</b>	OR	45.6	-122.6	12
24230	<b>Redmond/Bend</b>	OR	44.3	-121.2	940
24232	Salem	OR	44.9	-123.0	61
24233	<b>Seattle/Tacoma</b>	WA	47.5	-122.3	122
24243	Yakima	WA	46.6	-120.5	325
24283	<i>Arcata</i>	CA	41.0	-124.1	69
24284	North Bend	OR	43.4	-124.3	5
25308	Annette	AK	55.0	-131.6	34
25339	Yakutat	AK	59.5	-139.7	9
25501	Kodiak	AK	57.8	-152.3	34
25503	King Salmon	AK	58.7	-156.7	15
25624	Cold Bay	AK	55.2	-162.7	29
25713	St Paul Is.	AK	57.2	-170.2	7
26411	<b>Fairbanks</b>	AK	64.8	-147.9	138
26415	Big Delta	AK	64.0	-145.7	388
26425	Gulkana	AK	62.2	-145.5	481
26451	Anchorage	AK	61.2	-150.0	35
26510	Mcgrath	AK	63.0	-155.6	103
26528	Talkeetna	AK	62.3	-150.1	105
26533	<i>Bettles</i>	AK	66.9	-151.5	205
26615	<i>Bethel</i>	AK	60.8	-161.8	46
26616	<i>Kotzebue</i>	AK	66.9	-162.6	5
26617	Nome	AK	64.5	-165.4	7
27502	<i>Barrow</i>	AK	71.3	-156.8	4
41415	<b>Guam</b>	PI	13.6	-144.8	110
93037	Colorado Springs	CO	38.8	-104.7	1881
93058	Pueblo	CO	38.3	-104.5	1439
93129	<i>Cedar City</i>	UT	37.7	-113.1	1712
93193	<b>Fresno</b>	CA	36.8	-119.7	100

WBAN	Station	State	Latitude	Longitude	Elevation (m)
93721	Baltimore	MD	39.2	-76.7	47
93729	<b>Cape Hatteras</b>	NC	35.3	-75.6	2
93730	Atlantic City	NJ	39.5	-74.6	20
93738	<i>Sterling (Washington-Dulles Airpt.)</i>	VA	39.0	-77.5	82
93805	<b>Tallahassee/Apalachicola</b>	FL	30.4	-84.4	21
93814	Covington	KY	39.1	-84.7	271
93815	Dayton	OH	39.9	-84.2	306
93817	Evansville	IN	38.1	-87.5	118
93819	<b>Indianapolis</b>	IN	39.7	-86.3	246
93820	Lexington	KY	38.0	-84.6	301
93821	Louisville	KY	38.2	-85.7	149
93822	Springfield	IL	39.8	-89.7	187
93842	Columbus	GA	32.5	-85.0	136
93987	<i>Lufkin</i>	TX	31.2	-94.8	96
94008	Glasgow	MT	48.2	-106.6	700
94018/23062	<b>Boulder/Denver</b>	CO	39.8	-104.9	1610
94185	<b>Burns</b>	OR	43.6	-119.1	1271
94224	Astoria	OR	46.2	-123.9	7
94240	Quillayute	WA	48.0	-124.6	55
94702	Bridgeport	CT	41.2	-73.1	2
94725	<i>Massena</i>	NY	44.9	-74.9	63
94728/14732	<b>New York (LaGuardia Airpt.)</b>	NY	40.8	-73.9	11
94746	Worcester	MA	42.3	-71.9	301
94814	Houghton	MI	47.2	-88.5	329
94822	Rockford	IL	42.2	-89.1	221
94823	<b>Pittsburgh</b>	PA	40.5	-80.2	373
94830	Toledo	OH	41.6	-83.8	211
94846	Chicago	IL	41.8	-87.8	190
94847	Detroit	MI	42.4	-83.0	191
94849	Alpena	MI	45.1	-83.6	210

WBAN	Station	State	Latitude	Longitude	Elevation (m)
94860	Grand Rapids	MI	42.9	-85.5	245
94910	Waterloo	IA	42.6	-92.4	265
94918/14942	<b>Omaha</b>	NE	41.3	-95.9	298

## References

- Allen, R. G. 1996. Assessing integrity of weather data for reference evapotranspiration estimation. *Journal of Irrigation and Drainage Engineering* **122**:97-106.
- Allen, R. G. 2000. REF-ET: Reference Evapotranspiration Calculation Software for FAO and ASCE Standardized Equations, Version 2.0. University of Idaho, Kimberly.
- Allen, R. G., L. S. Pereira, D. Raes, and M. Smith. 1998. Crop evapotranspiration: Guidelines for computing crop water requirements. FAO Irrigation and Drainage Paper 56, Food and Agriculture Organization of the United Nations, Rome.
- ASCE. 1996. Hydrology Handbook (Manual No. 28), Second edition. American Society of Civil Engineers, New York.
- Brutsaert, W. 1982. Evaporation into the Atmosphere: Theory, History, and Applications. D. Reidel Publishing Co., Dordrecht, The Netherlands.
- Burman, R., and L. O. Pochop. 1994. Evaporation, Evapotranspiration and Climatic Data. Elsevier, Amsterdam.
- Burns, L. A. 2000. Exposure Analysis Modeling System (Exams): User Manual and System Documentation. EPA/600/R-00/081, U.S. Environmental Protection Agency, Office of Research and Development, National Exposure Research Laboratory, Research Triangle Park, North Carolina, USA.
- Carousel, R. F., J. C. Imhoff, P. R. Hummel, J. M. Cheplick, and A. S. Donigian, Jr. 2005. PRZM-3, A Model for Predicting Pesticide and Nitrogen Fate in the Crop Root and Unsaturated Soil Zones: Users Manual for Release 3.12.2. Athens, Georgia.
- EPA. 2000. Meteorological Monitoring Guidance for Regulatory Modeling Applications. EPA-454/R-99-005, U.S. Environmental Protection Agency, Office of Air Quality Planning and Standards, Research Triangle Park.
- Farnsworth, R. K., E. S. Thompson, and E. L. Peck. 1982. Evaporation Atlas for the Contiguous 48 United States. NOAA Technical Report NWS 33, Office of Hydrology, National Weather Service, Washington.
- Hanson, C. L., K. A. Cumming, D. A. Woolhiser, and C. W. Richardson. 1993. Program for Daily Weather Simulation. Water Resources Investigations Rep. 93-4018, U.S. Geological Survey, Denver.
- Hanson, C. L., K. A. Cumming, D. A. Woolhiser, and C. W. Richardson. 1994. Microcomputer Program for Daily Weather Simulation in the Contiguous United States. Agricultural Research Service ARS-114, U.S. Department of Agriculture, Boise.
- Harbeck, G. E., and F. W. Kennon. 1954. Lake Hefner Studies Technical Report. Professional Paper 269, U.S. Geological Survey, Washington.
- Harrison, L. P. 1963. Fundamental concepts and definitions relating to humidity. Pages 3-80 in A. Wexler, editor. Humidity and Moisture. Reinhold Publishing Company, New York.
- Johnson, G. L., C. L. Hanson, S. P. Hardegree, and E. B. Ballard. 1996. Stochastic weather simulation: Overview and analysis of two commonly used models. *Journal of Applied Meteorology* **35**:1878-1896.
- Kohler, M. A., T. J. Nordenson, and W. E. Fox. 1955. Evaporation from Pans and Lakes. Research Paper 38, U.S. Department of Commerce, Weather Bureau, Washington.
- Kohler, M. A., and L. H. Parmele. 1967. Generalized estimates of free-water evaporation. *Water*

- Resources Research **3**:997-1005.
- Lamoreux, W. W. 1962. Modern evaporation formulae adapted to computer use. *Monthly Weather Review* **90**:26-28.
- Merkel, W. H. 1988. Appendix A, Revision of Reservoir Operations Study Computer Program & User Manual. Technical Release 210-VI-TR-19A, U.S. Department of Agriculture, Soil Conservation Service, Washington.
- Nicks, A. D., and G. A. Gander. 1994. CLIGEN: A weather generator for climate inputs to water resource and other models. *in* Proceedings of the Fifth International Conference on Computers in Agriculture. American Society of Agricultural Engineers, Orlando.
- Penman, H. L. 1948. Natural evaporation from open water, bare soil and grass. *Proceedings of the Royal Society of London, Series A: Mathematical and Physical Sciences* **193**:120-145.
- Penman, H. L. 1963. *Vegetation and Hydrology*. Technical Communication 53, Commonwealth Agricultural Bureaux, Farnham Royal, Bucks, England.
- Richardson, C. W., and D. A. Wright. 1984. WGEN: A Model for Generating Daily Weather Variables. Agricultural Research Service ARS-8, U.S. Department of Agriculture, Washington.
- Semenov, M. A., and E. M. Barrow. 1997. Use of a stochastic weather generator in the development of climate change scenarios. *Climate Change* **35**:397-414.
- Semenov, M. A., R. J. Brooks, E. M. Barrow, and C. W. Richardson. 1998. Comparison of the WGEN and LARS-WG stochastic weather generators for diverse climates. *Climate Research* **10**:95-107.



United States  
Environmental Protection  
Agency

Office of Research  
and Development (8101R)  
Washington, DC 20460

Official Business  
Penalty for Private Use  
\$300

EPA 600/R-07/053  
May 2007  
[www.epa.gov](http://www.epa.gov)

Please make all necessary changes on the below label,  
detach or copy, and return to the address in the upper  
left-hand corner.

If you do not wish to receive these reports CHECK HERE

; detach, or copy this cover, and return to the address in  
the upper left-hand corner.

PRESORTED STANDARD  
POSTAGE & FEES PAID  
EPA  
PERMIT No. G-35



**Recycled/Recyclable**  
Printed with vegetable-based ink on  
paper that contains a minimum of  
50% post-consumer fiber content  
processed chlorine free