Food and Agriculture Organization of the United Nations

WaPOR Data Manual

Biomass production v2

Final Draft v2.2

eleaf FEED THE WORLD

vito

ITC

UNIVERSITY OF TWENTE.

WATERWATCH FOUNDATION

Preface

Achieving Food Security in the future while using water resources in a sustainable manner will be a major challenge for current and future generations. Increasing population, economic growth and climate change all add to increasing pressure on available resources. Agriculture is a key water user and careful monitoring of water productivity in agriculture and exploring opportunities to increase it is required. Improving water productivity often represents the most important avenue to cope with increased water demand in agriculture. Systematic monitoring of water productivity through the use of Remote Sensing techniques can help to identify water productivity gaps and evaluate appropriate solutions to close these gaps.

The FAO portal to monitor Water Productivity through Open access of Remotely sensed derived data (WaPOR) provides access to 10 years of continued observations over Africa and the Near East. The portal provides open access to various spatial data layers related to land and water use for agricultural production and allows for direct data queries, time series analyses, area statistics and data download of key variables to estimate water and land productivity gaps in irrigated and rain fed agriculture.

WaPOR Version 2 became available starting from June 2019. This manual explains the processing chain for the production of the biomass production data components distributed through WaPOR portal. It can be used in combination with the WaPOR Database methodology documents.

This version 2.2 document differs from version 2.1 for additional clarification on i) LST retrieval from Landsat products and ii) Proba-V / Sentinel 2 calibration.

Acknowledgements

FAO, in partnership with and with funding from the Government of the Netherlands, is developing a programme to monitor and improve the use of water in agricultural production. This document is part of the final output of the programme: the development of an operational methodology to develop an open-access database to monitor land and water productivity.

The methodology was developed by the FRAME[1] consortium, consisting of eLEAF, VITO, ITC, University of Twente and Waterwatch foundation, commissioned by and in partnership with the Land and Water Division of FAO.

Substantial contributions to the eventual methodology were provided during the first Methodology Review workshop, held in FAO Headquarters in October 2016 and during the second *beta* methodology review workshop, in January 2018. Participants in these workshops were: Henk Pelgrum, Karin Viergever, Maurits Voogt and Steven Wonink (eLEAF), Sergio Bogazzi, Amy Davidson, Jippe Hoogeveen, Michela Marinelli, Karl Morteo, Livia Peiser, Pasquale Steduto, Erik Van Ingen (FAO), Megan Blatchford, Chris Mannaerts, Sammy Muchiri Njuki, Hamideh Nouri, Zeng Yijan (ITC), Lisa-Maria Rebelo (IWMI), Job Kleijn (Ministry of Foreign Affairs, the Netherlands), Wim Bastiaanssen, Gonzalo Espinoza, Jonna Van Opstal (UNESCO-IHE), Herman Eerens, Sven Gilliams, Laurent Tits (VITO) and Koen Verberne (Waterwatch foundation).

[1]For more information regarding FRAME, contact eLEAF (http://www.eleaf.com/). Contact persons. FRAME project manager: Steven Wonink (steven.wonink@eleaf.com). Managing Director: Maurits Voogt (maurits.voogt@eleaf.com)

Contents

Abbreviations

AET	Actual Evapotranspiration
DMP	Dry Matter Production
ET	Evapotranspiration
FAPAR	Fraction Absorbed Photosynthetically Active Radiation
LST	Land Surface Temperature
LUE	Light Use efficiency
MODIS	MODerate-resolution Imaging Spectroradiometer
MOS	Maximum of Season
NDVI	Normalised Difference Vegetation Index
NIR	Near Infrared
NPP	Net Primary Production
NRT	Near Real Time
SMC	Soil Moisture Content
SMS	Soil Moisture Stress
TBP	Total Biomass Production
TIR	Thermal Infrared

1 Introduction

This document provides a detailed description of the processing chain applied for the production of the Biomass production data components distributed through the WaPOR portal. Whereas the WaPOR methodology document sets out the theory that underlies the applied methodology, this document provides details on the input data sources used at all levels and sets out the processing chain for the production of the biomass production data components NPP, AGBP and phenology.

The WaPOR methodology for producing biomass production data can therefore be found in the WaPOR methodology document, which should be used in conjunction with this data manual. To avoid confusion, in this data manual document we use the same terminology used in the methodology documents to refer to the different data components and dataset levels. Therefore 'Level 1' denotes the continental dataset at 250 m resolution, 'Level 2' denotes the national dataset at 100m resolution and 'Level 3' denotes the sub-national dataset at 30 m resolution. Furthermore, 'intermediate data components' refer to datasets that are created during pre-processing steps and which are used as input to the final processing of the biomass production data components. Whereas data components (such as NPP and AGBP) are available for download from WaPOR, intermediate data components are not available on WaPOR.

The WaPOR data components are mainly derived from freely available remote sensing satellite and other data sources and can be produced using open source software and tools. The WaPOR database manual specifies the input data sources and sets out how the processing is done.

Section 2 provides an overview of the processing chain for the biomass production data components, with detail on the different input data sources used at the 3 different levels given in section 3. The pre-processing steps applied for the production of intermediate data components that are used as input to the final processing is provided in section 4. Finally, section 5 gives a detailed description of the processing chain and functions used for the production of the biomass production data components.

2 Overview of the processing chain

2.1 NPP

The Net Primary Production component (NPP) is produced using the same processing chain at all resolution levels. The data are delivered on a dekadal basis at all levels.

The full processing chain starts with input data sources which can be either used directly as input to the processing chain, or during the pre-processing phase as input to produce intermediate data components that are in turn used as inputs to the processing chain. Figure 1 presents a flow chart that shows the different input datasets, both static external data and intermediate data components that are needed to produce NPP. Details of the input data sources can be found in section 3 and the pre-processing steps for producing intermediate data components are given in section 4. The final processing is described in detail in section 5.

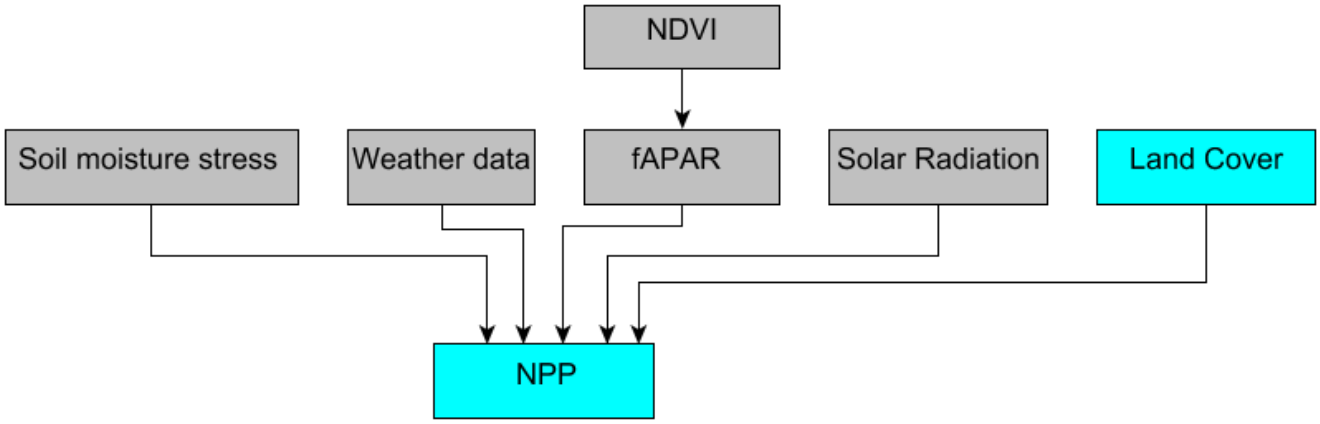

Figure 1: (Intermediate) data components that are used as input data for the production of the NPP data component. The grey boxes represent intermediate data components that convert external data into standardised input. Blue boxes represent data components that are distributed through WaPOR.

2.2 Phenology

The phenology data component is produced at level 2 and 3. It is provided on an annual basis, showing the start, maximum and end of up to two growing seasons per year. Figure 2 shows the input data components for the production of the phenology data. Details for the input data sources can be found in section 3 and the pre-processing steps for producing intermediate data components are given in section 4. The final processing is described in detail in section 5.

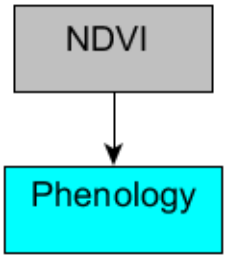

Figure 2: Intermediate data component used as input data for the production of the phenology data component. The grey box represents an intermediate data component that converts external data into standardised input. The blue box represents the phenology data component that is distributed through WaPOR.

2.3 TBP

The Total Biomass Production (TBP) represents the total biomass production in a given year or growing season, and is produced at all resolution levels, yet with a different methodology for Level 1. The data are delivered on an annual basis at Level 1 and on a seasonal basis at levels 2 and 3.

The processing chain starts with combining the information on the growing season with the dekadal NPP information. Figure 1 presents a flow chart that shows the different input datasets, both static external data and intermediate data components that are needed to produce NPP. Details of the input data sources can be found in section 3 and the pre-processing steps for producing intermediate data components are given in section 4. The final processing is described in detail in section 5.

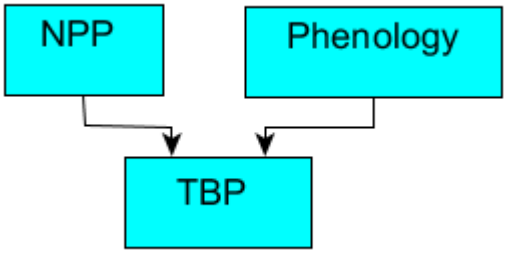

Figure 3: Data components that are used as input data for the production of the TBP data component. Blue boxes represent data components that are distributed through WaPOR.

3 Input data

The data required to calculate biomass production data components is explained in this paragraph. It is used to calculate a number of variables, part of the biomass production equations. These variables are shown in table 1. Three tables below specify the input data used for each level to derive these variables. Many input data sources are the same for each level. Therefore, the input data sources are explained once in separate paragraphs, starting with the sensor data in paragraph 3.1. The biomass production algorithm also uses model data, for example for weather data. These data source are discussed in paragraph 3.2. Finally, static data, such as land cover, is discussed in paragraph 3.3.

Table 1: variables required to calculate the biomass production data components

Term	Meaning	Unit	Range	Purpose	Temporal resolution
NDVI	**Normalized Difference Vegetation Index**	-	-1-1	Characterises vegetation condition to define start and end of a growing season	Dekadal
fAPAR	**Fraction absorbed photosynthetic active radiation**	-	0-1	Derive amount of incoming solar radiation absorbed by the vegetation	Dekadal
R_s	**Solar radiation**	Wm^{-2}	0-500	Incoming solar radiation	Daily
T_{min}	**Minimum air Temperature**	K	270-320		Daily
T_{max}	**Maximum air Temperature**	K	270-320		Daily
LUE	**Light use efficiency**	m		Land cover specific variable to calculate the energy conversion efficiency	Static
SMS	**Soil moisture Stress**	-	0-1	Reduction in biomass production due to limited water availability	Dekadal

Tables 2, 3 and 4 specify the input data sources used at levels 1, 2 and 3 respectively. The table also indicates whether the input data source is obtained from satellite sensor, model or static data sources.

Table 2: Input data sources for the production of biomass data components (NPP and TBP) at Level 1

Input data components	Type of input	Sensor	Data product	Comment
Weather data (temp)	Model		MERRA/GEOS-5	MERRA used prior to start of GEOS-5 (21-2-2014)
NDVI	Sensor	MODIS	MOD09GQ	
fAPAR	Sensor	MODIS	MOD09GQ	
Land Surface Temperature	Sensor	MODIS	MOD11A1, MYD11A1	Used to derive Soil Moisture Stress
Elevation, slope and aspect	Static		SRTM	Elevation, slope and aspect are derived from the DEM
Transmissivity	Model	MSG		Transmissivity is derived from MSG shortwave radiation products
Land Cover	Static		WaPOR L1 Land Cover Classification	

Table 3: Input data sources for the production of biomass data components (NPP, TBP and Phenology) at Level 2

Input data components	Type of input	Sensor	Data product	Comment
Weather data (temp)	Model		MERRA/GEOS-5	MERRA used prior to start of GEOS-5 (21-2-2014)
NDVI	Sensor	Proba-V	ProbaV_L3_S5_TOC_100m	
fAPAR	Sensor	Proba-V	ProbaV_L3_S5_TOC_100m	
Land Surface Temperature	Sensor	MODIS	MOD11A1, MYD11A1	Used to derive Soil Moisture Stress
Elevation, slope and aspect	Static		SRTM	Elevation, slope and aspect are derived from the DEM
Transmissivity	Model	MSG		Transmissivity is derived from MSG shortwave radiation products
Land Cover			WaPOR L2 Land Cover Classification	

Table 4: Input data sources for the production of biomass data components (NPP, TBP and Phenology) at Level 3

Input data components	Type of input	Sensor	Data product	Comment
Weather data (temp)	Model		MERRA/GEOS-5	MERRA used prior to start of GEOS-5 (21-2-2014)
NDVI	Sensor	Landsat 5 TM Landsat 7 ETM+ Landsat 8 OLI	L1TP	
fAPAR	Sensor	Landsat 5 TM Landsat 7 ETM+ Landsat 8 OLI	L1TP	
Land Surface Temperature	Sensor	Landsat 5 TM Landsat 7 ETM+ Landsat 8 OLI	L1TP	Used to derived Soil Moisture Stress
Elevation, slope and aspect			SRTM	Elevation, slope and aspect are derived from the DEM
Transmissivity	Model	MSG		Transmissivity is derived from MSG shortwave radiation products
Land Cover			WaPOR L3 Land Cover Classification	If L3 LC was not yet available, preliminary LC obtained from ESA CCI 20m LC (for 2016).

3.1 Sensor input data

3.1.1 MODIS TERRA

NASA's Moderate-resolution Imaging Spectroradiometer (MODIS) sensor on board the TERRA platform is selected as the primary remote sensing data source for optical data at 250m resolution. It covers the entire Earth on a near-daily basis, recording in 36 bands:

- At 250m (0.00223^o) resolution: 2 bands (RED/NIR).
- At 500m (0.00446^o) resolution: 5 bands in the shortwave range.
- At 1km (0.00892^o) resolution: 29 bands in the full spectrum (shortwave to TIR).

Having been active since early 2000, MODIS TERRA has generated a large archive with historical remote sensing data covering the whole FRAME project period. This makes it a stable and consistent data source for Level 1 data components.

Purpose of data

MODIS TERRA data is used for the production of the following data components:

- NDVI/fAPAR composites for Level 1
- Relative soil moisture content for Level 1 and 2 (based on the Thermal infrared bands at 1km)

Acquired data

A host of data products are generate by NASA on the basis of MODIS TERRA observations, including NDVI, fAPAR and NPP. However these derived products do not meet the required temporal and spatial resolution. Therefore basic data will be acquired to generate these products with the specifications needed for the WaPOR database. Specifics on the MODIS data acquired for the production of Land surface temperature at

Level 1 can be found in the Data Manual for Evapotranspiration. Additional MODIS data required for the production of biomass production data at Level 1 is as follows:

- **MxD09GQ**: Daily composites of TOC-reflectances in Red and NIR, plus some ancillary information, all at 250m resolution
- **MxD09GA**: These files contain two types of information:
    - At 1km resolution: The observation angles and the status information (clear, cloud, snow, land/sea, errors, etc.) used for quality control.
    - At 500m resolution: The reflectances in the 7 shortwave bands and some other information. These data are used for the calculation of the surface albedo, after resampling to 250m.
- **MxD11A1:** Land Surface Temperature and Emissivity (LST/E) products provide per-pixel temperature and emissivity values on a daily basis at 1 km spatial resolution. It also includes a quality rating, used for automated cloud-masking of the temperature data.

The above mentioned are all processing-level2 products, comprising of calibrated and atmospherically corrected images, requiring no additional pre-processing. The data is provided in the form of "tiles" of roughly 10°x10° mapped to the equal-area, sinusoidal projection. The files are in EOS-HDF5 format. An overview of the 7 bands is shown below.

MODIS Terra Band	Bandwidth (nm)	Resolution (m)
1	620 – 670	250
2	841 – 876	250
3	459 – 479	500
4	545 – 565	500
5	1230 – 1250	500
6	1628 – 1652	500
7	2105 – 2155	500

Source of the data

All MODIS data are freely available from the Land Processes Distributed Active Archive Center (LP-DAAC), located at USGS in Sioux Falls, South Dakota (https://lpdaac.usgs.gov/).

Challenges

MODIS data contains spatial artefacts when creating ten-daily NDVI-max composites, due to the influence of the viewing angle on the NDVI value, as well as the well-known 'bow-tie'-effect due to the arrangement of sensors on the MODIS instrument. Both effects can be reduced - but not removed - by imposing a threshold on the view zenith angle. Furthermore, its long lifespan means that it is expected to go out of service within a number of years. Fortunately, credible alternatives exist which can serve as back up.

Alternative sensors

In case of failure of the MODIS TERRA sensor, following back-ups are foreseen, listed here in order of importance:

- MODIS AQUA, with the same spectral, temporal and spatial characteristics, only with a different overpass time compared to the MODIS TERRA platform.

- Visible Infrared Imaging Radiometer Suite (VIIRS), Launched in 2011 as a successor to AVHRR and MODIS, with a spatial resolution of 375m. Major advantage is the spatial resolution of the thermal infrared bands.
- Sentinel 3A and B, launched in February 2016 and April 2018 respectively, will provide through the Ocean and Land Colour Instrument (OLCI) 300m data with a revisit time between one to two days.
- PROBA V 300m, which also provides daily global coverage with the required spectral bands to obtain the NDVI and derived products.

3.1.2 Proba-V

The Proba-V sensor is selected as the primary source for the 100m resolution optical data. It collects global imagery in four spectral bands (Blue, Red, NIR, SWIR) and in three resolutions:

- 1km (0.00892°) resolution with near-daily revisit time
- 300m (0.002976 °) resolution with near-daily revisit time
- 100m (0.0000992°) resolution with a revisit time of approximately 5 days.

The 100m resolution imagery of Proba-V will be used as primary source for the derivation of the different data components in Level 2. The system was launched in May 2013, but the 100m data only became available since March 2014.

Purpose of data

Proba-V data is used for the production of the following data components:

- NDVI/fAPAR composites for Level 2

Acquired data

Proba-V S5-TOC products will be used. These 5-daily composites are calibrated, atmospherically-corrected and mapped to Lon/Lat at resolution of 0.000992°. The product comprises of a number of data layers: 4 reflectances, 4 angles (sun/view zenith/azimuth), status map (indication of deviate observations: cloud, snow, water, error).
The format is either HDF4/5 or GeoTIFF, distributed in 10° x 10° tiles.

Acquisition of the data

VITO's subgroup CVB is responsible for the pre-processing (calibration, atmospheric correction, mapping to Geographic Lon/Lat) and the distribution of all the data via the portal http://www.vito-eodata.be.

Challenges

Proba-V only started delivering 100m resolution data in March 2014. Prior to this date no 100m resolution data is available. The gap in 100m resolution data for the historical data will be filled by resampling MODIS data. This will have effect on the data quality as some of the spatial variation available at 100m resolution is not available at 250m resolution.

Alternative sources

The Proba-V instrument (satellite and sensor) is still in good shape. Current operations will be continued until March 2020. After that period, no 100m alternative source is available, as is also the case with the historical data prior to March 2014. Resampled from a higher and lower resolution are therefore the only alternatives. The lower resolution MODIS data can be used in a similar way as is proposed for the historical data prior to March 2014. The higher resolution alternative will be the new Sentinel-2 mission, with a revisit time of 5 days, the same as the Proba-V 100m product (see 3.1.3). Landsat is not considered as a viable high resolution alternative, due to its low revisiting time.

3.1.3 Sentinel-2

The Sentinel 2 satellites are the secondary source for 100m optical data. As the Proba-V satellite will be decommissioned in April 2020, the 100m data products will be produced based on Sentinel 2. The switch from Proba-V to Sentinel-2 input data will start from the first dekad of 2020.

Sentinel-2 is a European wide-swath, high-resolution, multi-spectral imaging mission. Currently, two satellites are in orbit (Sentinel-2A and Sentinel-2B), flying in the same orbit but with a 180 degrees phase difference. The MSI sensor samples 13 spectral bands, with four bands at 10m, six bands at 20m and three bands at 60m spatial resolution. Sentinel-2A has been in orbit since 23 June 2015, and Sentinel-2B launched on 7 March 2017, together providing a five-daily global coverage.

Purpose of data

Sentinel-2 data is used for the production of the following data components:

- NDVI/fAPAR composites for Level 2

Acquired data

The Sentinel-2 data used is level-2A, representing Bottom-Of-Atmosphere reflectances in cartographic geometry. These images are thus already atmospherically corrected with the Sen2Cor atmospheric correction methodology. These data are provided in 100x100km^2 tiles in UTM/WGS84 projection.

Source of the data

The European Commission has deployed cloud-based platforms to provide centralised access to the Sentinel-2 data, as well as processing tools, i.e. the DIAS, or Data and Information Access Services. The current pre-processing and download-workflow have been deployed on the SOBLOO DIAS, but can easily be transferred to other cloud-based platforms in case issues with the SOBLOO DIAS occur.

Challenges

The operational use of Sentinel-2 at a continental scale comes with a few challenges. One is the large amount of data. A total of 1776 tiles need to be processed to cover the Level 2 area of WaPOR. With each tile approximately 800 MB large, and with approximetaly 2 images are available per dekad, a total of 1.42 TB of input data is generated per dekad. These data sizes will result in longer processing time, with a possible later delivery time compared to the Proba-V workflow.

In addition, many changes are still ongoing in the data availability and data delivery, both by the Sentinel hubs and the DIAS platforms. This is currently complicating the development of operational and NRT workflows for the Sentinel data. For example, at the time of writing this document, no official release terms were available, stating how soon acquired images would be available at either Level-1C or Level-2A (atmospherically corrected or not). This could severely impact the NRT data availability of Level 2 data in WaPOR.

Alternative sources

The Sentinel-2 satellites are the best currently available at the sub-100m resolution. No alternatives are currently foreseen. As Sentinel-2 is itself an alternative for the Proba-V sensor, we refer to the Proba-V alternatives for more information.

3.1.4 Landsat

The Landsat satellites are the primary source for 30m optical data. The Landsat program started in 1972 with the Landsat 1 satellite. Currently the 8th Landsat satellite (launched in 2013) is orbiting the Earth, carrying the Operational Land Imager (OLI) sensor. The cover the whole FRAME period, data from Landsat 5 (2009-2011) and Landsat 7 (2009-present) will also be used.

Purpose of data

Landsat data is used for the production of the following data components:

- NDVI/fAPAR composites for Level 3
- Land surface temperature at Level 3[2]
- Land Cover and Crop Classification for Level 3

Acquired data

All Landsat optical and thermal bands except the panchromatic band are used.

Source of the data

The data can be acquired from the National Satellite Land Remote Sensing Data Archive at the USGS EROS Centre (http://landsat.usgs.gov). Thanks to the Landsat Global Archive Consolidation[3] (LGAC) effort (since 2010), this centre holds the most geographically and temporally rich collection of Landsat data.

Challenges

Several problems exist with the Landsat data archive. Firstly, not all data is accessible, this varies by location of the Level 3 areas. Secondly, Landsat 7 developed a problem with its scan-line corrector, leading to reduced data quality, affecting Landsat 7 data from June 2003 onwards. Thirdly, a malfunction in the Landsat 5 TM sensor prompted reactivation of the MSS sensor, which made only limited acquisitions, affecting the period Nov 2011 – Jan 2013.

Alternative sources

Alternative sources of freely available, moderate resolution multi-spectral (in VNIR) satellite sensors exist that could be used as alternatives if significant data gaps in the Landsat archives occur for a Level 3 ROI. These are as follows:

- Sentinel-2A and 2B data are available from 2015 and 2017 respectively. Its VNIR bands are at 10m resolution and compare well with those of Landsat 8. The Sentinel-2 constellation can serve as back up for recent data acquisitions. Unfortunately, Sentinel-2 satellites do not have thermal bands.
- ASTER has VNIR data at 15 m resolution and thermal infrared bands at 90m resolution. It has a temporal extent from 2000 to present but data has only been acquired for areas that have been requested in the past. Spectrally, ASTER's Green, Red and NIR bands compare well with those of Landsat 7 ETM.
- CBERS (China Brazil Resource Satellite) has VNIR and thermal sensors (CCD, PANMUX, IRMSS) which are spectrally similar to Landsat. , CBERS-2B and CBERS 4 form potential alternative data sources within the period of interest. Limited data is freely available from INPE for Africa.
- The ALI sensor on board EO-1 was decommissioned on 30 March 2017, and could therefore be a possible additional source of historic data. The sensor mimics the Landsat ETM spatial and spectral resolution. Note that data does not exist in the archives for all the Level 3 areas.
- Archived SPOT data older than 5 years has been made freely available for research purposes by CNES. This data can also be used to fill in significant data gaps in historic datasets.

[2] For Landsat 5 and 7, the method described in Jiménez-Muñoz et al. (2009) is used and for Landsat 8, the method described in Rozenstein, et al. (2014) is used.

[3] https://www.usgs.gov/land-resources/nli/landsat/landsat-global-archive-consolidation?qt-science_support_page_related_con=3#qt-science_support_page_related_con

3.2 Model data

3.2.1 MSG

Atmospheric transmissivity determines the fraction of solar radiation that transverses through the atmosphere and reaches the Earth's surface. Cloud cover information is used to quantify the transmissivity of the atmosphere for shortwave solar radiation. The Meteosat Second Generation (MSG) geostationary satellite measures cloud cover with time steps of 15 minutes. Cleaned-up and geo-corrected cloud cover data is provided as part of the Cloud Physical Properties (CPP) algorithm development by the Koninklijk Nederlands Meteorologisch Instituut (KNMI) in the Netherlands. The down-welling surface fluxes provided by the algorithm are reliable inputs for calculating daily transmissivity.

Purpose of data

The MSG shortwave radiation product is used to calculate daily transmissivity which, in turn, is used to produce the solar radiation data component for all levels.

Acquired data

KNMI produces several products within the Cloud Physical Properties algorithm based on the MSG Spinning Enhanced Visible and InfraRed Imager (SEVIRI) sensor. One of these products is a shortwave radiation product in the form of 15-minute surface downwelling solar radiation ("sds"), in total 48 files per day. This product has a resolution of 4 km.

Acquisition of the data

Data can be obtained from EUMETSAT's Climate Monitoring Satellite Application Facility (CM-SAF).

Challenges

The coarse resolution of the data will have an effect on the estimation of solar radiation at higher levels. At the moment no higher resolution data is available for the project area.

Alternative sources

The GEOS-5 data assimilation system may deliver the same information but at a coarse resolution.

3.2.2 GEOS-5

The Goddard Earth Observing System Model, Version 5 (GEOS-5) uses the Earth System Modeling Framework (ESMF). The GEOS-5 Data Assimilation System (GEOS-5 DAS) integrates the GEOS-5 Atmospheric Global Climate Model (GEOS-5 AGCM) with the Gridpoint Statistical Interpolation (GSI) atmospheric analysis developed jointly with NOAA/NCEP/EMC.

Purpose of data

Air temperature, relative humidity and wind speed are derived from GOES-5. This data is used to produce NPP and relative soil moisture content for Level 1, 2 and 3.

Acquired data

Temperature at 2m height, specific humidity and wind speed in the east-west and north-south direction.

Acquistion of the data

The GEOS-5 systems are being developed by the Global Modeling and Assimilation Office (GMAO) to support NASA's earth science research. The data are available for download from the NASA ftp-site: ftp://gmao_ops@ftp.nccs.nasa.gov/fp/das/. Data is freely available with a time-step of 6 hours. The data is also available as OpenDap data on the http://opendap.nccs.nasa.gov/ server.

Challenges

The coarse resolution of the data makes it impossible to obtain location-specific meteo data. In mountainous regions corrections are made for elevation using a digital elevation model.

Alternative sources

These data can be obtained from MERRA for the period spanning 1979 to February 2016.

3.3 Static input data

3.3.1 Land Cover

Land cover is one of the data components available on WaPOR for Levels 1, 2 and 3. The legends distinguish different levels of detail, with seasonal agricultural land mapped at Level 2 and main crop types mapped for 2 seasons per year at Level 3 (see the WaPOR Methodology Documents for detail on the Land cover map legends).

Purpose of data

Land cover information is required to determine the land cover-specific light use efficiency, i.e. the efficiency by which the vegetation converts solar energy into biomass. The light use efficiency is explained in more detail in section 3.3.2.

Source

Land cover maps are produced at all 3 levels for WaPOR.

Alternative sources

Several alternative data sources exist. For example ESA CCI Land Cover (20m, for 2016) and GlobCover (300m, for 2009).

3.3.2 Light use efficiency

Light or radiation use efficiency denotes how well the vegetation can convert the energy from the absorbed sunlight into biomass, and is expressed in kg dry matter produced per GJ of PAR absorbed.

Biome-specific LUE values were estimated by parameterizing the Copernicus GDMP against flux tower GPP measurements. The land cover information applied for the flux towers is the IGBP (International Geosphere-Biosphere Programme) land cover class as provided by the flux tower investigator. Table 5 gives an overview of the optimized LUE values per IGBP land cover class.

Table 5: WaPOR land cover classes with parameterized Light Use Efficiency (LUE) values for all three levels

WaPOR land cover Level 1	*WaPOR land cover Level 2*	*WaPOR land cover Level 3*	*LUE optimized [kgDM/GJ$_{PA}$]*
Natural vegetation	-	-	1.93
-	Tree	Tree cover (closed)	2.23
		Tree cover (open)	2.23
	Tree cover: closed needleleaved, evergreen (>40%)		1.98
	Tree cover: closed broadleaved, evergreen (>40%)		2.56

	Tree cover: closed needleleaved, deciduous (>40%)		1.85
	Tree cover: closed broadleaved, deciduous (>40%)		1.99
	Tree cover: closed, mixed leaf type (broadleaved and needleleaved)		2.23
	Tree cover: closed unknown forest type		2.23
	Tree cover: open needleleaved, evergreen (15-40%)		1.98
	Tree cover: open broadleaved, evergreen (15-40%)		2.56
	Tree cover: open needleleaved, deciduous 15-40%)		1.57
	Tree cover: open broadleaved, deciduous 15-40%)		2.48
	Tree cover: open, mixed leaf type (broadleaved and needleleaved)		2.23
	Tree cover: open unknown forest type		2.23
-	Shrub	Shrubland	2.10
-	Grassland	Grassland	2.39
Built-up	Urban	Urban	1.42
Bare soil/sparse vegetation	Bare soil/sparse vegetation	Bare soil	1.42
Waterbodies	Waterbodies	Water	0
-	Wetlands	Wetlands	1.43
Cropland	Cropland	Other or unknown crop	2.70
-	-	Wheat	1.8
-	-	Maize	2.6
-	-	Potatoes	1.8
-	-	Vegetables	1.7
-	Fallow	Fallow	2.7
-	-	Orchard (closed)	1.8
-	-	Olive	1.8
-	-	Grapes	1.8
-	-	Orchard (open)	1.8
-	-	Rice	1.8
-	-	Sugar cane	2.6
-	-	Other perennials	1.8

Purpose of data

Light use efficiency is used as an input to convert the (potential) maximum NPP value to actual NPP.

Source

The LUE values for non-crop land covers have been derived from the *Copernicus Global Land Component – Dry Matter Productivity* algorithm theoretical basis document[4]. For the crop types, LUE values were derived from Villalobos & Fereres (2016)[5] who proposed for seasonal calculations some LUE intervals for "well-adapted crops under normal cropping conditions". Non-leguminous C3 plants were given the value 1.8 g/(MJ PAR) as the mean of the proposed interval 1.6-2.0 g/(MJ PAR). C4 plants were given the value 2.6 g/(MJ PAR) as the mean of the proposed interval 2.2-3.0 g/(MJ PAR). As vegetables are composed of leguminous and non-leguminous C3 plants, we have given the value 1.7 g/(MJ PAR) as mean of the two intervals 1.6-2.0 g/(MJ PAR) for non-leguminous C3 plants and 1.5-1.8 g/(MJ PAR) for leguminous C3 plants. When the LUE cannot be determined (for unknown crops, perennial crops and fallow), the default LUE value for crops at Level 2, i.e. 2.7 g/(MJ PAR) is used.

4 Pre-processing chain

Pre-processing is the phase in the production process where the acquired data is prepared for the algorithms that calculate the final data components. The input data undergoes one of two different actions:

- The input data is pre-processed to the correct format to be used directly in the processing chain, for example gridding of weather data or resampling of a lower resolution raster input data to a higher resolution, or
- The input data is used in several pre-processing steps to produce intermediate data components that will be used to calculate the final biomass production data components. An example is the production of NDVI and fAPAR which involves outlier detection, smoothing and gap-filling of the input data time series. This is described in Section 4.1.

Error! Reference source not found.-3 in section 2 show the input data components needed for the production f biomass production (NPP, TBP, Phenology). Since Land Cover is a data component that is available on WaPOR, the methods for their production are described separately in the WaPOR methodology documents.

The processing steps needed for the production of the different intermediate data components that are needed as input data during the processing chain for evapotranspiration are discussed in separate subsections below.

4.1 NDVI

[4] https://land.copernicus.eu/global/sites/cgls.vito.be/files/products/GIOGL1_ATBD_DMP_I2.00.pdf

[5] Villalobos, F. J., & Fereres, E. (Eds.). (2016). *Principles of agronomy for sustainable agriculture*. Springer.

The required input data for the derivation of the NDVI is the daily composites of TOC-reflectances in RED and NIR for all three levels, regardless of which sensor/platform in used. For Sentinel-2, a separate pre-processing step is included, to convert the original 10m data, projected in UTM, to the Proba-V 100m grid in Latitude/longitude projection. Reprojection and resampling are combined in one step, and performed directly on the required product (i.e. NDVI).

For Level 1 and 2, daily reflectance products are converted to dekadal NDVI images, using a constrained Max-NDVI compositing rule. The viewing angle is limited to be less than 35 degrees. This S10 (dekadal image) comprises the "best" observation extracted from the available S1 (daily) scenes. "Constrained" means that the flagged observations (e.g. clouds, snow) are not included in the selection.

The dekadal series of NDVI are still perturbed by undetected noise and missing values. Dekadal time series are smoothed based on Swets et al. (1999)[6], where first the unreliable observations (mostly local minima) are detected and then all missing or unreliable values are replaced by means of interpolation. The resulting images are completely filled with valid data. In the NDVI quality layer, an indication of the length of the filled gap is denoted, showing the reliability of the filled value.

NDVI at Level 3 is based on Landsat data. The Landsat data is first pre-processed in a number of steps shown in Figure 3.

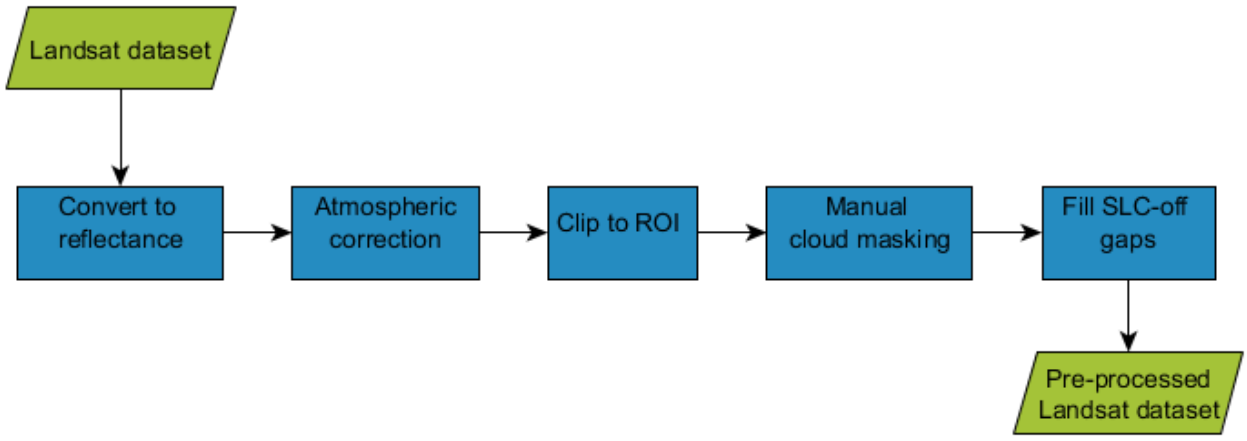

Figure 3: Flow chart showing the pre-processing steps that are applied to Landsat data at Level 3 for producing NDVI and Surface Albedo. Green parallelograms represent datasets and blue rectangles represent processing steps.

The pre-processing steps are as follows:

- Acquire Landsat data from the USGS as level 1 standard terrain corrected product.
- First, Landsat 5 and 7 data that are obtained in Digital Numbers are converted to radiance using the methodology prescribed by USGS[7]. This step is necessary in order to apply the atmospheric correction for Landsat 5 and 7, but is not necessary for Landsat 8.

[6]Swets, D.L, Reed, B.C., Rowland, J.D., Marko, S.E., 1999. A weighted least-squares approach to temporal NDVI smoothing. In: Proceedings of the 1999 ASPRS Annual Conference, Portland, Oregon, pp. 526-536.

[7] https://landsat.usgs.gov/landsat-7-data-users-handbook-section-5

- In order to remove the effects of varying atmospheric conditions across all Landsat images in the time series, atmospheric correction is done using SMAC (Simplified Model for Atmospheric Correction) after H. Rahman & G. Dedieu (1994)[8].
- The data is then clipped to the extent of the Level 3 ROI plus a buffer area. This ensures that the manual cloud masking step does not take unnecessarily long.
- Since there are no automatic cloud masking procedures available for Landsat data that consistently produce accurate results, cloud masking is carried out manually using the QGIS interface.
- All gaps in Landsat 7 data that are affected by the scan line corrector (SLC) error are filled using the method proposed by Chen et al (2011)[9].

Once the Landsat input data is pre-processed, the NDVI ratio is calculated using the Red and NIR bands of the Landsat data. The missing data in the NDVI time series (resulting from cloud masking) is then modelled using the gap-filling approach developed by Weiss, Daniel J., et al. (2014)[10]. This approach uses both spatial and temporal information within the Landsat-based NDVI time-series to fill the missing pixels. The noise in the filled NDVI time series is later smoothed out by applying the Savitzky–Golay filter method developed by Chen, Jin, et al. (2004)[11]. As a final step, dekadal NDVI composites are created from the smoothed NDVI inputs using a simple average method: a dekadal NDVI is calculated as the mean of the NDVI data falling within this dekad.

4.2 fAPAR

fAPAR at Level 1 and 2 is estimated using a direct relationship between the NDVI and a global fAPAR product, shown in the overview below. As smoothed and gap-filled NDVIs are used as input, no further processing of the fAPAR is needed.

- Level 1: $fAPAR = 3.9365NDVI^4 – 7.7984NDVI^3 + 5.6477NDVI^2 – 0.6931NDVI + 0.22$
- Level 2 (Proba-V): $fAPAR = 8.205NDVI^4 – 15.241NDVI^3 + 9.4734NDVI^2 – 1.1857NDVI + 0.0425$
- Level 2 (Sentinel2): $fAPAR = 7.110NDVI^4 – 12.436NDVI^3 + 7.055NDVI^2 – 0.387NDVI – 0.004$
- Level 3: $fAPAR = 8.205NDVI^4 – 15.241NDVI^3 + 9.4734NDVI^2 – 1.1857NDVI + 0.0425$

4.3 Solar radiation

Solar radiation is derived using MSG and the DEM. The MSG shortwave radiation product (see 4.2.1) is used to calculate daily transmissivity. The reason for not using the MSG shortwave radiation data directly is because of the limited resolution (> 3km) and of the lack of surface relief effects. The slope and aspect maps derived from the DEM are used to calculate the solar radiation for inclined surfaces, based on Tasumi, et al. (2006)[12].

[8] Rahman, H. and G. Dedieu, SMAC: a simplified method for the atmospheric correction of satellite measurements in the solar spectrum, International Journal of Remote Sensing 1994, 15(1), 123-143 , doi.org/10.1080/01431169408954055

[9] Chen, J., et al., "A simple and effective method for filling gaps in Landsat ETM+ SLC-off images" Remote Sensing of Environment 115 1053-1064 (2011)

[10] Weiss, D.J., Atkinson, P.M., Bhatt, S., Mappin, B., Hay, S.I. and Gething, P.W., “An effective approach for gap-filling continental scale remotely sensed time-series” ISPRS Journal of Photogrammetry and Remote Sensing 98:106-118 (2014). doi.org/10.1016/j.isprsjprs.2014.10.001.

[11] Chen, J., et al. "A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky-Golay filter" Remote Sensing of Environment, 91, 332-344 (2004).

[12] Tasumi, M. Allen, R. G, and R. Trezza. 2006. DEM based solar radiation estimation model for hydrological studies. Hydrological Science and Technology 22:197-208.

The transmissivity is used as a proxy to distinguish between direct and diffuse radiation. At lower transmissivity values a larger percentage of the solar radiation is considered diffuse. Diffuse solar radiation is estimated by ignoring the aspect and slope of the underlying terrain. At high transmissivity values the effects of the aspect and slope of the terrain are much more visible. At Level 3, the NPP is calculated using the actual Level 2 solar radiation as weather input. It is resampled at 30 meters using bilinear interpolation.

4.4 Weather data

Weather data, i.e. temperature, available on an hourly basis, are aggregated to an average daily value, resampled to the level resolution and extent. The temperature data are corrected for elevation using a Digital Elevation Model. The difference in elevation between a smoothed coarse scale DEM (>10 km) and a high resolution DEM (250 m) is used to upscale the coarse scale air temperature to a higher resolution. A default lapse rate of -6 °C km^{-1} is used to account for elevation differences. At Level 3, the NPP is calculated using the actual Level 2 minimum and maximum temperatures as weather input. They are both resampled at 30 meters using bilinear interpolation before use.

5 Description of biomass data component functions

This chapter contains the documentation for the algorithms used to implement the biomass data components, i.e. NPPmax, Soil moisture stress, NPP, phenology and TBP. The connection between the different functions will also be shown. The computation has been split up into different groups of functions, each of which calculates one or more intermediate and final results. The documentation will start at describing the final outputs. From these outputs the user may drill down into the different calculations. The core of the methodology has been detailed in Veroustraete et al. (2002)[13], whilst the practical implementation is described in Eerens et al. (2004)[14].

5.1 NPP max

First, based on the daily meteo data, NPP max is calculated on a daily basis (NPPmax1). For every dekad, the average NPPmax1 is calculated to derive NPPmax10 (mean composite). The code below shows the calculation for the NPPmax1.

```
#define   NclParms 20
short     Pmax(char **fil, long date, short Otyp, double ARint, double ARslo,
double PhotEff, short CO2year, short Screen);
void      Plut(short Otyp, double ARint, double ARslo, double PhotEff, double
CO2act, double Tc1, double Tc2, double TcDelt, float *LUTeff, float *LUTco2, float
*LUTar);

//*************************************************************************
                        void main( int argc, char **argv )
{
//*************************************************************************
char             s[HDRlmax+1], ss[10], m[2], stage[30],    pre[4][_MAX_PATH],
suf[4][_MAX_PATH], **fil;

short            yyyy,  yy,  mm,  dd,  dd_st,  ttm,  tty,  dpm,  doy;

short            i, j, Nd, Nv, period, DfMax, Dft[4], Otyp, CO2year, Screen;

long             l, date, date1, date2;

double           PhotEff, CO2act, ARint, ARslo;

double            progr40, progr40_step, GUIdone, GUIdone_step;


                                                          //============
                                                          // A. INITIAL
                                                          //============
if(argc==1) { bat=0; } else {   if(stricmp(argv[1], "GUI")) { bat = 1; } else {
bat = 2; ++argv; } }

strcpy(PROGNAME, "Pmax"); PROGVERSION = 1703; strcpy(ERRmess, "");
get_GLIMPSE_settings();

if(bat==1 && argc != (NclParms+1) && argc != (NclParms+2)) { if(argc==2) { bat =
0;} CLerror(1); }   //Only 1 parm => Interactive HELP
```

[13] Veroustraete, F., Sabbe, H., & Eerens, H. (2002). Estimation of carbon mass fluxes over Europe using the C-Fix model and Euroflux data. *Remote Sensing of Environment*, *83*(3), 376-399.

[14] Eerens, H., Piccard, I., Royer, A., & Orlandi, S. (2004). Methodology of the MARS crop yield forecasting system. Vol. 3: Remote sensing information, data processing and analysis. *Eds. Royer A. and Genovese G., EUR*, *21291*

```
if(bat==2 && argc != (NclParms+2) && argc != (NclParms+3)) {
CLerror(1); }
CLerror(0);

//Table fil[4][_MAX_PATH] holds names of I/O-IMGs for a given date (3=OUT).
//NB: dynamic allocation necessary because staticly allocated version can not be
transferred to function Pmax()
                    fil    = (char **)calloc(        4, sizeof(char *)); if(fil
==NULL){ sprintf(ERRmess, "Insufficient RAM for IMG-names"); ERROR(stage, ERRmess,
1); }
                    for(i=0;i<4;i++){ fil[i] = (char * )calloc(_MAX_PATH,
sizeof(char  )); if(fil[i]==NULL){ sprintf(ERRmess, "Insufficient RAM for IMG-
names"); ERROR(stage, ERRmess, 1); } }


                                                    //==========
                                                    // B. INPUT
                                                    //==========
strcpy(stage, "INPUT STAGE");

//------------------
// B1. GENERAL SPECS
//------------------
printf("    OUTPUT SPECIFICATIONS:\n");
printf(" p1.Type: 1=DMP, 2=NPP, 3=GDMP, 4=GPP  =>"); Otyp    = Qshort (1,    4,
0,  0, &argv);

if(Otyp < 3)
{
printf(" p2.A in AR = A + B.Tk   (def=-3.0490) =>"); ARint   = Qdouble(-1000,
1000, -3.049 , -9999, &argv);
printf(" p3.B in AR = A + B.Tk   (def=0.01145) =>"); ARslo   = Qdouble(-1000,
1000, 0.01145, -9999, &argv);
}
else { ARint = 0.0; ARslo = 0.0; if (bat) { ++argv; ++argv; }  }

printf(" p4.RUE [gDM/MJ Absorbed PAR,def=2.45] =>"); PhotEff = Qdouble(0,  100,
2.45, -1, &argv);
// printf(" p5.CO2 : 0=Variable, YYYY=Fix to year =>"); CO2year = Qshort (0, 2100,
1994, -1, &argv);
printf(" p5.CO2 : 0=Variable, YYYY=Fix to year =>"); CO2year = Qshort (0, 2100,
2010, -1, &argv);
if(CO2year > 0)
{
 //CO2act = 1.175 * CO2year - 1988;
// Veroustraete, 1994. For year 1994, this gives CO2 = 355.61.
   CO2act = 2.0775 * CO2year - 3785.783;
   printf("    Est. CO2-level [ppmv] for %4d     =>%g\n", CO2year, CO2act);
}
printf("\n");

//---------------------
// B2. TEMPORAL ASPECTS
//---------------------
printf("     TEMPORAL ASPECTS: Period & Range of Input-IMGs/dates to Consider\n");

   //Period (Time Increment)
   //"""""""
printf(" p6.Period: 1=Day, 10=dekad, 30=month  =>"); i = Qshort(1, 30, 0, 0,
&argv);
if(i!=1 && i!=10 && i!=30) { sprintf(ERRmess, "Periodicity (%d) beyond allowed
range [1,10,30]", i); ERROR(stage, ERRmess, 1); }
```

```
period = i;
switch(period)
{
   case  1: DfMax= 4; break;
   case 10: DfMax= 6; break;
   case 30: DfMax=10; break;
}

   //Fst Date
   //"""""""""
printf(" p7.First date in series    [YYYYMMDD] =>"); l = Qlong(19500101, 20491231,
0, 0, &argv);
date_test(l, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
//fatal=1 => exit(1) if error
switch(period)
{
   case   1: date1 = l                              ; break;
   case  10: date1 = 10000L*yyyy + 100*mm + dd_st; break;
//reset to start of dekad
   case  30: date1 = 10000L*yyyy + 100*mm + 1     ; break;
//reset to start of month
}
date_test(date1, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);

   //Lst Date
   //"""""""""
printf(" p8.Last  date in series    [YYYYMMDD] =>"); l = Qlong(date1, 20491231, 0,
0, &argv);
date_test(l, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
//fatal=1 => exit(1) if error
switch(period)
{
   case   1: date2 = l                              ; break;
   case  10: date2 = 10000L*yyyy + 100*mm + dd_st; break;
   case  30: date2 = 10000L*yyyy + 100*mm + 1     ; break;
}
date_test(date2, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
printf("\n");

//--------------
// B3. IMG NAMES
//--------------
printf("     1.YYYYMMDD     7.YYYYMM     YYYY = Year            [   1950  ...
2049]\n");
printf("     2.YYMMDD       8.YYMM         YY = Year            [50=1950  ...
49=2049]\n");
printf("     3.YYYYmDD      9.YYYYm        MM = Month  in Year  [01=Jan   ...
12=Dec ]\n");
printf("     4.YYmDD       10.YYm           m = Month  in Year  [ A=Jan   ...
L=Dec ]\n");
printf("     5.YYYYTT                      TT = Decade in Year  [01=First ...
36=Last]\n");
printf("     6.YYTT                        DD = Day    in Month [01=First ...
31=Last]\n");
printf("\n");

printf("    NAME IN-IMG(s) with SOLAR RADIATION [kJ/m2/d]: PrDrSr.img\n");
printf(" p9.Prefix      Pr                      =>");           Qstr(pre[0], "",
&argv);
printf("p10.Date-Format Dr [1-%2d]              =>", DfMax);   Dft[0] = Qshort(1,
DfMax, 0, 0, &argv) - 1;
```

```
printf("p11.Suffix      Sr                      =>");           Qstr(suf[0], "",
&argv);
printf("\n");

printf("    NAME IN-IMG(s) with MIN/MEAN TEMPERATURE [decigrade]: P1D1S1.img\n");
printf("p12.Prefix      P1                      =>");           Qstr(pre[1], "",
&argv);
printf("p13.Date-Format D1 [1-%2d]               =>", DfMax);   Dft[1] = Qshort(1,
DfMax, 0, 0, &argv) - 1;
printf("p14.Suffix      S1                      =>");           Qstr(suf[1], "",
&argv);
printf("\n");

printf("    NAME IN-IMG(s) with MAX TEMPERATURE [decigrade]: P2D2S2.img (OPTIONAL
!)\n");
printf("p15.Prefix      P2                      =>");           Qstr(pre[2], "",
&argv);
printf("p16.Date-Format D2 [1-%2d] (0=none!)     =>", DfMax);   Dft[2] = Qshort(0,
DfMax, 0, 0, &argv) - 1;
printf("p17.Suffix      S2                      =>");           Qstr(suf[2], "",
&argv);
printf("\n");

printf("    NAME OUT-IMG(s) with MAXIMUM DMP/NPP: PoDoSo.img\n");
printf("p18.Prefix      Po                      =>");           Qstr(pre[3], "",
&argv);
printf("p19.Date-Format Do [1-%2d]               =>", DfMax);   Dft[3] = Qshort(1,
DfMax, 0, 0, &argv) - 1;
printf("p20.Suffix      So                      =>");           Qstr(suf[3], "",
&argv);
printf("\n");

if(Dft[2] == -1) { strcpy(pre[2], pre[1]); Dft[2] = Dft[1]; strcpy(suf[2],
suf[1]); }        //CRUCIAL: Tmin = Tmax <= Tmean !!!

//------------------
// B4. Screen Output
//------------------
printf("p21.Screen OUT: 0=Short (def), 1=Long  =>");
if(argc == NclParms + bat)  { Screen = 0; printf("0\n"); } else { Screen =
Qshort(0, 1, 0, -1, &argv); }

                                                //==============
                                                //  C. PROCESS
                                                //==============
printf("\nNima YYYYMMDD STATUS\n");

Nd = 0;         //Tested dates/potential images
Nv = 0;         //Really available       images
date_test(date1, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
//Reset to Date1

while(1)
{
    //Next date: Find/Test/Show
   //"""""""""""""""""""""""""""
   date = 10000L*yyyy + 100*mm + dd;
    if(date > date2) { break; }
//EXIT WHILE-loop
   date_test(date, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
//fatal=1
   Nd++; printf("%4d %ld: ", Nd, date);
```

```
   //Full names (without extension) of 4 IMG's: 0=RAD, 1=Tmin, 2=Tmax, 3=OUT
   //""""""""""""""""""""""""""""""""""""""""""""
   for(i=0;i<4;i++) { img_date_name(pre[i], date, Dft[i], suf[i], fil[i]);  }
//fil[i] = IMGname without extension
   printf("Creating %s ... ", fil[3]);
//Stay on same screen line

   //Create OUT-IMG
   //"""""""""""""""
   j = bat; if(bat==0) { bat = 1; }
//TMP switch to batch mode => Existing OUT-IMGs will be overwritten, without
mercy.

   i = Pmax(fil, date, Otyp, ARint, ARslo, PhotEff, CO2year, Screen);
//MAJOR PROCEDURE: CREATES PRODmax-IMG FOR C

   if(i==0)
   {
      Nv++; if(Screen==0) { printf("OK\n"); }
//If error (i>0): Already handled by Pmax()
   }
   bat = j;

   //Adapt dd/mm/yyyy for next date
   //""""""""""""""""""""""""""""""""
   switch(period)
   {
      case   1: dd=dd+ 1; if(dd>dpm) { dd=1; mm++; } if(mm>12) { yyyy++; mm=1; }
break;
      case  10: dd=dd+10; if(dd>21 ) { dd=1; mm++; } if(mm>12) { yyyy++; mm=1; }
break;
      case  30:                                mm++;   if(mm>12) { yyyy++; mm=1; }
break;
   }
}
printf("\n");
printf("IMAGES/DATES: Total tested: %d\n", Nd);
printf("              Succeeded   : %d\n", Nv);

progression(2, 0, 0, 0, 0, "", "", "", &GUIdone, &GUIdone_step);
exit(0);
//***************************
}  //      END OF MAIN
//***************************


//*************************************************************************************
**********************************
 short Pmax(char **fil, long date, short Otyp, double ARint, double ARslo, double
PhotEff, short CO2year, short Screen) {
//*************************************************************************************
**********************************
//NB: bat > 0 (TMP reset by MAIN)

char              s[HDRlmax+1];
short             i, v, e, ok, TcShift, i12, i24, Ni, OFLAG, **b2;
long              rec, col, year;
float             x, fvi[3], T12, T24;
double            CO2act, Tc1, Tc2, TcDelt, fac;
float             *LUTeff, *LUTco2, *LUTar;                          //Allocated
here, filled by Plut()
```

```
double             progr40, progr40_step, GUIdone, GUIdone_step;

struct ENVIHDR    h[4];
FILE                 *fp[4];

//--------------
//1. TEST 4 IMGs
//--------------
for(v=0; v<3; v++)                                                    //3 IN-IMGs:
0=Radiation, 1=Tmin, 2=Tmax
{
   if(img_exist(fil[v], 1, 0, s))          { e=1; goto PmaxEND; }     //NB: s = fi
+ ".IMG"

   envi_hdr_read(s, &h[v], 1, 0, 1, 1, 4, 1, 1, 1);                   //everything
allowed
      if(h[v].date    == 0) { h[v].date = date;   }
    //if(h[v].days    == 0) { h[v].days = period; }

   if(h[v].bands   != 1)                  { e=2; goto PmaxEND; }
   if(h[v].bpp     != 2)                  { e=3; goto PmaxEND; }
   if(h[v].flip    != 0)                  { e=4; goto PmaxEND; }
   if(h[v].classes != 0)                  { e=5; goto PmaxEND; }
   if(map_diff(&h[0].mi, &h[v].mi) > 1)   { e=6; goto PmaxEND; }
   if(h[v].date    != date)               { e=7; goto PmaxEND; }
 //if(h[v].days    != period)             { e=8; goto PmaxEND; }     //1605:
Skipped!
   strcat(fil[v], ".img");                                            //fil[v] has
length MAX_PATH
}
img_exist(fil[3], 2, 0, s); strcat(fil[3], ".img");                   //bat > 0 =>
Existing OUT-IMG deleted!

//--------------------
//2. COMPUTE Pmax-IMG
//--------------------
//Compute 3 Lookup-Tables                                             //NB: LUTco2
can vary for all processed images/dates)
//"""""""""""""""""""""""""
Tc1 = -80; Tc2 = 60; TcDelt = 0.1;
//Temperatures in C: Range and step widt. NVER CHANGE TcDelt !!!
Ni = (short) ((Tc2 - Tc1)/TcDelt);                                    //1400
Integer temperature steps (excl. 0 !!!)
TcShift = (short) floor(0.5 - Tc1/TcDelt);                            //TcShift =
TMP-class of 0C. TcShift = INT(0.5 - (-80/0.1)) = INT(0.5 + 800) = 800

sprintf(ERRmess, "Insufficient RAM for Lookup-Tables");
LUTeff = (float *)calloc(Ni+1, sizeof(float)); if(LUTeff==NULL) {
ERROR("PROCESSING", ERRmess, 1); }
LUTco2 = (float *)calloc(Ni+1, sizeof(float)); if(LUTco2==NULL) {
ERROR("PROCESSING", ERRmess, 1); }
LUTar  = (float *)calloc(Ni+1, sizeof(float)); if(LUTar ==NULL) {
ERROR("PROCESSING", ERRmess, 1); }

   year = CO2year; if(CO2year == 0) { year = date / 10000; }          //year YYYY
to compute CO2-level (no variation over months)
// CO2act = 1.175  * year - 1988;
//Veroustraete, 1994. For year 1994, this gives CO2 = 355.61.
   CO2act = 2.0775 * year - 3785.783;

Plut(Otyp, ARint, ARslo, PhotEff, CO2act, Tc1, Tc2, TcDelt, LUTeff, LUTco2,
LUTar);    //Fill the three LUTs.
```

```
/*
   printf("\n");
   i=0   ; printf("i=%d, LUTeff=%g   LUTco2=%g   LUTar=%g\n", i, LUTeff[i],
LUTco2[i], LUTar[i]);
   i=800; printf("i=%d, LUTeff=%g   LUTco2=%g   LUTar=%g\n", i, LUTeff[i],
LUTco2[i], LUTar[i]);
   PressAnyKey(); exit(7);
*/

//Files & Buffers
//"""""""""""""""""
sprintf(ERRmess, "Insufficient RAM for IN-buffers");
b2 = (short **)calloc(3, sizeof(short *)); if(b2==NULL)
{ ERROR("PROCESSING", ERRmess, 1); }
for(v=0; v<3; v++)
{
      b2[v] = (short *)calloc(h[0].samples, 2); if(b2[v]==NULL)
{ ERROR("PROCESSING", ERRmess, 1); }
      if((fp[v]=fopen(fil[v],"rb"))==NULL) { sprintf(ERRmess, "Opening IN-IMG %s",
fil[v]); ERROR("PROCESSING", ERRmess, 1); }
      fseek(fp[v], h[v].offset, SEEK_SET);
}
v=3;  if((fp[v]=fopen(fil[v],"wb"))==NULL) { sprintf(ERRmess, "Opening OUT-IMG
%s", fil[v]); ERROR("PROCESSING", ERRmess, 1); }

//Process per Record
//""""""""""""""""""""
fac = 1.0; if(Otyp == 3) { fac = 2.0; }
//GDMP: divide V by 2.0 to avoid too much V > 32767
h[0].Vmin = 32767; h[0].Vmax = -32768;
//Output-extremes
OFLAG = -1;
//Single OUT-flag

if(Screen==1)
{
   printf("\n"); progression(0, h[0].lines, &progr40, &progr40_step, 1, "Creating
OUT-IMG", "records", "Processing", &GUIdone, &GUIdone_step);
}
for(rec=0; rec<h[0].lines; rec++)
{
   //Read record of 3 IN-files in buffers. NB: If Tmean, then twice the same!
   for(v=0; v<3; v++) { fread(b2[v], 2, h[0].samples, fp[v]); }

   //Process pixels in record
   for(col=0; col<h[0].samples; col++)
   {
      ok = 1;
      for(v=0; v<3; v++)
      {
         fvi[v] = (float)b2[v][col];
//Digital Nr.
         if(fvi[v] < h[v].Vlo || fvi[v] > h[v].Vhi) { ok = 0; b2[0][col] = OFLAG;
break; }
         fvi[v] = h[v].Vint + h[v].Vslo * fvi[v];
//Convert to Physical units: kJ/m/day or Decigrades Celsius (NEO: V1608)
      }
      if(ok == 0) { continue; }

      T24 = (fvi[1] + fvi[2]) / 2.;
//Mean TMP over day + night       (NB: 1=Tmin, 2=Tmax)
```

```
      T12 = 0.25 * fvi[1] + 0.75 * fvi[2];
//Mean TMP over sunshine period

      i24 = TcShift + (short)floor(0.5 + T24); if(i24 < 0) {i24 = 0;}   if(i24 >
Ni) {i24 = Ni;}   //Entries [0 - 1400] into the three LUTs
      i12 = TcShift + (short)floor(0.5 + T12); if(i12 < 0) {i12 = 0;}   if(i12 >
Ni) {i12 = Ni;}

      x = fvi[0] * LUTeff[i12] * LUTco2[i12] * LUTar[i24] / fac;
//NB: If GDMP/GPP: LUTar[i] = 1.0!
      x = max(0, x); x = min(32767, x);

      x = (short)floor(0.5 + x);
      b2[0][col] = (short) x;
      h[0].Vmin = min(x, h[0].Vmin); h[0].Vmax = max(x, h[0].Vmax);
   }

   //Write buffer to OUT-IMG
   fwrite(b2[0], 2, h[0].samples, fp[3]);
   if(Screen==1) { if((rec+1) >= (long)floor(progr40)) { progression(1,
h[0].lines, &progr40, &progr40_step, 1, "", "", "", &GUIdone, &GUIdone_step); } }
}

//Close & Free RAM
//"""""""""""""""""
_fcloseall();
for(v=0; v<3; v++) { free(b2[v]); } free(b2);
free(LUTeff); free(LUTco2); free(LUTar);
if(Screen==1) { printf("\n\n"); }

//-----------------
//3. CREATE OUT-HDR (georef + sensor from RAD-IMG, date/days adapted)
//-----------------
h[0].offset     = 0;
h[0].bands      = 1;
h[0].interleave = 1;       // BSQ
h[0].data_type  = 2;       // INTEGER
h[0].classes=0; h[0].cnames=NULL; h[0].colors=NULL;

strcpy(h[0].file_type, "ENVI Standard");

sprintf(h[0].program,"%s.exe (V%d/%d)", PROGNAME, PROGVERSION, LIBVERSION);

h[0].Vlo  = 0; h[0].Vhi  = 32767;   // significant DN-range (obs. range OK)
switch(Otyp)
{
   case 1:  sprintf(s, "Dry Matter Productivity DMP, Maximum for fAPAR=1.0,
RUE=%g, AutoResp=%g + %g*Tk", PhotEff, ARint, ARslo);
            strcpy(h[0].Vname,"DMPmax"); strcpy(h[0].Vunit, "kgDM/ha/day");
            h[0].Vint = 0; h[0].Vslo = 0.01;
// Y[kgDM/ha/day] = 0 + 0.01 * V  with V in [DgDM/ha/day] = [mgDM/m2/day]
            break;
   case 2:  sprintf(s, "Net Primary Productivity NPP, Maximum for fAPAR=1.0,
RUE=%g, AutoResp=%g + %g*Tk", PhotEff, ARint, ARslo);
            strcpy(h[0].Vname,"NPPmax"); strcpy(h[0].Vunit, "gC/m2/day");
            h[0].Vint = 0; h[0].Vslo = 0.001;
// Y[gC/m2/day] = 0 + 0.001 * V  with V in [mgC/m2/day]
            break;

   case 3:  sprintf(s, "Gross Dry Matter Productivity GDMP, Maximum for fAPAR=1.0,
RUE=%g", PhotEff);
            strcpy(h[0].Vname,"GDMPmax"); strcpy(h[0].Vunit, "kgDM/ha/day");
```

```
            h[0].Vint = 0; h[0].Vslo = 0.01 * fac;
// Y[kgDM/ha/day] = 0 + 0.02 * V / 2  with V in [DgDM/ha/day] = [mgDM/m2/day]
            break;
   case 4:  sprintf(s, "Gross Primary Productivity GPP, Maximum for fAPAR=1.0,
RUE=%g", PhotEff);
            strcpy(h[0].Vname,"GPPmax"); strcpy(h[0].Vunit, "gC/m2/day");
            h[0].Vint = 0; h[0].Vslo = 0.001;
// Y[gC/m2/day] = 0 + 0.001 * V  with V in [mgC/m2/day]
            break;
}
switch(CO2year)
{
   case  0: sprintf(h[0].description, "%s, CO2=%g ppmv (var~year)"   , s, CO2act
); break;
   default: sprintf(h[0].description, "%s, CO2=%g ppmv (fixed to %d)", s, CO2act,
CO2year); break;
}
if(stricmp(fil[1], fil[2])) { sprintf(h[0].comment, "RAD=%s, Tmin=%s, Tmax=%s",
fil[0], fil[1], fil[2]); }
else                        { sprintf(h[0].comment, "RAD=%s, Tmean=%s",
fil[0], fil[1]        ); }

sprintf(h[0].flags, "%d=Missing", OFLAG);

envi_hdr_create(fil[3], &h[0], IDRISI, ARCVIEW);
return 0;

//---------
//4. ERRORS    NB: All trapped initially, when testing the three requested IN-
IMGs!
//---------
PmaxEND:
switch(v)
{
   case 0: strcpy(s, "RAD-IMG" ); break;
   case 1: strcpy(s, "Tmin-IMG"); break;
   case 2: strcpy(s, "Tmax-IMG"); break;
}
switch(e)
{
   case 1: printf("%s: Not found\n"       , s); break;
   case 2: printf("%s: IMG 3D\n"          , s); break;
   case 3: printf("%s: Not INTEGER\n"     , s); break;
   case 4: printf("%s: Byte-flipped\n"    , s); break;
   case 5: printf("%s: Classes present\n" , s); break;
   case 6: printf("%s: Spatial error\n"   , s); break;
   case 7: printf("%s: Wrong date\n"      , s); break;
   case 8: printf("%s: Wrong period\n"    , s); break;
}
return e;
//****************************
}  //       END OF Pmax
//****************************

//*******************************************************************************
*****************************************************
 void Plut(short Otyp, double ARint, double ARslo, double PhotEff, double CO2act,
double Tc1, double Tc2, double TcDelt, float *LUTeff, float *LUTco2, float *LUTar)
{
/*******************************************************************************
*****************************************************
Adapted from Frank Veroustraete, Vito
```

```
- NB: Errors detected in previous version: NOW CORRECTED!
  . 2014-March: 2.54 has to be 2.45 (detected by Ozum).
  . 2010      : CO2 in GLIMPSE.SET holds for 1994 and not for 2000!
*****************************************************************************
*************************************************/
//---------------
//1. DECLARATIONS
//---------------
//Constants
//"""""""""""
double KELVIN   = 273.13   ;
double Rgas     = 8.3144   ;     // Universal gas constant, J/K/mol

                                 // For general Monteith-equation :
double ClimEff  = 0.48     ;     // kJ(PAR)/kJ(Totalshortwave), climatic
efficiency, McCree, 72
//double PhotEff = 2.54    ;     // Old ERROR: photosynthetic efficiency
(gDM/MJ(Absorbed PAR) = mgDM/kJ(APAR)) Wolfsy et al., 93
//double PhotEff = 2.45    ;     // That is the correct default value. But now
PhotEff is input parameter!
double gCgDM    = 0.45     ;     // gram carbon/gram DM acc (Atjay et al., 83).
Needed to convert DMP/GDMP to NPP/GPP.

                                 // For Normalized Temperature Dependency Factor
pT (LUTeff)
double C1       = 21.77    ;     // constant
double DHap     = 52750.   ;     // activation energy (J/mol)
double DHdp     = 211000.  ;     // deactivation energy (J/mol)
double Ds       = 704.98   ;     // entropy of CO2 denaturation equilibrium

                                          // FOR C02fert (LUTco2)
double O2       = 20.9     ;     // O2 concentration
double CO2ref   = 281.     ;     // CO2 mixing ratio for reference year 1833 (ppmv
= mol/mol)
double A1 = 2.419e+13;           // [%CO2]
double E1 = 59400.0  ;           // [J/mol] (Tavg ge 288.13)    CORRECTED (was
54900.0)
double A2 = 1.976e+22;           // [%CO2]
double E2 = 109600.0 ;           // [J/mol] (Tavg lt 288.13)
double A0 = 8240.    ;           // [%O2]
double E0 = 13913.5  ;           // [J/mol]
double At = 7.87e-5  ;
double Et =-42896.9  ;           // [J/mol]  - In RSE-publication : -42869.9

//General
//"""""""""
short   i, Ni;
double  x, y, z, constant, Tk, Km, K0, tau, A;

//Conversion ENERGY->MATTER
//Assumed Solar Radiation is in kJ(TOT)/m2/day [0-32767] (32767 can occur
exceptionally, but saturated to 32767)
//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                         constant = ClimEff * PhotEff;      // DMP/GDMP: 0-32767
mgDM/m/day = DgDM/ha/day (Dg=Decagram)
if(Otyp==2 || Otyp==4) { constant = constant * gCgDM; }     // NPP/GPP:  0-14745
mgC /m/day

//--------------
//2. CREATE LUTs
//--------------
```

```
Ni = (short) (1 + (Tc2 - Tc1)/TcDelt);          //1401 integer temperature steps
(incl. 0)
for(i=0; i<Ni; i++)                             //from Tc1 to Tc2 in steps of
TcDelt
{
    // Temperature-variants
   Tk = (i * TcDelt + Tc1) + KELVIN; x = Rgas * Tk;

   //LUTeff = pT = Normalized temperature dependency factor for photosynthetic
efficiency
    y  = C1 - DHap / x; z  = (Ds * Tk - DHdp) / x;
   LUTeff[i] = (float)(constant * exp(y) / (1 + exp(z)));

   //LUTco2 = CO2fert = Normalized CO2-fertilization factor
    if (Tk >= 288.13) { Km  = A1 * exp(-E1 / x); }
    else              { Km  = A2 * exp(-E2 / x); }
                       K0  = A0 * exp(-E0 / x);
                        tau = At * exp(-Et / x);
   y = O2/(2.*tau); z = Km * (1 + O2/K0);
    LUTco2[i] = (float)(((CO2act - y)/(CO2ref - y)) * ((z + CO2ref)/(z +
CO2act)));

   //LUTar = A = Fraction lost to Autotrophic Respiration (maintenance). NB: For
GDMP/GPP: ARint = ARslo = 0.0   => LUTar[i] = 1.0. Correct.
 //A = -3.049 + 0.01145 * Tk;    //Defaults
   A = ARint + ARslo * Tk;
   if(A < 0.) { A = 0.; }        //A cannot be smaller than zero (occurs below -
7C)
   if(A > 1.) { A = 1.; }        //A may not be greater than 1 (impossible for
normal T-range)
   LUTar[i] = (float)(1. - A);
}
//****************************
}  //      END OF Plut
//****************************
```

5.2 Soil moisture stress

The functions used for producing soil moisture stress are provided in the Data Manual for Evapotranspiration.

5.3 NPP

After the derivation of the maximum NPP, based on the meteorological data, the NPPmax10 is further scaled based on the factors limiting the biomass production. These consist of the fAPAR, Light use efficiency, autothropic respiration (set as a constant of 0.5) and the soil moisture stress. Combining these products with the NPPmax10 results in NPP, provided on a dekadal basis.

```
//*************************************************************************
                        void main( int argc, char **argv )
{
//*************************************************************************
```

```
char             s[HDRlmax + 1], str[HDRlmax + 1], stage[30], fi[3][_MAX_PATH],
fi_txt[_MAX_PATH], fo_img[_MAX_PATH], Oname[30];

unsigned char    c,                 *ba1, *bc1;

short            i, v, NDVI, Oflag, Otyp, k, Nk,  *bp2;

long             col, Ncol, rec, Nrec;

double            f, A, B, fAPAR, RUE, RUElut[256],    progr40, progr40_step,
GUIdone, GUIdone_step;

__int64          N64ok, N64[6];      //Counters for 5 OUT-Flags

struct ENVIHDR    hi[3], ho;

FILE             *fpi[3], *fpo, *fp;

                                     //============
                                     // A. INITIAL
                                     //============
if(argc==1) { bat=0; } else {   if(stricmp(argv[1], "GUI")) { bat = 1; } else {
bat = 2; ++argv; } }
strcpy(PROGNAME, "FRAME_Pact"); PROGVERSION = 1805; strcpy(ERRmess, "");
get_GLIMPSE_settings();
if(bat==1 && argc != (NclParms+1)) { if(argc==2) { bat = 0;} CLerror(1); }
//Only 1 parm => Interactive HELP
if(bat==2 && argc != (NclParms+2)) {                          CLerror(1); }
CLerror(0);
                                        //==========
                                        // B. INPUT
                                        //==========
strcpy(stage, "INPUT STAGE");
//-------------
//B1. fAPAR-IMG
//-------------
//IMG-Name
//"""""""""
v = 0;
printf("p1.IN-IMG1: fAPAR or NDVI          =>"); Qstr(s, "No IN-IMG1: fAPAR/NDVI",
&argv); img_exist(s, 1, 1, fi[v]);
envi_hdr_read(fi[v], &hi[v], 1, 0, 0, 1, 1, 1, 0, 0);                //BYTE,
offset OK

printf("   -Title  : %.66s\n"                                    ,
hi[v].description);
printf("   -MapSys : %s\n"                                  , hi[v].mi.name);
printf("   -Size   : %ld cols x %ld recs\n"                  , hi[v].samples,
hi[v].lines);
printf("   -Values : %s [%s]\n"                             , hi[v].Vname,
hi[v].Vunit);
printf("   -Scale  : %s [%s] = %g + %g * V [%g - %g]\n"   , hi[v].Vname,
hi[v].Vunit, hi[v].Vint, hi[v].Vslo, hi[v].Vlo, hi[v].Vhi);
printf("   -Flags  : %s\n"                                  , hi[v].flags);
printf("   -Date   : %ld\n"                                     , hi[v].date);
printf("   -Sensor : %.66s\n"                                      , hi[v].sensor);
printf("\n");

if(!strlen(hi[v].Vname)) { sprintf(ERRmess, "HDR of IN-IMG1 should contain section
VALUES={...}"); ERROR(stage, ERRmess, 1); }

//Conversion NDVI => fAPAR
```

```
//"""""""""""""""""""""""""
printf("   Conversion fAPAR [-] = A + B.NDVI [-]\n");
printf("p2.Intercept A        (def. = 0.0) =>"); A = Qdouble(-9999, -9999, 0.0, -
9999, &argv);
printf("p3.Slope     B        (def. = 1.0) =>"); B = Qdouble(-9999, -9999, 1.0, -
9999, &argv);
printf("\n");

NDVI = 0; if(A != 0.0 || B != 1.0) { NDVI = 1; }
hi[v].Vint = A + B * hi[v].Vint; hi[v].Vslo = B * hi[v].Vslo;
if(NDVI == 0)                                                    // If needed:
Convert fAPAR from [%] to [-].  NB: Don't do this if NDVI=1 !!!
{
   while(hi[v].Vint + hi[v].Vslo * hi[v].Vhi >  1.1) { hi[v].Vint = hi[v].Vint /
10.; hi[v].Vslo = hi[v].Vslo / 10.; }
}
//------------
//B2. Pmax-IMG
//------------
v = 1;
printf("p4.IN-IMG2: Pmax                    =>"); Qstr(s, "No IN-IMG2: Pmax",
&argv); img_exist(s, 1, 1, fi[v]);
envi_hdr_read(fi[v], &hi[v], 1, 0, 0, 2, 2, 1, 0, 0);                //SHORT,
offset OK

printf("   -Title  : %.66s\n"                                    ,
hi[v].description);
printf("   -MapSys : %s\n"                                  , hi[v].mi.name);
printf("   -Size   : %ld cols x %ld recs\n"                  , hi[v].samples,
hi[v].lines);
printf("   -Values : %s (%s)\n"                             , hi[v].Vname,
hi[v].Vunit);
printf("   -Scale  : Pmax = %g + %g * V [%g - %g]\n"        , hi[v].Vint, hi[v].Vslo,
hi[v].Vlo, hi[v].Vhi);
printf("   -Date   : %ld\n"                                      , hi[v].date);
printf("\n");

if(map_diff(&hi[0].mi, &hi[v].mi) > 1)    { sprintf(ERRmess, "IN-IMGs of
fAPAR/Pmax have different 'Map Info'");           ERROR(stage, ERRmess, 1); }
if(!strlen(hi[v].Vname))                  { sprintf(ERRmess, "Pmax-HDR should
contain section VALUES={...}");             ERROR(stage, ERRmess, 1); }

strtrim(hi[v].Vname, 2); strmid(hi[v].Vname, 1, 3, s); Otyp = 0;
if (strnicmp(s, "DMP", 3) == 0) { Otyp = 1; strcpy(Oname, "DMP" ); }
if (strnicmp(s, "NPP", 3) == 0) { Otyp = 2; strcpy(Oname, "NPP" ); }
if (strnicmp(s, "GDM", 3) == 0) { Otyp = 3; strcpy(Oname, "GDMP"); }
if (strnicmp(s, "GPP", 3) == 0) { Otyp = 4; strcpy(Oname, "GPP" ); }
if(Otyp == 0)                             { sprintf(ERRmess, "Vname in Pmax-HDR
should be DMPmax/NPPmax/GDMPmax/GPPmax");   ERROR(stage, ERRmess, 1); }

if(hi[v].date == 0)                       { hi[v].date = hi[0].date; }      //Both
can be zero!
if(hi[v].date != hi[0].date)              { sprintf(ERRmess, "IN-IMGs of
fAPAR/Pmax have different DATEs");                ERROR(stage, ERRmess, 1); }

printf("\n");

//------------------
//B3. Classification
//------------------
v = 2;
```

```
printf("p5.IN-IMG3: Classification          =>"); Qstr(s, "No IN-IMG3:
Classification", &argv); img_exist(s, 1, 1, fi[v]);
envi_hdr_read(fi[v], &hi[v], 1, 0, 0, 1, 1, 1, 0, 1);                  //BYTE,
offset OK, classes OK (but NOT mandatory at this stage!)

printf("   -Title   : %.66s\n"                                          ,
hi[v].description);
printf("   -MapSys  : %s\n"                                        , hi[v].mi.name);
printf("   -Size    : %ld cols x %ld recs\n"                        , hi[v].samples,
hi[v].lines);
printf("   -Values  : %s (%s)\n"                                    , hi[v].Vname,
hi[v].Vunit);
printf("   -Classes: %d (incl. 0 = OUT)\n"                             , hi[v].classes);
printf("\n");

if(map_diff(&hi[0].mi, &hi[v].mi) > 1)     { sprintf(ERRmess, "IN-IMGs of
fAPAR/Classification have different 'Map Info'");             ERROR(stage, ERRmess,
1); }

//----------------------------
//B4. TXT-FILE: Class_ID, RUE
//----------------------------
printf("p6.TXT-File (+ext.): LU-class, RUE =>"); Qstr(fi_txt, "No TXT-file with
RUE per class", &argv); file_exist(fi_txt, 1, 1);
if( (fp=fopen(fi_txt, "rt")) == NULL ) { sprintf(ERRmess, "Opening TXT-file: %s",
fi_txt); ERROR(stage, ERRmess, 1); }

for(k=0; k<256; k++) { RUElut[k] = -1.0; }      //Default: class k not present

Nk = 0; v = 0;
while (!feof(fp))
{
    str[0] = 0; fgets(str, HDRlmax, fp);
   if(strsplit(str, ",", 0, s) < 2)              { continue; }

   strsplit(str, ",", 1, s); if(strlen(s)==0)    { continue; }
   if(!ThisStringIsNumeric(s))                   { continue; }
   k = atoi(s);
   if(k < 0 || k > 255)                          { continue; }

   strsplit(str, ",", 2, s); if(strlen(s)==0)    { continue; }
   if(!ThisStringIsNumeric(s))                   { continue; }
   RUE = atof(s);
   if(RUE < 0.0 || RUE > 10.0)                   { sprintf(ERRmess, "Invalid RUE-
value (%g) for class_ID=%d in TXT-file: %s", RUE, k, fi_txt); ERROR(stage,
ERRmess, 1); }

   RUElut[k] = RUE; Nk++;                        //If RUElut[k]=0.0, then k=water
   if(RUE == 0.0) { v++; }
}
fclose(fp);
printf("   -Classes: %d (%d water classes)\n", Nk, v);
if(Nk==0)                                        { sprintf(ERRmess, "No valid
records found in TXT-file %s", fi_txt);                       ERROR(stage,
ERRmess, 1); }
printf("\n");

//--------------------
//B5. OUTPUT-IMG & HDR
//--------------------
    //OUT-IMG & Flag
    //"""""""""""""""
```

```
printf("p7.OUT-IMG: Pact                    =>"); Qstr(s, "No OUT-IMG specified",
&argv); img_exist(s, 2, 1, fo_img);
printf("\n");
printf("p8.OUT-Flag            (def=-9999) =>"); Oflag = Qshort(-32768, -1, -9999,
0, &argv);
printf("\n");

    //OUT-HDR: Define
    //""""""""
ho = hi[1];                                 // That is the Pmax IN-IMG (SHORT),
Values OK, except Vmin/Vmax

sprintf(ho.program,"%s.exe (V%d/%d)", PROGNAME, PROGVERSION, LIBVERSION);
ho.offset = 0;
strcpy(ho.sensor, hi[0].sensor);            // Sensor of fAPAR
strcpy(ho.Vname, Oname); ho.Vmin = ho.Vhi; ho.Vmax = ho.Vlo;

switch(Otyp)
{
   case 1:  sprintf(s, "Dry Matter Productivity DMP")       ; break;
   case 2:  sprintf(s, "Net Primary Productivity NPP")      ; break;
   case 3:  sprintf(s, "Gross Dry Matter Productivity GDMP"); break;
   case 4:  sprintf(s, "Gross Primary Productivity GPP")    ; break;
}
s[sizeof(ho.description) - 1] = 0; strcpy(ho.description, s);

if(NDVI == 0) { sprintf(s, "fAPAR=%s, Pmax=%s, LU=%s, TXT=%s", fi[0], fi[1],
fi[2], fi_txt); }
else          { sprintf(s, "NDVI=%s, fAPAR=%g + %g * NDVI, Pmax=%s, LU=%s,
TXT=%s", fi[0], A, B, fi[1], fi[2], fi_txt); }
s[sizeof(ho.comment) - 1] = 0; strcpy(ho.comment, s);

sprintf(ho.flags, "%d=nodata",  Oflag);

    //OUT-HDR: Show
    //""""""""
printf("   -Title  : %.66s\n"                                  , ho.description);
printf("   -MapSys : %s\n"                                  , ho.mi.name);
printf("   -Size   : %ld cols x %ld recs\n"                  , ho.samples, ho.lines);
printf("   -Values : %s (%s)\n"                             , ho.Vname, ho.Vunit);
printf("   -Scale  : Pact = %g + %g * V [%g - %g]\n"        , ho.Vint, ho.Vslo,
ho.Vlo, ho.Vhi);
printf("   -Flags  : %s\n"                                  , ho.flags);
printf("   -Date   : %ld\n"                                     , ho.date);
printf("   -Sensor : %.66s\n"                                      , ho.sensor);
printf("\n");


                                //=================
                                //  C. PROCESSING
                                //=================
//--------------------
// C1. Files & Buffers
//--------------------
strcpy(stage, "PREPROCESSING");

Ncol = ho.samples; Nrec = ho.lines;                 //That's easier

for(v=0; v<3; v++)                                  //0=fAPAR/NDVI, 1=Pmax, 2=LU-
map
{
```

```
      if((fpi[v]=fopen(fi[v],"rb"))==NULL) { sprintf(ERRmess, "Opening IN-IMG %s",
fi[v]); ERROR(stage, ERRmess, 1); } fseek(fpi[v], hi[v].offset, SEEK_SET);
}
      if((fpo=fopen(fo_img  ,"wb"))==NULL) { sprintf(ERRmess, "Opening OUT-IMG
%s", fo_img); ERROR(stage, ERRmess, 1); }

strcpy(ERRmess, "Insufficient RAM for Record-buffers");
ba1 = (unsigned char *)calloc(Ncol, 1); if(ba1==NULL)
{ ERROR(stage, ERRmess, 1); }  //fAPAR or NDVI
bp2 = (short         *)calloc(Ncol, 2); if(bp2==NULL)
{ ERROR(stage, ERRmess, 1); }  //Pmax & OUT-Pact
bc1 = (unsigned char *)calloc(Ncol, 1); if(bc1==NULL)
{ ERROR(stage, ERRmess, 1); }  //LU-class

//-------------------------
//C2. Process IMG per Record
//-------------------------
strcpy(stage, "PROCESSING");
progression(0, Nrec, &progr40, &progr40_step, 1, "Creating OUT-IMG", "records",
"Processing", &GUIdone, &GUIdone_step);
for(i=0;i<6;i++) { N64[i] = 0; }

                              //============================
                                for (rec=0; rec<Nrec; rec++) {
                              //============================
//Read IN-records
//"""""""""""""""
fread(ba1, 1, Ncol, fpi[0]);  //fAPAR/NDVI
fread(bp2, 2, Ncol, fpi[1]);  //Pmax
fread(bc1, 1, Ncol, fpi[2]);  //LU-class

//Process pixels
//"""""""""""""""
for(col=0; col<Ncol; col++)
{
   k = (short)bc1[col];
//k=LU-class
   if(RUElut[k] < 0.0)
   {
      _fcloseall(); unlink(fo_img); sprintf(ERRmess, "No RUE-value for class_ID=%d
in TXT-file: %s", k, fi_txt); ERROR(stage, ERRmess, 1);
   }
   if(RUElut[k] == 0.0)               { bp2[col] =   Oflag;         N64[2]++;
continue; }         //Water

   c = ba1[col];
//c=DN of fAPAR or NDVI
   if(c == 254)                       { bp2[col] =   Oflag;         N64[2]++;
continue; }         //Water

   f = (double)bp2[col];
//f=DN of Pmax
   if(f < hi[1].Vlo || f > hi[1].Vhi)  { bp2[col] =   Oflag;         N64[1]++;
continue; }         //OUT: No Meteo/Pmax

   if(c < hi[0].Vlo || c > hi[0].Vhi)  { bp2[col] =   Oflag;     N64[256-c]++;
continue; }         //Any UNIflag: 1 to 5 (2=water already trapped)

   fAPAR = hi[0].Vint + hi[0].Vslo * c;
//fAPAR now always as fraction [-]
   fAPAR = max(fAPAR, 0.0); fAPAR = min(fAPAR, 1.0);
//Certainly needed, especially when NDVI=1
```

```
// f     = hi[1].Vint + hi[1].Vslo * f;
//Keep Pmax in terms of digital V [0-32767] !

   f = f * RUElut[k] * fAPAR;
   f = max(0, f); f = min(32767, f);
//Keep in range [0 - 32767]
   i = (short)floor(0.5 + f);
//Convert to SHORT

   bp2[col] = i;
   ho.Vmin = min(i, ho.Vmin); ho.Vmax = max(i, ho.Vmax);
}

//Write record to OUT-IMG
//"""""""""""""""""""""""""
fwrite(bp2, 2, Ncol, fpo);
if ((rec+1) >= (long)floor(progr40)) { progression(1, Nrec, &progr40,
&progr40_step, 1, "Creating OUT-IMG", "records", "Processing", &GUIdone,
&GUIdone_step); }
                                //============================
                                        } // Next Rec
                                //============================
printf("\n"); printf("\n");
_fcloseall();

//---------------
//C3. Termination
//---------------
envi_hdr_create(fo_img, &ho, IDRISI, ARCVIEW);

N64ok = ho.pixels; for(i=1;i<6;i++) { N64ok = N64ok - N64[i]; }
f = (double) (100.0/ho.pixels);

   printf("Pixels: Total in IMGs      : %10I64d  (%7.3f%%)\n", ho.pixels,
f*ho.pixels  );
   printf("        Normally treated   : %10I64d  (%7.3f%%)\n", N64ok    , f*N64ok
);
   printf("        Water              : %10I64d  (%7.3f%%)\n", N64[2]   , f*N64[2]
);
   printf("        Missing Pmax/Meteo : %10I64d  (%7.3f%%)\n", N64[1]   , f*N64[1]
);
   printf("        Snow/Ice           : %10I64d  (%7.3f%%)\n", N64[3]   , f*N64[3]
);
   printf("        Cloud/Shadow       : %10I64d  (%7.3f%%)\n", N64[4]   , f*N64[4]
);
   printf("        Missing fAPAR      : %10I64d  (%7.3f%%)\n", N64[5]   , f*N64[5]
);

progression(2, 0, 0, 0, 0, "", "", "", &GUIdone, &GUIdone_step);
exit(0);
//***************************
}  //      END OF MAIN
//***************************
```

5.4 Phenology

To accumulate the total biomass produced over a certain area, it is key to detect the number of growing seasons, and define the start and end of each growing season. This information is used to from total annual biomass production (Level 1) to seasonal biomass production (Level 2 and 3). In this section the workflow is given to go from the dekadal NDVI time series to the number of growing seasons within a calendar year, as well as their start and end. Important to note is that if a growing season spans over different calendar years, it is assigned to the year where the end of the growing season occurs.

```
//***************************************************************************
                          void main( int argc, char **argv )                    {
//***************************************************************************
struct HIstruc
{
   char          *fi;                          //IMG-name, incl. extension
   long          offset;                       //BYTE-offset
   long          date;                         //date
   short         type;                         //1=present ACT-IMG, 2=present HIS-IMG
in tails, 0=lacking ACT-IMG in central gaps
}                *HI;                          //                  -1=lacking HIS-
IMGs (fatal error)

struct SPFparms                                //All settings in the SPF-file
{
   short          RMl, RMn, RMl2,                     Keep, dT2, dT5, dT6,
S2YM, S2YDt, Dek1, Dek2, S2YC, S2YDd,    o1, oYsm;
   double         RMw,       Edes, Eevg, Erg, Ecv,   dY1, dY2, Max3, Rat4,   Fsos,
Feos,   S2YT, S2YO , Ur, Uf,                   oMUl, oMUd, oRGl, oRGd;
}                FEN;

struct SEASON
{
   short          SOS,  MOS,  EOS,  L,    Keep;
   double         Ysos, Ymos, Yeos, Area;
}                *CYC, *SSN;

char                fo_suf[NvO][4]  = { "Ya0", "Yr0", "KK0",
// 00-02: Annual values (independent of seasons) : Y-mean, Y-range, Classification
                                        "s1",  "m1",  "e1",  "L1",  "s2",  "m2",
"e2",  "L2",    // 03-10: Dekads of SOS, MOS, EOS & Season length,  for 2 seasons
                                       "Ys1", "Ym1", "Ye1", "Ya1", "Ys2", "Ym2",
"Ye2", "Ya2",    // 11-18: Values at SOS, MOS, EOS & Season average, for 2 seasons
                                       "LT0", "LR0", "SI0" };
// 19-21: Annual values for both seasons together: length Tl1+Tl2, Tl1/(Tl1+Tl2),
Seasonality Index

char            s[HDRlmax+1], t[HDRlmax+1], m[2], stage[30], fi_VAR[_MAX_PATH],
fi_SPF[_MAX_PATH], fi_MSK[_MAX_PATH], **fi_img, **fi_cod,
                fi_pre[_MAX_PATH], fh_pre[_MAX_PATH], fo_pre[_MAX_PATH],
                fi_suf[_MAX_PATH], fh_suf[_MAX_PATH], **fo,
Emes[18][80];

unsigned char   U, **buf, *bm;
```

```
short            yyyy, yy, mm, dd, dd_st, ttm, tty, dpm, doy;

short            i, j, v, Nv, k, fi_df, fh_df, msk, M1, M2, Vm=0,   W, Ltot=0,   C,
Nc,   ss, Nss, NssOri,  e, Ee, Ne0, Ne1, E1, E2, T, Tmin, Tsos, Tmos, Teos, kk0,
*fi_usr;

short                yyyytar, NvACT, NvLTA, NvGAP, InGap, MaxInGap, NviMaxMis,
Nbad, vP, vAct1=0, vAct2=0, foM[NvO],    Un, Ut;

long             l, rec, col, date, Idate2, Ncol, Nrec,                  TEST,
TESTcol, TESTrec, TESTpix;

__int64           i64, Ns1=0, Ns2=0, NerrT=0,   N255=0, N254=0, N253=0, N252=0,
N251a=0, N251b=0, NE[7], Nerr[18];

double           x, NviMaxMisPerc,   sum, sumw,   A1, A2,   D, Dmin, Dmax;

double           *Yi, *Yj, *Yw, *Ys, **SGM, **SGM0, Y, Ymin, Ymax, Ymis, Ya0, Yr0,
Ycv, Ysos, Ymos, Yeos,           Uy;

double            f, progr40, progr40_step, GUIdone, GUIdone_step;

struct ENVIHDR   hi, hh, ho[NvO], *hi_VAR;

FILE               *fp, **fpi, **fpo, *fpe;

                                                                  //============
                                                                  // A. INITIAL
                                                                  //============
if(argc==1) { bat=0; } else {   if(stricmp(argv[1], "GUI")) { bat = 1; } else { bat
= 2; ++argv; } }
strcpy(PROGNAME, "PHENOdef18"); PROGVERSION = 1802; strcpy(ERRmess, "");
get_GLIMPSE_settings();
if(bat==1 && argc != (NclParms+1)) { if(argc==2) { bat = 0;} CLerror(1); }
if(bat==2 && argc != (NclParms+2)) {                        CLerror(1); }
CLerror(0);

TEST = 0; TESTcol = 1446; TESTrec = 33;         //Col/Rec here 1-based!  Skip if
TEST=0. If TEST=1: details of this pixel added to TXT-file with errors.

                                                                  //===============
                                                                  // B. INPUT STAGE
                                                                  //===============
strcpy(stage, "INPUT STAGE");

//----------------
// B1. TARGET YEAR
//----------------
   //Ask yyyytar
   //"""""""""""
printf(" p1.Target year [YYYY], 0=LTA            =>"); yyyytar = Qshort(0, 2050, -
1, -1, &argv);
if(yyyytar && (yyyytar<1950 || yyyytar>2050)) { sprintf(ERRmess, "Target year (%d)
must be 0 or in range 1950-2050", yyyytar); ERROR(stage, ERRmess, 1); }

   //Table with info on 108 IN-IMGs: fi, offset, date, type
   //""""""""""""""""""""""""""""""""
```

```
HI = (struct HIstruc *)calloc(NvI, sizeof(struct HIstruc)); if(HI == NULL) {
sprintf(ERRmess, "Insufficient RAM for IN-HDRs"); ERROR(stage, ERRmess, 1); }

//--------------------------------
// B2. DEFINE IN-IMGs via VAR-FILE
//--------------------------------
   //Ask fi_VAR      NB: If blank, use EXPLICIT method.
   //"""""""""""
printf(" p2.VAR-file with IN-IMGs (no ext. *.VAR) =>" ); Qstr(fi_VAR, "", &argv);
if(strlen(fi_VAR)) { strcat(fi_VAR, ".VAR"); file_exist(fi_VAR, 1, 1); } else { goto
EXPLICIT; }

   //Get names of VAR-IMGs in HI[Nv].fi, with Nv = 36/108.
   //"""""""""""""""""""""""""""""""""""""
strcpy(hh.mi.name, "");
// No geo-spatial comparison with other IMG
i = VARread(fi_VAR, 1, 1, 0, 0, 1, 0, &hh, 1, 1, &Nv, &fi_img, &fi_usr, &fi_cod,
&hi_VAR);   // Show=1, Only Byte, Offset OK, If error: FATAL
printf("\n");
if(Nv != 36 && Nv != 108) { sprintf(ERRmess, "VAR/MTA-file should contain 108 or 36
IN-IMGs. Here %d.", Nv); ERROR(stage, ERRmess, 1); }

hi = hi_VAR[0];
//Memorize first IN-HDRs for fixed mapinfo, scaling and significant range

for (v=0;v<Nv;v++)
{
   strcpy(s, fi_img[v]);
   HI[v].fi = (char *)calloc(1+strlen(s), 1); if(HI[v].fi == NULL) {
sprintf(ERRmess, "Insufficient RAM for IN-IMG names"); ERROR(stage, ERRmess, 1); }
   strcpy(HI[v].fi, s);
//Inclusive extension '.IMG' !!!
   HI[v].offset = hi_VAR[v].offset;
   free(fi_img[v]);
   free(fi_cod[v]);
}
free(fi_cod); free(fi_usr); free(fi_img); free(hi_VAR);

   //If Nv=36 (LTA): Expand to 108 dekads    Now from 0 to 107
   //'"""""""""""""""""""""""""""""""""""""
if(Nv == 36)
{
   for (v=0;v<36;v++) { HI[v + 36] = HI[v]; HI[v + 72] = HI[v];}
}

   //Set DATE & TYPE
//TYPE: 1=ACT-IMG, 2=HIS-IMG (absent IMGs are excluded here)
   //"""""""""""""""
if(yyyytar == 0)
//PHENO for LTA
{
    yyyy=1962; mm=1; dd=-9;
    for(v=0;v<NvI;v++)
    {
        dd = dd + 10; if(dd > 21) { dd = 1; mm = mm + 1; }  if(mm > 12) { mm = 1; }
//Stay on same year 1962!
        HI[v].date = 10000L * yyyy + 100L * mm + dd;
//For three years: 36 dekads in LTA-year "1962"
```

```
        HI[v].type = 2;
//That means: IN-IMG = LTA
    }
    NvACT = 0; NvLTA = NvI; NvGAP = 0;
}
else
//PHENO for specific year
{
   date = (yyyytar - 1) * 10000L +  101;
//Start with January 1 of Year1. NB: All later dekads will start with DD=01/11/21!
   date_test(date, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy); dd =
dd - 10;   //Reset date counters to start -10 days
   for(v=0;v<NvI;v++)
//NvI = 108 (3 years)
   {
      dd = dd + 10; if(dd > 21) { dd = 1; mm = mm + 1; }  if(mm > 12) { mm = 1; yyyy
= yyyy + 1; }
      HI[v].date = 10000L * yyyy + 100L * mm + dd;
      HI[v].type = 1;
//that means: This is an ACT-IMG
   }
   NvACT = NvI; NvLTA = 0; NvGAP = 0;
}

   //Skip EXPLICIT declaration of IN-IMGs
   //""""""""""""""""""""""""""""""""""""""
if (bat) { for(i=0;i<8;i++) {++argv;} }
//Skip p3-p10
goto SHOWinIMGs;

//-----------------------------------------------------
// B3. DEFINE IN-IMGs EXPLICITLY: Part 1 = ACTUAL IMGs
//-----------------------------------------------------
EXPLICIT:

   //If target year = LTA
   //"""""""""""""""""""""
if(yyyytar==0)
{
    if (bat) { for(i=0;i<5;i++) {++argv;} }
//Skip p3-p7
    yyyy=1962; mm=1; dd=-9;
    for(v=0;v<NvI;v++)
    {
        dd = dd + 10; if(dd > 21) { dd = 1; mm = mm + 1; }  if(mm > 12) { mm = 1; }
//Stay on same year 1962!
        HI[v].date = 10000L * yyyy + 100L * mm + dd;
//For three years: 36 dekads in LTA-year "1962"
        HI[v].type = -1;
//That means: IN-IMGs to be replaced with LTA!
    }
    NvLTA = NvI; MaxInGap = 0;
//No lacking LTA-IMGs allowed!
    goto LTAbegin;
}

   //For "real" target year (p1=YYYY)
   //"""""""""""""""""""""""
```

```
printf("\n");
printf("    NAMES OF ACTUAL IN-IMGs: PiDiSi.img\n");
printf(" p2.Prefix Pi  (add drive/path if needed) =>"); Qstr(fi_pre, "", &argv);
printf("    1.YYYYMMDD                  YYYY = Year           [   1950  ...
2049]\n");
printf("    2.YYMMDD                      YY = Year           [50=1950  ...
49=2049]\n");
printf("    3.YYYYmDD                     MM = Month in Year  [01=Jan   ... 12=Dec
]\n");
printf("    4.YYmDD                        m = Month in Year  [ A=Jan   ...  L=Dec
]\n");
printf("    5.YYYYTT                      TT = Dekad in Year  [01=First ...
36=Last]\n");
printf("    6.YYTT                        DD = Day   in Month [01=First ...
31=Last]\n");
printf(" p3.Date-Format Di (see above)      [1-6] =>"); fi_df = Qshort(1, 6, 0, 0,
&argv) - 1; //range [0-5]
printf(" p4.Suffix Si       (no extension: *.IMG) =>"); Qstr(fi_suf, "", &argv);

date = (yyyytar + 1) * 10000L +  1231;
//date=Max for Idate2
printf(" p5.Last ACT IMG  [YYYYMMDD / 0=%ld] =>", date); Idate2 = Qlong(0, date, 0,
-1, &argv);
printf("\n");
if(Idate2==0) { Idate2 = date; }
//Default: dekad 36 of third year
date_test(Idate2, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
//fatal=1 => exit(1) if error
if(Idate2 < (yyyytar * 10000L + 101)) { sprintf(ERRmess, "Last ACT IMG (%ld) must be
in target year or later (%ld-%ld)", Idate2, yyyytar, yyyytar+1); ERROR(stage,
ERRmess, 1); }

printf(" p6.Max. missing IMGs in centre   [def=0] =>"); MaxInGap = Qshort(0, 9, 0, -
1, &argv);

   //Check ACTUAL IMGs & Store results in HI
   //"""""""""""""""""""""""""""""""""""""""""""
date = (yyyytar - 1) * 10000L +  101;
//Start with January 1 of Year1. NB: All later dekads will start with DD=01/11/21!
date_test(date, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy); dd = dd
- 10;  //Reset date counters to start -10 days
NvACT = 0;
//Existing Actual IMGs found
for(v=0;v<NvI;v++)
//NvI = 108 (3 years)
{
    HI[v].type = 0;
//Default: This IN-IMG lacks

    //Define/store date
   dd = dd + 10; if(dd > 21) { dd = 1; mm = mm + 1; }  if(mm > 12) { mm = 1; yyyy =
yyyy + 1; }
   date = 10000L * yyyy + 100L * mm + dd;
   date_test(date, 1, &yyyy,  &yy,  &mm,  m, &dd,  &dd_st,  &ttm,  &tty,  &dpm,
&doy);
   HI[v].date = date;
//Date for ALL NvI IN-IMGs (all types)
```

```
    if(date > Idate2) { continue; }
//Rest will be filled with LTA. Don't use BREAK because HI[v].date must be completed
for all!

   //Find t=name of IN-IMG & Test if it exists
    img_date_name(fi_pre, date, fi_df, fi_suf, t);
//t = IN-IMG without extension
   if(img_exist(t, 1, 0, s)) { continue; }
//skip if IMG/HDR does not exist (typ remains 0)

   NvACT++;
//OK, IMG/HDR do exist & s=IN-IMG with extension '.IMG'
   envi_hdr_read(s, &hh, 1, 0, 0, 1, 4, 1, 0, 1);
//Allowed: 4 DTs (BYTE-test follows later), offset, classes. NOT: 3D, FLIP.

   if(hh.date < 19500100) { hh.date = date; }
//Date lacks in HDR =>accept & correct temporarily

   if(hh.date != date)
//Always OK if HDR does not contain Date
   {
        sprintf(ERRmess, "Wrong HDR-date (%ld instead of %ld) for IMG %s", hh.date,
date, s);   ERROR(stage, ERRmess, 1);
   }
   if(hh.data_type != 1)
   {
      sprintf(ERRmess, "Wrong datatype (%d - should be 1=BYTE) for IMG %s",
hh.data_type, s); ERROR(stage, ERRmess, 1);
   }

   if(NvACT==1) { hi = hh; }
//Memorize first IN-HDRs for fixed mapinfo, scaling and significant range
   if(map_diff(&hh.mi, &hi.mi) > 1)
   {
      sprintf(ERRmess, "IMG with deviate mapinfo: %s", s); ERROR(stage, ERRmess, 1);
   }
   if (hh.Vlo != hi.Vlo || hh.Vhi != hi.Vhi)
   {
      printf("     \nERROR: All IN-IMGs must have same significant range (Vlo-
Vhi)\n");
        printf("IMG of %ld: Vlo=%g, Vhi=%g\n", hi.date, hi.Vlo, hi.Vhi);
      printf("IMG of %ld: Vlo=%g, Vhi=%g\n", hh.date, hh.Vlo, hh.Vhi);
        sprintf(ERRmess, "IN-IMG with deviate significant range (Vlo-Vhi): %s", s);
ERROR(stage, ERRmess, 1);
   }
   if (hh.Vint != hi.Vint || hh.Vslo != hi.Vslo)
   {
        printf("     \nERROR: All IN-IMGs must have same scaling (Vint, Vslo)\n");
      printf("IMG of %ld: Vint=%g, Vslo=%g\n", hi.date, hi.Vint, hi.Vslo);
      printf("IMG of %ld: Vint=%g, Vslo=%g\n", hh.date, hh.Vint, hh.Vslo);
        sprintf(ERRmess, "IN-IMG with deviate scaling (Vint, Vslo): %s", s);
ERROR(stage, ERRmess, 1);
   }
   HI[v].type   = 1;
//OK: Actual IN-IMG present
   HI[v].offset = hh.offset;
```

```
   HI[v].fi = (char *)calloc(1+strlen(s), 1); if(HI[v].fi == NULL) {
sprintf(ERRmess, "Insufficient RAM for IN-IMG names"); ERROR(stage, ERRmess, 1); }
   strcpy(HI[v].fi, s);
//Inclusive extension '.IMG' !!!
}
printf("     Nr. of ACTUAL IN-IMGs found: %d\n", NvACT);
if(NvACT<36) { sprintf(ERRmess, "Insufficient actual IN-IMGs (%d) found (at least
36)", NvACT); ERROR(stage, ERRmess, 1); }

   //Reset TYPE=-1 for lacking IMGs in TAILS (1=OK, 0=missing IMG in centre)
    //Also find v-codes (vAct1, vAct2) of first/last ACT IMG in series
   //"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
NvLTA = 0;
//Lacking IN-IMGs in Head/Tail (to be replaced with LTA)
for(v=0;    v<NvI;v++) { if(HI[v].type==0) { HI[v].type=-1;  NvLTA++; } else { vAct1
= v; break; } }
for(v=NvI-1;v>=0 ;v--) { if(HI[v].type==0) { HI[v].type=-1;  NvLTA++; } else { vAct2
= v; break; } }

printf("     Dates first/last ACT IMGs  : %ld ... %ld\n", HI[vAct1].date,
HI[vAct2].date);
if(vAct2 < 36) { sprintf(ERRmess, "No ACT IMGs found for target year %d", yyyytar);
ERROR(stage, ERRmess, 1); }

   //Check on MaxInGap  Now, IMGs with type=0 are lacking in the profile centre
(NOAA!!!)
    //""""""""""""""""""
NvGAP = 0; InGap = 0;                                    //NvGAP=Total lacking IMGs
in centre (those with type=0). InGap=Max.gap found.
i = HI[0].type; k = 0;                                   //i=type of previous IMG,
k=number of IMGs in current gap
for(v=0;v<NvI;v++)
{
   j = HI[v].type;                                       //type of current IMG
(1=OK-ACT, 0=lacking in centre, -1=Replace by LTA)
   if(i==0 && j==0) { k++; NvGAP++; }                    //existing GAP continues
   if(i==1 && j==0) { k=1; NvGAP++; }                    //start of new GAP
   if(i==0 && j!=0) { InGap = max(InGap, k); k = 0;}     //end of GAP => Evaluate
   i = j;
}
InGap = max(InGap, k);                                   //evaluation again needed
at the end
printf("     IN-IMGs in central gaps    : %d\n", NvGAP);
printf("     Widest gap                 : %d dekads\n", InGap);
if(InGap > MaxInGap) { sprintf(ERRmess, "Gap of lacking IMGs (%d) wider than allowed
(%d)", InGap, MaxInGap); ERROR(stage, ERRmess, 1); }

//-------------------------------------------------------------
// B4. DEFINE IN-IMGs EXPLICITLY: Part 2 = Historical LTA IMGs        (Here no gaps
allowed, that is tested at the end!)
//-------------------------------------------------------------
LTAbegin:
   //If yyyytar=YYYY but no lacking IN-IMGs in both tails
   //"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
if(NvLTA == 0) { if (bat) { for(i=0;i<3;i++) {++argv;} } goto LTAend; }    //Skip
p7-p9

   //If ACT-IMGs are lacking in both tails. Or always if yyyytar=0.
```

```
   //"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
printf("\n");
printf("    NAMES OF HISTORICAL IN-IMGs with LTA (year 1962): PhDhSh.img \n");
printf(" p7.Prefix Ph  (add drive/path if needed) =>"); Qstr(fh_pre, "", &argv);
printf(" p8.Date-Format Dh (see above)      [1-6] =>"); fh_df = Qshort(1, 6,
fi_df+1, 0, &argv) - 1; //range [0-5]
printf(" p9.Suffix Sh       (no extension: *.IMG) =>"); Qstr(fh_suf, "", &argv);

   //Check LTA-IMGs
   //"""""""""""""""
j = 0; if(yyyytar==0) { j = 1; }                                  //j=1 means hi still
undefined. So impossible to compare hh <> hi.
for(v=0;v<NvI;v++)
{
   if(HI[v].type != -1) { continue; }                        //Only substitute
lacking IN-IMGs in tails (or ALL if yyyytar=0)!

   date = HI[v].date;                                     //date has already been
defined before!
   date_test(date, 1, &yyyy,  &yy,  &mm,  m, &dd,  &dd_st,  &ttm,  &tty,  &dpm,
&doy);
   yyyy = 1962;
    l = 10000L * yyyy + 100L * mm + dd_st;              //l = the real date of LTA-
IMG

   img_date_name(fh_pre, l, fh_df, fh_suf, t);          //t = IN-IMG without
extension

   if(img_exist(t, 1, 0, s)) { continue; }             //FATAL but Message comes
later. TYPE remains -1.

    envi_hdr_read(s, &hh, 1, 0, 0, 1, 4, 1, 0, 1);     //Allowed: 4 DTs (BYTE-test
follows below), offset, classes,. NOT: 3D, FLIP.

    if(j) { j=0; hi = hh; }                                     //From here onwards:
hi defined

   if(hh.date < 19500100) { hh.date = l; }             //HDR-date lacks => accept OK
& correct temporarily

    if(hh.date != l)                                                //Always OK if
HDR does not contain date
    {
        sprintf(ERRmess, "Wrong HDR-date (%ld instead of %ld) for LTA-IMG %s",
hh.date, l, s);      ERROR(stage, ERRmess, 1);
    }
    if(hh.data_type != 1)
    {
        sprintf(ERRmess, "Wrong datatype (%d - should be 1=BYTE) for LTA-IMG %s",
hh.data_type, s); ERROR(stage, ERRmess, 1);
    }
   if(map_diff(&hh.mi, &hi.mi) > 1)
   {
      sprintf(ERRmess, "LTA-IMG with deviate mapinfo: %s", s); ERROR(stage, ERRmess,
1);
   }
   if (hh.Vlo != hi.Vlo || hh.Vhi != hi.Vhi)
   {
```

```
      printf("     \nERROR: All IN-IMGs must have same significant range (Vlo-
Vhi)\n");
        printf("IMG of %ld: Vlo=%g, Vhi=%g\n", hi.date, hi.Vlo, hi.Vhi);
      printf("IMG of %ld: Vlo=%g, Vhi=%g\n", hh.date, hh.Vlo, hh.Vhi);
        sprintf(ERRmess, "IN-IMG with deviate significant range (Vlo-Vhi): %s", s);
ERROR(stage, ERRmess, 1);
   }
   if (hh.Vint != hi.Vint || hh.Vslo != hi.Vslo)
   {
        printf("     \nERROR: All IN-IMGs must have same scaling (Vint, Vslo)\n");
      printf("IMG of %ld: Vint=%g, Vslo=%g\n", hi.date, hi.Vint, hi.Vslo);
      printf("IMG of %ld: Vint=%g, Vslo=%g\n", hh.date, hh.Vint, hh.Vslo);
        sprintf(ERRmess, "IN-IMG with deviate scaling (Vint, Vslo): %s", s);
ERROR(stage, ERRmess, 1);
   }

   HI[v].type   = 2;                                    //Reset -1 to 2 (LTA-present)
   HI[v].offset = hh.offset;

    HI[v].fi = (char *)calloc(1+strlen(s), 1); if(HI[v].fi == NULL) {
sprintf(ERRmess, "Insufficient RAM for IN-IMG names"); ERROR(stage, ERRmess, 1); }
    strcpy(HI[v].fi, s);                                 //Inclusive extension '.IMG'
!!!
}
LTAend:

//------------------------
//B5. SHOW FINAL IN-SERIES                              //NB: Now always for NvI=108
IN-IMGs. All LTA-IMGs must be present: yyyytar=0 (all), yyyytar=YYYY (those in
tails).
//------------------------
SHOWinIMGs:

i = 0;                                                  //Nr. of errors found
printf("\n");
printf(" DEK  YYYYMMDD  TYPE  IN-IMG\n");
for(v=0; v<NvI; v++)
{
   switch(HI[v].type)
   {
      case -1: strcpy(t, "?   "); strcpy(s, "Error: Missing IMG!"  ); i++; break;
      case  0: strcpy(t, "?   "); strcpy(s, "Allowed GAP in Centre");      break;
      case  1: strcpy(t, "ACT "); strcpy(s, HI[v].fi); break;
      case  2: strcpy(t, "HIST"); strcpy(s, HI[v].fi); break;
   }
   printf("%4d  %ld  %s  %s\n", v+1, HI[v].date, t, s);

   if(v > 0 && v % 36 != 0) { PressAnyKey(); }      // Skipped if BAT=1/2
}
printf("     Total Nr. of DEKADs/IN-IMGs: %d   (%d ACT-IMGs, %d GAPS, %d HIS-
IMGs)\n", NvI, NvACT, NvGAP, NvLTA);
if(i) { sprintf(ERRmess, "Lacking LTA-IMG(s) of Historical Year=%d", i);
ERROR(stage, ERRmess, 1); }
printf("\n");

//---------------
//B6. OTHER PARMS
//---------------
```

```
printf("p11.Max %% missing per pixel      [def=15] =>"); NviMaxMisPerc = Qdouble(0,
95, 15, -1, &argv);
   NviMaxMis = (short)floor(0.5 + NvI * NviMaxMisPerc / 100.);                    //
Max. Nr. of Missing values (eg. 108 * 15 / 100 = 16 dekads missing)
   printf("\n");
   if(NvGAP > NviMaxMis) { sprintf(ERRmess, "More lacking IN-IMGs (%d) than allowed
Missing Values (%d)", NvGAP, NviMaxMis); ERROR(stage, ERRmess, 1); }

printf("p12.SPF-file with specs   (no ext. *.SPF) =>" ); Qstr(fi_SPF, "No SPF-file",
&argv); strcat(fi_SPF, ".SPF"); file_exist(fi_SPF, 1, 1);

//------------------
// B7. READ SPF-FILE       NB: No defaults, all keywords must be present! R%12,
Dek1, Dek2 are derived (nit in SPF-file).
//------------------
//struct SPFparms                                  //All settings in the SPF-file
//{
//    short          RMl, RMn, RMl2,                    Keep, dT2, dT5, dT6,
S2YM, S2YDt, Dek1, Dek2, S2YC,    o1, oYsm;
//    double         RMw,       Edes, Eevg, Erg, Ecv,   dY1, dY2, Max3, Rat4,
Fsos, Feos,   S2YT, S2YO , Ur, Uf,              oMUl, oMUd, oRGl, oRGd;
//}                 FEN;
//------------------
if((fp=fopen(fi_SPF, "rt")) == NULL) { sprintf(ERRmess, "Opening SPF-file: %s",
fi_SPF); ERROR(stage, ERRmess, 1); }
sprintf(ERRmess, "Reading SPF-file: %s", fi_SPF);

printf("    -Running Mean Filter  : ");
strSPX("FENrmL"    , s, "=", fp, 1, 1); FEN.RMl   = atoi(s); printf(  "FENrmL=%d",
FEN.RMl );
strSPX("FENrmN"    , s, "=", fp, 1, 1); FEN.RMn   = atoi(s); printf(", FENrmN=%d",
FEN.RMn );
strSPX("FENrmW"    , s, "=", fp, 1, 1); FEN.RMw   = atof(s); printf(", FENrmW=%g",
FEN.RMw );
printf("\n");

printf("    -PRE-eliminations     : ");
strSPX("FEN0des"   , s, "=", fp, 1, 1); FEN.Edes  = atof(s); printf(  "FEN0des=%g",
FEN.Edes );
strSPX("FEN0evg"   , s, "=", fp, 1, 1); FEN.Eevg  = atof(s); printf(", FEN0evg=%g",
FEN.Eevg );
strSPX("FEN0rg"    , s, "=", fp, 1, 1); FEN.Erg   = atof(s); printf(", FEN0rg=%g",
FEN.Erg  );
strSPX("FEN0cv"    , s, "=", fp, 1, 1); FEN.Ecv   = atof(s); printf(", FEN0cv=%g",
FEN.Ecv  );
printf("\n");

printf("    -Segment eliminations : ");
strSPX("FENdY1"    , s, "=", fp, 1, 1); FEN.dY1   = atof(s); printf(  "FENdY1=%g",
FEN.dY1  );
strSPX("FENdY2"    , s, "=", fp, 1, 1); FEN.dY2   = atof(s); printf(", FENdY2=%g",
FEN.dY2  );
strSPX("FENdT2"    , s, "=", fp, 1, 1); FEN.dT2   = atoi(s); printf(", FENdT2=%d",
FEN.dT2  );
strSPX("FENmax3"   , s, "=", fp, 1, 1); FEN.Max3  = atof(s); printf(", FENmax3=%g",
FEN.Max3 );
printf("\n                            ");
```

```
strSPX("FENrat4"    , s, "=", fp, 1, 1); FEN.Rat4  = atof(s); printf(  "FENrat4=%g",
FEN.Rat4 );
strSPX("FENdT5"     , s, "=", fp, 1, 1); FEN.dT5   = atoi(s); printf(", FENdT5=%d",
FEN.dT5  );
strSPX("FENdT6"     , s, "=", fp, 1, 1); FEN.dT6   = atoi(s); printf(", FENdT6=%d",
FEN.dT6  );
strSPX("FENkeep"    , s, "=", fp, 1, 1); FEN.Keep  = atoi(s); printf(", FENkeep=%d",
FEN.Keep );
printf("\n");

printf("    -SOS/EOS ratios       : ");
strSPX("FENsos"     , s, "=", fp, 1, 1); FEN.Fsos  = atof(s); printf(  "FENsos=%g",
FEN.Fsos );
strSPX("FENeos"     , s, "=", fp, 1, 1); FEN.Feos  = atof(s); printf(", FENeos=%g",
FEN.Feos );
strSPX("FENuR"      , s, "=", fp, 1, 1); FEN.Ur    = atof(s); printf(", FENuR=%g" ,
FEN.Ur   );
strSPX("FENuF"      , s, "=", fp, 1, 1); FEN.Uf    = atof(s); printf(", FENuF=%g" ,
FEN.Uf   );
printf("\n");

printf("    -SEASON => CIVIL YEAR : ");
strSPX("FENs2yM"    , s, "=", fp, 1, 1); FEN.S2YM  = atoi(s); printf(  "FENs2YM=%d",
FEN.S2YM );         //0=based on MOS, 1=based on EOS
strSPX("FENs2yDt"   , s, "=", fp, 1, 1); FEN.S2YDt = atoi(s); printf(", FENs2YDt=%d",
FEN.S2YDt);         //Shift limits of civil year (normally - if S2YdT=0:  36-71, 0-
based)
FEN.Dek1 = 36 + FEN.S2YDt; FEN.Dek2 = 71 + FEN.S2YDt;        printf(", DEK1/2=%d/%",
FEN.Dek1, FEN.Dek2);
printf("\n");

if(FEN.S2YM == 0)     { FEN.S2YC = 0; }
//Modifications for EOS~based
else
{
   printf("                                ");
                      strSPX("FENs2yC"    , s, "=", fp, 1, 1); FEN.S2YC  = atoi(s);
printf(  "FENs2YC=%d",    FEN.S2YC );         //If 0 then skip all Adjustments
   if(FEN.S2YC == 1) { strSPX("FENs2yDd"   , s, "=", fp, 1, 1); FEN.S2YDd = atoi(s);
printf(", FENs2YDd=%d",   FEN.S2YDd); }
   if(FEN.S2YC >  1) { strSPX("FENs2yT"    , s, "=", fp, 1, 1); FEN.S2YT  = atof(s);
printf(", FENs2YT=%g",    FEN.S2YT ); }
   if(FEN.S2YC >  0) { strSPX("FENs2yO"    , s, "=", fp, 1, 1); FEN.S2YO  = atof(s);
printf(", FENs2YO=%g",    FEN.S2YO ); }
   printf("\n");
}

printf("    -OUT-MODALITIES       : ");
strSPX("FENo1"      , s, "=", fp, 1, 1); FEN.o1    = atoi(s); printf(  "FENo1=%d"  ,
FEN.o1   );
strSPX("FENoYsm"    , s, "=", fp, 1, 1); FEN.oYsm  = atoi(s); printf(", FENoYsm=%d",
FEN.oYsm );
printf("\n");

printf("    -Classification (kk0) : ");
strHDR("FENoKmu", t, fp, &i, 1, 1);
```

```
i = strsplit(t, ",", 0, s);                                          if(i != 2)     {
printf("\n"); sprintf(ERRmess, "KeyWord FENoKmu should contain 2 parameters (MUlo,
MUdelt)"); ERROR(stage, ERRmess, 1); }
strsplit(t, ",", 1, s); f = atof(s); printf(  "MUlo=%g"   , f); FEN.oMUl = f;
strsplit(t, ",", 2, s); f = atof(s); printf(", MUdelt=%g", f); if(f<0.001)    {
printf("\n"); sprintf(ERRmess, "MUdelt (%g) in KeyWord FENoKmu should be >= 0.001",
f);       ERROR(stage, ERRmess, 1); }
                                                                     FEN.oMUd = f;

strHDR("FENoKrg", t, fp, &i, 1, 1);
i = strsplit(t, ",", 0, s);                                          if(i != 2)     {
printf("\n"); sprintf(ERRmess, "KeyWord FENoKrg should contain 2 parameters (RGlo,
RGdelt)"); ERROR(stage, ERRmess, 1); }
strsplit(t, ",", 1, s); f = atof(s); printf(", RGlo=%g"   , f); FEN.oRGl = f;
strsplit(t, ",", 2, s); f = atof(s); printf(", RGdelt=%g", f); if(f<0.001)    {
printf("\n"); sprintf(ERRmess, "RGdelt (%g) in KeyWord FENoKrg should be >= 0.001",
f);       ERROR(stage, ERRmess, 1); }
                                                                     FEN.oRGd = f;

printf("\n");
fclose(fp);

//---------------
// B8. MASK-Image: msk=0/1, fi_MSK, hh, M1, M2
//---------------
msk = 0; M1 = 0; M2 = 0; strcpy(fi_MSK, "none");

printf("\n");
printf("    OPTIONAL MASK-IMAGE (must be BYTE)\n");
printf("p13.Mask-IMAGE       (no ext./blank=none) =>"); Qstr(s, "", &argv);
if(strlen(s))
{
   msk = 1; img_exist(s, 1, 1, fi_MSK);
   envi_hdr_read(fi_MSK, &hh, 1, 0, 0, 1, 1, 1, 0, 0);    //BYTE. Classes allowed
but Class Info is not read.

   printf("    Title  : %.64s\n"        , hh.description);
   printf("    Range  : %.7g ... %.7g\n", hh.Vlo, hh.Vhi);

   //envi_hdr_show(&hh, 4);
   //printf("\n");

   if(map_diff(&hh.mi, &hi.mi) > 1) { sprintf(ERRmess, "MASK and IN-IMGs have
different georeferencing"); ERROR(stage, ERRmess, 1); }

    printf("p14.LO Digital Value to Include [  0-255] =>");     M1 = Qshort( 0, 255,
-1, -1, &argv);
   printf("p15.HI Digital Value to Include [%3d-255] =>", M1); M2 = Qshort(M1, 255,
-1, -1, &argv);
}
else { if (bat) { ++argv;++argv; }  }
printf("\n");

//-------------
// B9. OUT-HDRs         NB1: All 22 HDRs are defined anyhow    NB2: We do this here
in advance because selection of OUT-IMGs (B10) needs ho[v].Vname (which is defined
here)
//-------------
sprintf(ERRmess, "Insufficient RAM for OUT-HDRs");
for(v=0;v<NvO;v++)
```

```
{
    // General items
   //
   ho[v] = hi;
    ho[v].offset = 0;
    sprintf(ho[v].program,"%s.exe (V%d/%d)", PROGNAME, PROGVERSION, LIBVERSION);
   strcpy(ho[v].file_type,"ENVI standard"); ho[v].classes = 0; ho[v].cnames = NULL;
ho[v].colors = NULL;              // Adapted later for v=2 (kk0)

                          sprintf(s, "VI=%s [%s], YYYYt=%ld", hi.Vname, hi.Vunit,
yyyytar); if(yyyytar==0) { strcat(s, " (LTA)"); }
   if(strlen(fi_VAR)) { sprintf(t, ", INvar=%s", fi_VAR);
strcat(s, t); }
   else               { sprintf(t, ", from %s[df=%d]%s.img", fi_pre, fi_df + 1,
fi_suf);            strcat(s, t); }
    if(yyyytar > 0)    { sprintf(t, ", ACT-series from %ld to %ld", HI[vAct1].date,
HI[vAct2].date); strcat(s, t); }
   s[sizeof(hi.description) - 1] = 0; strcpy(ho[v].description, s);

                          sprintf(s, "SPF=%s, MASK=%s", fi_SPF, fi_MSK);
   if(msk)            { sprintf(t, ", MSKlo=%d, MSKhi=%d", M1, M2);
strcat(s, t); }
                          sprintf(t, ", MaxMis%%=%g", NviMaxMisPerc);
strcat(s, t);
                          sprintf(t, ", Shift of target year=%d dekads", FEN.S2YDt);
strcat(s, t);
   s[sizeof(hi.comment    ) - 1] = 0; strcpy(ho[v].comment    , s);

   sprintf(ho[v].flags,  "255=Water, 254=Masked, 253=Insufficient data, 252=Error,
251=No seasons/No season2");

    ho[v].days = 360;   if(yyyytar==0) { ho[v].date = 19620101; } else { ho[v].date
= yyyytar * 10000L + 101; }

   // Values  NB: For all Y-IMGs, keep original scaling of IN-IMGs
   //
   //char               fo_suf[NvO][4]  = { "Ya0", "Yr0", "KK0",
// 00-02: Annual values (independent of seasons) : Y-mean, Y-range, Classification
   //                                        "s1",  "m1",  "e1",  "L1",  "s2",  "m2",
"e2",  "L2",    // 03-10: Dekads of SOS, MOS, EOS & Season length,  for 2 seasons
   //                                       "Ys1", "Ym1", "Ye1", "Ya1", "Ys2", "Ym2",
"Ye2", "Ya2",    // 11-18: Values at SOS, MOS, EOS & Season average, for 2 seasons
   //                                       "LT0", "LR0", "SI0" };
// 19-21: Annual values for both seasons together: length Tl1+Tl2, Tl1/(Tl1+Tl2),
Seasonality Index
   //
   switch(v)
    {
      case  0:
sprintf(s,"Annual Mean VI");
break;
       case  1:                                  ho[v].Vint =   0;
sprintf(s,"Annual VI-Range");
break;
    case  2: ho[v].Vlo = 0; ho[v].Vhi = 244; ho[v].Vint =   0; ho[v].Vslo = 1.0;
strcpy( s,"k=100.Ns[0-2]+10.MU[0-4]+RG[0-4]");  strcpy(ho[v].Vunit, "PHENO-class" );
break;
```

```
      case  3: ho[v].Vlo = 1; ho[v].Vhi = 108; ho[v].Vint = -36; ho[v].Vslo = 1.0;
strcpy (s,"SOS1");                                strcpy(ho[v].Vunit, "dekad");
break;
        case  4: ho[v].Vlo = 1; ho[v].Vhi = 108; ho[v].Vint = -36; ho[v].Vslo = 1.0;
strcpy (s,"MOS1");                                 strcpy(ho[v].Vunit, "dekad");
break;
      case  5: ho[v].Vlo = 1; ho[v].Vhi = 108; ho[v].Vint = -36; ho[v].Vslo = 1.0;
strcpy (s,"EOS1");                                strcpy(ho[v].Vunit, "dekad");
break;
    case  7: ho[v].Vlo = 1; ho[v].Vhi = 108; ho[v].Vint = -36; ho[v].Vslo = 1.0;
strcpy (s,"SOS2");                                 strcpy(ho[v].Vunit, "dekad");
break;
        case  8: ho[v].Vlo = 1; ho[v].Vhi = 108; ho[v].Vint = -36; ho[v].Vslo = 1.0;
strcpy (s,"MOS2");                                 strcpy(ho[v].Vunit, "dekad");
break;
        case  9: ho[v].Vlo = 1; ho[v].Vhi = 108; ho[v].Vint = -36; ho[v].Vslo = 1.0;
strcpy (s,"EOS2");                                 strcpy(ho[v].Vunit, "dekad");
break;

      case  6: ho[v].Vlo = 0; ho[v].Vhi = 108; ho[v].Vint =   0; ho[v].Vslo = 1.0;
strcpy (s,"LEN1");                                strcpy(ho[v].Vunit, "dekads");
break;
        case 10: ho[v].Vlo = 0; ho[v].Vhi = 108; ho[v].Vint =   0; ho[v].Vslo = 1.0;
strcpy (s,"LEN2");                                 strcpy(ho[v].Vunit, "dekads");
break;

        case 11:
sprintf(s,"VI at SOS1");
break;
        case 12:
sprintf(s,"VI at MOS1");
break;
        case 13:
sprintf(s,"VI at EOS1");
break;
        case 15:
sprintf(s,"VI at SOS2");
break;
        case 16:
sprintf(s,"VI at MOS2");
break;
        case 17:
sprintf(s,"VI at EOS2");
break;

        case 14:
sprintf(s,"VI-Mean over Season1");
break;
        case 18:
sprintf(s,"VI-Mean over Season2");
break;

        case 19: ho[v].Vlo = 0; ho[v].Vhi = 108; ho[v].Vint =   0; ho[v].Vslo = 1.0;
strcpy (s, "LEN1+LEN2");                          strcpy(ho[v].Vunit, "dekads");
break;
      case 20: ho[v].Vlo = 0; ho[v].Vhi = 200; ho[v].Vint =   0; ho[v].Vslo = 0.5;
strcpy (s, "100.LEN1/[LEN1+LEN2]");                strcpy(ho[v].Vunit, "%");
break;
```

```
        case 21: ho[v].Vlo = 0; ho[v].Vhi = 250; ho[v].Vint =   0; ho[v].Vslo = 1.0;
strcpy (s, "100.(Ygreen/Yannual - 1)");          strcpy(ho[v].Vunit, "%");
break;
   }
   s[sizeof(hi.Vname) - 1] = 0; strcpy(ho[v].Vname, s);


   // kk0 = Classification (v = 2)
   //
   if(v != 2) { continue; }

   strcpy(ho[v].file_type,"ENVI classification"); ho[v].classes = 256;
// NB: While Vhi=244 !

    ho[v].colors       = (unsigned char **)calloc(256, sizeof(unsigned char *));
if(ho[v].colors   ==NULL) { ERROR(stage, ERRmess, 1); }
    ho[v].cnames       = (         char **)calloc(256, sizeof(         char *));
if(ho[v].cnames   ==NULL) { ERROR(stage, ERRmess, 1); }
    for(k=0;k<256;k++)
    {
        ho[v].colors[k] = (unsigned char  *)calloc(  3, sizeof(unsigned char  ));
if(ho[v].colors[k]==NULL) { ERROR(stage, ERRmess, 1); }   //INIT=0
        ho[v].cnames[k] = (         char  *)calloc( 25, sizeof(         char  ));
if(ho[v].cnames[k]==NULL) { ERROR(stage, ERRmess, 1); }   //INIT=""
    }
    for(Nss=0;Nss<=2;Nss++)                                  // Seasons
    {
        for(i=0;i<5;i++)                                      // Mean-classes
        {
            for(j=0;j<5;j++)                                   // Range-classes
            {
                k = 100*Nss + 10*i + j;
                ho[v].colors[k][2] = 55 + Nss * 100.0;        // Blue  = Seasons or
Cycles           B = 55/    155/    255 for Nss =0/1/2
                ho[v].colors[k][1] = 55 + i   * 50.0;         // Green = VI-mean
G = 55/105/155/205/255 for Ymu=0/1/2/3/4
                ho[v].colors[k][0] = 55 + j   * 50.0;        // Red   = VI-range
R = 55/105/155/205/255 for Yrg=0/1/2/3/4
                sprintf(ho[v].cnames[k], "Ns=%d/2 Mu=%d/4 Range=%d/4", Nss, i, j);
            }
        }
    }

 //  for(k=251;k<256;k++)
    //{
 //     for(i=0;i<3;i++) { ho[v].colors[k][i] = 40; } sprintf(ho[v].cnames[k],
"Flag_%d", k);  //Colors & Names for FLAGS 251-255
    //}

   k = 255; ho[v].colors[k][0] = 255; ho[v].colors[k][1] = 255; ho[v].colors[k][2] =
255; sprintf(ho[v].cnames[k], "Water");                  //White
   k = 254; ho[v].colors[k][0] = 255; ho[v].colors[k][1] = 255; ho[v].colors[k][2] =
0; sprintf(ho[v].cnames[k], "Masked");                 //Yellow
   k = 253; ho[v].colors[k][0] = 255; ho[v].colors[k][1] =   0; ho[v].colors[k][2] =
0; sprintf(ho[v].cnames[k], "Insufficient inputs");   //Red
   k = 252; ho[v].colors[k][0] = 255; ho[v].colors[k][1] =   0; ho[v].colors[k][2] =
0; sprintf(ho[v].cnames[k], "Processing error");      //Red
}
```

```
//--------------
// B10. OUT-IMGs
//--------------
//Base-name (prefix) must be given via p16.
//Six OUT-IMGs are always made: SOS/MOS/EOS for seasons 1/2.
//For the others, selection can be made via p17-p22, as stored in FoM[NvO=22] = 0/1.
//But regardless settings of foM[NvO], all outputs are internally computed!
//Only difference: if foM[v]==0, the concerned OUT-buffer is not written to OUT-IMG
(and the OUT-HDR is not created at the end).
//------------
   //Inputs: Prefix Po and foM[NvO]
   //"""""""""""""""""""""""""""""""
   //char               fo_suf[NvO][4]  = { "Ya0", "Yr0", "KK0",
// 00-02: Annual values (independent of seasons) : Y-mean, Y-range, Classification
   //                                        "s1",  "m1",  "e1",  "L1",  "s2",  "m2",
"e2",  "L2",    // 03-10: Dekads of SOS, MOS, EOS & Season length,  for 2 seasons
   //                                       "Ys1", "Ym1", "Ye1", "Ya1", "Ys2", "Ym2",
"Ye2", "Ya2",    // 11-18: Values at SOS, MOS, EOS & Season average, for 2 seasons
   //                                       "LT0", "LR0", "SI0" };
// 19-21: Annual values for both seasons together: length Tl1+Tl2, Tl1/(Tl1+Tl2),
Seasonality Index
   //"""""""""""""""""""""""""""""""
printf("    OUT-IMGs TO CREATE: PoSo (the So-suffixes are Fixed)\n");
printf("p16.Prefix Po (add drive/path of needed)  =>"); Qstr(s, "", &argv);
if(_fullpath(fo_pre, s, _MAX_PATH)==NULL)     {sprintf(ERRmess, "Path not found for
Base Name PHENO-IMGs: %s", s); ERROR(stage, ERRmess, 1); }
printf("\n");

for(v=0;v<NvO;v++) { foM[v] = 0; }                          //default: all 0=NOT
for(v=3;v<6  ;v++) { foM[v] = 1; foM[v+4] = 1; }            //default: all Sn/Mn/En
for both seasons n=1/2 (v=3/4/5 and 7/8/9)

printf("    CREATE [0/1] following ADDITIONAL OUT-IMGs (default is always O=NO)\n");
printf("p17. 3 IMGs: annual features (overall)              (Ya0, Yr0, kk0)    =>");
i = Qshort(0, 1, 0, -1, &argv);  if(i) { foM[ 0] = 1; foM[ 1] = 1; foM[ 2] = 1;
}
printf("p18. 2 IMGs: for 2 seasons, Y-value at MOS          (Ym1/Ym2)          =>");
i = Qshort(0, 1, 0, -1, &argv);  if(i) { foM[12] = 1; foM[16] = 1;
}
printf("p19. 4 IMGs: for 2 seasons, Y-value at SOS and EOS (Ys1/Ys2, Ye1/Ye2) =>");
i = Qshort(0, 1, 0, -1, &argv);  if(i) { foM[11] = 1; foM[13] = 1; foM[15] = 1;
foM[17] = 1; }
printf("p20. 2 IMGs: for 2 seasons, season length           (L1/L2)            =>");
i = Qshort(0, 1, 0, -1, &argv);  if(i) { foM[ 6] = 1; foM[10] = 1;
}
printf("p21. 2 IMGs: for 2 seasons, mean Y-value            (Ya1/Ya2)          =>");
i = Qshort(0, 1, 0, -1, &argv);  if(i) { foM[14] = 1; foM[18] = 1;
}
printf("p22. 3 IMGs: annual features (season-dependent)     (LT0, LR0, SI0)    =>");
i = Qshort(0, 1, 0, -1, &argv);  if(i) { foM[19] = 1; foM[20] = 1; foM[21] = 1;
}
printf("\n");

   //Override via FEN.o1   (only outputs for 1 season)
   //"""""""""""""""""""
if(FEN.o1 > 0)
{
```

```
   for(i= 7;i<=10;i++) { foM[i] = 0; }
   for(i=15;i<=18;i++) { foM[i] = 0; }
   for(i=19;i<=20;i++) { foM[i] = 0; }
}

   //OUT-IMGs: names, tests & show
   //""""""""""""""""""""""""""""""
sprintf(ERRmess, "Insufficient RAM for OUT-IMG names\n");
fo = (char **)calloc(NvO, sizeof(char *)); if(fo==NULL) { ERROR(stage, ERRmess, 1);
}

printf(" N %-35.35s %s\n", "CONTENTS", "OUT-IMG or FILE");
//Truncate CONTENTS = ho[v].Vname to 35 characters
i = 0;
for(v=0; v<NvO;v++)
{
   if(foM[v]==0) { continue; }
   i++; sprintf(s, "%s%s", fo_pre, fo_suf[v]);
   printf("%2d %-35.35s %s\n", i,  ho[v].Vname, s);
   img_exist(s, 2, 1, t);
//Fatal=1: If user keeps existing OUT-IMG => exit(1);  NB: t has extension ".img"
    fo[v] = (char *)calloc(strlen(t) + 1, 1); if(fo[v] == NULL) { ERROR(stage,
ERRmess, 1); }
   strcpy(fo[v], t);                                                    //Incl.
'.IMG' !!!
}
printf("\n");

   //Three additional TXT-files (always created)
   //""""""""""""""""""""""""""""
sprintf(s, "%s%s", fo_pre,".VAR"); strcpy(t, "VAR-file OUT-IMGs"); printf("   %-
35.35s %s\n", t, s);
sprintf(s, "%s%s", fo_pre,".MTA"); strcpy(t, "MTA-file OUT-IMGs"); printf("   %-
35.35s %s\n", t, s);
sprintf(s, "%s%s", fo_pre,".TXT"); strcpy(t, "TXT-file ERRORS  "); printf("   %-
35.35s %s\n", t, s);
printf("\n");


                                                        //====================
                                                        //  C. PRE-PROCESSING
                                                        //====================

strcpy(stage, "PREPROCESSING"); strcpy(ERRmess, "Insufficient RAM for
buffers/tables");
Ncol = hi.samples; Nrec = hi.lines;

//--------------------
// C1. FILES & BUFFERS
//--------------------
   //Open Nvi IN-IMGs: Can be either ACT (type=1) or LTA (type=2)
   //"""""""""""""""""
fpi = (FILE **)calloc(NvI, sizeof(FILE *)); if(fpi==NULL) { ERROR(stage, ERRmess,
1); }
for(v=0; v<NvI; v++)
{
    if(HI[v].type < 1) { continue; }          //TYPE=0: Lacking IN-IMG in central
part of the series
```

```
   if((fpi[v]=fopen(HI[v].fi,"rb"))==NULL) { sprintf(ERRmess, "Opening IN-IMG %s",
HI[v].fi); ERROR(stage, ERRmess, 1); }
   fseek(fpi[v], HI[v].offset, SEEK_SET);   //skip header bytes
}

    //Open Nvo OUT-IMGs: Only those which were asked via foM[NvO]
   //""""""""""""""""""
fpo = (FILE **)calloc(NvO, sizeof(FILE *)); if(fpo==NULL) { ERROR(stage, ERRmess,
1); }
for(v=0; v<NvO; v++)
{
   if(foM[v]) { if((fpo[v]=fopen(fo[v],"wb"))==NULL) { sprintf(ERRmess, "Opening
OUT-IMG %s", fo[v]); ERROR(stage, ERRmess, 1); } }
}

   //General IO-buffers: NvI=108 BYTE buffers with Ncol elements. First NvO buffers
also used for OUTput.
   //""""""""""""""""""
buf = (unsigned char **)calloc(NvI, sizeof(unsigned char *)); if(buf   ==NULL) {
ERROR(stage, ERRmess, 1); }
for(v=0; v<NvI; v++)
{
    buf[v] = (unsigned char *)calloc(Ncol, 1);                 if(buf[v]==NULL) {
ERROR(stage, ERRmess, 1); }
}
    //Optional MASK: Open & Buffer bm
    //"""""""""""""
if(msk)
{
    if((fp=fopen(fi_MSK, "rb")) == NULL) { sprintf(ERRmess, "Opening MASK: %s",
fi_MSK); ERROR(stage, ERRmess, 1); }
    fseek(fp, hh.offset, SEEK_SET);
    bm = (unsigned char *)calloc(Ncol, 1); if(bm==NULL) { ERROR(stage, ERRmess, 1);
}
}

//---------------------------------
// C2. TABLES WITH PROFILE OF 1 PIXEL
//---------------------------------
Yi  = (double  *)calloc(NvI + 2 * FEN.RMl, sizeof(double)); if(Yi ==NULL) {
ERROR(stage, ERRmess, 1); }     //Raw IN-profile, but later interpolated. Keep tails
for RMF-filter.
Yj  = (double  *)calloc(NvI + 2 * FEN.RMl, sizeof(double)); if(Yj ==NULL) {
ERROR(stage, ERRmess, 1); }     //TMP for iterative RMF-smoothing
Yw  = (double  *)calloc(NvI + 2 * FEN.RMl, sizeof(double)); if(Yw ==NULL) {
ERROR(stage, ERRmess, 1); }     //Smoothing weights
Ys  = (double  *)calloc(NvI              , sizeof(double)); if(Ys ==NULL) {
ERROR(stage, ERRmess, 1); }     //Smoothed (running mean of length RMF2)

   //Table SGM[NvI][3] contains for all extremes in smoothed Ys-profile: 0=Position
[0 -> 107], 1=Y-value, 2=Type (0=Min, 1=Max)
                        SGM    = (double **)calloc(NvI, sizeof(double*)); if(SGM
==NULL) { ERROR(stage, ERRmess, 1); }
for(v=0; v<NvI; v++) { SGM[v] = (double * )calloc(3   , sizeof(double )); if(SGM[v]
==NULL) { ERROR(stage, ERRmess, 1); } }

                        SGM0   = (double **)calloc(NvI, sizeof(double*)); if(SGM0
==NULL) { ERROR(stage, ERRmess, 1); }
```

```
for(v=0; v<NvI; v++) { SGM0[v]= (double * )calloc(3   , sizeof(double ));
if(SGM0[v]==NULL) { ERROR(stage, ERRmess, 1); } }

CYC = (struct SEASON *)calloc(20, sizeof(struct SEASON)); if(CYC == NULL) {
ERROR(stage, ERRmess, 1); }     //Info on all cycles (possible seasons in 3 years:
20 should be enough)
SSN = (struct SEASON *)calloc( 2, sizeof(struct SEASON)); if(SSN == NULL) {
ERROR(stage, ERRmess, 1); }     //Info on 1/2 cycles in Central/Target Year Yt


Ymis     = hi.Vint + hi.Vslo * hi.Vlo - 1000;
//Internal flag to label missing IN-values in Yi(108)
FEN.RMl2  = 2 * FEN.RMl + 1;
//RM-smoothing: full length of Running Mean Filter (1 becomes 3, 2 becomes 5, ...
Not used for FEN.RMl=0 => FEN.RMl2=1.

//----------------------
// C3. ERRORS & WARNINGS                                                  //NB: In
descending order of importance!
//----------------------
   //List of Errors/Warnings
   //"""""""""""""""""""""""""
strcpy(Emes[ 0], "No Errors");

strcpy(Emes[ 1], "E: Bad sequence MAX-MAX or MIN-MIN");
strcpy(Emes[ 2], "E: Good Profile but no seasons Found");
strcpy(Emes[ 3], "E: SOS1 or EOS1 undefined");                            //NO
ERROR: Possible for first/last season
strcpy(Emes[ 4], "E: SOS2 or EOS2 undefined");
strcpy(Emes[ 5], "E: SOS1 >= EOS1");
strcpy(Emes[ 6], "E: SOS2 >= EOS2");
strcpy(Emes[ 7], "E: MOS1 beyond [SOS1-EOS1]");
strcpy(Emes[ 8], "E: MOS2 beyond [SOS2-EOS2]");

strcpy(Emes[ 9], "W: LEN1 > 36 dekads");
strcpy(Emes[10], "W: LEN2 > 36 dekads");
strcpy(Emes[11], "W: 2 seasons & LEN1+2 > 36 dekads");                    //NB:
ALWAYS POSSIBLE (multi-annual crops)
strcpy(Emes[12], "W: 2 seasons & EOS1 >= SOS2 - 1");
strcpy(Emes[13], "W: MOS1=SOS1");
strcpy(Emes[14], "W: MOS2=SOS2");
strcpy(Emes[15], "W: MOS1=EOS1");
strcpy(Emes[16], "W: MOS2=EOS2");
strcpy(Emes[17], "W: No seasons found in Central Year");

   //TXT-File with Errors/Warnings
   //"""""""""""""""""""""""""""""""
sprintf(s, "%s%s", fo_pre, ".TXT");
if ((fpe = fopen(s, "wt")) == NULL) { sprintf(ERRmess, "Creating TXT-file: %s", s);
ERROR(stage, ERRmess, 1); }

fprintf(fpe, "ERRORS (E) & WARNINGS (W) in PHENO-IMGs generated by Program %s.exe
(V%d/%d)\n", PROGNAME, PROGVERSION, LIBVERSION);
fprintf(fpe, "- %s\n", ho[0].description);
fprintf(fpe, "- %s\n", ho[0].comment);
fprintf(fpe, "See overall statistics at the end of this file.\n");
```

```
fprintf(fpe,
"*************************************************************************\n")
;
fprintf(fpe, "        N,    COL,    REC, EW, ERROR/WARNING                    ,
MSK\n");
fflush(fpe);

   //Initialize counters
   //""""""""""""""""""""
N255 = 0; N254 = 0; N253 = 0; N252 = 0; N251a = 0; N251b = 0;              //Nr. of
flagged pixels to report at the end. For 251 (No Season) there are two cases
Ns1  = 0; Ns2 = 0;                                                         //Normal
land pixels with 1 or 2 seasons
NerrT = 0;                                                                 //For OUT
TXT-file: Total Nr. or of Errors + Warnings
for(i=0;i< 7;i++) { NE[i]   = 0; }                                         //Scores
for each elimination TEST
for(i=0;i<18;i++) { Nerr[i] = 0; }                                         //Nr. of
pixels affected by different errors and warnings
for(v=0;v<NvO;v++) {    ho[v].Vmin = ho[v].Vhi; ho[v].Vmax = ho[v].Vlo; }
//Initialize OUT-Extremes
Vm = 0;                                                                    //Value
read from Mask


//=================
                                                          //  D. PROCESSING

//=================
strcpy(stage, "PROCESSING");
progression(0, Nrec, &progr40, &progr40_step, 1, "Computing phenological IMGs",
"lines", "Processing", &GUIdone, &GUIdone_step);
for(rec=0; rec < Nrec; rec++)
{
   //---------------------------
    //REC: Read all input buffers
   //---------------------------
    for(v=0; v<NvI; v++) { if(HI[v].type > 0) { fread(buf[v], 1, Ncol, fpi[v]); } }
    if(msk)                                  { fread(bm    , 1, Ncol, fp    ); }

   //-----------------------
    //REC: Process all pixels
   //-----------------------
    for(col=0; col<Ncol; col++)
    {
      W = 0;
//So far no errors (W=Wrong)
      //printf("REC/COL=%ld/%ld\n", rec+1, col+1);

      TESTpix = 0; if(TEST) { if(col + 1 == TESTcol && rec + 1 == TESTrec) { TESTpix
= 1; } }

      //============================
      //1. Y-PROFILES: ORI & SMOOTHED
      //============================
        //Scan IN-values, scale to Y & count Nbad
        //""""""""""""""""""""""""""""""""""""""""
```

```
        for(v=0; v<FEN.RMl; v++) { Yi[v] = Ymis; Yi[FEN.RMl + NvI + v] = Ymis; }
//Reset tails to Ymis (needed after InterPolMissing on previous pixel)

        Nbad = 0; j = FEN.RMl - 1;
//Place 108 values in Yi[j] keeping FEN.RMl free places in front and at end
        for(v=0; v<NvI; v++)
        {
            j++;
         if(HI[v].type < 1)               { Yi[j] = Ymis; Nbad++; }
//Lacking IN-IMG
         else
         {  i = buf[v][col];
            if(i < hi.Vlo || i > hi.Vhi)  { Yi[j] = Ymis; Nbad++; }
//OUT-range
                 else                                    { Yi[j] = hi.Vint + hi.Vslo
* (double)i; }  //Normal
         }
      }

      //Skip special pixels & Jump to next pixel. NB1: These flags hold for all the
OUT-IMGs                        NB2: Order is important: SEA > MASK > LAND with
insufficient data
        //""""""""""""""""""""""""""""""""""""""""""""
        if(Nbad == NvI)                          { N255++; for(v=0; v<NvO; v++){
buf[v][col] = 255; } continue; }        //Missing values: ALL, probably water
        if(msk)
        {
            Vm = bm[col];
         if(Vm < M1 || Vm > M2)           { N254++; for(v=0; v<NvO; v++){
buf[v][col] = 254; } continue; }        //OUT-masked
        }
        if(Nbad > NviMaxMis)                          { N253++; for(v=0; v<NvO; v++){
buf[v][col] = 253; } continue; }        //LAND with too much missing values

      //Interpolate MISSING values (also needed to extrapolate tails for later
Running Mean Filtering)             NB: What remains are LAND pixels with sifficient
valid inputs
        //""""""""""""""""""""""""""""
        i = 1; j = 1; InterPolMissing(Yi, NvI + 2 * FEN.RMl, Ymis + 10.0, &i, &j);
//Ymis+10.0 is in-between Ymis and Ylo. Return values of i/j are not used

        //Ys = RM-Smoothed profile => Only 108 values (no more FEN.RMl-tails like Yi
has)
      //NB:  FEN.RMl2 = 2 * FEN.RMl + 1 = full length of Running Mean Filter (1
becomes 3, 2 becomes 5,...). Not used for FEN.RMl=0 => FEN.RMl2=1.
      //     All values in Ys[NvI] are always replaced => No need for initialisation
to 0.
      //NB:  Now weighted, extremes get weight FENw (instead of default 1.0).

//""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
        if(FEN.RMl < 1)
//Skip smoothing
        {
         for(v=0;v<NvI;v++) { Ys[v] = Yi[FEN.RMl + v]; }
        }
        else
        {
```

```
         for(i=0;i<FEN.RMn;i++)
//Iterations!
         {
            //Yj = IN-series to smooth
            //
            if(i == 0)
//Yj = copy of Yi, incl. tails
            {
               for(v=0; v<NvI + 2 * FEN.RMl; v++) { Yj[v] = Yi[v]; }
            }
            else
//Yj = copy of Ys, excl. tails. But if Yi > Ys then select Yi.
            {
               for(v=0; v<NvI; v++)
               {
                  j = FEN.RMl + v; Yj[j] = Ys[v]; if(Yi[j] > Ys[v]) { Yj[j] = Yi[j];
}
               }
               for(v=0; v<FEN.RMl; v++) { Yj[v] = Yj[FEN.RMl]; Yj[FEN.RMl + NvI + v]
= Yj[FEN.RMl + NvI - 1]; } //Extrapolate tails
            }

            //Yw = Weights
            //
              for(v=0; v<NvI    ; v++) { Yw[FEN.RMl + v] = 1.0; }
//Reset all weights to 1.0...
              for(v=0; v<FEN.RMl; v++) { Yw[v] = 0.5; Yw[FEN.RMl + NvI + v] = 0.5; }
//But tails get W=0.5
              j = FEN.RMl - 1;
            for(v=0; v<NvI    ; v++)
            {
               j++; Y = Yj[j];
               if(Y > Yj[j-1] && Y > Yj[j+1]) { Yw[j] = FEN.RMw; }
//Extremes get W=FEN.RMw
               if(Y < Yj[j-1] && Y < Yj[j+1]) { Yw[j] = FEN.RMw; }
            }

            //Ys = Smoothed Series
            //
               sum = 0.0; sumw = 0.0;
            for(v=0;v<FEN.RMl2;v++) { sumw = sumw + Yw[v]; sum = sum + Yw[v] *
Yj[v]; } Ys[0] = sum / sumw;    //RMF-result for first FEN.RMl2 values in Yj =>
Holds for Ys[0]
               j = FEN.RMl2 - 1;  k = -1;
               for(v=1;v<NvI;v++)
//Now find RMF-results for next entries in Ys[j=1 to NvI-1]
               {
                   j++; k++;
//j = New dekad to include. k = dekad to be removed
               sumw = sumw - Yw[k]          + Yw[j];
               sum  = sum  - Yw[k] * Yj[k] + Yw[j] * Yj[j];
               Ys[v] = sum / sumw;
               }
         }
      }

if(TESTpix)
{
```

```
   fprintf(fpe, "\n");
   fprintf(fpe, "DETAILS for COL/REC=%ld/%ld\n", col+1, rec+1);
   fprintf(fpe, "108 Inputs (ORI & Smoothed):\n");
   for(v=0; v<NvI; v++)  { fprintf(fpe, "%3d, %10.7f, %10.7f\n", v+1, Yi[FEN.RMl +
v], Ys[v]); }
   fprintf(fpe, "\n\n"); fflush(fpe);
}

      //===========================
      //2. SKIP PIXELS WITH Ns = 0
      //===========================
    //General statistics (based on Ys) over 36 dekads in central year (FEN.Dek1 -
FEN.Dek2)
        //"""""""""""""""""""
      Ya0 = 0.0; Ymin = 3E+38; Ymax = -Ymin;
//      for(v=36; v<72; v++)
        for(v=FEN.Dek1; v<=FEN.Dek2; v++)
        {
             Y = Ys[v]; Ymin = min(Y, Ymin); Ymax = max(Y, Ymax); Ya0 = Ya0 + Y;
        }
        Ya0 = Ya0 / 36.; Yr0 = Ymax - Ymin; Ycv = Yr0 / Ya0;     //Overall annual Y-
mean (Ya0), Y-range (Yr0) and Coeff. of Variation (Ycv)

      //Skip pixels without seasonality                          NB: Outputs OK for
OUT-IMGs 1-3 (Ya0, Yr0, kk0), but flagged for the others!
      //"""""""""""""""""""""""""""""""""
      Nss = 1;                                                 //Seasons in central
year
      if(Ymax < FEN.Edes) { Nss = 0; }                         //Desert
      if(Ymin > FEN.Eevg) { Nss = 0; }                         //Evergreen
      if(Nss == 1)
      {
         if(Yr0 < FEN.Erg && Ycv < FEN.Ecv) { Nss = 0; }       //Insuffient
variability
      }

      if(Nss == 0)
      {
         N251a++; W = -1;                                      //W = -1: No
seasonality at all (this is no real Error/Warning)
         NssOri = 0;                                           //Memorize this for
KK0 OUT-IMG
         goto OUTPUTS;                                         //There we'll output:
1-3: Normal, 4-22: 251
      }

      //=====================================
      //3. SEGMENT PRUNING: MIN-MAX or MAX-MIN
      //=====================================
      //--------------
      //Initialisation
      //--------------
      //Table SGM[NvI][3] contains for all extremes in smoothed Ys-profile:
0=Position [0 - 107], 1=Ys-value, 2=Type (0=Min, 1=Max)
      //NB: InterPolMissing sometimes retrieves Yi-values = -0.000 instead of simply
0.000. That gives errors in table SGM (wrong S=slope). Now with Tolerance!
        //"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

```
        Ne0 = 0;                                            //Nr. of Min/Max-extremes
over the 3 years => Ne = 2 * Nss (Nss = Nr. of seasons or cycles)
        j = sgnd(Ys[1] - Ys[0]);                //j = slope of first segment
(initialisation): +/0/- = rising/flat/descending
        for(v=2;v<NvI;v++)
        {
            i = sgnd(Ys[v] - Ys[v-1]);          //sign of slope w.r.t. to previous
period
            if(i != 0 && i != j)                    //Previous point (v-1) was
extremum
            {
                SGM[Ne0][0] = v - 1;                //Position of extremum in
periods[0 -> NvI - 1]
                SGM[Ne0][1] = Ys[v - 1];        //Value of this extremum
                SGM[Ne0][2] = (i < 0);          //Type: Max=1 / Min=0
                Ne0++;
                j = i;                          //Previous slope
            }
        }
      //Now: we found Ne0 extremes, but that number can be odd or even.
Initialisation is weak.

      //SGM0()=Copy of SGM()
      //""""""""""""""""""""""
      for(e=0;e<Ne0;e++) { for(i=0;i<3;i++) { SGM0[e][i] = SGM[e][i]; } }

if(TESTpix)
{
   fprintf(fpe, "Ne0=%d (before Segment Pruning)\n", Ne0);
   for(e=0;e<Ne0;e++) { fprintf(fpe, "%3d, %7.3f, %7.3f, %7.3f\n", e+1,
SGM0[e][0]+1, SGM0[e][1], SGM0[e][2]); }
   fprintf(fpe, "\n\n"); fflush(fpe);
}

      //vP = Position of highest Max in Central Year => To be protected/kept in
elimination tests below
      //""""""""""""""""""""""""""""""""""""""""""""""""
      if(FEN.Keep == 0) { vP = -1; }
//Skip this protective measure
      else
      {
         Ymax = -3E+38; vP = -1;
         D = (FEN.Dek1 + FEN.Dek2) / 2.0;
//D=centre of central year. Normally: D = (36 + 71) / 2.0 = 53.5.
           for(e=0;e<Ne0;e++)
           {
            v = SGM[e][0];
//          if(SGM[e][2] && v > 35 && v < 72)
//e=Max in central year Yt
            if(SGM[e][2] && v >= FEN.Dek1 && v <= FEN.Dek2)
//e=Max in central year Yt  (FEN.Dek1 - FEN.Dek2)
            {
               Y = SGM[e][1];
               if(Y < Ymax) {                   continue; }
               if(Y > Ymax) { Ymax = Y; vP = v; continue; }
//Remains: Y = Ys[vP]. Which to select?
               if(Yi[FEN.RMl + v] > Yi[FEN.RMl + vP]) { vP = v; continue; }
//Doubt:  1. Check Yi
```

```
                if(fabs(v - D) < fabs(vP - D))          { vP = v; }
//          2. e is closer to centre of central year
              }
            }
        }

        //-----------------------------------------------------------
        //Eliminate irrelevant segments in SGM[Ne0][3] via 6 Tests
        //-----------------------------------------------------------
          Ne1 = Ne0;
//Nr. of extremes: Ne0=originally, Ne1=after elimination

           //TEST1: Low absolute Difference (FEN.dY1=0.025)
           //"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
          while(1)
//Repeat this iteratively, starting with with flattest segment
          {
              Dmin = 1E+38; Ee = -1;
              for(e=1;e<Ne1;e++)
              {
              if(SGM[e][0] == vP || SGM[e - 1][0] == vP) { continue; }
                  Y = fabs(SGM[e][1] - SGM[e-1][1]);
//Absolute value of Y-difference
                  if(Y < Dmin && Y < FEN.dY1)
//Best candidate to eliminate so far
                  {
                      Dmin = Y; Ee = e;
//Eliminate segment Ee-1 => Ee
                  }
              }
              if(Ee == -1) { break; }
//No more such bad segments found
              Ne1 = Ne1 - 2; NE[1] = NE[1] + 1;
//Eliminate two extrema
              for(e=Ee-1;e<Ne1;e++) { for(i=0;i<3;i++) { SGM[e][i] = SGM[e+2][i]; } }
//Shift good segments to start of table. Overwrite the eliminated one.
          }

           //TEST2: Low absolute Difference (FEN.dY2=0.050) and Short distance
(FEN.dT2=4)

//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
          while(1)
          {
              Dmin = 1E+38; Ee = -1;
              for(e=1;e<Ne1;e++)
              {
              if(SGM[e][0] == vP || SGM[e - 1][0] == vP) { continue; }
                  Y = fabs(SGM[e][1] - SGM[e-1][1]);
//Absolute value of Y-difference
              i =      SGM[e][0] - SGM[e-1][0];
//Time difference in dekads
                  if(Y < Dmin && Y < FEN.dY2 && i < FEN.dT2)
//Best candidate to eliminate so far
                  {
                      Dmin = Y; Ee = e;
//Eliminate segment Ee-1 => Ee
                  }
```

```
            }
            if(Ee == -1) { break; }
//No more such bad segments found
            Ne1 = Ne1 - 2; NE[2] = NE[2] + 1;
//Eliminate two extrema
            for(e=Ee-1;e<Ne1;e++) { for(i=0;i<3;i++) { SGM[e][i] = SGM[e+2][i]; } }
//Shift good segments to start of table. Overwrite the eliminated one.
        }

            //TEST3: For maxima with Y-value < FEN.Max3 (0.0): Remove that MAX +
Neighbour + highest MIN       NB: TEST3 is skipped, because FEN.Max3 = 0.0!
         //
      if(FEN.Max3 > 0.0)
      {
           while(1)
           {
            if(SGM[0][2]) { E1 = 0; } else { E1 = 1; }
//Position of first maximum in list
               Ee = -1;
               for(e=E1;e<Ne1;e=e+2)
//Scan maxima
               {
               if(SGM[e][0] == vP) { continue; }
                   if(SGM[e][1] < FEN.Max3)
//This e-maximum falls below the threshold FEN.Max3
                   {
                       Ee = e;
//Default             => Remove this MAX + previous MIN
                       if(e == 0)       { Ee++; break; }
//e is first extremum => Remove this MAX + next MIN
                       if(e == Ne1 - 1) {       break; }
//e is last  extremum => Keep default
                       if(SGM[e+1][1] > SGM[e-1][1]) { Ee++; }
//Remove this MAX + next MIN because that is higher than previous MIN
                       break;
                   }
               }
               if(Ee == -1) { break; }
               Ne1 = Ne1 - 2; NE[3] = NE[3] + 1;
               for(e=Ee-1;e<Ne1;e++) { for(i=0;i<3;i++) { SGM[e][i] = SGM[e+2][i]; }
}       //Shift good segments to start of table. Overwrite the eliminated one.
           }
      }

            //TEST4: For irrelevant maxima (FEN.Rat4=0.25): Remove that MAX +
Neighbour with highest MIN
         //
      if(FEN.Rat4 > 0.0)
      {
           Ymin = 3E+38; Ymax = -3E+38;
           for(e=0;e<Ne1;e++) { Ymin = min(SGM[e][1], Ymin); Ymax = max(SGM[e][1],
Ymax); } //Absolute extremes over 3 years
           Y = Ymin + FEN.Rat4 * (Ymax - Ymin);
//Remove all maxima below this threshold Y

           while(1)
//Again iteratively, starting with worst case
           {
```

```
            if(SGM[0][2]) { E1 = 0; } else { E1 = 1; }
//Position of first maximum in list
                Ee = -1;
                for(e=E1;e<Ne1;e=e+2)
//Scan maxima
                {
                if(SGM[e][0] == vP) { continue; }
                    if(SGM[e][1] < Y)
//This e is a maximum and it falls below the threshold
                    {
                        Ee = e;
//Default             => Remove this MAX + previous MIN
                        if(e == 0)       { Ee++; break; }
//e is first extremum => Remove this MAX + next MIN
                        if(e == Ne1 - 1) {       break; }
//e is last  extremum => Keep default
                        if(SGM[e+1][1] > SGM[e-1][1]) { Ee++; }
//Remove this MAX + next MIN because that is higher than previous MIN
                        break;
                    }
                }
                if(Ee == -1) { break; }
                Ne1 = Ne1 - 2; NE[4] = NE[4] + 1;
                for(e=Ee-1;e<Ne1;e++) { for(i=0;i<3;i++) { SGM[e][i] = SGM[e+2][i]; }
}       //Shift good segments to start of table. Overwrite the eliminated one.
            }
      }

            //TEST5: Maxima too close (FEN.dT5=6): Remove intermediate MIN and
Lowest MAX
         //
        while(1)
        {
           if(SGM[0][2]) { E1 = 2; } else { E1 = 3; }
//Position of 2nd maximum in list
            Ee = -1;
            for(e=E1;e<Ne1;e=e+2)
//Scan subsequent maxima (e+2), starting from this E1
            {
                if(SGM[e][0] - SGM[e-2][0] < FEN.dT5)
//Distance between both maxima is too short
                {
                    Ee = e;
//Certainly remove central MIN. And also current (following) MAX. That is default.
                    if(SGM[e][1] > SGM[e-2][1]) { Ee--; }
//Correct: remove previous MAX because that is lower
                if(SGM[Ee][0] == vP || SGM[Ee - 1][0] == vP) { Ee = -1; continue; }
                    break;
                }
            }
            if(Ee == -1) { break; }
            Ne1 = Ne1 - 2; NE[5] = NE[5] + 1;
            for(e=Ee-1;e<Ne1;e++) { for(i=0;i<3;i++) { SGM[e][i] = SGM[e+2][i]; } }
//Shift good segments to start of table. Overwrite the eliminated one.
        }

            //TEST6: Min-Max too close (FEN.dT6=3): Remove entire segment
         //
```

```
         //NB: FEN.dT6 must be > 2 dekads: This guarantees that later there will be
space in-between for SOS/EOS
         //NEO: Remove MAX + highest MIN, or MIN + Lowest MAX
         //
        while(1)
        {
            Ee = -1;
            for(e=1;e<Ne1;e++)
            {
            if(SGM[e][0] == vP || SGM[e - 1][0] == vP) { continue; }
                if(SGM[e][0] - SGM[e-1][0] < FEN.dT6)
//Segment indeed too short
                {
                    Ee = e;
//Default: Remove this e + previous extreme
                    if(e == Ne1 - 1) { break; }
//e is last  extremum => Keep default
               if(SGM[e][2])
//e=MAX: Also delete highest MIN.
               {
                  if(SGM[e + 1][1] > SGM[e - 1][1]) { Ee++; }
               }
               else
//e=MIN: Also delete lowest  MAX.
               {
                  if(SGM[e + 1][1] < SGM[e - 1][1]) { Ee++; }
               }
                    break;
                }
            }
            if(Ee == -1) { break; }
            Ne1 = Ne1 - 2; NE[6] = NE[6] + 1;
            for(e=Ee-1;e<Ne1;e++) { for(i=0;i<3;i++) { SGM[e][i] = SGM[e+2][i]; } }
//Shift good segments to start of table. Overwrite the eliminated one.
        }

         //Final Check (Redundant?)
         //"""""""""""
        for(e=1;e<Ne1;e++)
        {
         if(SGM[e][2] == SGM[e-1][2]) { W = 1; goto OUTPUTS; }
//ERROR: Non-allowed sequence of MIN-MIN or MAX-MAX
      }

      //--------------------------------------
      //Trap Error W= 17 (Nss=0 at this stage)
//NB: This is possible when all extrema have been eliminated above.
      //--------------------------------------
      if(Ne1 == 0)
      {
         N251b++; W = 17;
//W = 17: There is some seasonality in 3-yearly profile (see first test), but no
seasons in Yc
         Nss = 0; NssOri = 0;
//Memorize this for KK0 OUT-IMG
         goto OUTPUTS;
//There we'll output: 1-3: Normal, 4-22: 251
      }
```

```
      //-------------------------------------------------------------------------
-------------------------------------
      //First retained point is MAX => Search/add appropriate MIN before.
      //NB: This removes all or most of the W=3 Errors (SOS1 or EOS1 undefined). W=3
only occurs for SOS=undefined (not EOS)?
      //    It's here that we need table SGM0 with the original extrema (before
pruning)
      //-------------------------------------------------------------------------
-------------------------------------
        if(SGM[0][2])
      {
          for(E2=0;E2<Ne0;E2++) { if(SGM[0][0] == SGM0[E2][0]) { break; } }
//E2=Corresponding MAX in SGM0(). E2 always defined!

         Dmax = 0.0; i = 0; E1 = -1;
         for(e=E2-1;e>=0;e=e-2)
//Scan previous MINima. Skip if E2=0
         {
            j = SGM[0][0] - SGM0[e][0] + 1; if(j < FEN.dT6) { continue; }
            Y = SGM[0][1] - SGM0[e][1]    ; if(Y < FEN.dY1) { continue; }
            D = Y / j;                      if(D > Dmax)    { Dmax = D; E1 = e; }
//E1=Original MIN to add on top of SGM()
            i++; if(i >= 2 && E1 > -1) { break; }
         }

         if(E1 == -1) { E1 = E2 - 1; }
//Still possible: E2=0, hence E1=-1
         if(E1 > -1)
//Else: No solution & Error W=3 at end
         {
            for(e=Ne1-1; e>=0; e=e-1)
            {
               for(i=0;i<3;i++) { SGM[e + 1][i] = SGM[e][i]; }
//Shift downwards to make place on top
            }
            for(i=0;i<3;i++) { SGM[0][i] = SGM0[E1][i]; }
//Add MIN of E1 on top
            Ne1++;
         }
      }
      //if(rec==221 && col==1192) { message("STAP1 OK. Press ENTER to continue...",
1); }


      //-------------------------------------------------------------------------
-------------------------------------
      //Last retained point is MAX => Search/add appropriate MIN at the end
      //NB: This removes all or most of the W=17 Errors (No season found in Central
Year)
      //    Essential here that Ne1 > 0
      //-------------------------------------------------------------------------
-------------------------------------
        if(SGM[Ne1-1][2])
      {
          for(E2=Ne0-1;E2>=0;E2--) { if(SGM[Ne1-1][0] == SGM0[E2][0]) { break; } }
//E2=Corresponding MAX in SGM0(). E2 always defined (0-107)!
```

```
         Dmax = 0.0; i = 0; E1 = -1;
//But E1 is not always defined (still -1)!
         for(e=E2+1;e<Ne0;e=e+2)
//Scan following MINima. Skip if E2 = Ne0 - 1
         {
            j = SGM0[e][0] - SGM[Ne1-1][0] + 1; if(j < FEN.dT6) { continue; }
            Y = SGM[Ne1-1][1] - SGM0[e][1]    ; if(Y < FEN.dY1) { continue; }
            D = Y / j;                          if(D > Dmax)    { Dmax = D; E1 = e;
}        //E1=Original MIN to add at end of SGM()
            i++; if(i >= 2 && E1 > -1) { break; }
         }

         if(E1 == -1) { E1 = E2 + 1; }
//Simply keep next MIN. But still possible: E2=Ne0-1, hence E1=Ne0 (beyond range)
         if(E1 < Ne0)
//Else: No solution & Error at end
         {
            for(i=0;i<3;i++) { SGM[Ne1][i] = SGM0[E1][i]; }
//Add MIN of E1 at the end
            Ne1++;
         }
      }

if(TESTpix)
{
   fprintf(fpe, "Ne1=%d (after  Segment Pruning)\n", Ne1);
   for(e=0;e<Ne1;e++) { fprintf(fpe, "%3d, %7.3f, %7.3f, %7.3f\n", e+1, SGM[e][0]
+1, SGM[e][1] , SGM[e][2]);  }
   fprintf(fpe, "\n\n"); fflush(fpe);
}

      //================================
      //4. CHARACTERISTICS OF ALL CYCLES
      //================================
      //Nc = Nr. of Remaining Maxima/Cycles in Profile = Can be potential Seasons
Nss in cetral year Yr (two cycles at edges might be incomplete)
      //Redim as SEASON  CYC(1 to 20)                  Info on all cycles (possible
seasons in 3 years: 20 should be enough). REDIM not needed.
      //struct SEASON
      //{
      //   short          SOS,  MOS,  EOS,   L,    Keep;          NB: Keep=1 if
Cycle/Season belongs to Yt
      //   double         Ysos, Ymos, Yeos,  Area;
      //}               *CYC, *SSN;
      //================================
      //Nc = MAXima/seasons in 3-yearly profile     Redundant: Nc computed below
      //-------------------------------------
      //Nc = 0;
        //for(e=0;e<Ne1;e++)
        //{
        //  if(SGM[e][2]) { Nc++; }
        //}

      //-------------------------------------------------------------------------
      //For each of the Nc Maxima/Cycles:  Find SOS/MOS/EOS + associated Ys-values
      //-------------------------------------------------------------------------
      Nc = -1;
        for(e=0;e<Ne1;e++)
```

```
        {
            if(SGM[e][2] == 0) { continue; }
//Skip minima
         Nc++;
//Nc is zero-based

         //MOS: Dekad & Value
         //"""""""""""""""""""
         Tmos = SGM[e][0]; CYC[Nc].MOS  = Tmos;
//Alias Tmax
         Ymos = SGM[e][1]; CYC[Nc].Ymos = Ymos;
//Alias Ymax

         //SOS: Dekad & Value
         //"""""""""""""""""""
         if(e == 0) { CYC[Nc].SOS = -1; CYC[Nc].Ysos = -1; }
//MIN1 lacks => SOS undefined
         else
         {
              Tmin = SGM[e-1][0]; Ymin = SGM[e-1][1];
//Position & Value of preceding MIN
              Ysos  = Ymin + FEN.Fsos * (Ymos - Ymin);
//Y-threshold for SOS
            i = 0; Un = 0;

            if(FEN.Ur > FEN.Fsos)
            {
                 Ut = -1; Uy = Ymin + FEN.Ur * (Ymos - Ymin);
//Y-threshold (somewhat higher) to possibly Update SOS (if saddle between MIN1 and
MAX)
               for(T=Tmos; T>=Tmin; T--)
//Backwards from MAX to previous MIN
                {
                 if(Ys[T] < Uy  ) { Un++; if(Ut == -1)  { Ut = T; } }
//Ut/Un = First/Number of such saddle points
                 if(T     < Tmos) { if(Ys[T] > Ys[T+1]) { i = 1; } }            //i
= 0: Y-values are systematically decreasing.
                }
            }

            if(i == 1 && (((double)Un) / (Tmos - Tmin + 1)) > FEN.Uf) { Tsos = Ut; }
            else
            {
                 for(T=Tmos; T>=Tmin; T--)
//Backwards from MAX to previous MIN
                 {
                     Y = Ys[T];
                  if(Y <= Ysos)
                     {
                         Tsos = T;
//Closer to Tmin
                     if(Tsos==Tmos) { break; }
                         if( (Ys[T + 1] - Ysos) <= (Ysos - Y) ) { Tsos++; }
//Closer to Tmos
                         break;
                     }
                  if(T == Tmin) { Tsos = Tmin; break; }
//Precaution: Tsos cannot be < Tmin
```

```
                }
            }

            if(Tsos == Tmin) { Tsos++; }
//Must be:  Tmin < Tsos < Tmos
            if(Tsos == Tmos) { Tsos--; }
            CYC[Nc].SOS  = Tsos; CYC[Nc].Ysos = Ys[Tsos];
        }

        //EOS: Dekad & Value
        //"""""""""""""""""""
        if(e == Ne1 - 1) { CYC[Nc].EOS = -1; CYC[Nc].Yeos = -1; }
//MIN2 lacks => EOS undefined
        else
        {
             Tmin = SGM[e+1][0]; Ymin = SGM[e+1][1];
//Position & Value of following MIN.
             Yeos = Ymin + FEN.Feos * (Ymos - Ymin);
//Y-threshold for EOS
            i = 0; Un = 0;

            if(FEN.Ur > FEN.Feos)
            {
                 Ut = -1; Uy = Ymin + FEN.Ur * (Ymos - Ymin);
//Y-threshold (somewhat higher) to possibly Update EOS (if saddle between MAX and
MIN2)
               for(T=Tmos; T<=Tmin; T++)
//Forewards from MAX to following MIN2
                {
                 if(Ys[T] < Uy  ) { Un++; if(Ut == 0)   { Ut = T; } }
//Ut/Un = First/Number of such saddle points
                 if(T     > Tmos) { if(Ys[T] > Ys[T-1]) { i = 1;  } }           //i
= 0: Y-values are systematically decreasing.
                }
            }

            if(i == 1 && (((double)Un) / (Tmin - Tmos + 1)) > FEN.Uf) { Teos = Ut; }
            else
            {
                 for(T=Tmos; T<=Tmin; T++)
//Forwards from MAX to following MIN2
                 {
                     Y = Ys[T];
                  if(Y <= Yeos)
                     {
                         Teos = T;
//Closer to Tmin
                     if(Teos==Tmos) { break; }
                         if( (Ys[T - 1] - Yeos) <= (Yeos - Y) ) { Teos--; }
//Closer to Tmos
                         break;
                     }
                  if(T == Tmin) { Teos = Tmin; break; }
//Precaution: Teos cannot be > Tmin
                 }
            }
```

```
            if(Teos == Tmin) { Teos--; }
//Must be:   Tmos < Teos < Tmin
            if(Teos == Tmos) { Teos++; }
            CYC[Nc].EOS  = Teos; CYC[Nc].Yeos = Ys[Teos];
         }
      }
      Nc++;
//CRUCIAL: Now Nc=1-based.

      //NB: At this stage, still possible: Nc = 0 => CYC[Nc] empty/undefined &
Lengths/Areas skipped. That happens if Ne1=1 and if that is a MIN.

      //-------------------------------------
      //Compute Season Lenths (when possible)
      //-------------------------------------
      for(C=0;C<Nc;C++)
      {
         if(CYC[C].SOS == -1 || CYC[C].EOS == -1) { CYC[C].L = -1; }
         else                                     { CYC[C].L = CYC[C].EOS -
CYC[C].SOS + 1; }
      }
      //-------------------------------------
      //Compute Season Areas (when possible)     NB: Base = Mean of Ysos and Yeos
      //-------------------------------------
      for(C=0;C<Nc;C++)
      {
         if(CYC[C].SOS == -1 || CYC[C].EOS == -1) { CYC[C].Area = -1; continue; }
         CYC[C].Area = 0; Ymin = (CYC[C].Ysos + CYC[C].Yeos) / 2.0;
         for(v=CYC[C].SOS;v<=CYC[C].EOS;v++)
         {
            Y = Ys[v] - Ymin; if(Y > 0.0) { CYC[C].Area = CYC[C].Area + Y; }
         }
      }

if(TESTpix)
{
   fprintf(fpe, "Nc=%d  (Cycles after Characteristics)\n", Nc);
   for(C=0;C<Nc;C++) { fprintf(fpe, "%3d, %3d, %3d, %3d, %3d, %3d, %7.3f, %7.3f,
%7.3f, %7.3f\n", C+1, CYC[C].SOS+1, CYC[C].MOS+1, CYC[C].EOS+1, CYC[C].L,
CYC[C].Keep, CYC[C].Ysos, CYC[C].Ymos, CYC[C].Yeos, CYC[C].Area); }
   fprintf(fpe, "\n\n");
}

      //=================================================
      //5. SEASONS in CENTRAL YEAR: SELECT & REDUCE to TWO
//Define CYC[Nc].Keep = 1 for Cycles belonging to Central Year Yt
      //=================================================
      //---------------------------
      //Initial/Default assignments
//Normally: FEN.Dek1=36, FEN.Dek2=71. But possibly shifted via FEN.s2yDt.
      //---------------------------
      Nc = -1;
//We'll scan the maxima in SGM() and re-count Nc = Nr. of maxima/cycles (0-based)
        for(e=0;e<Ne1;e++)
//Scan Maxima/cycles
        {
         if(SGM[e][2] == 0) { continue; }                                  //Skip
MINima
```

```
         Nc++;                                                                  //This
is a new MAX/Cycle    NB: Including the bad ones with SOS/EOS undefined

         CYC[Nc].Keep = 0;
//Default = Don't keep. Always OK for cycles with EOS in Previous Year Yp, or MOS in
Next Year Yn.
         if(FEN.S2YM == 0) { T = CYC[Nc].MOS; } else  { T = CYC[Nc].EOS; }
//Default assignment based on: 0=MOS or 1=EOS
         if(T >= FEN.Dek1 && T <= FEN.Dek2) { CYC[Nc].Keep = 1; }             //T
falls in Yc.
      }
      Nc++;
//CRUCIAL: Now Nc=1-based.

      //---------------------------------
      //Possible adjustments for EOS-based
      //---------------------------------
      if(FEN.S2YC == 0) { goto SKIPadjust; }
//Always if S2YM = 0 (MOS-based)

      Nc = -1;
//We'll again scan the maxima in SGM() and re-count Nc = Nr. of maxima/cycles (0-
based)
      E1 = -1; E2 = -1;                                                       //For
FEN.S2YM=1 (EOS): Dubious seasons with MOS-EOS around NewYear & assigned to Yt, at
start (E1) or end (E2)
        for(e=0;e<Ne1;e++)
//Scan Maxima/cycles
        {
         if(SGM[e][2] == 0) { continue; }                                      //Skip
MINima
         Nc++;                                                                 //This
is a new MAX/cycle    NB: Including the bad ones with SOS/EOS undefined

         //Re-assignment1:                                                     //EOS
falls in Yt, MOS in Previous Year Yp, but only small fraction of cycle lies in Yt =>
KEEP=0
         //"""""""""""""""
//Deficient cycles: MOS always defined and EOS was OK (see above).
         if(CYC[Nc].Keep == 1 && CYC[Nc].MOS < FEN.Dek1)                       //EOS
falls in Yt (via default), but MOS in Yp
         {
            E1 = Nc;                                                           //E1 =
This cycle is assigned to Yt, but that might be dubious
            if(FEN.S2YC == 1)
            {
               if(CYC[Nc].EOS < FEN.Dek1 + FEN.S2YDd) { E1 = 0; CYC[Nc].Keep = 0; }
//Correct: This cycle belongs to previous year Yp. No longer dubious!
            }
            else
            {
               if(FEN.S2YC == 2)
               {
                  A1 = FEN.Dek1 - CYC[Nc].MOS;
//A1=Length from MOS to NewYear
                  A2 = CYC[Nc].EOS - FEN.Dek1 + 1;
//A2=Length from NewYear to EOS
               }
```

```
                else
                {
                   A1 = 0; A2 = 0; Ymin = CYC[Nc].Yeos;
                   for(v=CYC[Nc].MOS;v<FEN.Dek1;v++)
                   {
                      Y = Ys[v] - Ymin; if(Y > 0.0) { A1 = A1 + Ys[v]; }
//A1=Area from MOS to NewYear
                   }
                   for(v=FEN.Dek1;v<=CYC[Nc].EOS;v++)
                   {
                      Y = Ys[v] - Ymin; if(Y > 0.0) { A2 = A2 + Ys[v]; }
//A2=Area from NewYear to EOS
                   }
                }
                Y = A2 / (A1 + A2);                                                    //Y
= Fraction falling in Yt
                if(Y < FEN.S2YT) { E1 = -1; CYC[Nc].Keep = 0; }
//Too small => Remove cycle and reset E1: This cycle belongs to previous year Yp (no
longer dubious)
             }
          }

          //Re-assignment2:                                                          //EOS
falls in next year Yn, MOS in central Year Yt, but largest fraction of cycle lies in
Yt => KEEP=1
          //"""""""""""""""
//Deficient cycles: MOS always defined and EOS was OK (see above).
          if(CYC[Nc].EOS > FEN.Dek2 && CYC[Nc].MOS <= FEN.Dek2)
          {
             if(FEN.S2YC == 1)
             {
                if(CYC[Nc].EOS <= FEN.Dek2 + FEN.S2YDd) { E2 = 1; CYC[Nc].Keep = 1; }
//Correct: This cycle belongs to central year Yt. Dubious case!
             }
             else
             {
                if(FEN.S2YC == 2)
                {
                   A1 = FEN.Dek2 - CYC[Nc].MOS + 1;
//A1=Length from MOS to NewYear
                   A2 = CYC[Nc].EOS - FEN.Dek2;
//A2=Length from NewYear to EOS
                }
                else
                {
                   A1 = 0; A2 = 0; Ymin = CYC[Nc].Yeos;
                   for(v=CYC[Nc].MOS;v<=FEN.Dek2;v++)
                   {
                      Y = Ys[v] - Ymin; if(Y > 0.0) { A1 = A1 + Ys[v]; }
//A1=Area from MOS to NewYear
                   }
                   for(v=FEN.Dek2+1;v<=CYC[Nc].EOS;v++)
                   {
                      Y = Ys[v] - Ymin; if(Y > 0.0) { A2 = A2 + Ys[v]; }
//A2=Area from NewYear to EOS
                   }
                }
```

```
                Y = A2 / (A1 + A2);                                                    //Y
= Fraction falling in Yn
                if(Y < FEN.S2YT) { E2 = Nc; CYC[Nc].Keep = 1; }
//This cycle is re-assigned to central year Yt. Correct E2: might be dubious.
             }
          }
       }
       Nc++;
//CRUCIAL: Because we started from zero, now again Nc=1-based.

       //-------------------------------------------------------------------------
---------
       //Two dubious assignments (around NewYear) to Yc. Most probably, one must be
eliminated.     NB: Holds for S2YC=1/2/3.
       //-------------------------------------------------------------------------
---------
       if(FEN.S2YO > 0.0 && E1 > -1 && E2 > -1)
       {
          j = CYC[E1].L; i = CYC[E2].L;
//Length of both cycles
          if(j > -1 && i > -1)
//Elimination impossible if one of both seasons is deficient (at edges)
          {
                                                    if(i > j    ) { j    = i; }
//J =length of longest season
             Tsos = CYC[E1].SOS; T = CYC[E2].SOS - 36; if(T > Tsos) { Tsos = T; }
//Tsos=is latest   dekad of both SOS
             Teos = CYC[E1].EOS; T = CYC[E2].EOS - 36; if(T < Teos) { Teos = T; }
//Teos=is earliest dekad of both EOS
             i    = Teos - Tsos + 1;
//i =length of overlapping period wrt longest season
             if(((double)i) / j >= FEN.S2YO) { CYC[E1].Keep = 0; }
//Eliminate first of both seasons. Else: Weak overlap and nothing happens
          }
       }

SKIPadjust:

       //-----------------------------------------
       //Nss = Nr. of Cycles/Seasons in Central Year
       //-----------------------------------------
       Nss = 0;
       for(C=0;C<Nc;C++)
       {
          if(CYC[C].Keep == 1) { Nss++; }
       }

if(TESTpix)
{
   fprintf(fpe, "Nc=%d (Cycles, now with KEEP=1)\n", Nc);
   for(C=0;C<Nc;C++) { fprintf(fpe, "%3d, %3d, %3d, %3d, %3d, %3d, %7.3f, %7.3f,
%7.3f, %7.3f\n", C+1, CYC[C].SOS+1, CYC[C].MOS+1, CYC[C].EOS+1, CYC[C].L,
CYC[C].Keep, CYC[C].Ysos, CYC[C].Ymos, CYC[C].Yeos, CYC[C].Area); }
   fprintf(fpe, "\n");
   fprintf(fpe, "Ns=%d,  Dek1/2=%d/%d, S2YM=%d\n", Nss, FEN.Dek1, FEN.Dek2,
FEN.S2YM);
   fprintf(fpe, "\n\n");
}
```

```
      //-------------------------------------
      //Trap Error W= 17 (Nss=0 at this stage)                        //NB: This is
still possible, eg. when original seasons in Yt were all assigned to Yp and/or Yn.
      //-------------------------------------
      if(Nss == 0)
      {
         N251b++; W = 17;                                             //W = 17: There
is seasonality in 3-yearly profile, but no seasons in Yc
         NssOri = 0;                                                  //Memorize this
for KK0 OUT-IMG
         goto OUTPUTS;                                                //There we'll
output: 1-3: Normal, 4-22: 251
      }

      //-------------------------------------------------
      //If Nss > 2 Then Remove Seasons with Smallest Area             At the end:
Nss=2
      //-------------------------------------------------
      while(Nss > 2)
      {
         A1 = 1E+38; i = -1;                                          //NB: Deficient
seasons at edges (no Tsos/Teos) have A=-1. Hence they are first removed.
         for(C=0;C<Nc;C++)
         {
            if(CYC[C].Keep == 0) { continue; }
            if(CYC[C].Area < A1) { A1 = CYC[C].Area; i = C; }
         }
         CYC[i].Keep = 0; Nss--;
      }

      //=================================================
      //6. SEASONS in CENTRAL YEAR: OPTIONAL ADAPTATIONS
NB: At this stage Nss = 1/2.
      //=================================================
      //Copy CYC(Nc) to SSN (Nss)
      //-------------------------
      Nss = 0;
      for(C=0;C<Nc;C++)
      {
         if(CYC[C].Keep == 0) { continue; }
         SSN[Nss] = CYC[C];
         Nss++;
      }

      //---------------------------------
      //Optionally only Keep 1 OUT-Season
NB: At this stage Ns=1/2. And finally table SSN(1 to Ns) is kept intact.
      //---------------------------------
      NssOri = Nss;
//Memorize this for KK0 OUT-IMG
      if(Nss == 2 && FEN.o1 > 0)
      {
         Nss = 1; i = 0;
//Output only for 1 season
         switch(FEN.o1)
         {
```

```
            case 1:                                                 ; break;
//Keep 1st season
            case 2:                                       i = 1     ; break;
//Keep 2nd season
            case 3: if(SSN[0].L    < SSN[1].L   ) { i = 1; }; break;
//Keep Longest season
            case 4: if(SSN[0].Area < SSN[1].Area) { i = 1; }; break;
//Keep Biggest season (~ Area)
          }
          if(i) { SSN[0] = SSN[1]; }
      }


//=====================================================================================
====================================
      //7. FIND ERRORS (1-8) & WARNINGS (9-17). As Before but now no more
Corrections are made (Most errors are removed before)

//=====================================================================================
====================================
      //NB: As in previous program versions, but now:
      //    - Most former errors are avoided
      //    - No more corrections at the end: If Error then Flag OUT-IMGs (252). If
Warning then retrieve normal outputs.
      //      In both cases: mention in TXT-file with Errors/Warnings.
      //NB: Three cases already trapped above (W <> 0) and then directly sent to
OUTPUTS:
      //    - W=-1: No seasonality at all. Special case. Counted via N251a.
      //    - W= 1: Non-allowed sequence of MIN-MIN or MAX-MAX. Real error.
      //    - W=17: Good profile but no seasons in central year Yt. Counted via
N251b.
      //'"""""""""""
      //strcpy(Emes[ 0], "No Errors");
      //
      //strcpy(Emes[ 1], "E: Bad sequence MAX-MAX or MIN-MIN");
      //strcpy(Emes[ 2], "E: Good Profile but no seasons Found");
      //strcpy(Emes[ 3], "E: SOS1 or EOS1 undefined");
//NO ERROR: Possible for first/last season
      //strcpy(Emes[ 4], "E: SOS2 or EOS2 undefined");
      //strcpy(Emes[ 5], "E: SOS1 >= EOS1");
      //strcpy(Emes[ 6], "E: SOS2 >= EOS2");
      //strcpy(Emes[ 7], "E: MOS1 beyond [SOS1-EOS1]");
      //strcpy(Emes[ 8], "E: MOS2 beyond [SOS2-EOS2]");
      //
      //strcpy(Emes[ 9], "W: LEN1 > 36 dekads");
      //strcpy(Emes[10], "W: LEN2 > 36 dekads");
      //strcpy(Emes[11], "W: 2 seasons & LEN1+2 > 36 dekads");
//NB: ALWAYS POSSIBLE (multi-annual crops)
      //strcpy(Emes[12], "W: 2 seasons & EOS1 >= SOS2 - 1");
      //strcpy(Emes[13], "W: MOS1=SOS1");
      //strcpy(Emes[14], "W: MOS2=SOS2");
      //strcpy(Emes[15], "W: MOS1=EOS1");
      //strcpy(Emes[16], "W: MOS2=EOS2");
      //strcpy(Emes[17], "W: No seasons found in Central Year");

//=====================================================================================
====================================
      //General
```

```
      //********
      if(Nss == 0)                              { W = 2;       goto OUTPUTS; }  //Quasi
impossible now

      //Per Season
      //**********
      Ltot = 0;
//Length of both seasons
      for(ss=0;ss<Nss;ss++)
      {
         Tsos = SSN[ss].SOS; Tmos = SSN[ss].MOS; Teos = SSN[ss].EOS;

         if(Tsos == -1 || Teos == -1)           { W =  3 + ss; goto OUTPUTS; }  //W=3/
4: Tsos/Teos unknown
         if(Tsos >= Teos)                       { W =  5 + ss; goto OUTPUTS; }  //W=5/
6: Tsos >= Teos
         if(Tmos <  Tsos || Tmos > Teos)        { W =  7 + ss; goto OUTPUTS; }  //W=7/
8: MOS beyond [SOS-EOS]

         if(Tmos == Tsos)                       { W = 13 + ss; goto OUTPUTS; }
//W=13/14: MOS=SOS
         if(Tmos == Teos)                       { W = 15 + ss; goto OUTPUTS; }
//W=15/16: MOS=EOS

         j = SSN[ss].L; Ltot = Ltot + j;
         if(j > 36)                             { W =  9 + ss; goto OUTPUTS; }
//W=9/10: Season longer than 36 dekads
      }

      //If two seasons
      //***************
      if(Nss == 2)
      {
         if(Ltot > 36)                          { W = 11;      goto OUTPUTS; }
//W=11: Total season length too long
         if(SSN[0].EOS >= SSN[1].SOS - 1)       { W = 12;      goto OUTPUTS; }
//W=12: EOS1 >= SOS2 - 1
      }


//=============================================================================
====================================
      //8. OUTPUTs: Store Results in OUT-BUFFERS

//=============================================================================
====================================
      OUTPUTS:

      if(Nss==2) { Ns2++; }
//Pixels with 2 seasons

      //-------------------------------------
      //Output Problematic Pixels to TXT-File
      //-------------------------------------
      //fprintf(fpe, "        N,    COL,    REC, EW, ERROR/WARNING
, MSK\n");
      if(W > 0)
//Exclude W = -1 (no seasonality at all), but Keep W = 17 (no season in Yc)
```

```
      {
         NerrT = NerrT + 1; Nerr[W] = Nerr[W] + 1;
         fprintf(fpe, "%9I64d,%6ld,%6ld,%2d, %s, %d\n", NerrT, col+1, rec+1, W,
Emes[W], Vm);
      }

      //-----------------------------------------------------
      //Optionally Replace smoothed Ys by Original Yi-Values (but Yi is interpolated
and hence free of missing values)
      //-----------------------------------------------------
      if(FEN.oYsm == 0)
      {
         for(v=0;v<NvI;v++) { Ys[v] = Yi[FEN.RMl + v]; }

         for(ss=0;ss<Nss;ss++)
         {
            i = SSN[ss].SOS; if(i > -1) { SSN[ss].Ysos = Ys[i]; }
            i = SSN[ss].MOS; if(i > -1) { SSN[ss].Ymos = Ys[i]; }
            i = SSN[ss].EOS; if(i > -1) { SSN[ss].Yeos = Ys[i]; }
         }

         Ya0 = 0.0; Ymin = 3E+38; Ymax = -Ymin;
//        for(v=36; v<72; v++)
          for(v=FEN.Dek1; v<=FEN.Dek2; v++)
          {
               Y = Ys[v]; Ymin = min(Y, Ymin); Ymax = max(Y, Ymax); Ya0 = Ya0 + Y;
          }
          Ya0 = Ya0 / 36.; Yr0 = Ymax - Ymin;
// Overall annual Y-mean (Ya0) and Y-range (Yr0)
      }

//"""""""""""""""""""""""""""""""
//char              fo_suf[NvO][4]  = { "Ya0", "Yr0", "KK0",
// 00-02: Annual values (independent of seasons) : Y-mean, Y-range, Classification
//                                          "s1",  "m1",  "e1",  "L1",  "s2",  "m2",
"e2",  "L2",    // 03-10: Dekads of SOS, MOS, EOS & Season length,  for 2 seasons
//                                         "Ys1", "Ym1", "Ye1", "Ya1", "Ys2", "Ym2",
"Ye2", "Ya2",    // 11-18: Values at SOS, MOS, EOS & Season average, for 2 seasons
//                                         "LT0", "LR0", "SI0" };
// 19-21: Annual values for both seasons together: length Tl1+Tl2, Tl1/(Tl1+Tl2),
Seasonality Index
//NB: Three first OUT-IMGs are always filled, also in case of errors/warnings.
//"""""""""""""""""""""""""""""""
      //-------------------------------------
      //First three OUT-IMGs are always filled - also in case of errors/warnings.
      //-------------------------------------
      //Ya0 (Ymu)
      //""""""""""
       v = 0; x = floor(0.5 + (Ya0 - ho[v].Vint) / ho[v].Vslo); x = max(ho[v].Vlo,
min(ho[v].Vhi, x));
        ho[v].Vmin = min(ho[v].Vmin, x); ho[v].Vmax = max(ho[v].Vmax, x);
        buf[v][col] = (unsigned char)x;

      //Yr0 (Yrg)
      //""""""""""
        v = 1; x = floor(0.5 + (Yr0 - ho[v].Vint) / ho[v].Vslo); x = max(ho[v].Vlo,
min(ho[v].Vhi, x));
        ho[v].Vmin = min(ho[v].Vmin, x); ho[v].Vmax = max(ho[v].Vmax, x);
```

```
        buf[v][col] = (unsigned char)x;

      //kk0 (PHENO-CLASS)
      //""""""""""""""""""
        v = 2;
        i = floor((Ya0 - FEN.oMUl) / FEN.oMUd); i = max(0, min(i, 4));
//i=Class of Ymean , forced to [0-4]
      j = floor((Yr0 - FEN.oRGl) / FEN.oRGd); j = max(0, min(j, 4));
//j=Class of Yrange, forced to [0-4]
        kk0 = 100 * NssOri + 10 * i + j;
//kk0 [0-244]. NB: Use original NssOri (1/2), in case only 1 OUT-season is desired.

      U = (unsigned char)max(ho[v].Vlo, min(ho[v].Vhi, kk0));
//Force to range 0-244
        ho[v].Vmin = min(ho[v].Vmin, U); ho[v].Vmax = max(ho[v].Vmax, U);
      buf[v][col] = U;

      //-----------------------------------------------
      //Flag other OUT-IMGs for special cases & Errors    NB: For Warnings (W>8):
keep normal outputs.
      //-----------------------------------------------
      if(W < 0 ||  W == 17) { for(v=3;v<NvO;v++) { buf[v][col] = 251; }
continue; } //251: No seasons at all (W=-1) or in central year (W=17)
      if(W > 0 &&  W <   9) { for(v=3;v<NvO;v++) { buf[v][col] = 252; }  N252++;
continue; } //252: Processing error

     //-------------
      //OUT-IMGs 3-18:           For Nss Seasons (1/2): SOS/MOS/EOS: dekads & Y-
values + Season Lenths
      //-------------
      for(ss=0;ss<2;ss++)
      {
         //What with ss=1 (second season)?
         //""""""""""""""""""""""""""""""""
         if(ss == 1)
         {
            if(FEN.o1 > 0) { break; }                                    //Only 1
season asked
            if(Nss == 1)
            {
               for(v= 7;v<=10;v++) { buf[v][col] = 251; }                //251: No
season2
               for(v=15;v<=18;v++) { buf[v][col] = 251; }
               break;                                                    //Leave ss-
Loop
            }
         }

         //Dekads of SOS/MOS/EOS and L=Season Length                     //So far in
range [0 - 107], so we must add 1.
         //""""""""""""""""""""""""""""""""""""""""""""""
         C = ss * 4 + 2;
         for(i=1;i<5;i++)
         {
            switch(i)
            {
               case 1: j = SSN[ss].SOS; k = j + 1; break;
               case 2: j = SSN[ss].MOS; k = j + 1; break;
```

```
               case 3: j = SSN[ss].EOS; k = j + 1; break;
               case 4: j = SSN[ss].L  ; k = j    ; break;
            }
            v = C + i;                                                    //Season1: v
= 3 - 6, Season2: v = 7 - 10
            if(j < 0)  { U = 252; }                                       //252:
Processing error NB: Should be impossible here.
            else
            {
               U = (unsigned char)max(ho[v].Vlo, min(ho[v].Vhi, k));
                 ho[v].Vmin = min(ho[v].Vmin, U); ho[v].Vmax = max(ho[v].Vmax, U);
            }
            buf[v][col] = U;
         }

         //VI-values at SOS/MOS/EOS and Mean over Season
         //""""""""""""""""""""""""""""""""""""""""""""""""""
         C = ss * 4 + 10;
         for(i=1;i<5;i++)
         {
            switch(i)
            {
               case 1: j = SSN[ss].SOS; Y = SSN[ss].Ysos; break;
               case 2: j = SSN[ss].MOS; Y = SSN[ss].Ymos; break;
               case 3: j = SSN[ss].EOS; Y = SSN[ss].Yeos; break;
               case 4: j = SSN[ss].L  ;
                       if(j < 1) { j = -1; }
                       else      { Y = 0.0; for(v=SSN[ss].SOS;v<=SSN[ss].EOS;v++) {
Y = Y + Ys[v]; } Y = Y / j; }     //Y = VI-Mean over season
                       break;
            }
            v = C + i;                                                    //Season1: v
= 11 - 14, Season2: v = 15 - 18
            if(j < 0)  { x = 252; }                                       //252:
Processing error NB: Should be impossible here.
            else
            {
                x = floor(0.5 + (Y - ho[v].Vint) / ho[v].Vslo); x = max(ho[v].Vlo,
min(ho[v].Vhi, x));
                 ho[v].Vmin = min(ho[v].Vmin, x); ho[v].Vmax = max(ho[v].Vmax, x);
            }
              buf[v][col] = (unsigned char)x;
         }
      }

      //----------------------------------------
      //Last three OUT-IMGs with Annual Statistics
      //----------------------------------------
      //LT0 (LEN1+2)          NB: Already computed before in Ltot [0-108].  For
pixels with Ns=1: LT0 is identical to L1
      //"""""""""""""
       v = 19; x = floor(0.5 + (Ltot - ho[v].Vint) / ho[v].Vslo); x = max(ho[v].Vlo,
min(ho[v].Vhi, x));
        ho[v].Vmin = min(ho[v].Vmin, x); ho[v].Vmax = max(ho[v].Vmax, x);
        buf[v][col] = (unsigned char)x;


      //LR0 (LEN1 / LEN1+2)    NB: For pixels with Ns=1: LR0 is 100% (V=200).
```

```
      //"""""""""""""""""""""
       v = 20; Y = (100.0 * SSN[0].L) / Ltot; x = floor(0.5 + (Y - ho[v].Vint) /
ho[v].Vslo); x = max(ho[v].Vlo, min(ho[v].Vhi, x));
        ho[v].Vmin = min(ho[v].Vmin, x); ho[v].Vmax = max(ho[v].Vmax, x);
        buf[v][col] = (unsigned char)x;

      //SI0 (Yg/Ymu - 1)    NB: Yg=Mean over 1/2 seasons. Yg and Ymu both computed
over 36 dekads of central year (FEN.Dek1/2).
      //"""""""""""""""""""
        v = 21;
      if(foM[v])
      {
         if(fabs(Ya0) < 0.001) { buf[v][col] = 252; continue; }
//Division by 0 =>Error flag only for this OUT-IMG. This seems IMPOSSIBLE!
         Y = 0.0; Ltot = 0;
//Mean of Yi over green dekads in central year in seasons 1/2
         for(ss=0;ss<Nss;ss++)
         {
//          for(i=SSN[ss].SOS;i<=SSN[ss].EOS;i++) { if(i > 35 && i < 72) { Y = Y +
Ys[i]; Ltot++; } }
            for(i=SSN[ss].SOS;i<=SSN[ss].EOS;i++) { if(i >= FEN.Dek1 && i <=
FEN.Dek2) { Y = Y + Ys[i]; Ltot++; } }
         }
         Y = Y / Ltot; Y = 100. * ((Y / Ya0) - 1.0);
          x = floor(0.5 + (Y - ho[v].Vint) / ho[v].Vslo); x = max(ho[v].Vlo,
min(ho[v].Vhi, x));
           ho[v].Vmin = min(ho[v].Vmin, x); ho[v].Vmax = max(ho[v].Vmax, x);
           buf[v][col] = (unsigned char)x;
      }

   }  // NEXT Col

   //---------------------------------
    //REC: Write NvO buffers to outfiles
   //---------------------------------
   for(v=0; v<NvO; v++) { if(foM[v]) { fwrite(buf[v], 1, Ncol, fpo[v]); } }
    if ((rec+1) >= (long)floor(progr40)) { progression(1, Nrec, &progr40,
&progr40_step, 1, "", "", "", &GUIdone, &GUIdone_step); }

}  // Next Rec

printf("\n"); printf("\n");
_fcloseall();



//======================
                                                          //  E. POST-
PROCESSING

//======================
strcpy(stage, "POSTPROCESSING");
printf("CREATED:\n");

//----------------------
// E1. Create OUT-Headers
//----------------------
```

```
for(v=0; v<NvO; v++) { if(foM[v]) { envi_hdr_create(fo[v], &ho[v], IDRISI, ARCVIEW);
} }
printf("- HDR-files\n");

//---------------------------------
// E2. Create VAR-file with OUT-IMGs
//---------------------------------
sprintf(t, "%s%s", fo_pre, ".VAR");
if ((fp = fopen(t, "wt")) == NULL) { sprintf(ERRmess, "Creating VAR-file: %s", t);
ERROR(stage, ERRmess, 1); }
fprintf(fp, "PHENO-IMGs generated by program %s.exe (V%d/%d)\n", PROGNAME,
PROGVERSION, LIBVERSION);
fprintf(fp, "- %s\n", ho[0].description);
fprintf(fp, "- %s\n", ho[0].comment);

fprintf(fp, "  ---  ---  --------\n");
fprintf(fp, "   Vu  COD  IMG-NAME (without extension *.img/hdr)\n");
fprintf(fp, "  ---  ---  --------\n");
for(v=0; v<NvO; v++)
{
   if(foM[v]==0) { continue; }
   strcpy(s, fo[v]); s[strlen(s) - 4] = 0;                //Remove extension ".img"
    fprintf(fp, "%5d%5s  %s\n", v+1, fo_suf[v], s);
}
fclose(fp);
printf("- VAR-file: %s\n", t);

//---------------------------------
// E3. Create MTA-file with OUT-IMGs
//---------------------------------
sprintf(t, "%s%s", fo_pre,".MTA");
if ((fp = fopen(t, "wt")) == NULL) { sprintf(ERRmess, "Creating ENVI MTA-file: %s",
t); ERROR(stage, ERRmess, 1); }
fprintf(fp, "ENVI META FILE\n");
for(v=0; v<NvO; v++)
{
   if(foM[v]==0) { continue; }
   fprintf(fp, "File : %s\n", fo[v]);                     //Keep extension "*.img"
   fprintf(fp, "Bands: 1\n");
   fprintf(fp, "Dims : 1-%ld,1-%ld\n\n", Ncol, Nrec);
}
fclose(fp);
printf("- MTA-file: %s\n", t);

//-----------------------------------------------
// E4. File with Errors/Warnings: Add Statistics
//-----------------------------------------------
sprintf(t, "%s%s", fo_pre, ".TXT");
if ((fpe = fopen(t, "at")) == NULL) { sprintf(ERRmess, "Addings Statistics to TXT-
file: %s", t); ERROR(stage, ERRmess, 1); }
fprintf(fpe,
"*****************************************************************************\n")
;
fprintf(fpe, "ERRORS (OUT=252) & WARNINGS (OUT=Normal):\n");
fprintf(fpe, " COD          Npix  MEANING\n");
for(W=1;W<18;W++)
{
```

```
   fprintf(fpe, "-%3d  %12I64d  %s\n", W, Nerr[W], Emes[W]); if(W == 8) {
fprintf(fpe, "\n"); }
}
fprintf(fpe, "-------------------------------------------------------------------
---------\n");
Y = (double) (100. / hi.pixels);
i64 = hi.pixels - N255 - N254 - N253 - N252 - N251a - N251b;
//Land pixels with normal results (Nss=1/2)
Ns1 = i64 - Ns2;

fprintf(fpe, "        CATEGORY                          :     PIXELS    PIXELS%%\n");
fprintf(fpe, "0-255 = TOTAL                             : %10I64d   %7.3f%%\n",
hi.pixels, Y*hi.pixels);
fprintf(fpe, "  255 = Sea : all IN-values flagged       : %10I64d   %7.3f%%\n", N255
, Y*N255);
fprintf(fpe, "  254 = Land: pixels removed by the mask: %10I64d   %7.3f%%\n", N254
, Y*N254);
fprintf(fpe, "  253 = Land: too many flagged inputs     : %10I64d   %7.3f%%\n", N253
, Y*N253);
fprintf(fpe, "  252 = Land: processing error            : %10I64d   %7.3f%%\n", N252
, Y*N252);
fprintf(fpe, "  251 = Land: no seasons                  : %10I64d   %7.3f%%\n", N251a
, Y*N251a);
fprintf(fpe, "              no seasons in target year : %10I64d   %7.3f%%\n", N251b
, Y*N251b);
fprintf(fpe, "1-108 = Land: Normal outputs              : %10I64d   %7.3f%%\n", i64
, Y*i64 );
fprintf(fpe, "        - One single season               : %10I64d   %7.3f%%\n", Ns1
, Y*Ns1 );
fprintf(fpe, "        - Two seasons                     : %10I64d   %7.3f%%\n", Ns2
, Y*Ns2 );
fprintf(fpe, "-------------------------------------------------------------------
---------\n");
fprintf(fpe, "ELIMINATION of EXTREMES in Pixel Profiles: Nr. of times each test was
Applied (see file %s):\n", fi_SPF);
for (i=1;i<7;i++)
{
   fprintf(fpe, "- Test %d: %20I64d\n", i, NE[i]);
}

fclose(fpe);
printf("- TXT-file: %s\n", t);
printf("\n");

//----------------
// C5. Termination
//----------------
   //Pixel distribution
   //"""""""""""""""""""
Y = (double) (100. / hi.pixels);
i64 = hi.pixels - N255 - N254 - N253 - N252 - N251a - N251b;
//Land pixels with normal results (Nss=1/2)
Ns1 = i64 - Ns2;

printf("                                        :     PIXELS    PIXELS%%\n");
printf("0-255 = TOTAL                           : %10I64d   %7.3f%%\n", hi.pixels,
Y*hi.pixels);
```

```
printf("  255 = Sea : all IN-values flagged     : %10I64d   %7.3f%%\n", N255     ,
Y*N255);
printf("  254 = Land: pixels removed by the mask: %10I64d   %7.3f%%\n", N254     ,
Y*N254);
printf("  253 = Land: too many flagged inputs   : %10I64d   %7.3f%%\n", N253     ,
Y*N253);
printf("  252 = Land: processing error          : %10I64d   %7.3f%%\n", N252     ,
Y*N252);
printf("  251 = Land: no seasons                : %10I64d   %7.3f%%\n", N251a    ,
Y*N251a);
printf("              no seasons in target year : %10I64d   %7.3f%%\n", N251b    ,
Y*N251b);
printf("1-108 = Land: Normal outputs            : %10I64d   %7.3f%%\n", i64      ,
Y*i64 );
printf("        - One single season             : %10I64d   %7.3f%%\n", Ns1      ,
Y*Ns1 );
printf("        - Two seasons                   : %10I64d   %7.3f%%\n", Ns2      ,
Y*Ns2 );
printf("\n");

   //Errors & Warnings
   //""""""""""""""""""
printf("PROBLEMATIC PIXELS (see OUT-file %s.txt):\n", fo_pre);
printf(" COD          Npix  MEANING\n");
for(W=1;W<18;W++)
{
   printf("-%3d  %12I64d  %s\n", W, Nerr[W], Emes[W]); if(W == 8) { printf("\n"); }
}
printf("\n");

   //Elimination tests
   //""""""""""""""""""
printf("ELIMINATION of EXTREMES in Pixel Profiles: Nr. of times each test was
Applied (see file %s):\n", fi_SPF);
for (i=1;i<7;i++)
{
   printf("- Test %d: %20I64d\n", i, NE[i]);
}

progression(2, 0, 0, 0, 0, "", "", "", &GUIdone, &GUIdone_step);
exit(0);
//***************************
}  //      END OF MAIN
//***************************

//*****************************************************************************
  void InterPolMissing(double *Y, short N, double MISSING, short *Jb, short *Je) {
//*****************************************************************************
//In situ replacement of all values <= MISSING (=Global variable = -100000)
//                    in 1D double array Y() which has N elements Y(0 to N-1)
//                    by extra/inter-polation between "good" values with > MISSING
//At Input : Jb/Je = 0 / 1 = also don't do / do EXTRApolation at left/right
//At Output: Jb/Je = first/last "good" point found in IN-series
//                  Jb=Je=-1 means no good points at all !
//*****************************************************************************
short   i, j, J1, J2, DoLeft, DoRight;
double  A, B;
```

```
DoLeft = *Jb; DoRight = *Je; *Jb = - 1; *Je = -1;

// Left Tail: extrapolation
//"""""""""""""""""""""""""""
for(i=0; i<N; i++)
{
    if(Y[i] > MISSING )
   {
      *Jb=i; if(DoLeft) { for(j=0; j<i; j++) { Y[j] = Y[i]; } } break;
   }
}
if(*Jb == -1) return;  // all values <= MISSING

// Right Tail: extrapolation
//"""""""""""""""""""""""""""
for(i=N-1; i>=0; i--)
{
    if(Y[i] > MISSING)
   {
      *Je=i; if(DoRight) { for(j=i+1; j<N; j++) {Y[j] = Y[i]; } }  break;
   }
}

// Middle: Interpolation
//""""""""""""""""""""""""
J1 = *Jb;                                  // first good value in series
while(1)
{
    J1++;
    if(Y[J1] <= MISSING)                   // first missing value => to be
interpolated
    {
        J2 = J1; J1--;                      // J1=First  good point for interpolation
        while(Y[J2] <= MISSING) { J2++; } // J2=Next

        B=(Y[J2] - Y[J1]) / (J2 - J1);     // Slope of Y = A + B * J
        A= Y[J1] - B * J1;                  // Intercept
        for(i = J1+1; i < J2; i++) { Y[i] = A + B * i; } // Interpolate in-between
values
        J1 = J2;                            // Continue with good value
    }
   if(J1 >= *Je) { break; }
}

//Test      Eliminated here.
//"""""
//for(i=0; i<N; i++)
//{
//  if(Y[i] <= MISSING ) {  ERROR("FUNCTION InterpolMissing", "Fatal error", 200); }
//}
//***************************
}  // END OF InterPolMissing
//***************************

//*************************
  short sgnd(double x) {
/*************************
Returns SIGN (+1, 0, -1) of double x
```

```
**************************/
if(x > 0.0) {return ( 1);}
if(x < 0.0) {return (-1);}
             return ( 0);
//************************
} //     END OF sgnd
//************************
```

5.5 TBP

Total Biomass Production (TBP) is defined as the sum of the above-ground dry matter produced during the growing season. TBP is expressed in kgDM/ha/day, and has thus different biomass units compared to NPP, with 1 gC/m²/day (NPP) = 22.222 kgDM/ha/day (DMP). The required inputs are the dekadal NPP values, as well as information on the start and the end of the growing season. At level 1, this start- and end of season is set to the beginning and the end of the target year, for Level 2 and 3 the phenology data components are used. In addition, a vegetation check is included to account for vegetated areas where no clear growing season is detected (e.g. evergreen forest). In the absence of a growing season, the annual mean NPP value is used to determine if vegetation is present. If for a given pixel the threshold value of 0.75 gC/m²/day is exceeded, the start- and end of the growing season in that pixel will be set to the start and end of the calendar year.

```
//****************************************************************************
                        void main( int argc, char **argv )                    {
//****************************************************************************
char             s[HDRlmax+1], t[_MAX_PATH], stage[30], ss[10], m[2],
fi_Kc[_MAX_PATH], fs_pre[_MAX_PATH], fs_img[3][_MAX_PATH], fi_pre[_MAX_PATH],
fi_suf[_MAX_PATH], fo_img[_MAX_PATH];

unsigned char    Vt[NiT];

short            yyyy, yy, mm, dd, dd_st, ttm, tty, dpm, doy;

short            i, j, k, v, ok, MET, season, fi_df, Ni, NiGOOD, NiMISS, NdaysI,
MaxInGap, InGap, NiMaxMiss, TypeI, TypeO, Base, KcW, TYP[256], typ;

short                 Nbad, pX1, pX2, pSOS, pMOS, pEOS, pC1, pC2, J1, J2, yyyytar,
yyyycen, yyyypre, tt, ti1, ti2, to;

long             l, rec, col, Ncol, Nrec, date, Idate1, Idate2, Odate;

__int64           N, Nsea, Nmsk, Nnos, Nmis, Nok, NsatLO, NsatHI;

double           f, V, Vlo, Vhi, Vbase, Vi[NiT], Ymin, Ymax, Ybase, T2, A, B,
NiMaxMissPerc, Fsea, Fmsk, Fnos, Fmis, Fout, Kc[256][3], Sv, Sw, W[NiT];

double            progr40, progr40_step, GUIdone, GUIdone_step;

struct ENVIHDR   hi       , ho  , hs     , hh  ;

FILE               *fpi[NiT], *fpo, *fps[3], *fpk;

unsigned char    **bi1    , *bo1, *bs1[3], *bk1;
short            **bi2    , *bo2;
long             **bi3    , *bo3;
float            **bi4    , *bo4;

struct HIstruc
{
   long          date;                        //startdate of dekad
    float             days;                             //days in this dekad. NB:
FLOAT because days also used as weighting factors!
    short         lack;                         //0=IN-IMG present, 1=IN-IMG lacking
   char          *fi;                          //IMG-name, incl. extension
   long          offset;                       //BYTE-offset
}                *HI;                          //For (at most) 0 + 108 IN-IMGs
```

```
//=============================================================
                                    //                          A. INITIAL

//=============================================================
if(argc==1) { bat=0; } else {   if(stricmp(argv[1], "GUI")) { bat = 1; } else { bat
= 2; ++argv; } }
strcpy(PROGNAME, "PHENOcum"); PROGVERSION = 1611; strcpy(ERRmess, "");
get_GLIMPSE_settings();
if(bat==1 && argc != (NclParms+1)) { if(argc==2) { bat = 0;} CLerror(1); }   //Only
1 parm => Interactive HELP
if(bat==2 && argc != (NclParms+2)) {                          CLerror(1); }
CLerror(0);


//=============================================================
                                    //                          B. INPUTS

//=============================================================
strcpy(stage, "INPUT STAGE");
//--------------------------
// B1. IMGs with SOS/MOS/EOS
//--------------------------
   //Ask basename & Season
   //""""""""""""""""""""""
printf("    PHENOLOGICAL IN-IMGs WITH SOSs/MOSs/EOSs for SEASON s
(~PHENOdef/PHENOfix):\n");
printf(" p1.Base Name PHENO-IMGs (+drive/path if needed) =>"); Qstr(s, "No Base Name
PHENO-IMGs", &argv);
if(_fullpath(fs_pre, s, _MAX_PATH)==NULL)     {sprintf(ERRmess, "Full path not found
for Base Name PHENO-IMGs: %s", s); ERROR(stage, ERRmess, 1); }
printf(" p2.Season to treat [0, 1, 2, 3, ...]            =>"); season = Qshort(0,
10, -1, -1, &argv);
printf("\n");

   //Show/test IMGs
   //""""""""""""""""
printf("    N TYPE IN-IMG\n");
for(v=0;v<3;v++)
{
    switch(v)
    {
        case 0: sprintf(s, "%ss%d", fs_pre, season); sprintf(t, "SOS%d", season);
break;
        case 1: sprintf(s, "%sm%d", fs_pre, season); sprintf(t, "MOS%d", season);
break;
        case 2: sprintf(s, "%se%d", fs_pre, season); sprintf(t, "EOS%d", season);
break;
    }
    printf("%5d %s %s.img\n", v + 1, t, s);
    img_exist(s, 1, 1, fs_img[v]);                                  //Fatal=1  => If
IMG not found, stop.
   envi_hdr_read(fs_img[v], &hh, 1, 0, 0, 1, 1, 0, 0, 0);   //1-band, BYTE, OFFSET
NOT allowed

    if(v==0)
```

```
   {
      hs = hh;
      yyyytar = hs.date / 10000L; strcpy(t, ""); if(yyyytar==0) { strcpy(t, "
(LTA)"); }
        printf("    -Title   : %.64s\n"                 , hs.description);
      printf("    -MapSys  : %s\n"                  , hs.mi.name);
      printf("    -Size    : %ld cols x %ld recs\n", hs.samples, hs.lines);
      printf("    -Tgt Year: %d %s\n"                 , yyyytar, t);
   }
   if(map_diff(&hh.mi, &hs.mi) > 1)                                                {
sprintf(ERRmess, "Deviate mapinfo in PHENO-IMG %s", s);
ERROR(stage, ERRmess, 1); }
    if (hh.Vlo > 1 || hh.Vhi != 108 || hh.Vint != -36 || hh.Vslo != 1.)  {
sprintf(ERRmess, "Wrong VALUES (must be Vlo=0/1,Vhi=108,Vint=-36,Vslo=1)in PHENO-IMG
%s", s); ERROR(stage, ERRmess, 1); }
    if (hh.date != hs.date)
{ sprintf(ERRmess, "Deviant date in PHENO-IMG %s", s);
ERROR(stage, ERRmess, 1); }
}
printf("\n");
//  See PHENOdef/fix:   hs.flags = "255=water[,254=masked],253=insufficient
data,252=error,251=no seasonality" (and 251=no season2 )

//---------------------------------
// B2.IN-IMGs to be SUMMED/AVERAGED
//---------------------------------
    //Names
    //"""""
printf("    SERIES OF DEKADAL (S10) IN-IMGs TO BE AVERAGED/SUMMED (Names:
PiDiSi.img): \n");
printf(" p3.Prefix      Pi                              =>"); Qstr(fi_pre, "", &argv);
printf(" p4.Date-Format Di [1-6]                        =>"); fi_df = Qshort(1, 6, 0,
0, &argv) - 1;
printf(" p5.Suffix      Si                              =>"); Qstr(fi_suf, "", &argv);

    //Start/End Dates
    //"""""""""""""""""
if(yyyytar)
//PHENOdef was applied on specific year
{
    l = 10000L * (yyyytar - 1) +  101;
//default for Idate1
    printf(" p6.Start IN-series  [YYYYMMDD, def=%ld] =>", l);       Idate1 =
Qlong(19500101, 20491231, l, 0, &argv);
}
else
//PHENOdef was applied on LTA
{
    printf(" p6.Start IN-series    [YYYYMMDD, no default] =>");    Idate1 =
Qlong(19500101, 20491231, 0, 0, &argv);
}
date_test(Idate1, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
Idate1 = 10000L*yyyy + 100*mm + dd_st;                                  // Reset
to start of dekad!
yyyypre = yyyy;                                                          //
First year (basic assumption!!!)
yyyycen = yyyypre + 1;                                                   //
Central year (not necessarily same as yyyytar=year of OUT-IMG)
```

```
l = 10000L * (yyyycen + 1) + 1231;                                          //
Default and Maximum for Idate2

printf(" p7.End   IN-series  [YYYYMMDD, def=%ld] =>", l);       Idate2 =
Qlong(Idate1, l, l, 0, &argv);
date_test(Idate2, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
Idate2 = 10000L*yyyy + 100*mm + dd_st;                                      //Reset
to start of dekad

if(yyyy == yyyypre) { sprintf(ERRmess, "End of IN-series (%ld) must be 1 or 2 years
after year of start (%d)", Idate2, yyyypre); ERROR(stage, ERRmess, 1); }

//-------------------
// B3.IN-IMGs: CHECKS      NB: NiT = 109 (0, 1-108)
//-------------------
HI = (struct HIstruc *)calloc(NiT, sizeof(struct HIstruc)); if(HI == NULL) {
sprintf(ERRmess, "Insufficient RAM for IN-HDRs"); ERROR(stage, ERRmess, 1); }
Ni = 0;
// Dekads in concerned period (max. 108)
NiGOOD = 0;
// Existing IMGs found
NdaysI = 0;                                                                 //
Total nr. of days in series. NOT REALLY USED IN COMPUTATIONS!!!
ti1 = 0; ti2 = 0;
// Sequence nr (1-108) of first/last IN-IMG (present or lacking).

date = 10000L * yyyypre + 101;                                  // We start
from dekad 1 in previous year (not simply from Idate1)
date_test(date, 1, &yyyy,  &yy,  &mm,  m, &dd,  &dd_st,  &ttm,  &tty,  &dpm,  &doy);
dd = dd - 10;
for(tt=1;tt<NiT;tt++)                                                       //
Scan all possible dekads 1-108
{
   dd=dd+10; if(dd>21) { dd=1; mm++; } if(mm>12) { yyyy++; mm=1; }
    date = 10000L*yyyy + 100*mm + dd;

    if(date < Idate1 || date > Idate2) { continue; }
    Ni++;
// OK, inside IN-period => New dekad
    ti2=tt;  if(ti1==0) { ti1 = tt; }                                       //
Monitor first/last dekad in specified IN-series

    date_test(date, 1, &yyyy,  &yy,  &mm,  m, &dd,  &dd_st,  &ttm,  &tty,  &dpm,
&doy);
    HI[tt].date = date;                                                     //
That is is the nominal date
   i = 10; if(ttm==3){ i = dpm - 20;} HI[tt].days = i;              // Days in this
dekad (NB: leap years are accounted for!)
    NdaysI = NdaysI + i;
// Total days since start of ti1

    img_date_name(fi_pre, date, fi_df, fi_suf, t);                  // t = IN-
IMG without extension
    if(img_exist(t, 1, 0, s))    { HI[tt].lack = 1; continue; }      // s = IN-IMG
with    extension. This IN-IMG lacks!

    NiGOOD++; HI[tt].lack = 0;                                              //
OK, this IN-IMG exists.
```

```
   envi_hdr_read(s, &hh, 1, 0, 0, 1, 4, 1, 0, 1);                          // Allowed:
4 DTs, offset, classes. NOT: 3D, FLIP.

   if(hh.date < 19500100) { hh.date = date; }                              // Date
lacks in HDR => accept & replace with nominal date
   if(hh.date != date)                                                          //
Always OK if HDR does not contain date
   {
        sprintf(ERRmess, "Wrong HDR-date (%ld instead of %ld) for IMG %s", hh.date,
date, s); ERROR(stage, ERRmess, 1);
   }

   if(NiGOOD == 1) {    hi = hh; }                                             //
Memorize first IN-HDR in hi

   if(map_diff(&hh.mi, &hs.mi) > 1)                                  // Compare mapinfo
   {
      sprintf(ERRmess, "IN-IMG with mapinfo different from PHENO-IMGs: %s", s);
ERROR(stage, ERRmess, 1);
   }

// Datatype must be constant
    if(hh.data_type != hi.data_type)
   {
      printf("     \nERROR: All IN-IMGs must have same datatype [1-4]\n");
        printf("IMG of %ld: datatype=%d\n", hi.date, hi.data_type);
      printf("IMG of %ld: datatype=%d\n", hh.date, hh.data_type);
        sprintf(ERRmess, "IMG with deviate datatype (%d): %s", hh.data_type, s);
ERROR(stage, ERRmess, 1);
   }
    if (hh.Vlo != hi.Vlo || hh.Vhi != hi.Vhi)                          // Significant
range must be constant
   {
      printf("     \nERROR: All IN-IMGs must have same significant range (Vlo-
Vhi)\n");
        printf("IMG of %ld: Vlo=%.12g, Vhi=%.12g\n", hi.date, hi.Vlo, hi.Vhi);
      printf("IMG of %ld: Vlo=%.12g, Vhi=%.12g\n", hh.date, hh.Vlo, hh.Vhi);
        sprintf(ERRmess, "IN-IMG with deviate significant range (Vlo-Vhi): %s", s);
ERROR(stage, ERRmess, 1);
   }
   if (hh.Vint != hi.Vint || hh.Vslo != hi.Vslo)                     // Scaling must be
constant
   {
        printf("     \nERROR: All IN-IMGs must have same scaling (Vint, Vslo)\n");
      printf("IMG of %ld: Vint=%.12g, Vslo=%.12g\n", hi.date, hi.Vint, hi.Vslo);
      printf("IMG of %ld: Vint=%.12g, Vslo=%.12g\n", hh.date, hh.Vint, hh.Vslo);
        sprintf(ERRmess, "IN-IMG with deviate scaling (Vint, Vslo): %s", s);
ERROR(stage, ERRmess, 1);
   }

   HI[tt].offset = hh.offset;                                                   //
Memorize offset & IMG-name, incl. extension "*.IMG"
   HI[tt].fi = (char *)calloc(1+strlen(s), 1); if(HI[tt].fi == NULL) {
sprintf(ERRmess, "Insufficient RAM for IN-IMG names"); ERROR(stage, ERRmess, 1); }
   strcpy(HI[tt].fi, s);
}
NiMISS = Ni - NiGOOD;
printf("    Dekads in IN-series : %d\n", Ni);
```

```
printf("    Nr. of IN-IMGs found: %d\n", NiGOOD);
printf("    Total Days covered  : %d\n", NdaysI);                          // FURTHER
NOT USED IN COMPUTATIONS!!!

if(NiGOOD<2) { sprintf(ERRmess, "Only %d IN-IMGs found (minimum=2)", NiGOOD);
ERROR(stage, ERRmess, 1); }

    //First and last IN-IMG must be present                          // NB: Maybe this
annoying constraint should be removed! Anyway always possible that fist/last IN-
values are flagged!
    //""""""""""""""""""""""""""""""""""""""""
if(HI[ti1].lack) { sprintf(ERRmess, "IN-IMG of start date (%ld) is lacking",
Idate1); ERROR(stage, ERRmess, 1); }
if(HI[ti2].lack) { sprintf(ERRmess,   "IN-IMG of end date (%ld) is lacking",
Idate2); ERROR(stage, ERRmess, 1); }

    //Find widest gap (series of consecutively missing IN-IMGs, in centre of
series). NB: This also seems to work when first/last IN-IMG are lacking!
    //"""""""""""""""""
InGap = 0;
// InGap=Max.gap found
i = HI[ti1].lack; k = 0;                                                    //
i=type of previous IMG, k=number of IMGs in current gap
for(tt=ti1;tt<=ti2;tt++)
{
   j = HI[tt].lack;                                                         //
j=type of current IMG (0=OK, 1=lacking)
   if(i==1 && j==1) { k++; }                                                //
existing GAP continues
   if(i==0 && j==1) { k=1; }                                                //
start of new GAP
   if(i==1 && j==0) { InGap = max(InGap, k); k = 0;}                        // end of
GAP => Evaluate
   i = j;
}
InGap = max(InGap, k);                                                      //
evaluation again needed at the end
printf("    Widest gap          : %d consecutively lacking IN-IMGs\n", InGap);
printf("\n");

//---------------------
// B4.GENERAL OBJECTIVE
//---------------------
printf(" p8.GENERAL METHOD              [1/2, def=1] =>"); MET = Qshort(1, 2, 1, -
1, &argv);
printf("\n");

//-----------------------
// B5.IN-IMGs: MORE SPECS
//-----------------------
printf("    MORE SPECIFICATIONS on the DEKADAL IN-IMGs:\n");

   //MAXgap
   //"""""""
printf(" p9.Max. consecutively missing IMGs   [def=0] =>");  MaxInGap = Qshort(0,
Ni, 0, -1, &argv);
    if(InGap > MaxInGap) { sprintf(ERRmess, "Gap of lacking IN-IMGs (%d) wider than
allowed (%d)", InGap, MaxInGap); ERROR(stage, ERRmess, 1); }
```

```
   //Ybase-Variants
   //"""""""""""""""
printf("p10.Use Ybase: 0=no, 1=Reset, 2=Tsum  [def=0] =>"); Base = Qshort(0, 2, 0, -
1, &argv);
printf("p11.Minimum physical value Ybase [no default] =>");
if(Base==0)
{
    Ybase = 0; Vbase = 0; printf("none\n"); if (bat) { ++argv; }
}
if(Base==1)                                                    // Reset all
Y=Ybase when Y<Ybase. NB: Ybase must be in range Ylo-Yhi.
{
    Ymin = hi.Vint + hi.Vslo * hi.Vlo;
    Ymax = hi.Vint + hi.Vslo * hi.Vhi;
    Ybase = Qfloat(Ymin, Ymax, Ymin, Ymin - 100., &argv);
    Vbase = (Ybase - hi.Vint) / hi.Vslo; if(hi.data_type < 4) { Vbase = floor(0.5 +
Vbase); }
}
if(Base==2)                                                    // Work with
Ydif=Y-Ybase, with Ydif=0 if Y<Ybase (for Temp.Sums)
{
    Ybase = Qfloat(-1, -1, -1, -1, &argv);                       // Can be any
value, even beyond Ylo-Yhi
    Vbase = (Ybase - hi.Vint) / hi.Vslo; if(hi.data_type < 4) { Vbase = floor(0.5 +
Vbase); }
    hi.Vint = 0;                                               // Yd = Y-Yb =
(a+b.V)-(a+b.Vb) = b(V-Vb) = b.Vd. SUM(Yd) = b.SUM(Vd). If a is reset to zero (even
if Yb=0 or Vb=0), the rest remains the same.
}

   //Type of IN-Values
   //"""""""""""""""""
printf("p12.IN-Type: 1=daily values, 2=sums   [def=1] =>"); TypeI = Qshort(1, 2, 1,
-1, &argv);

   //Kc-Weighting ((see program PHENOtyp)
   //"""""""""""""
printf("p13.IMG with Kc-Types/Weights    (blank=none) =>"); Qstr(s, "", &argv);
if(strlen(s) == 0)
{
   KcW = 0; strcpy(fi_Kc, "none");
}
else
{
   //Check Kc-IMG: Byte classification, with Kc-Types 0-250. The weights are stored
in 'Class Lookup'. Kc-Type 0 = default = equal weights (100, 100, 100)
   KcW = 1; img_exist(s, 1, 1, fi_Kc); envi_hdr_read(fi_Kc, &hh, 1, 0, 0, 1, 1, 1,
0, 1);
   printf("    -Title   : %.64s\n"         , hh.description);
   printf("    -Range   : %.7g ... %.7g\n", hh.Vlo, hh.Vhi);

   if(map_diff(&hh.mi, &hi.mi) > 1) { sprintf(ERRmess, "Kc-IMG has deviant
georeferencing: %s"     , fi_Kc); ERROR(stage, ERRmess, 1); }
   if (hh.Vlo != 0 || hh.Vhi > 250) { sprintf(ERRmess, "Kc-IMG must have Vlo=0 and
Vhi <= 250: %s", fi_Kc); ERROR(stage, ERRmess, 1); }
```

```
   //Read Kc-Weights from hh.colors in table Kc[0 - 250, 251-255][0 - 2]. And
discern defined (1) from undefined (0) Kc-Types in table TYP[0-250, 251-255].
   for(k=0;k<256;k++) { TYP[k] = 0; for(i=0;i<3;i++) { Kc[k][i] = 0; } }
// Initialize both tables. That's needed because declared STATICly.
   j = 0;
   for(k=0;k<hh.classes;k++)
   {
      ok = 0;
      for(i=0;i<3;i++)
      {
         if(hh.colors[k][i] > 250) { ok = 0; break; }
         ok = ok + hh.colors[k][i];
         Kc[k][i] = hh.colors[k][i] / 200.;
// Scale from BYTE 0-200 to fractions 0.0 ... 1.0
      }
      TYP[k] = 0; if(ok > 0) { TYP[k] = 1; j++; }
   }
   printf("    -Kc-Types: %d\n", j);
   if(j==0)       { sprintf(ERRmess,       "No valid Kc-Types found in IMG: %s",
fi_Kc); ERROR(stage, ERRmess, 1); }
   if(TYP[0]==0)  { sprintf(ERRmess, "Kc-Type=0 (default) not found in IMG: %s",
fi_Kc); ERROR(stage, ERRmess, 1); }
   if(j==1)       { sprintf(ERRmess,    "No specific Kc-Types found in IMG: %s",
fi_Kc); ERROR(stage, ERRmess, 1); }

   //for(k=0;k<256;k++) { TYP[k] = 0; for(i=0;i<3;i++) { Kc[k][i] = 0; } }
// Initialize both tables. That's needed because declared STATICly.
   //j = 0;
   //for(k=0;k<hh.classes;k++)
   //{
   //   ok = 0; for(i=0;i<3;i++) { ok = ok + hh.colors[k][i]; Kc[k][i] =
hh.colors[k][i] / 200.; }     // Scale from BYTE 0-200 to fractions 0.0 ... 1.0
   //   TYP[k] = 0; if(ok > 0) { TYP[k] = 1; j++; }
   //}
   //printf("    -Kc-Types: %d\n", j);
   //if(j==0)       { sprintf(ERRmess,       "No valid Kc-Types found in IMG: %s",
fi_Kc); ERROR(stage, ERRmess, 1); }
   //if(TYP[0]==0)  { sprintf(ERRmess, "Kc-Type=0 (default) not found in IMG: %s",
fi_Kc); ERROR(stage, ERRmess, 1); }
}
printf("\n");

//---------------
// B6. Output-IMG: Date, Name, Type, Flags,...
//---------------
printf("    SPECIFICATIONS OF THE OUT-IMG:\n");

   //Odate  V1612  NB: Must be in Central Year and Odate <= Idate2
   //""""""
//date = 10000L * yyyycen + 0101;
// Minimum for Odate
//l = min(10000L * (yyyycen + MET -1) + 1231, Idate2);
// Maximum and default for Odate: MET=1: End of YYYYc, MET=2: End of YYYYc + 1
//printf("p14.OUT-date          [%ld-%ld=def] =>", date, l); Odate = Qlong(date, l,
l, 0, &argv);
//date_test(Odate, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
//Odate = 10000L*yyyy + 100*mm + dd_st;
// Reset to start of dekad
```

```
//  Correction in V1602:
//   OLD: to = 36 + tty;
// to= Sequence number of OUT-dekad [37-72]. That's OK for MET=1, and for MET=2 if
Odate in YYYYcen.
//   NEW: to = 36 + (yyyy - yyyycen) * 36 + tty;
// to= Sequence number of OUT-dekad [37-108]. Now always OK (yyyy = year of Odate)
//to = 36 + (yyyy - yyyycen) * 36 + tty;

   //Odate  V1611:  MET=1: In Yc (max 36 dekads), MET=2 in Yc-1 to Yc+1 (max 108
dekads). But always limited to [p6 - p7].
   //"""""""""""""
if(MET==1)
{
   date = max(Idate1, 10000L * yyyycen + 0101);
// Minimum for Odate
   l    = min(Idate2, 10000L * yyyycen + 1231);
// Maximum and default
}
else
{
   date = max(Idate1, 10000L * (yyyycen - 1) + 0101);
// Minimum for Odate
   l    = min(Idate2, 10000L * (yyyycen + 1) + 1231);
// Maximum and default
}
date_test(l    , 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
// In case l=Idate2: Reset l=Max/Def to end of dekad
switch(ttm)
{
    case 1: l = 10000L * yyyy + 100 * mm +  10; break;
    case 2: l = 10000L * yyyy + 100 * mm +  20; break;
    case 3: l = 10000L * yyyy + 100 * mm + dpm; break;
}
printf("p14.OUT-date          [%ld-%ld=def] =>", date, l); Odate = Qlong(date, l, l,
0, &argv);
date_test(Odate, 1, &yyyy, &yy, &mm, m, &dd, &dd_st, &ttm, &tty, &dpm, &doy);
Odate = 10000L*yyyy + 100*mm + dd_st;
// Reset to start of dekad
to = 36 + (yyyy - yyyycen) * 36 + tty;

   //TypeO
   //""""""
printf("p15.OUT-Type: 1=MEAN, 2=SUM           [def=1] =>"); TypeO = Qshort(1, 2, 1,
-1, &argv);

   //IMG-Name
   //"""""""""
printf("p16.IMG Base name  (add drive/path if needed) =>"); Qstr(t, "No base name
OUT-IMGs", &argv);
sprintf(s, "%s%d", t, season);
printf("    OUT-IMG                                  =>%s.img\n", s);
img_exist(s, 2, 1, fo_img);

   //OUT-Flagging
   //"""""""""""""
printf("    Flag pixels with incomplete profiles, if their IN-series ... \n");
```

```
printf("p17.contains > T1%% missing values        [0-100,def=85%%] =>");
NiMaxMissPerc = Qfloat(0, 100, 85, -1, &argv);

    NiMaxMiss = (short)floor(0.5 + Ni * NiMaxMissPerc / 100.);
// Max. Nr. of Missing values (eg. 108 * 85 / 100 = 91.8 = 92 dekads missing out of
108)
    if(NiMaxMissPerc > 0. && NiMaxMiss == 0) { NiMaxMiss = 1; }
    if(NiMaxMiss > (Ni - 2)) { NiMaxMiss = Ni - 2; }
// At least two good observations always needed!
    if(NiMISS > NiMaxMiss) { sprintf(ERRmess, "More lacking IN-IMGs (%d) than
allowed Missing Values (%d)", NiMISS, NiMaxMiss); ERROR(stage, ERRmess, 1); }

printf("p18.covers   < T2%% of full season length  [0-100,def=0%%] =>"); T2 =
Qfloat(0, 100, 0, -1, &argv) / 100.;   // PHENOcumTAR has default 50%

//---------------
// B7. Output-HDR
//---------------
    //General      NB: ho.sensor comes from hi (if specified). And of course also
ho.data_type.
    //********
ho = hi;
// Start with HDR of first IN-IMG
sprintf(ho.program,"%s.exe (V%d/%d)", PROGNAME, PROGVERSION, LIBVERSION);
ho.offset = 0;
ho.days = 10; ho.date = Odate;

switch(fi_df)
{
    case  0: strcpy(ss, "yyyymmdd"); break;
// strings ss and t only needed in DESCRIPTION!
    case  1: strcpy(ss, "yymmdd")  ; break;
    case  2: strcpy(ss, "yyyymdd") ; break;
    case  3: strcpy(ss, "yymdd")   ; break;
    case  4: strcpy(ss, "yyyytt")  ; break;
    case  5: strcpy(ss, "yytt")    ; break;
}
if(TypeO == 1) { strcpy (t, "MEAN"); } else { strcpy (t, "SUM"); }

sprintf(s, "%s of 10-daily IMGs %s[%s]%s from %ld to %ld over season %d of %d,
holding for %ld, METHOD=%d", t, _strupr(fi_pre), ss, _strupr(fi_suf), Idate1,
Idate2, season, yyyycen, Odate, MET);
s[sizeof(ho.description) - 1] = 0; strcpy(ho.description, s);

sprintf(s, "PHENO-IMGs=%s, MaxMis=%d, UseYbase=%d, Ybase=%g, INtype=%d, OUTtype=%d,
T1=%g%%, T2=%g%%, KcIMG=%s", fs_pre, MaxInGap, Base, Ybase, TypeI, TypeO,
NiMaxMissPerc, T2 *100, fi_Kc);
s[sizeof(ho.comment    ) - 1] = 0; strcpy(ho.comment    , s);

    //Spectral
    //*********
strcpy(ho.file_type,"ENVI standard");
// Remove all CLASS-info (if any): Initially accepted, but TO BE AVOIDED!
ho.classes = 0; ho.cnames = NULL; ho.colors = NULL;

if(TypeO == 1)
// OUT=MEAN => Keep DT/Scaling/Vunit from IN-IMGs. Adapt Vname & initially keep
Vlo/Vhi!
```

```
{
   if(MET==1) { sprintf(s,"MEAN of [%s] over season %d at %ld"          , hi.Vname,
season,          Odate); }
   if(MET==2) { sprintf(s,"MEAN of [%s] over season %d of %d - at %ld" , hi.Vname,
season, yyyycen, Odate); }
   s[sizeof(ho.Vname) - 1] = 0; strcpy(ho.Vname, s);

    switch(hi.data_type)
    {
        case 1: Fsea=     255; Fmsk=     254; Fnos=     253; Fmis=     252; Fout=
251;   Vlo =         0; Vhi =         250; break;
        case 2: Fsea=  -32765; Fmsk=  -32764; Fnos=  -32763; Fmis=  -32762; Fout=  -
32761;   Vlo =   -32760; Vhi =       32767; break;
        case 3: Fsea=-1000005; Fmsk=-1000004; Fnos=-1000003; Fmis=-1000002; Fout=-
1000001;   Vlo = -1000000; Vhi = 2147483647; break;
        case 4: Fsea=-1000005; Fmsk=-1000004; Fnos=-1000003; Fmis=-1000002; Fout=-
1000001;   Vlo = -1000000; Vhi =     hi.Vhi; break;
    }

    k=0;        // No saturation possible
    if(ho.Vlo < Vlo) { k=1; sprintf(s, "Vlo reset from %.12g to %.12g to make place
for 5 flags", ho.Vlo, Vlo); message(s, 1); ho.Vlo = Vlo; }    // Impossible for
BYTE!
    if(ho.Vhi > Vhi) { k=1; sprintf(s, "Vhi reset from %.12g to %.12g to make place
for 5 flags", ho.Vhi, Vhi); message(s, 1); ho.Vhi = Vhi; }    // Impossible for
FLOAT!
}
else
// OUT=SUM => DT=4=FLOAT, Remove scaling, Adapt Vname/Vunit & arbitrarily set
Vlo/Vhi!
{
    strcpy(t, "dekad"); if(TypeI == 1) { strcpy(t, "day"); }
// Only needed for Vunit.

   if(MET==1) { sprintf(s, "SUM of [%s] over season %d on %ld", hi.Vname, season,
Odate  ); }
   if(MET==2) { sprintf(s, "SUM of [%s] over season %d of %d" , hi.Vname, season,
yyyycen); }
                                        s[sizeof(ho.Vname) - 1] = 0; strcpy(ho.Vname,
s);
   sprintf(s,"[%s]*%ss", hi.Vunit, t); s[sizeof(ho.Vunit) - 1] = 0; strcpy(ho.Vunit,
s);

    ho.data_type = 4;
    ho.Vint = 0.; ho.Vslo = 1.;
    ho.Vlo = -1000000; ho.Vhi = 1.0E+30;
// BEWARE!!! AND NO TEST NEEDED ON OVERLAP WITH FLAGS!
    Fsea=-1000005; Fmsk=-1000004; Fnos=-1000003; Fmis=-1000002; Fout=-1000001;
}

if(MET==1) { sprintf(s, "%.12g=water,%.12g=missing PHENO,%.12g=no season
%d,%.12g=insufficient data,%.12g=off season %d"        ,
Fsea,Fmsk,Fnos,season,Fmis,Fout, season); }
if(MET==2) { sprintf(s, "%.12g=water,%.12g=missing PHENO,%.12g=no season
%d,%.12g=insufficient data,%.12g=season %d not started",
Fsea,Fmsk,Fnos,season,Fmis,Fout, season); }
s[sizeof(ho.flags) - 1] = 0; strcpy(ho.flags, s);
```

```
//NB: PHENOdef/fix:   hs.flags = "255=water[,254=masked],253=insufficient
data,252=error,251=no seasonality" (or 251=no season2 )



//============================================================
                                    //                          C. PRE-PROCESSING

//============================================================
strcpy(stage, "PREPROCESSING");
strcpy(ERRmess, "Insufficient RAM for IO-buffers");
Ncol = hi.samples; Nrec = hi.lines;
// Easier
//-------------------------------------------------
// C1. Open all IMGs: 3 PHENO, Ni IN [, Kc], 1 OUT
//-------------------------------------------------
   //Ni IN-IMGs
   //""""""""""
for(tt=ti1;tt<=ti2;tt++)
// ti1/2 in range [1-108]
{
    if(HI[tt].lack) { continue; }
   if((fpi[tt]=fopen(HI[tt].fi,"rb"))==NULL) { sprintf(ERRmess, "Opening IN-IMG %s",
HI[tt].fi); ERROR(stage, ERRmess, 1); }
   fseek(fpi[tt], HI[tt].offset, SEEK_SET);
// Skip header bytes
}
   //Others
   //"""""""
for(v=0;v<3;v++) { if((fps[v]=fopen(fs_img[v],"rb"))==NULL) { sprintf(ERRmess,
"Opening PHENO-IMG %s", fs_img[v]); ERROR(stage, ERRmess, 1); } }      // PHENO-IMGs
if(KcW)          { if((fpk=fopen(fi_Kc       ,"rb"))==NULL) { sprintf(ERRmess,
"Opening Kc-IMG %s",    fi_Kc    ); ERROR(stage, ERRmess, 1); } }      // Kc IN-IMGs
                   if((fpo=fopen(fo_img      ,"wb"))==NULL) { sprintf(ERRmess,
"Opening OUT-IMG %s",    fo_img  ); ERROR(stage, ERRmess, 1); }        // OUT-IMG

//----------------
// C2. I/O-Buffers
//----------------
   //IN-buffers         (Any DT, but fixed for all Ni IN-IMGs)
   //""""""""""
switch(hi.data_type)
{
   case 1:                                  bi1    =(unsigned char **)calloc(NiT,
sizeof(unsigned char *)); if(bi1    ==NULL){ ERROR(stage, ERRmess, 1); }
            for(tt=ti1;tt<=ti2;tt++) { bi1[tt]=(unsigned char  *)calloc(Ncol,
1); if(bi1[tt]==NULL){ ERROR(stage, ERRmess, 1); } } break;
   case 2:                                  bi2    =(short         **)calloc(NiT,
sizeof(short         *)); if(bi2    ==NULL){ ERROR(stage, ERRmess, 1); }
            for(tt=ti1;tt<=ti2;tt++) { bi2[tt]=(short         *)calloc(Ncol,
2); if(bi2[tt]==NULL){ ERROR(stage, ERRmess, 1); } } break;
   case 3:                                  bi3    =(long          **)calloc(NiT,
sizeof(long          *)); if(bi3    ==NULL){ ERROR(stage, ERRmess, 1); }
            for(tt=ti1;tt<=ti2;tt++) { bi3[tt]=(long          *)calloc(Ncol,
4); if(bi3[tt]==NULL){ ERROR(stage, ERRmess, 1); } } break;
   case 4:                                  bi4    =(float         **)calloc(NiT,
sizeof(float         *)); if(bi4    ==NULL){ ERROR(stage, ERRmess, 1); }
```

```
            for(tt=ti1;tt<=ti2;tt++) { bi4[tt]=(float          *)calloc(Ncol,
4); if(bi4[tt]==NULL){ ERROR(stage, ERRmess, 1); } } break;
}

   //Others
   //"""""""
for(i=0;i<3;i++) { bs1[i]=(unsigned char *)calloc(Ncol, 1); if(bs1[i]==NULL){
ERROR(stage, ERRmess, 1); } }     // PHENO-buffers (BYTE)
if(Kc)           { bk1   =(unsigned char *)calloc(Ncol, 1); if(bk1   ==NULL){
ERROR(stage, ERRmess, 1); } }     // Kc-buffer     (BYTE)
switch(ho.data_type)
{
   case 1:           bo1   =(unsigned char *)calloc(Ncol, 1); if(bo1   ==NULL){
ERROR(stage, ERRmess, 1); }
   case 2:           bo2   =(short         *)calloc(Ncol, 2); if(bo2   ==NULL){
ERROR(stage, ERRmess, 1); }
   case 3:           bo3   =(long          *)calloc(Ncol, 4); if(bo3   ==NULL){
ERROR(stage, ERRmess, 1); }
   case 4:           bo4   =(float         *)calloc(Ncol, 4); if(bo4   ==NULL){
ERROR(stage, ERRmess, 1); }
}

//-----------------------
// C3. Other Preparations
//-----------------------
if(TypeI==2) { for(i=1;i<NiT;i++) { HI[i].days = 1.0; } }            // In-
Values=[X/dekad] => All dekads receive equal weight

ho.Vmin = ho.Vhi; ho.Vmax = ho.Vlo;                                  // Initialize
Vmin/Vmax for all OUT-IMGs

Nsea=0, Nmsk=0, Nnos=0, Nmis=0; Nok=0;                               // Counters for
OUT-Flagged pixels, and the good ones (Nok). The remainder has 251=OUT=beyond
season.

NsatLO = 0; NsatHI = 0;                                              // Good pixels
(included in Nok) but with OUT-saturation


//============================================================
                                    //                        D. PROCESSING

//============================================================
//-----------------------
// D1. Process per Record
//-----------------------
strcpy(stage, "PROCESSING");
progression(0, Nrec, &progr40, &progr40_step, 1, "Computing OUT-IMG", "lines",
"Processing", &GUIdone, &GUIdone_step);
for(rec=0; rec < Nrec; rec++)
{
    //REC: Read all IN-buffers
   //"""""""""""""""""""""""""
   for(v=0;v<3;v++)                  {                      fread(bs1[ v], 1,
Ncol, fps[ v]); }              // SOS/MOS/EOS of concerned season
   if(KcW)                           {                      fread(bk1    , 1,
Ncol, fpk    ); }           // Kc-Types for season 1/2
```

```
    switch(hi.data_type)
// IN-IMGs to sum/average
    {
        case 1: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { fread(bi1[tt], 1,
Ncol, fpi[tt]); } } break;
        case 2: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { fread(bi2[tt], 2,
Ncol, fpi[tt]); } } break;
        case 3: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { fread(bi3[tt], 4,
Ncol, fpi[tt]); } } break;
        case 4: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { fread(bi4[tt], 4,
Ncol, fpi[tt]); } } break;
    }

    //REC: Process all pixels
   //""""""""""""""""""""""""
    for(col=0; col<Ncol; col++)
    {
        //COL: Read SOS/MOS/EOS & Skip pixels without PHENO-Info
// NB: IN -PHENOdef-flags = "255=water    [,254=masked],253=insufficient
data,252=error,    251=no seasonality or no season s"
      //같같같같같같같같같같같같같같같같같같같같같같같?              //     OUT-
PHENOcum-flags =  255=SEA      254=MSK (MISSING PHENO)
253=NOS
        pSOS = bs1[0][col];                                                    //
Dekads of SOS/MOS/EOS [1-108 or flag]
        pMOS = bs1[1][col];
        pEOS = bs1[2][col];
        if(pSOS==255)
// PHENOdef: 255=Water
        {
            switch(ho.data_type)
            {
            case 1: bo1[col] = (unsigned char)Fsea; break;
            case 2: bo2[col] = (short        )Fsea; break;
            case 3: bo3[col] = (long         )Fsea; break;
            case 4: bo4[col] = (float        )Fsea; break;
            }
            Nsea++; continue;
        }
        if(pSOS > 251 || pEOS > 251)
// PHENOdef: 254=Masked, 253=insufficient data, 252=error => GROUPED HERE AS 254 =
"MISSING PHENO-INFO OVER LAND"
        {
            switch(ho.data_type)
            {
            case 1: bo1[col] = (unsigned char)Fmsk; break;
            case 2: bo2[col] = (short        )Fmsk; break;
            case 3: bo3[col] = (long         )Fmsk; break;
            case 4: bo4[col] = (float        )Fmsk; break;
            }
            Nmsk++; continue;
        }
        if(pSOS > 250)
// PHENOdef: 251=No seasonality at all (or no SOS/EOS in the concerned season) =>
Now becomes 253=NoS
        {
            switch(ho.data_type)
            {
```

```
            case 1: bo1[col] = (unsigned char)Fnos; break;
            case 2: bo2[col] = (short        )Fnos; break;
            case 3: bo3[col] = (long         )Fnos; break;
            case 4: bo4[col] = (float        )Fnos; break;
            }
            Nnos++; continue;
        }

        //COL: Read pixel values in Vi[NiT=0-108]
// NB: Processing done on digital V, not on physical Y!
      //같같같같같같같같같같같같같같같같같같같같같같같같같같?
      switch(hi.data_type)
      {
            case 1: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { Vi[tt] =
(double)bi1[tt][col]; } } break;
            case 2: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { Vi[tt] =
(double)bi2[tt][col]; } } break;
            case 3: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { Vi[tt] =
(double)bi3[tt][col]; } } break;
            case 4: for(tt=ti1;tt<=ti2;tt++) { if(HI[tt].lack==0) { Vi[tt] =
(double)bi4[tt][col]; } } break;
      }

      //COL: Define Type of each Value in Vt[NiT]
// Type: 0=good value, 1=missing value (IN-flagged or lacking IN-IMG). Only needed
for later INTERPOLATION
      //같같같같같같같같같같같같같같같같같같같같같같같같같같?
      Nbad = 0; pX1 = -1; pX2 = -1;
// position of first/last good value (X), for this pixel
      for(tt=ti1;tt<=ti2;tt++)
      {
         if(HI[tt].lack)                          { Vt[tt] = 1; Nbad++; continue; }
// IN-IMG lacks (L), thus SKIP. ELSE: IN-IMG present (X or F)
         if(Vi[tt] < hi.Vlo || Vi[tt] > hi.Vhi) { Vt[tt] = 1; Nbad++; continue; }
// Flagged IN-value F
                                                  Vt[tt] = 0;
// Good IN-value X
                                                      if(pX1 < 0) { pX1 = tt; }
pX2 = tt;    // pX1/pX2=first and last good values.  NB: ti1 <= pX1 <= pX2 <= ti2
      }

      //COL: Skip pixels with incomplete profiles
// NB: Fmis = 252 = INSUFFICIENT DATA for PHENOcum
        //같같같같같같같같같같같같같같같같같같같같같같같같같같?
        if(Nbad > NiMaxMiss || pX1 < 0)
        {
            switch(ho.data_type)
            {
            case 1: bo1[col] = (unsigned char)Fmis; break;
            case 2: bo2[col] = (short        )Fmis; break;
            case 3: bo3[col] = (long         )Fmis; break;
            case 4: bo4[col] = (float        )Fmis; break;
            }
            Nmis++; continue;
        }
```

```
        //COL: Interpolate missing values
// NB: Only between pX1 and pX2 => No extrapolation
        //같같같같같같같같같같같같같같같?
        J1 = pX1;
// Profile centre => Interpolation  (J1=dekad of first good value in series)
      while(J1 < pX2)
      {
          J1++;
          if(Vt[J1])
// missing value (F/L) => to be interpolated
          {
              J1--; J2 = J1+1;
// J1=First good X-value
              while(Vt[J2]) { J2++; }
// J2=Next  good X-value

              B = (Vi[J2] - Vi[J1]) / (J2 - J1);
// Slope of Y = A + B * J
              A = Vi[J1] - B * J1;
// Intercept
              for(v = J1+1; v < J2; v++)
            { Vi[v] = A + B * v; Vt[v] = 0; }
// Interpolate in-between values & Reset Vt-type
              J1=J2;
// Continue with last good value
          }
      }
// CONCLUSION: now we have 'valid' data between dekads pX1 and pX2

      //COL: Adapt Vi-values for Base (p9/p10)
        //같같같같같같같같같같같같같같같같같
        if(Base == 1) { for(tt=pX1;tt<=pX2;tt++) { Vi[tt] = max(Vi[tt],  Vbase   );
} }    // 1=Reset all Y=Ybase when Y<Ybase                  => Saturate lower values
to Vbase
        if(Base == 2) { for(tt=pX1;tt<=pX2;tt++) { Vi[tt] = max(Vi[tt] - Vbase, 0);
} }    // 2=Use Ydif=Y-Ybase, with Ydif=0 if Y<Ybase (Tsum) => Replace values by
(non-negative) differences

      //COL: Set default outputs to 251="off Season" (MET=1) or "Season not started"
(MET=2)
      //""""""""""""""""""""""""""""""같같같같같같같같같같같같같같같같같같같같같같같같같?
        switch(ho.data_type)
        {
         case 1: bo1[col] = (unsigned char)Fout; break;
         case 2: bo2[col] = (short        )Fout; break;
         case 3: bo3[col] = (long         )Fout; break;
         case 4: bo4[col] = (float        )Fout; break;
        }

      //COL: Define pC1/pC2 = Dekads [1-108] to accumulate, given the position of
the OUT-dekad (to)
      //     NB: Definition of PHENOdef: MOS always falls in range [37-72], SOS can
be earlier [1-36], EOS later [73-108]

//같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같
같같같같같같같?
```

```
      if(MET==1)
// Odate lies in central year (by definition), hence 'to' is always resticted to
range of [37-72]
      {
         pC1 = -1;
         if(pSOS > 36 && pEOS <= 72)
// SOS/MOS/EOS in same central year
           {
              if(to >= pSOS && to <= pEOS) { pC1 = pSOS     ; }
// Else: pC1 beyond season
           }
           if(pSOS < 37)
// SOS in first year => MOS/EOS in central year
           {
              if(                 to <= pEOS) { pC1 = pSOS     ; }
              if(to >= pSOS + 36           ) { pC1 = pSOS + 36; }
           }
           if(pEOS > 72)
// EOS in third year => SOS/MOS in central year
           {
              if(to <= pEOS - 36           ) { pC1 = pSOS - 36; }
              if(                 to >= pSOS) { pC1 = pSOS     ; }
           }
           if(pC1 == -1) { continue; }
// OUT-dekad 'to' falls beyond SOS-EOS => Keep default Fout=251 (off season)
         pC2 = to;
      }
      else
// Here 'to' can be in range [37-108]
      {
         if(to < pSOS) { continue; }
// OUT-dekad comes before season       => Keep default Fout=251 (season not yet
started)

         pC1 = pSOS;
// Accumulate from SOS to ...
         pC2 = min(pEOS, to);
// 'to' or at most EOS (if EOS < to)
      }

      //COL: Thus, "to" falls within the season and we have to find the SUM of
values from pC1 to pC2.
      //     But if this range is not fully covered with valid data (pX1 - pX2), ...
      //     then the pixel must be flagged with Fmis = 252 = "Insufficient data"

//같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같?
      ok = 1; if(pC1 < pX1 || pC1 > pX2 || pC2 < pX1 || pC2 > pX2 ) { ok = 0; }
      if(ok == 0)
      {
         switch(ho.data_type)
            {
            case 1: bo1[col] = (unsigned char)Fmis; break;
            case 2: bo2[col] = (short        )Fmis; break;
            case 3: bo3[col] = (long         )Fmis; break;
            case 4: bo4[col] = (float        )Fmis; break;
            }
            Nmis++; continue;
      }
```

```
      //COL: Partial overlap between pC1-pC2 and SOS-EOS is too small. Worst case:
pC1 = pC2.
      //     Then keep default OUT-FLAG (251 = OFF-SEASON). What else?
      //     But with default T2 = 0%, this event should never happen!
      //같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같?
      if( ((float)(pC2 - pC1)) / (pEOS - pSOS) < T2) { continue; }

      Nok++;                                                              // This
pixel will have a normal output (no more failures/flaggings possible)

      //COL: Adapt vector W[109] with combined weights (days x Kc-weights) // NB:
THESE OVERALL WEIGHTS W[tt] ARE NOT YET NORMALIZED!
      //같같같같같같같같같같같같같같같같같같같같같같같같같같
        for(tt=pC1;tt<=pC2;tt++) { W[tt] = HI[tt].days; }                   // If
TypeI=2 (X/dekad), all days/weights were already reset to 1!
      W[pSOS] = HI[pSOS].days / 2.0; W[pEOS] = HI[pEOS].days / 2.0;       // NB:
Dekads of SOS and EOS only count for 50% !!!
      if(KcW)
      {
         typ = bk1[col]; if(TYP[typ] == 0) { typ = 0; }                    // All
pixels in Regions/Classes not covered by PHENOtyp => Default Typ=0 (equal weights)

         W[pMOS] = W[pMOS] * Kc[typ][1];                                   //
Overall weight at MOS (redundant if MOS > C2 => no problem)

                                                                          // All
dekads from SOS to MOS
            B = (Kc[typ][1] - Kc[typ][0]) / (pMOS - pSOS);                 //
Increment for Kc-weights (difference between two dekads). Normally POSITIVE.
            V =  Kc[typ][0] + B * (pC1 - pSOS);                            // Kc-
weight at pC1
            W[pC1] = W[pC1] * V;                                           //
Overall weight at pC1
            for(tt=pC1+1;tt<pMOS;tt++) { V = V + B; W[tt] = W[tt] * V; }   // For
next dekads (not MOS!): Kc-weight V and overall weight

         if(pC2 > pMOS)                                                    // Not
necessarily: pC2=to can be < pMOS
         {
            B = (Kc[typ][2] - Kc[typ][1]) / (pEOS - pMOS);                 //
Increment for Kc-weights (difference between two dekads). Normally NEGATIVE.
            V = Kc[typ][2] + B * (pC2 - pEOS);                             // Kc-
weight at pC2
            W[pC2] = W[pC2] * V;                                           //
Overall weight at pC2
            for(tt=pC2-1;tt>pMOS;tt--) { V = V - B; W[tt] = W[tt] * V; }   // For
previous dekads (not MOS!): Kc-weight V and overall weight
         }
      }

      //COL: Compute Sums of weights and of weighted values                // Over
all dekads pC1 - pC2
      //같같같같같같같같같같같같같같같같같같같같같같같?
        Sw = 0.; Sv = 0.;
        for(tt=pC1;tt<=pC2;tt++)
        {
```

```
            Sw = Sw + W[tt]; Sv = Sv + W[tt] * Vi[tt];
        }

      //COL: Compute OUT-values V (MEAN or SUM) at OUT-dekad "to"          // NB: If
MET=2 and to>pEOS, the value of EOS is written to OUT-buffer!
      //같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같같?
        if(TypeO==1) { V = Sv / Sw; }                                        //
MEAN ? = a + b.?       Output ?  &  maintain scaling a/b.
        else         { V = hi.Vint * Sw + hi.Vslo * Sv; }                  // SUM
Sy = a.Sw + b.SV    Output Sy  &  remove scaling

      //COL: OUTPUT to buffer
      //같같같같같같같같같?
      if(V < ho.Vlo) { V = ho.Vlo; NsatLO++; }                               //
Check for saturation
        if(V > ho.Vhi) { V = ho.Vhi; NsatHI++; }                              //
Possible for TypeO=1 when Vlo/Vhi were reset for flags. Very unlikely for TypeO=2
(floating point).

        ho.Vmin = min(V, ho.Vmin);
        ho.Vmax = max(V, ho.Vmax);
// Still in DOUBLE => Rounding to OUT-DT done at the end!!!

        switch(ho.data_type)
        {
            case 1: bo1[col] = (unsigned char)floor(0.5 + V); break;
            case 2: bo2[col] = (short        )floor(0.5 + V); break;
            case 3: bo3[col] = (long         )floor(0.5 + V); break;
            case 4: bo4[col] =                        V ; break;
        }
   }                                                                         // NEXT
COLUMN COL

    //REC: Write OUT-buffer
   //""""""""""""""""""""""
    switch(ho.data_type)
    {
        case 1: fwrite(bo1, 1, Ncol, fpo);  break;
        case 2: fwrite(bo2, 2, Ncol, fpo);  break;
        case 3: fwrite(bo3, 4, Ncol, fpo);  break;
        case 4: fwrite(bo4, 4, Ncol, fpo);  break;
    }
    if ((rec+1) >= (long)floor(progr40)) { progression(1, Nrec, &progr40,
&progr40_step, 1, "Computing OUT-IMG", "lines", "Processing", &GUIdone,
&GUIdone_step);  }
}
printf("\n"); printf("\n");
_fcloseall();

//-----------------------
// D2. Create OUT-Headers
//-----------------------
if(ho.Vmin > ho.Vmax) { ho.Vmin = ho.Vlo; ho.Vmax = ho.Vlo; }
// Might happen if no good pixels found (Nok=0), early OUT-date & short IN-series.
switch(ho.data_type)
{
```

```
    case 1: ho.Vmin=(unsigned char)floor(0.5 + ho.Vmin); ho.Vmax=(unsigned
char)floor(0.5 + ho.Vmax); break;
    case 2: ho.Vmin=        (short)floor(0.5 + ho.Vmin); ho.Vmax=
(short)floor(0.5 + ho.Vmax); break;
    case 3: ho.Vmin=         (long)floor(0.5 + ho.Vmin); ho.Vmax=
(long)floor(0.5 + ho.Vmax); break;
    case 4:
; break;
}
sprintf(s, "%s, NsaturLO=%I64d, NsaturHI=%I64d", ho.comment, NsatLO, NsatHI);
s[sizeof(ho.comment) - 1] = 0; strcpy(ho.comment, s);
envi_hdr_create(fo_img, &ho, IDRISI, ARCVIEW);

//-----------------
// D3. Termination
//-----------------
f = (double) (100./hi.pixels); N = hi.pixels - Nok;            // N = all flagged
pixels

                  printf("TOTAL PIXELS                    : %12I64d  (%7.3f%%)
\n", hi.pixels, f*hi.pixels       );
                  printf("- FLAGGED                       : %12I64d  (%7.3f%%)
\n", N         , f*N              );
                  printf("   . PHENOdef Sea               : %12I64d  (%7.3f%%)
Flag=%.12g\n", Nsea      , f*Nsea       , Fsea);
                  printf("             Masked/Missing/Error: %12I64d  (%7.3f%%)
Flag=%.12g\n", Nmsk      , f*Nmsk       , Fmsk);
                  printf("             No seasonality      : %12I64d  (%7.3f%%)
Flag=%.12g\n", Nnos      , f*Nnos       , Fnos);
                  printf("   . Incomplete IN-profile       : %12I64d  (%7.3f%%)
Flag=%.12g\n", Nmis      , f*Nmis       , Fmis);
N = hi.pixels - (Nsea + Nmsk + Nnos + Nmis + Nok);
if(MET==1)      { printf("   . Off season                  : %12I64d  (%7.3f%%)
Flag=%.12g\n", N          , f*N          , Fout); }
if(MET==2)      { printf("   . Season not yet started      : %12I64d  (%7.3f%%)
Flag=%.12g\n", N          , f*N          , Fout); }
                  printf("- Pixels with NORMAL VALUES      : %12I64d  (%7.3f%%)
\n", Nok       , f*Nok            );
                  printf("   . Saturation to adapted Vlo   : %12I64d  (%7.3f%%)
\n", NsatLO    , f*NsatLO         );
                  printf("   . Saturation to adapted Vhi   : %12I64d  (%7.3f%%)
\n", NsatHI    , f*NsatHI         );

k = 0; if(NsatLO || NsatHI) { k = 1; }
// Saturation happened!
if(k)
{
                   printf("\n");
                        printf("Some pixels have saturated values:\n");
                  printf("Good OUT-values had to be reset to adjusted OUT-range
Vlo=%.12g -> Vhi=%.12g\n", ho.Vlo, ho.Vhi);
    if(TypeO==1) {  printf("Because IN-range had to be shrinked to make place for 5
flags.\n"); }
    else            { printf("Strange, this is very unlikely for OUT-type=2
(Sums).\n");                  }
}
progression(2, 0, 0, 0, 0, "", "", "", &GUIdone, &GUIdone_step);
exit(0);
```

```
//****************************
}  //      END OF MAIN
//****************************
```

References

Copernicus Gio Global Land Component – Dry Matter Productivity algorithm theoretical basis document https://land.copernicus.eu/global/sites/cgls.vito.be/files/products/GIOGL1_ATBD_DMP_I2.00.pdf

Eerens, H., Piccard, I., Royer, A., & Orlandi, S. (2004). Methodology of the MARS crop yield forecasting system. Vol. 3: Remote sensing information, data processing and analysis. *Eds. Royer A. and Genovese G., EUR*, *21291*

Jiménez-Muñoz, J.C., Cristóbal, J., Sobrino, J.A., Sòria, G., Ninyerola, M. and Pons X. "Revision of the Single-Channel Algorithm for Land Surface Temperature Retrieval From Landsat Thermal-Infrared Data" IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING 47 (1): 339-349 (2009)

Rozenstein, O., Qin, Z., Derimian, Y. And Karnieli, A. Derivation of Land Surface Temperature for Landsat-8 TIRS Using a Split Window Algorithm, Sensors, 2014, 14(4):5768-5780, https://doi.org/10.3390/s140405768

Swets, D.L, Reed, B.C., Rowland, J.D., Marko, S.E., 1999. A weighted least-squares approach to temporal NDVI smoothing. In: Proceedings of the 1999 ASPRS Annual Conference, Portland, Oregon, pp. 526-536.

Tasumi, M. Allen, R. G, and R. Trezza. 2006. DEM based solar radiation estimation model for hydrological studies. Hydrological Science and Technology 22:197-208

Veroustraete, F., Sabbe, H., & Eerens, H. (2002). Estimation of carbon mass fluxes over Europe using the C-Fix model and Euroflux data. *Remote Sensing of Environment*, *83*(3), 376-399.