RE-ENGINEERING OF MISSION ANALYSIS SOFTWARE FOR ENVISAT-1

PPF_VISIBILITY SOFTWARE USER MANUAL

PO-IS-DMS-GS-0560

Code:	PO-IS-DMS-GS-0560
Issue:	3.9
Date:	30/05/11

	Name	Function	Signature
Prepared by:	Noelia Sánchez-Ortiz Juan Jose Borrego Bote Carlos Villanueva Muñoz	Project Engineer Project Engineer Project Engineer	
Checked by:	José Antonio González Abeytua	Project Manager	
Approved by:	José Antonio González Abeytua	Project Manager	

DEIMOS Space S.L.U.
Ronda de Poniente, 19
Edificio Fiteni VI, Portal 2, 2ª Planta
28760 Tres Cantos(Madrid), SPAIN
Tel.: +34 91 806 34 50
Fax: +34 91 806 34 51
E-mail: deimos@deimos-space.com

Document Information

Contract Data		Classification	
Contract Number:	Contract number	Internal	
		Public	
Contract Issuer:	ESA / ESTEC	Industry	X
		Confidential	

Internal Distribution		
Name	Unit	Copies

External Distribution		
Name	Organisation	Copies

Archiving	
Word Processor:	Framemaker 6.0
File name:	VisibilitySum
Archive Code	P/SUM/DMS/01/043-018

Document Status Log

Issue	Change Description	Date	Approval
1.0	• First Version	31/01/97	
2.0	• It includes the modifications with respect the prototype functions	20/11/97	
2.1	• It includes additional information on algorithms	05/12/97	
2.2	• Runtime performances improved • New pv_orbitinfo CFI function • Known problems of release 2.1 solved	18/05/98	
2.3	• It includes the new CFI function STARVISTIME • port to Windows	19/11/98	
2.4	• Completely new ZONEVISTIME implementation: - Counterclockwise-defined zones are allowed. - Zones greater than half a sphere allowed - Error handling completely modified. Messages and names are changed. • STAVISTIME update. Use new Zonevistime CFI function within it. Error handling updated. • Error handling update for pv_orbitinfo. • DUT1 range updated to [-1.0,1.0]	31/05/99	
2.5	• STARVISTIME update: - Known problems solved (SPR 44 & 52). - Input Star coordinates now in degrees (SPR 58). • STAVISTIME update: - SPR 56 fixed (Allowed multiple transitions in an orbit). - Runtime improvement. • ZONEVISTIME update: - SPR 43 fixed (Reading Zone DB file) • ORBITINFO update: - SPR 65 fixed (Wrong calculation of the absolute orbit number in PV_REL_INFO mode)	14/04/00	
2.6	• Unreleased	22/06/01	
2.7	• Unreleased	31/07/01	
2.8	• Unreleased • New document from previous version by: - L.J. Alvarez (GMV) - M. Sánchez-Gestido (GMV) - R. Martínez Iturbe (GMV) - B. Duesmann (ESA) - P. Viau (ESA)	22/10/01	

Issue	Change Description	Date	Approval
3.0	• Updates in DRSVISTIME - New angular constraints for the antenna movement - Computation of stop and start-up trajectory - Use of an Auxiliary Az/El Mask	18/01/02	
3.1	• Unrelease	25/11/02	
3.2	• See change bars	26/05/03	
3.3	• Maintenance release	13/12/04	
3.3.1	• Maintenance release	15/02/05	
3.4	• Added Time Segment Manipulation functions • Modified behaviour of DRSVISTIME to take into account inclination changes in Artemis orbit	17/05/05	
3.5	• Maintenance release • Library Optimization	30/01/06	
3.6	• Maintenance release • New library for LINUX 64-bits	22/12/06	
3.7	• Changes for Envisat extended life • New library for MAC OS on Intel platforms • Maintenance release	11/04/08	
3.8	• Changes for Envisat extended life • New function pv_compute_mlst_drift • Maintenance release	15/09/09	
3.9	• New reference orbit in which there is no DRS visibility for the mission extension • Maintenance release	30/05/11	

Table of Contents

List of Tables

List of Figures

1 SCOPE

The Software User Manual (SUM) of the Envisat-1 mission CFI software is composed of

- a general document describing the sections common to all the CFI software libraries
- a specific document for each of those libraries.

This document is the PPF_VISIBILITY Software User Manual. It provides a detailed description of the use of the CFI functions included within the PPF_VISIBILITY CFI software library.

2 ACRONYMS AND NOMENCLATURE

2.1 Acronyms

AOCS	Attitude and Orbit Control System
AOS	Acquisition of Signal
ANX	Ascending Node Crossing
APM	Antenna Pointing Mechanism
CFI	Customer Furnished Item
CS	Coordinate System
DRS	Data Relay Satellite
ESA	European Space Agency
ESTEC	European Space Technology and Research Centre
FOS	Flight Operation Segment
GS	Ground Station
H/W	Hardware
I/F	Interface
LOS	Line Of Sight (in DRS context)
LOS	Loss Of Signal (in ground station context)
MCD	Mission Convention Document
OEF	Orbit Event File
OSF	Orbit Scenario File
ROEF	Reference Orbit Event File
SBT	Satellite Binary Time
SUM	Software User Manual
S/W	Software
UTC	Universal Time Coordinated
UT1	Universal Time UT1
SSP	Sub Satellite Point

2.2 Nomenclature

CFI	A group of CFI functions, and related software and documentation, that will be distributed by ESA to the users as an independent unit
CFI function	A single function within a CFI that can be called by the user
Library	A software library containing all the CFI functions included within a CFI plus the supporting functions used by those CFI functions (transparently to the user)

3 APPLICABLE AND REFERENCE DOCUMENTS

3.1 Applicable documents

AD 1 Finalization of the re-engineering of Mission Analysis Software and of the ROP Generation Tool for Envisat: Statement of Work.PO-SW-ESA-SY-1242. ESA/ESTEC/APP. Issue 1.1. 03/10/2001.

AD 2 ESA Software Engineering Standards. ESA PSS-05-0. ESA. Issue 2. February 1991

3.2 Reference documents

RD 1 Envisat-1 Mission CFI Software Description and Interface Definition Document. PO-ID-ESA-SY-00412

RD 2 Envisat-1 Mission CFI Software. Mission Conventions Document. PO-IS-GMV-GS-0561

RD 3 Envisat-1 Mission CFI Software General Software User Manual. PO-IS-DMS-GS-0556

RD 4 Envisat-1 Mission CFI Software PPF_GENREF Software User Manual. PO-IS-DMS-GS-0609

RD 5 Envisat-1 Reference Operation Plan (ROP). EN-PL-ESA-GS-00334.

RD 6 DRS Visibility and Dynamical Constraints in DRS Antenna. PO-TN-ESA-GS-980

RD 7 pv_drsvistime URD. PO-TN-ESA-GS-1160 1.0. 28 February 2001

4 INTRODUCTION

This software library contains the CFI functions required to compute time segments at which Envisat-1 or one of its instruments is in view of various targets:

- zones (defined as polygons or circles, on the earth ellipsoid or at a given altitude)
- ground stations
- data relay satellites
- stars

This library is to be used for planning of Envisat-1 operations.

The PPF_VISIBILITY library includes the following CFI functions:

- **pv_stavistime**: computes visibility time segments for a ground station
- **pv_drsvistime**: computes visibility time segments for a data relay satellite
- **pv_zonevistime**: computes visibility time segments for an instrument swath in visibility of a zone.
- **pv_swathcalc**: computes location of a swath at a given time (additional routine to help refine the results of **pv_zonevistime**)
- **pv_starvistime**: computes visibility time segments for a star.
- **pv_orbitinfo**: returns all relevant orbital information related to a user specified orbit.
- Time Segments Manipulation Routines:
  - **pv_time_segments_not:** returns the complement of 1 vector of time segments.
  - **pv_time_segments_and:** returns the intersection segments from 2 vectors of time segments.
  - **pv_time_segments_or:** returns the joined segments from 2 vectors of time segments
  - **pv_time_segments_delta:** add or subtract time durations at the beginning and end of each time segment in a vector.
  - **pv_time_segments_sort:** returns the vector of time segments sorted according to absolute or relative orbits.
  - **pv_time_segments_merge:** merges all the overlapped segments in a list.
- **pv_compute_mlst_drift**: computes the MLST and MLST drift for a given orbit.

Several files are required to operate properly the above functions:

- Reference Orbit Event File (all functions) [1]
- Swath Template Files (**pv_stavistime, pv_zonevistime, pv_swathcalc**)
- Ground Stations Database File (**pv_stavistime**)
- (optionally) Zones Database File (**pv_zonevistime**)
- (optionally) Star Database File (**pv_starvistime**)

1. It is also possible to use as input, instead of the Orbit Event File, the Orbit Scenario file that was used to generate the Orbit Event File.

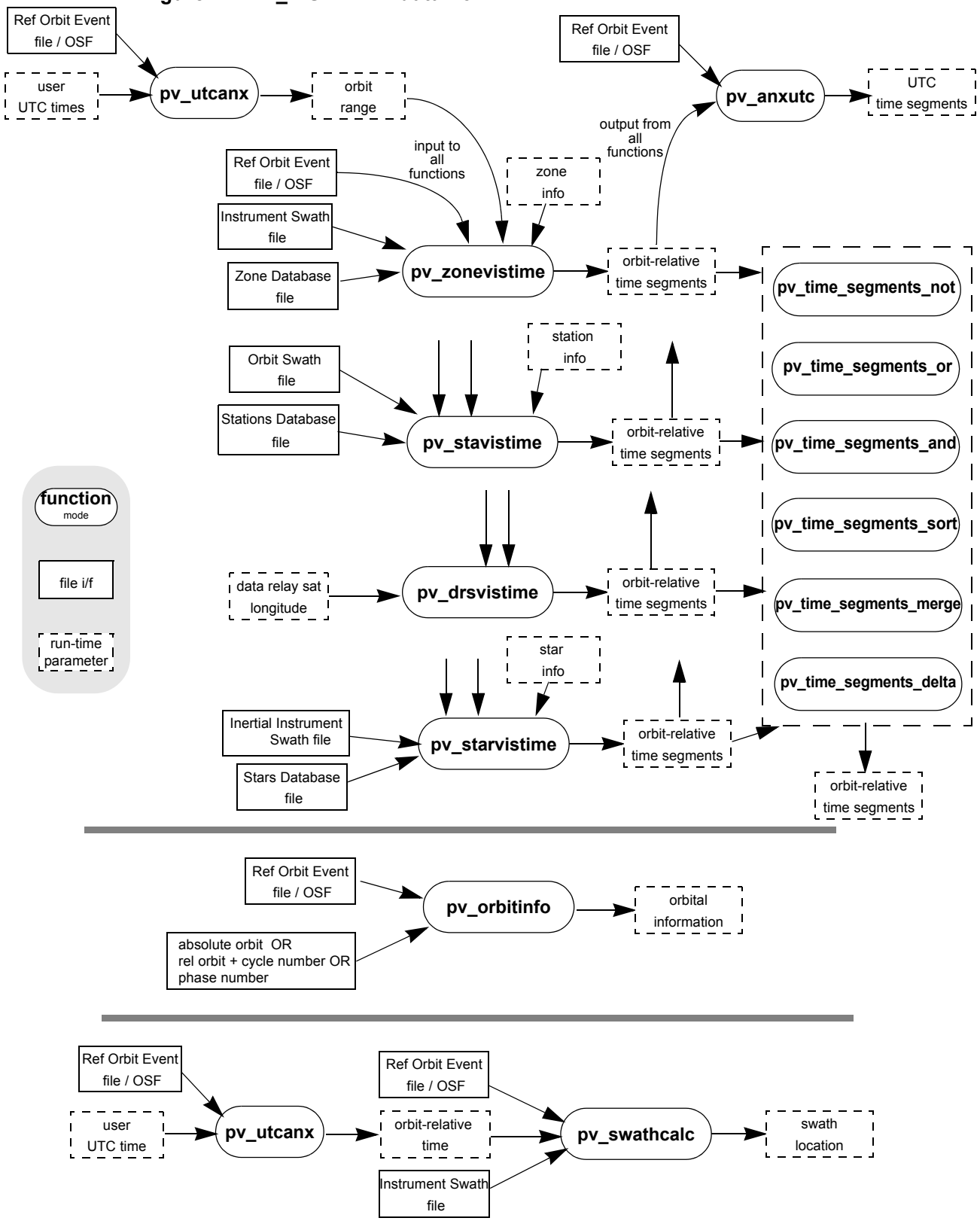
Note that all the above routines use orbit-relative time parameters (i.e. the time parameters are represented as orbit number + time since ascending node). For that reason, 2 ancillary routines are provided to convert to/from UTC:

- **pv_utcanx**: converts from UTC time to orbit-relative time
- **pv_anxutc**: converts from orbit-relative time to UTC time

Note that the 2 time conversion routines above require, like all other routines in PPF_VISIBILITY, a Reference Orbit Event File (or Orbit Scenario File) to operate. For this reason, they have been placed in PPF_VISIBILITY rather than in PPF_LIB where all other (simpler) time-conversion routines are located.

An overview of the PPF_VISIBILITY data flow is presented in Figure 1:

Figure 1: PPF_VISIBILITY data flow



A detailed description of each function is provided in section 7.

Please refer also to:

- RD 2 for a detailed description of the time references and formats, coordinate systems, parameters and models used in this document
- RD 3 for a complete overview of the CFI

5 LIBRARY INSTALLATION

For a detailed description of the installation of any CFI library, please refer to RD 3.

6 LIBRARY USAGE

Note that to use the PPF_VISIBILITY software library, the following other CFI software libraries are required:

- `PPF_LIB` (version 5.9, see RD 3).
- `PPF_ORBIT` (version 5.9, see RD 3).
- `PPF_POINTING` (version 5.9, see RD 3).

To use the PPF_VISIBILITY software library in a user application, that application must include in his source code either:

- `ppf_visibility.h` (for a C application)
- `ppf_visibility.inc` (for a Fortran application under SOLARIS/AIX/LINUX/ MacOS)
- `ppf_visibility_win.inc` (for a Fortran application under Windows 95/NT)

To link correctly his application, the user must include in his linking command flags like (assuming *cfi_libs_dir* and *cfi_include_dir* are the directories where respectively all CFI libraries and include files have been installed, see RD 3 for installation procedures):

- SOLARIS / AIX

  `-I`*cfi_include_dir* `-L`*cfi_lib_dir*

  -lppf_visibility

  -lppf_pointing -lppf_orbit -lppf_lib
- WINDOWS

  `/I` *"cfi_include_dir"* /libpath:*"cfi_lib_dir"* `libppf_pointing.lib`

  libppf_orbit.lib libppf_lib.lib

All functions described in this document have a name starting with the prefix `pv_`

To avoid problems in linking a user application with the PPF_VISIBILITY software library due to the existence of names multiple defined, the user application should avoid naming any global software item beginning with either the prefix `PV_` or `pv_`.

To preserve compatibility with the historical CFI function names, it is possible to call the CFI functions described in this document from a user application with or without the `pv_` prefix.

This is summarized in the table below.

Function Name	Enumeration value	long
Main CFI Functions		
pv_zonevistime zonevistime	PV_ZONEVISTIME_ID	0
pv_stavistime stavistime	PV_STAVISTIME_ID	1

Function Name	Enumeration value	long
pv_drsvistime drsvistime	PV_STAVISTIME_ID	2
pv_swathcalc swathcalc	PV_SWATHCALC_ID	3
pv_anxutc anxutc	PV_UTCANX_ID	4
pv_utcanx utcanx	PV_ANXUTC_ID	5
pv_orbitinfo	PV_ORBITINFO_ID	6
pv_starvistime starvistime	PV_STARVISTIME_ID	7
pv_time_segments_not	PV_TIME_SEGMENTS_NOT_ID	8
pv_time_segments_or	PV_TIME_SEGMENTS_OR_ID	9
pv_time_segments_and	PV_TIME_SEGMENTS_AND_ID	10
pv_time_segments_sort	PV_TIME_SEGMENTS_SORT_ID	11
pv_time_segments_merge	PV_TIME_SEGMENTS_MERGE_ID	12
pv_time_segments_delta	PV_TIME_SEGMENTS_DELTA_ID	13
Error Handling Functions		
pv_verbose	not applicable	
pv_silent		
pv_vector_code		
pv_vector_msg		
pv_print_msg		

Notes about the table:

- to transform the error vector returned by a CFI function to either a list of error codes or list of error messages, the enumeration value (or the corresponding long value) described in the table must be used
- the error handling functions have no enumerated values

6.1 General enumerations

The aim of the current section is to present some of the enumeration values that can be used rather than integer parameters for some of the input parameters of the PPF_VISIBILITY routines, as shown in the table below. The enumerations presented in RD 3 are also applicable, as well as the specific enumerations included for individual functions.

Table 1: Some enumerations within PPF_VISIBILITY library

Input	Description	Enumeration value	Long
Orbit type / Order Criteria	Absolute Orbit	PV_ORBIT_ABS	0
	Relative Orbit	PV_ORBIT_REL	1
Order enumeration	Input Segments ordered by start time	PV_TIME_ORDER	0
	Input Segments not ordered by start time	PV_NO_TIME_ORDER	1

The use of the previous enumeration values could be restricted by the particular usage within the different CFI functions. The actual range to be used is indicated within a dedicated reference named **allowed range**. When there are not restrictions to be mentioned, the allowed range column is populated with the label **complete**.

7 CFI FUNCTIONS DESCRIPTION

The following sections describe each CFI function.

The calling interfaces are described both for C users and Fortran users.

Input and output parameters of each CFI function are described in tables, where C programming language syntax is used to specify:

- parameter types (e.g. long, double)
- array sizes of N elements (e.g. param[N])
- array element M (e.g. [M])

Fortran users should adapt the tables using Fortran syntax equivalent terms:

- parameter types (e.g. long <=> INTEGER*4, double <=>REAL*8)
- array sizes of N elements (e.g. param[N] <=> param (N))
- array element M (e.g. [M] <=> (M+1))

7.1 pv_zonevistime

7.1.1 Overview

The **pv_zonevistime** function computes all the orbital segments for which a given instrument swath intercepts a user-defined zone at the surface of the Earth ellipsoid.

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as seconds (and microseconds) elapsed since the ascending node crossing.

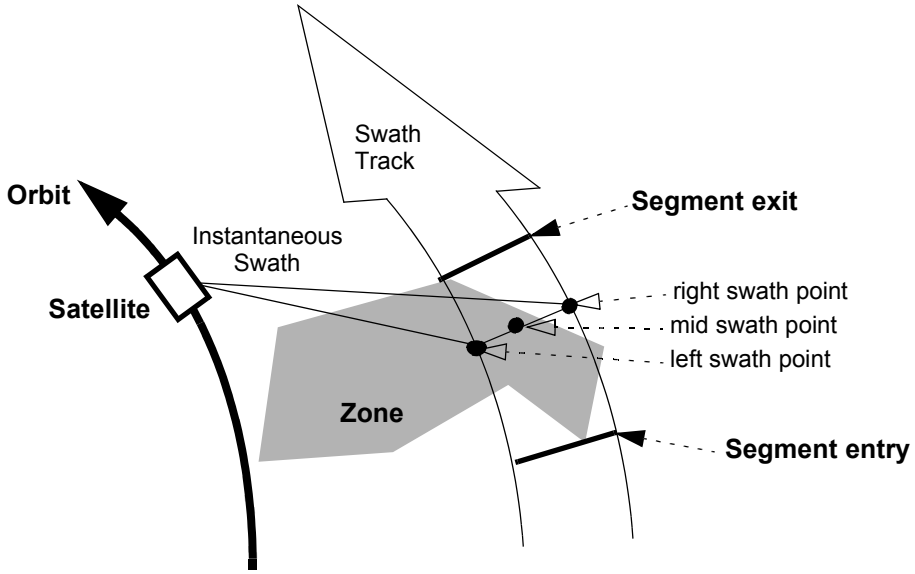
A user-defined zone can be:

- a polygon specified by a set of latitude and longitude points
- a circle specified by the centre latitude, longitude, and the diameter

Note that particular cases of the above can be used to define the zone as:

- a point
- a line

Figure 2 Segment Definition pv_zonevistime



pv_zonevistime requires access to several files to produce its results:

- the Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (**pg_genoef** function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.
- the Instrument Swath File, excluding inertial swath files, describing the area seen by the relevant instrument all along the current orbit. It is produced off-line by the PPF_GENREF CFI software (**pg_genswath** function)
- optionally, a Zone Database File, containing the zone description. The user can either specify a zone identifier referring to a zone in the file, or provide the zone parameters directly to **pv_zonevistime**

Those files are produced off-line by the Envisat-1 Project Team, and delivered to users of PPF_VISIBILITY.

As for the optional Zone Database File:

- the Envisat-1 Project Team can deliver a Zone Database File containing the definition of zones used by the Envisat-1 Global Mission and Background Regional Mission, as these are zones guaranteed to be observed
- the user will usually produce his own Zone Database File if he wants to use this option

The time intervals used by **pv_zonevistime** are expressed in absolute orbit numbers. This is valid for both:

- input parameter "Orbit Range": first and last absolute orbit to be considered
- output parameter "Zone Visibility Segments": time segments with time expressed as {absolute orbit number, number of seconds since ascending node, number of microseconds}

Users who need to use UTC times must make use of the conversion routines provided in PPF_VISIBILITY (**pv_utcanx** and **pv_anxutc** functions).

NOTE: Since the swath template file is generated from a reference orbit, it is not recommended to use **pv_zonevistime** for a range of orbits that includes an orbital change (e.g. change in the repeat cycle or cycle length). If this would happen, **pv_zonevistime** automatically will ignore those orbits that do not correspond with the template file (i.e. no visibility segments will be generated for those orbits). For version 2 of Orbit Event/Orbit Scenario and Swath Template files, only the visibility segments of orbits corresponding to the orbital change of the Swath Template file reference orbit are returned.

7.1.2 Swath Definition

Envisat-1 has 3 main categories of instruments:

- earth-observing instruments ('nadir line' or 'nadir point')
- limb-sounding instruments ('limb', narrow or wide)
- limb-sounding instruments observing inertial objects ('inertial')

These 3 types of instruments have different swath definitions. **pg_genswath** is designed to generate any.

table 2 lists all instrument modes and the relevance of the swaths. It shows also:

- the prefix to be used when generating the swath template file name
- the different types of algorithms to be used by pg_genswath (this is transparent to the user)

The following sub-sections provide some details on the various swath definitions.

7.1.2.1 Earth-observing Instruments Swath Definition

The term swath must be clearly defined to understand the explanations in this document:

- instantaneous swath: the part of the earth surface observed by an instrument at a given time
- swath track: represents the track made on the earth surface by the instantaneous swath over a period of time

Instrument	Mode	File Prefix = swath	pg_genswath algorithm	Swath Type	Remarks
RA		RA_2__	POINT	Nadir point	Modeled as sub-satellite track
MERIS	Averaging / Direct & Averaging	MERIS_	LINE	Nadir line	
ASAR	Image Modes (IS1... IS7)	SARxIM (x=1...7)	ASAR	Nadir line	
	Alt. Polarization (IS1... IS7)				
	Wide Swath	SARWIM			
	Global Monitoring				
	Wave (IS1... IS7)	SARxWV (x=1...7)			Modeled as a continuous swath anywhere within the image swath
GOMOS	Occultation	GOMOIL GOMOIH	INERTIAL	Inertial direction	IFOV much smaller than swath. IFOV Very dependent on star availability. 2 swaths defined: - 1 for high altitude (GOMOIH) - 1 for low altitude (GOMOIL)
	Occultation	GOMO_H GOMO_L	LIMB	Limb wide	Same mode as above, now swath defined as Earth-fixed location. IFOV much smaller than swath. IFOV Very dependent on star availability. 2 swaths defined: - 1 for high altitude (GOMO_H) - 1 for low altitude (GOMO_L)
SCIAMACHY	Nadir / Nadir of Nadir & Limb	SCIAN_	LINE	Nadir line	Continuous Nadir swath modeled
	Limb / Limb of Nadir & Limb	SCIALH SCIALL		Limb wide	2 swaths defined: - 1 for high altitude (SCIALH) - 1 for low altitude (SCIALL)

Table 2: Envisat-1 Swaths

Instrument	Mode	File Prefix = swath	pg_genswath algorithm	Swath Type	Remarks
AATSR		ATSR_N ATSR_F	LINE	Nadir line	2 swaths defined: - 1 for nadir swath - 1 for forward swath
MWR		MWR___	POINT	Nadir point	Modeled as sub-satellite track
MIPAS	Nominal	MIPN_H MIPN_L	LIMB	Limb nar-row	2 swaths defined: - 1 for high altitude (MIPN_H) - 1 for low altitude (MIPN_L)
	Special Event Mode (across)	MIP_X_	LIMB	Limb nar-row	Modeled as an across track swath, in the middle of the MIPAS SEM acquisition scan.
	Special Event Mode (rearward)	MIP_RH MIP_RL	LIMB	Limb wide	IFOV much smaller than swath. 2 swaths defined: - 1 for high altitude (MIP_RH) - 1 for low altitude (MIP_RL)
	Rearward Sideward	MIPIRH MIPIRL MIPIXH MIPIXL	INERTIAL	Inertial direction	2 swaths defined for rearward mode: - 1 for high altitude (MIPIRH) - 1 for low altitude (MIPIRL) 3 swaths defined for sideward mode: - 1 for high altitude (MIPIXH) - 1 for back mode (MIPIXB) - 1 for forward mode (MIPIXF)

Table 2: Envisat-1 Swaths

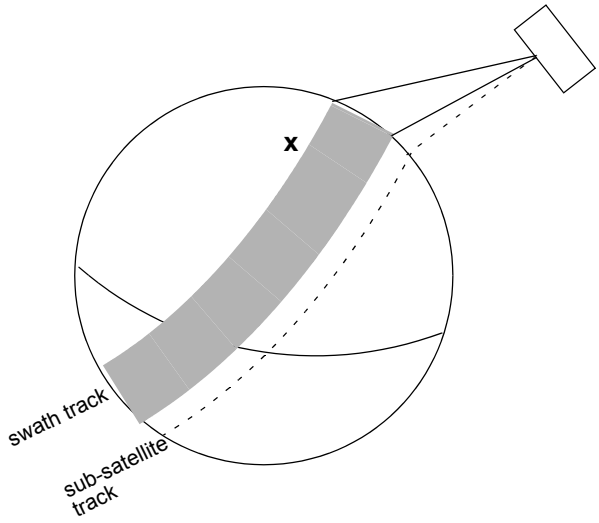
For instruments observing the surface of the earth, the instantaneous swath is constituted by the line (or by the point for instruments like the Radar Altimeter) on the ground observed by the instrument at a given time. It is calculated taking the earth ellipsoid as a reference for the earth surface. The wider the field-of-view of the instrument, the wider the swath on the ground.

When the satellite moves over a period of time, this line (or point) defines a band (or line) on the earth surface. This constitutes the swath track.

See Figure 3 for an illustration of these definitions.

Note that the terms line or point are an idealized view of the instrument FOV, which usually have a thickness.

Figure 3 Earth-observing instrument: swath definition

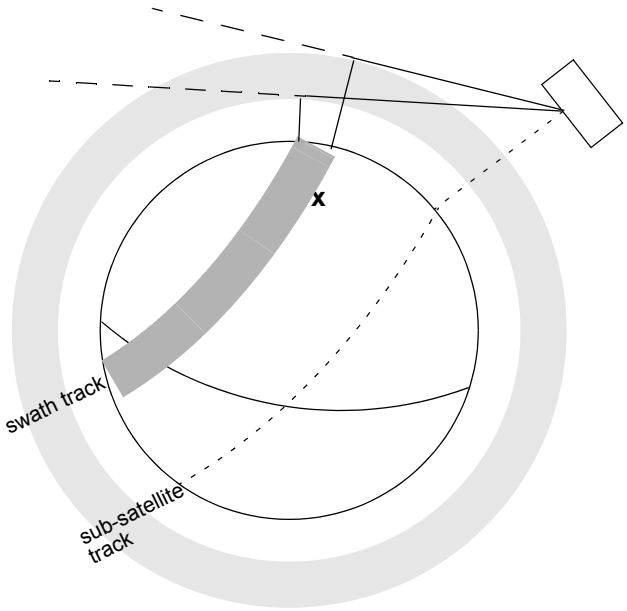


7.1.2.2 Limb-sounding Instruments Swath Definition

For limb sounding instruments, the concept can be generalized to define a “thick swath”. This is obtained by defining a minimum and a maximum altitude, and considering the tangent points to these altitudes as the edges of the swath. 2 cases have to be considered:

- deterministic (narrow) azimuth field of view (e.g. MIPAS sideward-looking): the swath projection on the earth surface is similar to a regular sideward-looking swath, with the lower altitude defining the further swath edge and the higher altitude defining the closer swath edge. See Figure 4.

Figure 4 Limb-sounding instrument: swath definition (1)

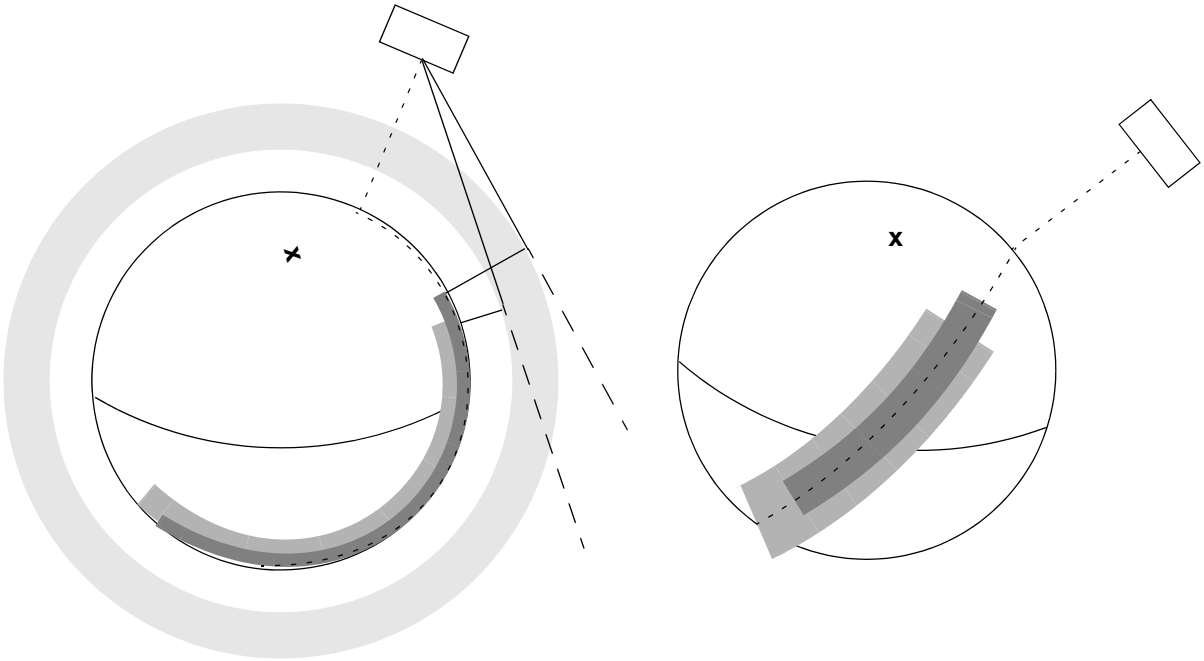


- non-deterministic (potentially wide) azimuth field of view (e.g. MIPAS rearward-looking): due to the potentially wide azimuth field of view, each altitude defines a swath projection on the earth surface. Depending on the altitude, these swaths are of different width across-track, and also at different distance from the satellite. See Figure 5.
- For these, 2 Instrument Swath Files are provided:
  - one at the highest altitude
  - one at the lowest altitude
- The user must handle both swath himself to determine his required visibility time segments.

7.1.2.3 Limb-sounding Instruments Inertial Swath Definition

Both Gomos occultation mode and Mipas Line of Sight mode, observe inertial targets. For the CFI function **pv_starvistime** the FOV direction in inertial coordinates must be available. Therefore for these instrument modes the direction in inertial space, for a given tangent altitude, is given in the swath template file.

Figure 5 Limb-sounding instrument: swath definition (2)



7.1.3 Zone Borders and Projection

When defining a polygon zone, the user is assumed to wish polygon sides as straight lines. But on the earth surface, a straight line is, at best, a confusing concept.

The only way to define unambiguously straight lines is to work in a 2-dimensional projection of the earth surface. There are many possible projections, each having advantages and drawbacks.

pv_zonevistime can handle zone borders in 2 different projections:

- rectangular projection, using longitude and latitude as the X and Y axis; this is appropriate to express zones where (some of) the edges follow constant latitude lines, and provide a reasonable approximation for straight lines at low-medium latitudes
- azimuthal gnomonic projection, where great circles are always projected as straight lines; this is better for high latitudes, where the rectangular projection suffers from too much distortion and the singularity at the poles.

pv_zonevistime allows the user to specify which projection he wants to work in, i.e. in which projection the polygon sides will be represented by **pv_zonevistime** as straight lines. The user is assumed to be aware of how the polygon sides behave on the Earth surface.

7.1.4 Zone Definition

The user-defined zone can be either (see table 3);

- a point
- a line
- a polygon
- a circle

A zone is defined by the area of the earth surface enclosed by the zone borders:

- in the case of a circular zone, the area inside the circle
- in the case of a polygonal zone, the area which is always to the right of any polygon side; if the polygon is defined as a sequence of N points, each polygon side is considered as a line from point i to point i+1; this unambiguously defines the right side of the polygon sides.

For the gnomonic projection, a side of a zone is always smaller than a half great circle, because two polygon points are considered to be joined by the shortest line.

For the rectangular projection, two consecutive points of the zone are also joined by the shortest line; so the difference in longitude must be less than 180 degrees.

The polygon zone can be closed (i.e. the first and last points are the same) or not. If the zone is not closed, pv_zonevistime closes it by joining the last point with the first one in its internal computations.

See Figure 6 for examples of zone definitions.

pv_zonevistime will issue an error on the zone definition if the polygon has intersecting sides ("butterfly" zone)

Zone definition	Zone_num	Zone_long Zone_lat	Zone_diam	Description
Circular Zone	1	[0]: centre point	yes zone_diam > 0.0	The zone is represented as a circle, around the centre point
Point Zone	1	[0]: Point	yes zone_diam = 0.0	The zone is defined by the point. Resulting segments will have a zero duration. The zone will always be completely covered by the swath.
Line Zone	2	[0], [1]: Line	no	The zone is defined by the line from point [0] to point [1].
Polygon Zone	>2	[i]	no	The zone is defined by the area right of the line from point [i] to point [i+1].

Table 3: Zone definition

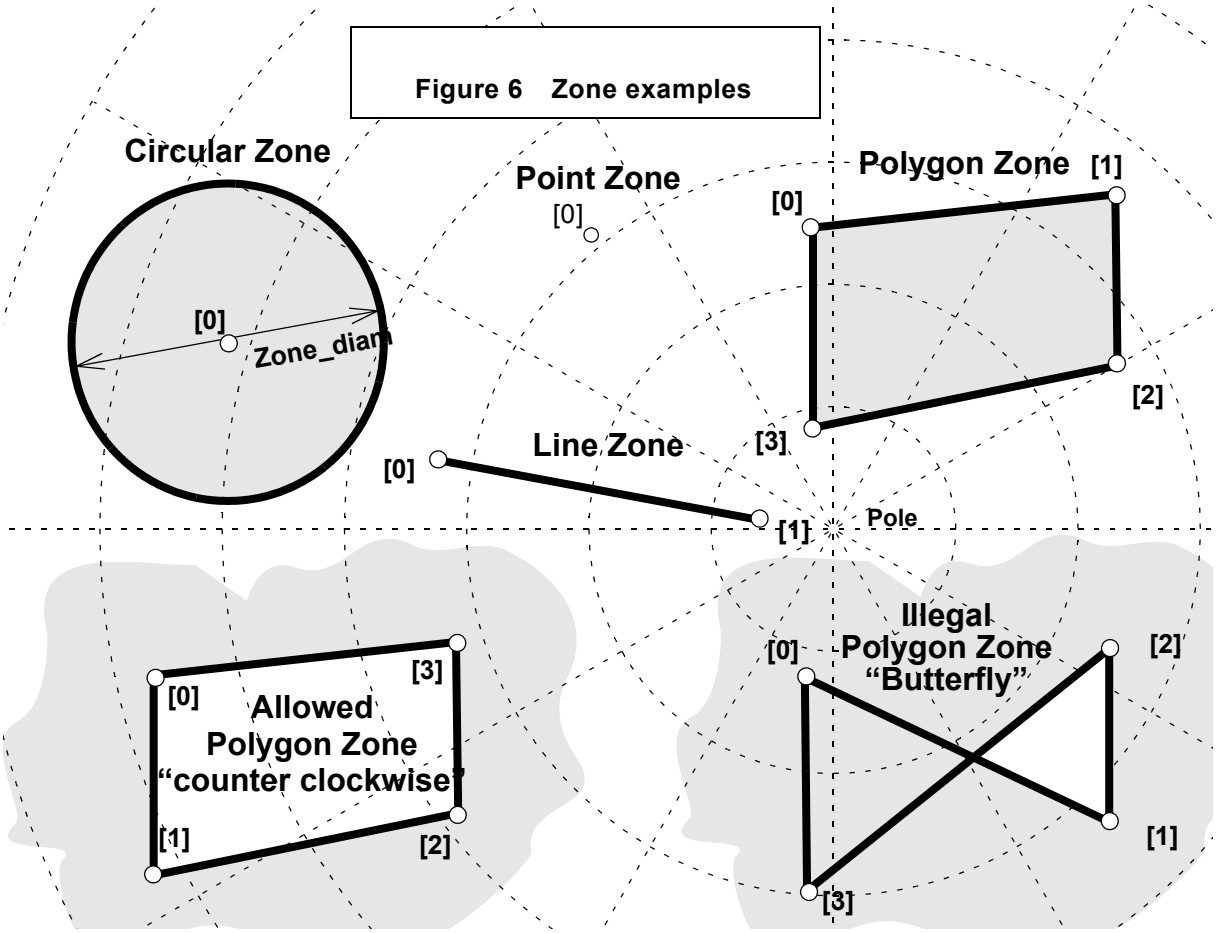


Figure 6 Zone examples

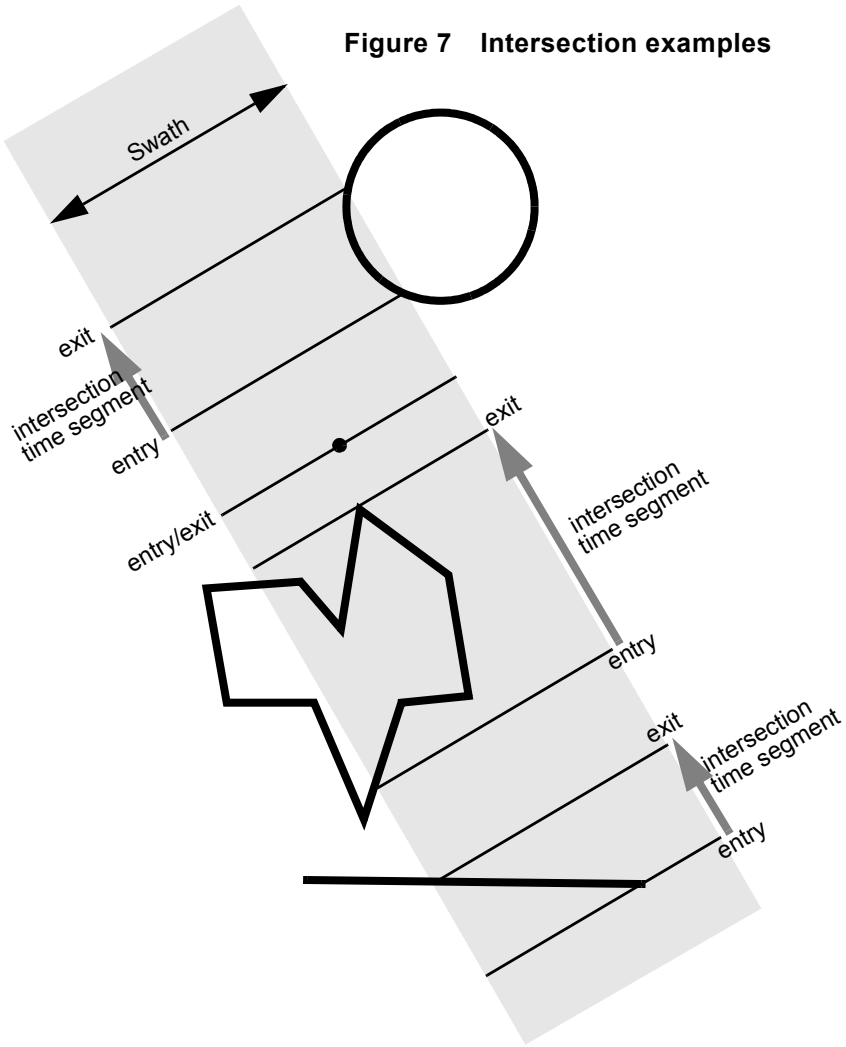
7.1.5 Intersection Definition

The **pv_zonevistime** intersection times between the instrument swath and the user-defined zone are defined as the first and last occurrence, in chronological order with respect to the satellite direction, of the geometrical super-position of any point belonging to the instrument swath with any single point belonging to the zone (including the zone border).

The entry and exit times for each intersection are given as elapsed seconds (and microseconds) since the ascending node crossing.

Figure 7 shows some typical intersections.

Figure 7 Intersection examples



7.1.6 Intersection Algorithm

The intersection of an Envisat-1 instrument swath and a user-defined zone is to be performed on the Earth projected to a map plane in one of the following projections:

- Rectangular projection
- Gnomonic projection

Although the projections are quite different, the intersection rules are identical. The algorithm can however be different, in order to take advantage of a particular feature of a projection.

The purpose of the CFI function ZONEVISTIME is to obtain quickly, accurate intersection segments with a low precision (1 second).

The algorithms assume that the polygon zones are closed and expects a wrap around between the first

and the last point. Thus ZONEVISTIME must first close the polygon if necessary.

For ZONEVISTIME the following swath types are defined:

- point swath: instantaneous swath is a point (e.g. RA-2)
- line swath: instantaneous swath is a line (e.g. ASAR)
- inertial swath: not used by ZONEVISTIME

The main concept in the algorithm is the transition, defined as the change in coverage of (part of) the swath and the zone (e.g. edge of the swath crosses one polygon side).

7.1.6.1 Intersection with a point swath.

The vertices of the polygon defining the area are connected by straight lines in the chosen projection, along track swath points are also connected by straight lines in the same projection.

Transitions are located by linear intersection of the zone sides and the swath along track lines. A transition is only valid if the intersection occurs inside both line segments. The polygon side from <i> to <j> is defined in a clockwise manner inclusive point <i> but exclusive point <j>. The swath line from time <k> to <l> is defined inclusive the template point at <k> but exclusive the template point at <l>.

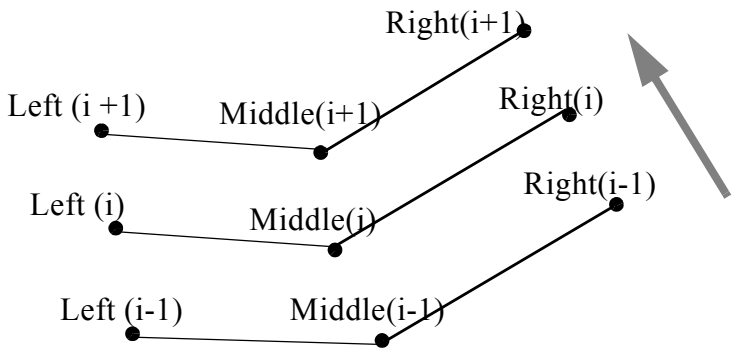
The fraction of the swath along track line determines the precise timing since time <k> of the intersection. Also the determination if the transition is a on- or off-transition is quite trivial. First a vector is defined, perpendicular to the along track swath line, such that the vector points left. Then, the dot product of the polygon side and this vector is calculated. If the dot product is positive, the transition is on, i.e. the swath enters the zone. If the result is negative, then the swath leaves the zone. If the result equals zero then the transition can be ignored (polygon side and swath overlay, a proper transition will be found with another pair of polygon side - swath line.) .

7.1.6.2 Intersection with a line swath

The left, middle and right side of the swath, are located using the same algorithm as for the point swath. Even left, middle and right time segments can be made based on the left, middle and right hand transitions.

The polygon vertices (and not the sides) are intersected with the along track moving line swath, in order to catch zones smaller than the swath, etc. Swaths for intermediate times between two consecutive times in Swath Template File are considered straight segments, the first one joining an intermediate point of the Left swath line from time <k> to time <l>, with an intermediate point in Middle swath line, and the other segment joining this intermediate point in Middle swath line with an intermediate point in Right swath line.

Figure 8 Swath points



7.1.7 Usage Hints

7.1.7.1 Limb-sounding Instruments Intersection

In the case of limb-sounding instrument with a potentially wide azimuth field of view, 2 swaths have to be considered (1 for minimum altitude, 1 for maximum altitude). Furthermore, these 2 swaths are offset in time (i.e. their projection on the earth intersect with a given point at different times). To cope with this, the user must do the following:

- call **pv_zonevistime** twice (once for each extreme altitude swath)
- merge/filter the 2 sets of time segments, depending on what he wants to achieve

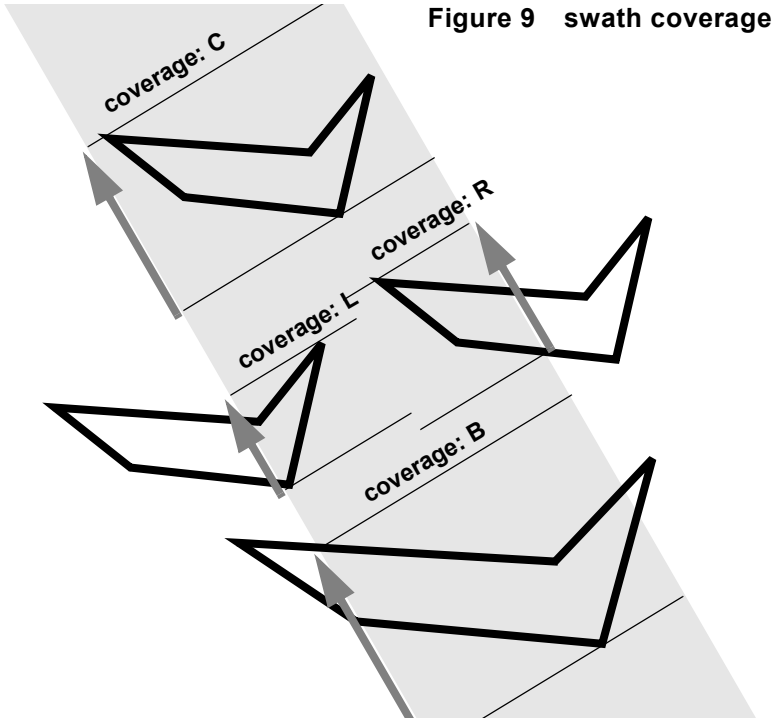
7.1.7.2 Zone Coverage

pv_zonevistime computes purely geometrical intersections. The resulting zone visibility segments might need some additional filtering by the user. In particular, instrument constraints (e.g. only working outside of sun eclipse) have to be considered by the user.

Furthermore, to help users to deal with zones wider than the swath (i.e. requiring several orbits to cover the whole zone), **pv_zonevistime** produces for each zone visibility segment an indication of the coverage type (see Figure 9);

- coverage = C: zone completely covered by the swath
- coverage = R: zone partially covered by the swath, extending over the right edge of the swath
- coverage = L: zone partially covered by the swath, extending over the left edge of the swath
- coverage = B: zone partially covered by the swath, extending over both edges of the swath

Figure 9 swath coverage definition



7.1.7.3 Combined use of pv_swathcalc and the coverage flag

The PPF_VISIBILITY function **pv_swathcalc** can be used to refine the work performed with

pv_zonevistime.

7.1.8 Calling sequence

For C programs, the call to **pv_zonevistime** is (input parameters are underlined):

```
#include"ppf_visibility.h"
#define MAX_SEGMENTS <your value here>
#define ZONE_NUM <your value here>
{
       long     start_orbit, stop_orbit, zone_num,
                projection, max_segments, number_segments,
                bgn_orbit[MAX_SEGMENTS],
                bgn_second[MAX_SEGMENTS],
                bgn_microsec[MAX_SEGMENTS],
                end_orbit[MAX_SEGMENTS],
                end_second[MAX_SEGMENTS],
                end_microsec[MAX_SEGMENTS],
                coverage[MAX_SEGMENTS], ierr[10], status;
       double   zone_long[ZONE_NUM], zone_lat[ZONE_NUM],
                zone_diam, min_duration;
       char     *orbit_event_file, *swath_file;
       char     zone_id[8], *zone_db_file;

       max_segments = MAX_SEGMENTS;

       status = pv_zonevistime (
                           orbit_event_file, &start_orbit, &stop_orbit,
                           swath_file, zone_id, zone_db_file,
                           &projection, &zone_num,
                           zone_long, zone_lat, &zone_diam,
                           &max_segments, &min_duration,
                           &number_segments,
                           bgn_orbit, bgn_second, bgn_microsec,
                           end_orbit, end_second, end_microsec,
                           coverage, ierr);

/* test status */
}
```

For FORTRAN programs **pv_zonevistime** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
        INTEGER*4   START_ORBIT, STOP_ORBIT, ZONE_NUM, PROJECTION,
       &            MAX_SEGMENTS, NUMBER_SEGMENTS,
       &            BGN_ORBIT(MAX_SEGMENTS),
       &            BGN_SECOND(MAX_SEGMENTS),
       &            BGN_MICROSEC(MAX_SEGMENTS),
       &            END_ORBIT(MAX_SEGMENTS),
       &            END_SECOND(MAX_SEGMENTS),
       &            END_MICROSEC(MAX_SEGMENTS),
```

```
      &             COVERAGE(MAX_SEGMENTS), IERR(10), STATUS
       REAL*8       ZONE_LONG(ZONE_NUM), ZONE_LAT(ZONE_NUM),
      &             ZONE_DIAM, MIN_DURATION
       CHARACTER*(*)    ORBIT_EVENT_FILE, SWATH_FILE, ZONE_DB_FILE
       CHARACTER*8      ZONE_ID

#include"ppf_visibility.inc"

       STATUS = PV_ZONEVISTIME (
                           ORBIT_EVENT_FILE, START_ORBIT, STOP_ORBIT,
      &                    SWATH_FILE, ZONE_ID, ZONE_DB_FILE,
      &                    PROJECTION, ZONE_NUM,
      &                    ZONE_LONG, ZONE_LAT, ZONE_DIAM,
      &                    MAX_SEGMENTS, MIN_DURATION,
      &                    NUMBER_SEGMENTS,
      &                    BGN_ORBIT, BGN_SECOND, BGN_MICROSEC,
      &                    END_ORBIT, END_SECOND, END_MICROSEC,
      &                    COVERAGE, IERR)


C test status
```

7.1.9 Input parameters pv_zonevistime

c name	c type	Array Ele-ment	Description	Units	Range
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
start_orbit	long		First orbit, segment filter. Segments will be filtered as from the beginning of first orbit (within orbit range from orbit_event_file) If set to zero then first orbit of stop_orbit orbital change is selected.	abso-lute orbit number	(=0) or (>= start_oef and <= stop_oef)
stop_orbit	long		Last orbit, segment filter. Segments will be filtered until the end of last orbit (within orbit range from orbit_event_file) If set to zero then last orbit of start_orbit orbital change is selected.	abso-lute orbit number	= 0 or >= start_orbit <= stop_oef
swath_file	char *		File name of the swath-file for the appropriate instrument mode If empty string (“”), the file last read in a previous call is used (saves computation time)		
zone_id[8]	char		Identification of the zone, as defined in zone_db_file. This parameter is used ONLY IF zone_num = 0		EXACTLY 8 characters
zone_db_file	char *		File name of the zone-database-file. This file is used ONLY IF zone_num = 0		

Table 4: Input parameters for pv_zonevistime

c name	c type	Array Element	Description	Units	Range
projection	long		projection used to define polygon sides as straight lines: = 0 Read projection from Zones DB (rectangular projection is used by default if the DB does not contain a projection) = 1 Azimuthal gnomonic = 2 Rectangular lat/long		
zone_num	long		Number of vertices of the zone provided in zone_long, zone_lat: = 0 no vertices provided, use zone_id / zone_db_file = 1 Point / Circular zone, = 2 Line zone > 2 Polygon zone		>= 0
zone_long [zone_num]	double	all	zone_long[i-1] Geocentric longitude of - circle centre, for circ. zone, i = 1 - point, for point zone, i = 1 - line-end, for line zone, i = 1 or 2 - vertices, for polygon zone, i = 1... zone_num		
zone_lat [zone_num]	double	all	zone_lat[i-1] Geodetic latitude of - circle centre, for circ. zone, i = 1 - point, for point zone, i = 1 - line-end, for line zone, i = 1 or 2 - vertices, for polygon zone, i = 1... zone_num		
zone_diam	double		Zone diameter for circular zones, dummy for other zones If diameter equals 0.0 then zone is Point Zone	m	>= 0.0
max_segments	long		Size of the segment arrays.		>0
min_duration	double		Minimum duration for segments. Only segments with a duration longer than min_duration will be given on output.	s	>= 0.0

Table 4: Input parameters for pv_zonevistime

It is also possible to use enumeration values rather than integer values for some of the input argu-

ments, as shown in the table below:

Input	Description	Enumeration value	long
projection	Read projection from the zones DB file	PV_READ_DB	0
		PV_GNOMONIC	1
	Azimuthal Gnomonic	PV_RECTANGULAR	2
	Rectangular long/lat		

7.1.10 Output parameters pv_zonevistime

c name	c type	Array Ele-ment	Description	Unit	Range
pv_zonevistime	long		Function status flag, 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
number_segments	long		Number of visibility segments returned to the user.		>= 0
bgn_orbit [max_segments]	long	all	Orbit number, begin of visibility segment i bgn_orbit[i-1], i = 1, number_segments		> 0
bgn_second [max_segments]	long	all	Seconds since ascending node, begin of visibility segment i bgn_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
bgn_microsec [max_segments]	long	all	Micro seconds within second begin of visibility segment i bgn_microsec[i-1], i = 1, number_segments	μs	0 =< =< 999999
end_orbit [max_segments]	long	all	Orbit number, end of visibility segment i end_orbit[i-1], i = 1, number_segments		> 0
end_second [max_segments]	long	all	Seconds since ascending node, end of visibility segment i end_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
end_microsec [max_segments]	long	all	Micro seconds within second end of visibility segment i end_microsec[i-1], i = 1, number_segments	μs	0 =< =< 999999

Table 5: Output parameters for pv_zonevistime

c name	c type	Array Ele-ment	Description	Unit	Range
coverage [max_segments]	long	all	Zone coverage flag for segment = 0 Zone completely covered by swath = 1 Zone not completely covered by swath, extending over the left edge of the swath. = 2 Zone not completely covered by swath, extending over the right edge of the swath. = 3 Zone not completely covered by swath, extending over both edges of the swath coverage[i], i = 0, (number_segments-1)		
ierr[10]	long		Error status flags		

Table 5: Output parameters for pv_zonevistime

It is also possible to use enumeration values rather than integer values for some of the output arguments, as shown in the table below:

Input	Description	Enumeration value	long
coverage	Zone completely covered by swath	PV_COMPLETE	0
	Left extreme transitions found	PV_LEFT	1
	Right extreme transitions found	PV_RIGHT	2
	Both extreme transitions found	PV_BOTH	3

7.1.11 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_zonevistime** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg** (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_zonevistime** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code** (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Input parameter "Number of ZONE points" cannot be negative.	Computation not performed	PV_CFI_ZONEVISTIME_NEGATIVE_NUM_ZONE_ERR	0
ERR	Input parameter "Maximum number of segments" cannot be negative.	Computation not performed	PV_CFI_ZONEVISTIME_NEGATIVE_MAX_SEGMENTS_ERR	1
ERR	Input parameter "Minimum duration" cannot be negative.	Computation not performed	PV_CFI_ZONEVISTIME_NEGATIVE_MIN_DURATION_ERR	2
ERR	Input parameter "Projection" out of range.	Computation not performed	PV_CFI_ZONEVISTIME_PROJECTION_OUT_OF_RANGE_ERR	3
ERR	Error reading Swath Template File.	Computation not performed	PV_CFI_ZONEVISTIME_READ_SWATH_FILE_ERR	4
ERR	Swath type not allowed	Computation not performed	PV_CFI_ZONEVISTIME_INCORRECT_SWATH_TYPE_ERR	5
ERR	Cannot allocate memory for the Swath Template File	Computation not performed	PV_CFI_ZONEVISTIME_ALLOCATE_SWATH_MEMORY_ERR	6
ERR	Input parameter "start_orbit" cannot be negative.	Computation not performed	PV_CFI_ZONEVISTIME_NEGATIVE_START_ORBIT_ERR	7
ERR	Error reading OEF/OSF file.	Computation not performed	PV_CFI_ZONEVISTIME_READ_OEF_OSF_ERR	8
WARN	"start_orbit" is before the first orbit in "orbit_event_file".	Computation performed from the first orbit of the OEF. Message to inform the user.	PV_CFI_ZONEVISTIME_EARLIER_START_ORBIT_WARN	9
WARN	"stop_orbit is after the last orbit in "orbit_event_file".	Computation performed until the stop orbit of the OEF. Message to inform the user.	PV_CFI_ZONEVISTIME_LATER_STOP_ORBIT_WARN	10

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Input parameter "start_orbit" cannot be greater than "stop_orbit" .	Computation not performed	PV_CFI_ZONEVISTIME_WRONG_ORBIT_RANGE_ERR	11
ERR	Error calling "pv_orbitinfo".	Computation not performed	PV_CFI_ZONEVISTIME_ORBITINFO_CALL_ERR	12
ERR	"cycle_length" read from the input "Swath Template File" is not equal to that of any orbits within the orbit range	Computation not performed	PV_CFI_ZONEVISTIME_INCONSISTENT_SWATH_ERR	13
WARN	There is at least one orbital change within the requested orbit range.	Some orbits are not taken into account, those inconsistent with the cycle length of the STF.	PV_CFI_ZONEVISTIME_ORBITAL_CHANGE_WARN	14
ERR	Input parameter "zone_id" is an empty string.	Computation not performed	PV_CFI_ZONEVISTIME_ZONE_ID_EMPTY_ERR	15
ERR	Number of characters in input string "zone_id" is different from n.	Computation not performed	PV_CFI_ZONEVISTIME_WRONG_ZONE_ID_LENGTH_ERR	16
ERR	Error reading the ZONE Database file.	Computation not performed	PV_CFI_ZONEVISTIME_READ_ZONE_DB_FILE_ERR	17
WARN	"Projection" parameter set to default.	Computation performed. Message to inform the user.	PV_CFI_ZONEVISTIME_DEFAULT_PROJECTION_WARN	18
ERR	Cannot allocate memory for the ZONE records.	Computation not performed	PV_CFI_ZONEVISTIME_ALLOCATE_ZONE_MEMORY_ERR	19
ERR	Latitude must be in the range [-90.0 ; 90.0].	Computation not performed	PV_CFI_ZONEVISTIME_WRONG_LATITUDE_RANGE_ERR	20
WARN	Two consecutive points are equal, only one point is used.	Computation performed. Message to inform the user.	PV_CFI_ZONEVISTIME_TWO_EQUAL_POINTS_WARN	21
ERR	Difference in longitude for 2 consecutive ZONE points is equal to 180.0 degrees (RECTANGULAR projection). Zone definition is ambiguous.	Computation not performed	PV_CFI_ZONEVISTIME_DIFF_LONG_180_ERR	22
ERR	Two consecutive ZONE points are antipodal (GNOMONIC projection). Zone definition is ambiguous.	Computation not performed	PV_CFI_ZONEVISTIME_ANTIPODAL_POINTS_ERR	23
ERR	Error precomputing intersection of two segments.	Computation not performed	PV_CFI_ZONEVISTIME_SEGMENT_INTERSECT_PREC_ERR	32

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error computing intersection of two segments.	Computation not performed	PV_CFI_ZONEVISTIME_SEGMENT_INTERSECT_COMP_ERR	33
ERR	Error computing gnomonic coordinates.	Computation not performed	PV_CFI_ZONEVISTIME_GNOMONIC_COORD_ERR	34
ERR	Two ZONE segments intersect.	Computation not performed	PV_CFI_ZONEVISTIME_TWO_SEGMENTS_INTERSECT_ERR	35
ERR	Two consecutive ZONE segments are aligned in the opposite direction.	Two segments cancel each other and ZONEVISTIME does not support this feature. Computation not performed	PV_CFI_ZONEVISTIME_ALIGNED_SEGMENTS_ERR	36
ERR	Input parameter "ZONE diameter" cannot be negative (POINT or CIRCLE zone).	Computation not performed	PV_CFI_ZONEVISTIME_ZONE_DIAM_NEGATIVE_ERR	37
ERR	SWATH contains the POLE (RECTANGULAR projection).	Computation not performed	PV_CFI_ZONEVISTIME_POLE_IN_SWATH_ERR	38
ERR	Not convex SWATH quadrilateral for the specified latitude range.	Computation not performed	PV_CFI_ZONEVISTIME_CUADRILATERAL_NOT_CONVEX_ERR	39
ERR	Error checking if a point is inside a cuadrilateral.	Computation not performed	PV_CFI_ZONEVISTIME_POINT_IN_CUADRILATERAL_ERR	40
ERR	Error sorting intersections.	Computation not performed	PV_CFI_ZONEVISTIME_SORT_INTERSECTIONS_ERR	41
ERR	Maximum number of segments (OFF's) reached.	Computation not performed. No valid output.	PV_CFI_ZONEVISTIME_MAX_OFF_NUMBER_REACHED_ERR	42
ERR	Maximum number of segments (ON's) reached.	Computation not performed No valid output.	PV_CFI_ZONEVISTIME_MAX_ON_NUMBER_REACHED_ERR	43
WARN	Warning checking the visibility segments.	Message to inform the user.	PV_CFI_ZONEVISTIME_CHECK_SEGMENTS_ERR	44
ERR	Error checking the visibility segments.	Computation not performed	PV_CFI_ZONEVISTIME_CHECK_SEGMENTS_ERR	45
ERR	Error computing the final segments for the POINT swath and POINT zone.	Computation not performed	PV_CFI_ZONEVISTIME_ANXUTC_CALL_ERR	46
ERR	Error checking orbit range cycle	Computation not performed. The input orbit range is incompatible with STF cycle	PV_CFI_ZONEVISTIME_CALL_ORBIT_RANGE_CYCLE_ERR	47

Error type	Error message	Cause and impact	Error Code	Error No
ERR	The requested orbit range is not contained in swath cycle	Computation not performed The input orbit range is not compatible with STF cycle	PV_CFI_ZONEVISTIME_ORBIT_RANGE_CYCLE_ERR	48
WARN	The orbit %li is not contained in swath cycle	Message to inform the user. The orbits not compatible with swath cycle are not taken into account in computation	PV_CFI_ZONEVISTIME_ORBIT_RANGE_CYCLE_WARN	49
ERR	Error calculating start/stop orbits	Computation not performed. Error trying to compute start/stop orbits when 0 is entered as start_orbit or stop_orbit.	PV_CFI_ZONEVISTIME_ORB_DEFAULT_VALUES_ERR	50

Note that error codes and messages have been completely modified since the last issue due to a completely new implementation of the CFI function.

7.1.12 Runtime performances

The following runtime performances have been measured:

orbits	case	Ultra Sparc [sec]
POINT ZONE:		
501	Point Swath + Rectangular projection	0.55
501	Point Swath + Gnomonic projection	0.66
501	Linear Swath + Rectangular projection	0.44
501	Linear Swath + Gnomonic projection	0.67
SEGMENT ZONE:		
501	Point Swath + Rectangular projection	2.0
501	Point Swath + Gnomonic projection	16.88
501	Linear Swath + Rectangular projection	9.89
501	Linear Swath + Gnomonic projection	53.09
CIRCLE ZONE:		
501	Point Swath + Rectangular projection	6.75
501	Point Swath + Gnomonic projection	6.67
501	Linear Swath + Rectangular projection	15.80
501	Linear Swath + Gnomonic projection	18.91
POLYGON ZONE (small quadrilateral):		
501	Point Swath + Rectangular projection	0.57
501	Point Swath + Gnomonic projection	0.71
501	Linear Swath + Rectangular projection	1.46
501	Linear Swath + Gnomonic projection	1.83
POLYGON ZONE (large quadrilateral):		
501	Point Swath + Rectangular projection	2.29
501	Point Swath + Gnomonic projection	7.13
501	Linear Swath + Rectangular projection	5.92
501	Linear Swath + Gnomonic projection	31.53
POLYGON ZONE (few points):		

orbits	case	Ultra Sparc [sec]
501	Point Swath + Rectangular projection	2.08
501	Point Swath + Gnomonic projection	2.64
501	Linear Swath + Rectangular projection	10.63
501	Linear Swath + Gnomonic projection	16.72
POLYGON (many points):		
501	Point Swath + Rectangular projection	5.63
501	Point Swath + Gnomonic projection	5.91
501	Linear Swath + Rectangular projection	18.92
501	Linear Swath + Gnomonic projection	22.02
POLYGON ZONE (complex zone)		
501	Point Swath + Rectangular projection	21.67
501	Point Swath + Gnomonic projection	23.15
501	Linear Swath + Rectangular projection	67.79
501	Linear Swath + Gnomonic projection	81.29

The above times depend strongly on:

- the number of sides of the polygon
- the type of swath (point or linear)
- the projection (gnomonic projection may be up to 10 times slower than rectangular).
- the width in longitude of each side of the polygon
- the width in latitude for the whole zone.

7.2 pv_stavistime

7.2.1 Overview

The **pv_stavistime** function computes groundstation visibility segments, the orbital segments for which the satellite is visible from a ground station located at the surface of the Earth.

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as seconds elapsed since the ascending node crossing.

In addition, **pv_stavistime** calculates for every visibility segment the time of zero-doppler (i.e. the time at which the range-rate to the station is zero).

pv_stavistime requires access to several files to produce its results:

- the Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (**pg_genoef** function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.
- the Station Database File, describing the location and the physical mask of each ground station
- the Orbit Swath File

The time intervals used by **pv_stavistime** are expressed in absolute orbit numbers. This is valid for both:

- input parameter “Orbit Range”: first and last absolute orbit to be considered
- output parameter “Station Visibility Segments”: time segments with time expressed as {absolute orbit number, number of seconds since ANX, number of microseconds}

Users who need to use UTC times must make use of the conversion routines provided in PPF_VISIBILITY (**pv_utcanx** and **pv_anxutc** functions).

NOTE: Since the orbit swath template file is generated from a reference orbit, it is not recommended to use **pv_stavistime** for a range of orbits that includes an orbital change (e.g. change in the repeat cycle or cycle length). If this would happen, **pv_stavistime** automatically will ignore those orbits that do not correspond with the template file (i.e. no visibility segments will be generated for those orbits). For version 2 of Orbit Event/Orbit Scenario and Swath Template files, only the visibility segments of orbits corresponding to the orbital change of the Swath Template file reference orbit are returned.

7.2.2 Calling sequence pv_stavistime

For C programs, the call to **pv_stavistime** is (input parameters are underlined):

```
#include"ppf_visibility.h"
#define MAX_SEGMENTS <your value here>
{
      long          start_orbit, stop_orbit,
                    max_segments, number_segments,
                    bgn_orbit[MAX_SEGMENTS],
                    bgn_second[MAX_SEGMENTS],
                    bgn_microsec[MAX_SEGMENTS],
                    end_orbit[MAX_SEGMENTS],
                    end_second[MAX_SEGMENTS],
                    end_microsec[MAX_SEGMENTS],
                    zdop_orbit[MAX_SEGMENTS],
                    zdop_second[MAX_SEGMENTS],
                    zdop_microsec[MAX_SEGMENTS],
                    ierr[10], mask, status;
      double        aos_elevation, los_elevation, min_duration;
      char          *orbit_event_file, *orbit_swath_file;
      char          sta_id[8],*sta_db_file;

      max_segments = MAX_SEGMENTS;

      status = pv_stavistime (
                        orbit_event_file, &start_orbit, &stop_orbit,
                        orbit_swath_file, sta_id, sta_db_file,
                        &mask, &aos_elevation, &los_elevation,
                        &max_segments, &min_duration,
                        &number_segments,
                        bgn_orbit, bgn_second, bgn_microsec,
                        end_orbit, end_second, end_microsec,
                        zdop_orbit, zdop_second, zdop_microsec,
                        ierr);

/* test status */
}
```

For FORTRAN programs **pv_stavistime** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
       INTEGER*4    START_ORBIT, STOP_ORBIT,
      &             MAX_SEGMENTS, NUMBER_SEGMENTS,
      &             BGN_ORBIT(MAX_SEGMENTS),
      &             BGN_SECOND(MAX_SEGMENTS),
      &             BGN_MICROSEC(MAX_SEGMENTS),
      &             END_ORBIT(MAX_SEGMENTS),
```

```
      &             END_SECOND(MAX_SEGMENTS),
      &             END_MICROSEC(MAX_SEGMENTS),
      &             ZDOP_ORBIT(MAX_SEGMENTS),
      &             ZDOP_SECOND(MAX_SEGMENTS),
      &             ZDOP_MICROSEC(MAX_SEGMENTS),
      &             MASK, IERR(10), STATUS
       REAL*8       AOS_ELEVATION, LOS_ELEVATION, MIN_DURATION
       CHARACTER*(*)    ORBIT_EVENT_FILE, ORBIT_SWATH_FILE, STA_DB_FILE
       CHARACTER*8      STA_ID

#include"ppf_visibility.inc"

       STATUS = PV_STAVISTIME (
                           ORBIT_EVENT_FILE, START_ORBIT, STOP_ORBIT,
      &                    ORBIT_SWATH_FILE, STA_ID, STA_DB_FILE,
      &                    MASK, AOS_ELEVATION, LOS_ELEVATION,
      &                    MAX_SEGMENTS, MIN_DURATION,
      &                    NUMBER_SEGMENTS,
      &                    BGN_ORBIT, BGN_SECOND, BGN_MICROSEC,
      &                    END_ORBIT, END_SECOND, END_MICROSEC,
      &                    ZDOP_ORBIT, ZDOP_SECOND, ZDOP_MICROSEC,
      &                    IERR)


C test status
```

7.2.3 Input parameters pv_stavistime

c name	c type	Array Ele-ment	Description	Units	Range
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves compu-tation time)		
start_orbit	long		First orbit, segment filter. Segments will be filtered as from the beginning of first orbit (within orbit range from orbit_event_file) If set to zero then first orbit of stop_orbit orbital change is selected.	abso-lute orbit number	(=0) or (>= start_oef and <= stop_oef)
stop_orbit	long		Last orbit, segment filter. Segments will be filtered until the end of last orbit (within orbit range from orbit_event_file) If set to zero then last orbit of orbital change of start_orbit is selected.	abso-lute orbit number	= 0 or >= start_orbit <= stop_oef
orbit_swath_file	char *		File name of the orbit swath-file If empty string (“”), the file last read in a previous call is used (saves computation time)		
sta_id[8]	char		identification name of the station		
sta_db_file	char *		File name of the station database file This file is read each time the func-tion is called		
aos_elevation	double		Minimum elevation to consider at AOS (i.e. before considering start of visibility).	deg	>= 0.0
los_elevation	double		Maximum elevation to consider at LOS (i.e. before considering end of visibility).	deg	>= 0.0 <= aos_elevati on

Table 6: Input parameters for pv_stavistime

c name	c type	Array Element	Description	Units	Range
mask	long		mask used to define visibility = 0 combine AOS/LOS elevations and physical mask (nominal mode) = 1 consider only AOS/LOS elevations = 2 consider only physical mask		>=0
max_segments	long		Size of the segment arrays. There is an internal limitation to 5000 segments. In case the internal limitation is not reached but more visibility segments than max_segments are detected only max_segments will be given on output.		>0 <5000
min_duration	double		Minimum duration for segments. Only segments with a duration longer than min_duration will be given on output.	s	>= 0.0

Table 6: Input parameters for pv_stavistime

It is also possible to use enumeration values rather than integer values for some of the input arguments, as shown in the table below:

Input	Description	Enumeration value	long
mask	Combine AOS/LOS and physical mask	PV_COMBINE	0
	Use only AOS/LOS	PV_AOS_LOS	1
	Use only physical mask	PV_PHYSICAL	2

7.2.4 Output parameters pv_stavistime

c name	c type	Array Ele-ment	Description	Unit	Range
pv_stavistime	long		Function status flag, 0 = No error > 0 Warnings, results generated < 0 Error, no results generated		
number_segments	long		Number of visibility segments returned to the user (it can be the number of visibility segments found or max_segments, when the number of visibility segments found is greater than max_segments)		>= 0
bgn_orbit [max_segments]	long	all	Orbit number, begin of visibility segment i bgn_orbit[i-1], i = 1, number_segments		> 0
bgn_second [max_segments]	long	all	Seconds since ascending node, begin of visibility segment i bgn_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
bgn_microsec [max_segments]	long	all	Micro seconds within second begin of visibility segment i bgn_microsec[i-1], i = 1, number_segments	μs	0 =< =< 999999
end_orbit [max_segments]	long	all	Orbit number, end of visibility segment i end_orbit[i-1], i = 1, number_segments		> 0
end_second [max_segments]	long	all	Seconds since ascending node, end of visibility segment i end_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
end_microsec [max_segments]	long	all	Micro seconds within second end of visibility segment i end_microsec[i-1], i = 1, number_segments	μs	0 =< =< 999999

Table 7: Output parameters for pv_stavistime

c name	c type	Array Ele-ment	Description	Unit	Range
zdop_orbit [max_segments]	long	all	Orbit number, time of zero doppler (-1 if no zero doppler within corresponding visibility segment) zdop_orbit[i-1], i = 1, number_segments		> 0
zdop_second [max_segments]	long	all	Seconds since ascending node, time of zero doppler (-1 if no zero doppler within corresponding visibility segment) zdop_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
zdop_microsec [max_segments]	long	all	Micro seconds within second time of zero doppler (-1 if no zero doppler within corresponding visibility segment) zdop_microsec[i-1], i = 1, number_segments	µs	0 =< =< 999999
ierr[10]	long		Error status flags		

Table 7: Output parameters for pv_stavistime

7.2.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_stavistime** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg** (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_stavistime** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code** (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error wrong swath type selected.	Computation not performed	PV_CFI_STAVISTIME_SWATH_TYPE_ERR	0
ERR	Error in input parameter to stavistime.	Computation not performed	PV_CFI_STAVISTIME_INPUTS_CHECK_ERR	1
ERR	Error reading the Orbit event file.	Computation not performed	PV_CFI_STAVISTIME_OEF_READ_ERR	2
WARN	Warning: start orbit is outside range of OEF/OSF.	Computation performed with start orbit = start_oef. Message to inform the user	PV_CFI_STAVISTIME_FIRST_ORBIT_WARN	3
WARN	Warning: stop orbit is outside range of OEF/OSF.	Computation performed with stop orbit = stop_oef. Message to inform the user	PV_CFI_STAVISTIME_LAST_ORBIT_WARN	4
ERR	Actual stop orbit is earlier than actual start orbit.	Computation not performed	PV_CFI_STAVISTIME_WRONG_INTERVAL_ERR	5
ERR	Error obtaining orbital information in orbit info .	Computation not performed Message to inform the user	PV_CFI_STAVISTIME_ORBIT_INFO_ERR	6
WARN	Warning: there is an orbital change within the requested orbits .	Computation performed.	PV_CFI_STAVISTIME_ORBIT_CHANGE_WARN	7
ERR	Error reading the swath template file.	Computation not performed	PV_CFI_STAVISTIME_SWATH_READ_ERR	8
ERR	There is a potential memory overload.Try with a smaller orbital interval.	Computation not performed	PV_CFI_STAVISTIME_POTENTIAL_MEMORY_ERR	9
ERR	Orbital information does not coincide with reference swath.	Computation not performed	PV_CFI_STAVISTIME_INCONSISTENT_SWATH_ERR	10
ERR	Error read info the ground station's mask data file.	Computation not performed	PV_CFI_STAVISTIME_READ_STA_ERR	11

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error transforming the station's mask into an equivalent zone .	Computation not performed	PV_CFI_STAVISTIME_AZEL2LONLAT_ERR	12
ERR	Error calling ZONEVISTIME to calculate transitions.	Computation not performed	PV_CFI_STAVISTIME_ZONEVISTIME_CALL_ERR	13
ERR	Error refining intersection time.	Computation not performed	PV_CFI_STAVISTIME_CALL_STAVIS_ERR	14
WARN	Accuracy of 0.001 deg in elevation not reached in orbit n. Orbit too close to the mask limit.	Computation performed. It is advised not to use this particular segment. This is acceptable since these segments are only a few seconds long.	PV_CFI_STAVISTIME_CALL_STAVIS_WARN	15
ERR	Error calculating zero doppler interval.	Computation not performed	PV_CFI_STAVISTIME_ZERO_DOPPLER_ERR	16
WARN	Segment longer than half nodal period deleted.	Computation performed. Message to inform the user	PV_CFI_STAVISTIME_LONG_SEGM_SKIPPED_WARN	17
ERR	Error checking orbit range cycle	Computation not performed. The input orbit range is incompatible with STF cycle	PV_CFI_STAVISTIME_CALL_ORBIT_RANGE_CYCLE_ERR	18
ERR	The requested orbit range is not contained in swath cycle	Computation not performed The input orbit range is not compatible with STF cycle	PV_CFI_STAVISTIME_ORBIT_RANGE_CYCLE_ERR	19
WARN	The orbit %li is not contained in swath cycle	Message to inform the user. The orbits not compatible with swath cycle are not taken into account in computation	PV_CFI_STAVISTIME_ORBIT_RANGE_CYCLE_WARN	20
ERR	Error calculating start/stop orbits	Computation not performed. Error trying to compute start/stop orbits when 0 is entered as start_orbit or stop_orbit.	PV_CFI_STAVISTIME_ORB_DEFAULT_VALUES_ERR	21

Note that some error numbers have changed since previous version. Error names have not changed except for PV_CFI_STAVISTIME_ORBIT_CHANGE_WARN.

7.2.6 Runtime performances

The following runtime performances have been measured:

orbits	Ultra Sparc [sec]
501	8.96

The above time depends on:

- the number of orbits
- the latitude and visibility size of the ground station

7.3 pv_drsvistime

7.3.1 Overview

The **pv_drsvistime** function computes all the orbital segments for which the satellite is visible from a data relay satellite located in a geostationary orbit.

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as seconds elapsed since the ascending node crossing.

pv_drsvistime requires access to one file to produce its results:

- the Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (**pg_genoef** function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.

The time intervals used by **pv_drsvistime** are expressed in absolute orbit numbers. This is valid for both:

- input parameter “Orbit Range”: first and last absolute orbit to be considered
- output parameter “Data Relay Satellite Visibility Segments”: time segments with time expressed as {absolute orbit number, number of seconds since ANX, number of microseconds}

Users who need to use UTC times must make use of the conversion routines provided in PPF_VISIBILITY (**pv_utcanx** and **pv_anxutc** functions).

The pv_drsvistime CFI function has been adapted to take into account the increasing inclination angle of the DRS (Artemis) orbit. Previously pv_drsvistime assumed that the DRS was located at the nominal zero inclination.

As the actual position is unknown during planning, the visibility constraints are checked for different latitudes in the range [-maximum inclination, +maximum inclination]. The estimation of the maximum inclination at epoch is based on a linear equation derived from the observed evolution of the inclination angle.

The resulting set of visibility segments, calculated for different latitudes, is then processed in order to determine the time windows in which there is visibility from the satellite to the DRS. Proceeding in this way, it is ensured that the visibility constraints will not be violated for any latitude in the range [-inclination, +inclination].

As a consequence of this approach, the duration of the orbital segments is reduced with respect the zero-latitude implementation.

Note that the DRS orbit inclination value corresponding to the time of the last relative orbit in the satellite cycle length is applied to all the orbits in the cycle length.

The **pv_drsvistime** function considers the following sources of occultation:

- Earth plus 20 km of atmosphere
- Fixed appendages: 1 deg half cone around:
  - Service Module
  - Payload Module
  - Module Interface
  - ASAR antenna

- AATSR Payload
- ATSR Radiator
- Mipas Payload
- Mipas Electronics
- Sciamachy Radiators A, B and C
- UMI
- Star Trackers, enlarged to have a 16 deg halfcone to protect against radiation.
- S Band Antennas

- Rotating appendices (solar array and its structure): 1 deg half cone around solar array and supporting structure
- Azimuth Blockage (165 deg to 195 deg, MCD convenction for the azimuth and elevation angles)
- Elevation Blockage (-86 deg to -90 deg, MCD convenction for the azimuth and elevation angles)

Operations of the antenna are also limited to the values (APM definition):

- Elevation from -30.0 deg to +90.0 deg
- Azimuth from -165.0 deg to +165.0 deg

These operations limitations are imposed considering margins of 1.0 deg.

In addition to these occultation sources, the function **pv_drsvistime** checks that the initial movement of the antenna (start-up trajectory) does not violate any mechanical constraints in order to reach the corresponding pointing to mthe DRS at the beginning time of the visibility segment. Similar computations are performed to be able to stop the antenna at the end point of the visibility segment.

In case the mechanical constraints are violated for a visibility segment, it is reduced by 1 second and the condition is checked again. The process is repeated until both trajectories are within the limits. A warning message is raised if the visibility segment duration comes to be smaller than the minimum duration defined by the user (*min_duration*).

The considerations assumed in the implementation of the start-up and stop trajectories are the following:

Concept	Start-up Trajectory	Stop Trajectory
Angular movements	Common time for azimuth and elevation movement	No common time for azimuth and elevation movement
Azimuth acceleration	AZ_{acc} = 0.015 deg/sec^2	Low Velocity: AZ_{acc} = 0.023 deg/sec^2
		High Velocity: AZ_{acc} = 0.043 deg/sec^2

Table 8: Assumptions for the start-up and stop trajectory computations

Concept	Start-up Trajectory	Stop Trajectory
Elevation acceleration	EL_{acc} = 0.004 deg/sec^2	Low Velocity: EL_{acc} = 0.02 deg/ sec^2
		High Velocity: EL_{acc} = 0.02 deg/ sec^2
Velocity limit	N/A	vel_{limit} = 0.11459 deg/sec

Table 8: Assumptions for the start-up and stop trajectory computations

7.3.2 Calling sequence pv_drsvistime

For C programs, the call to **pv_drsvistime** is (input parameters are underlined):

```
#include"ppf_visibility.h"
#define MAX_SEGMENTS <your value here>
{
       long          start_orbit, stop_orbit,
                     max_segments, number_segments,
                     bgn_orbit[MAX_SEGMENTS],
                     bgn_second[MAX_SEGMENTS],
                     bgn_microsec[MAX_SEGMENTS],
                     end_orbit[MAX_SEGMENTS],
                     end_second[MAX_SEGMENTS],
                     end_microsec[MAX_SEGMENTS],
                     ierr[10], status;
       double        min_duration, longitude;
       char          *orbit_event_file;

       max_segments = MAX_SEGMENTS;

       status = pv_drsvistime (
                        orbit_event_file, &start_orbit, &stop_orbit,
                        &longitude,
                        &max_segments, &min_duration,
                        &number_segments,
                        bgn_orbit, bgn_second, bgn_microsec,
                        end_orbit, end_second, end_microsec,
                        ierr);

* test status */
}
```

For FORTRAN programs **pv_drsvistime** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
        INTEGER*4    START_ORBIT, STOP_ORBIT,
       &             MAX_SEGMENTS, NUMBER_SEGMENTS,
       &             BGN_ORBIT(MAX_SEGMENTS),
       &             BGN_SECOND(MAX_SEGMENTS),
       &             BGN_MICROSEC(MAX_SEGMENTS),
       &             END_ORBIT(MAX_SEGMENTS),
       &             END_SECOND(MAX_SEGMENTS),
       &             END_MICROSEC(MAX_SEGMENTS),
       &             IERR(10), STATUS
        REAL*8       LONGITUDE, MIN_DURATION
        CHARACTER*(*)    ORBIT_EVENT_FILE
```

```
#include"ppf_visibility.inc"

        STATUS = PV_DRSVISTIME (
                              ORBIT_EVENT_FILE, START_ORBIT, STOP_ORBIT,
       &                      LONGITUDE,
       &                      MAX_SEGMENTS, MIN_DURATION,
       &                      NUMBER_SEGMENTS,
       &                      BGN_ORBIT, BGN_SECOND, BGN_MICROSEC,
       &                      END_ORBIT, END_SECOND, END_MICROSEC,
       &                      IERR)


C test status
```

7.3.3 Input parameters pv_drsvistime

c name	c type	Array Element	Description	Units	Range
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
start_orbit	long		First orbit, segment filter. Segments will be filtered as from the beginning of first orbit (within orbit range from orbit_event_file) If set to zero then first orbit of orbit_event_file is selected.	absolute orbit number	(=0) or (>= start_oef and <= stop_oef)
stop_orbit	long		Last orbit, segment filter. Segments will be filtered until the end of last orbit (within orbit range from orbit_event_file) If set to zero then last orbit of orbit_event_file is selected.	absolute orbit number	= 0 or >= start_orbit <= stop_oef
longitude	double		longitude of data relay satellite		[0, 360]
max_segments	long		Size of the segment arrays. If more segments are detected. Only max_segments segments will be given on output.		>0
min_duration	double		Minimum duration for segments. Only segments with a duration longer than min_duration will be given on output.	s	>= 0.0

Table 9: Input parameters for pv_drsvistime

7.3.4 Output parameters pv_drsvistime

c name	c type	Array Ele-ment	Description	Unit	Range
pv_drsvistime	long		Function status flag, 0 = No error > 0 Warnings, results generated < 0 Error, no results generated		
number_segments	long		Number of visibility segments returned to the user (it can be the number of visibility segments found or max_segments, when the number of visibility segments found is greater than max_segments)		>= 0
bgn_orbit [max_segments]	long	all	Orbit number, begin of visibility segment i bgn_orbit[i-1], i = 1, number_segments		> 0
bgn_second [max_segments]	long	all	Seconds since ascending node, begin of visibility segment i bgn_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
bgn_microsec [max_segments]	long	all	Micro seconds within second begin of visibility segment i bgn_microsec[i-1], i = 1, number_segments	µs	0 =< =< 999999
end_orbit [max_segments]	long	all	Orbit number, end of visibility segment i end_orbit[i-1], i = 1, number_segments		> 0
end_second [max_segments]	long	all	Seconds since ascending node, end of visibility segment i end_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
end_microsec [max_segments]	long	all	Micro seconds within second end of visibility segment i end_microsec[i-1], i = 1, number_segments	µs	0 =< =< 999999
ierr[10]	long		Error status flags		

Table 10: Output parameters for pv_drsvistime

7.3.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_drsvistime** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg** (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_drsvistime** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code** (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error in state vector computation. Orbit no: (%ld). [PO]	Computation not performed	PV_CFI_DRSVISTIME_PO_PPF_PREDICT_ERR	0
ERR	Error in rectifying Earth rotation. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PG_EF_TO_QEF_ERR	1
ERR	Error in coordinates transformation. Orbit no: (%ld). [PL]	Computation not performed	PV_CFI_DRSVISTIME_PL_CHANGE_CS_ERR	2
ERR	Error in direction computation. Orbit no: (%ld). [PL]	Computation not performed	PV_CFI_DRSVISTIME_PL_PT_DIR_RANGE_ERR	3
ERR	Error in azimuth-elevation computation. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_AZIM_ELEV_ERR	4
ERR	Error in physical mask checking. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_FIXED_CHECK_ERR	5
ERR	Error in Earth occultation checking. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_EARTH_CHECK_ERR	6
ERR	Error in solar panel position computation. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_ROTATING_POS_ERR	7
ERR	Error in solar panel occultation checking. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_ROTATING_SOLAR_PANEL_CHECK_ERR	8
ERR	Error in solar panel structure occultation checking. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_ROTATING_SOLAR_PANEL_STR_CHECK_ERR	9
ERR	Error in OSF/OEF reading.	Computation not performed	PV_CFI_DRSVISTIME_PV_OSF_RECORDS_READ_ERR	10
ERR	Error in input parameters.	Computation not performed	PV_CFI_DRSVISTIME_PV_DRSINPUTS_CHECK_ERR	11

Error type	Error message	Cause and impact	Error Code	Error No
WARN	Warning in input parameters.	Computation performed Message to inform the user	PV_CFI_DRSVISTIME_PV_DRSINPUTS_CHECK_WARN	12
ERR	Error in canonical position computation. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_CANON_POS_ERR	13
ERR	Error in orbit parameters computation. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_ORBIT_INFO_ERR	14
ERR	Error in ascending node parameters computation. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PG_GENSTATE_ERR	15
ERR	Maximum number of iterations. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_MAX_NUMBER_ITER_ERR	16
ERR	Error in time computations. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_TIME_SEC_ERR	17
ERR	Maximum number of segments exceeded. Execution interrupted. Values already loaded are correct. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_MAX_NUMBER_SEGMENTS_ERR	18
WARN	First orbit starts with visibility.	Computation performed Message to inform the user	PV_CFI_DRSVISTIME_FIRST_ORBIT_VIS_WARN	19
WARN	Last orbit ends with visibility.	Computation performed Message to inform the user	PV_CFI_DRSVISTIME_LAST_ORBIT_VIS_WARN	20
ERR	Error in antenna stop trajectory computations. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_CHECK_STOP_TRAJECTORY_ERR	21
WARN	No possible stop trajectory. Orbit no: (%ld)	Computation performed Message to inform the user	PV_CFI_DRSVISTIME_PV_CHECK_STOP_TRAJECTORY_WARN	22
ERR	Error in antenna start-up trajectory computations. Orbit no: (%ld).	Computation not performed	PV_CFI_DRSVISTIME_PV_CHECK_STARTUP_TRAJECTORY_ERR	23
WARN	No possible start-up trajectory. Orbit no: (%ld)	Computation performed Message to inform the user	PV_CFI_DRSVISTIME_PV_CHECK_STARTUP_TRAJECTORY_WARN	24
ERR	Memory allocation error	Computation not performed	PV_CFI_DRSVISTIME_MEMORY_ERR	25
ERR	Error getting the ANX Time at the reference orbit	Computation not performed	PV_CFI_DRSVISTIME_ANXUTC_ERR	26

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error getting visibility segments for the DRS at latitude: %f	Computation not performed	PV_CFI_DRSVISTIME_DRS_LAT_ERR	27

7.3.6 Runtime performances

The following runtime performances have been measured:

orbits	Ultra Sparc [sec]
501	187

The above time depends strongly on the number of orbits to be solved.

7.4 pv_swathcalc

7.4.1 Overview

The **pv_swathcalc** function computes the location of a swath at a given time.
Swath location is expressed as[2]:

- longitude
- latitude
- altitude

for up to 3 points, defined as follows with respect to satellite flight direction (see Figure 2):

- left-most point of the swath
- middle point of the swath
- right-most point of the swath

pv_swathcalc requires access to several files to produce its results:

- the Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (**pg_genoef** function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.
- the Instrument Swath File, describing the area seen by the relevant instrument all along the current orbit. It is produced off-line by the PPF_GENREF CFI software (**pg_genswath** function)

The input time used by **pv_swathcalc** is expressed in orbit-relative time.

Users who need to use UTC time must make use of the conversion routine provided in PPF_VISIBILITY (**pv_anxutc** functions).

NOTE: Since the swath template file is generated from a reference orbit, it is not allowed to use **pv_swathcalc** for an orbit in the orbit event file with different repeat cycle or cycle length. If this would happen, **pv_swathcalc** will return an error an no computation will be performed. For version 2 of Orbit Event/Orbit Scenario and Swath Template files, taken into account previous restriction, if the required orbit does not belong to the orbital change of Swath Template file reference orbit, a warning is raised.

2. For inertial swaths, right ascension and declination are used instead of longitude and latitude

7.4.2 Calling sequence pv_swathcalc

For C programs, the call to **pv_swathcalc** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{
       long          orbit, second, microsec,
                     ierr[10], status;
       double        longitude[3], latitude[3], altitude[3];
       char          *orbit_event_file, *swath_file;

       status = pv_swathcalc (
                         orbit_event_file, swath_file,
                         &orbit, &second, &microsec,
                         longitude, latitude, altitude,
                         ierr);

* test status */
}
```

For FORTRAN programs **pv_swathcalc** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
        INTEGER*4    ORBIT, SECOND, MICROSEC,
       &             IERR(10), STATUS
        REAL*8       LONGITUDE(3), LATITUDE(3), ALTITUDE(3)
        CHARACTER*(*)    ORBIT_EVENT_FILE, SWATH_FILE

#include"ppf_visibility.inc"

        STATUS = PV_SWATHCALC (
                            ORBIT_EVENT_FILE, SWATH_FILE,
       &                    ORBIT, SECOND, MICROSEC,
       &                    LONGITUDE, LATITUDE, ALTITUDE,
       &                    IERR)


C test status
```

7.4.3 Input parameters pv_swathcalc

c name	c type	Array Ele-ment	Description	Units	Range
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
swath_file	char *		File name of the swath-file for the appropriate instrument mode If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
orbit	long		Orbit number		> 0
second	long		Seconds since ascending node	s	>= 0 < orbital period
microsec	long		Micro seconds within second	μs	0 =< =< 999999

Table 11: Input parameters for pv_swathcalc

7.4.4 Output parameters pv_swathcalc

c name	c type	Array Ele-ment	Description	Unit	Range
pv_swathcalc	long		Function status flag, 0 = No error > 0 Warnings, results generated < 0 Error, no results generated		
longitude[3]	double	all	longitude (right ascension for inertial swaths) of point i: i = 0, left swath point i = 1, mid swath point i = 2, right swath point In case of point swath, only longitude[0] is useful; longitude[1] and longitude[2] are dummy	deg	[-180. 180]
latitude[3]	double	all	latitude (declination for inertial swaths) of point i: i = 0, left swath point i = 1, mid swath point i = 2, right swath point In case of point swath, only latitude[0] is useful; latitude[1] and latitude[2] are dummy	deg	[-90, 90]
altitude[3]	double	all	altitude of point i: i = 0, left swath point i = 1, mid swath point i = 2, right swath point In case of point swath, only altitude[0] is useful; altitude[1] and altitude[2] are dummy	m	
ierr[10]	long		Error status flags		

Table 12: Output parameters for pv_swathcalc

7.4.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_swathcalc** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg** (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_swathcalc** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code** (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Orbit number must be positive	Computation not performed	PV_CFI_SWATHCALC_ORB_NUM_LIM_ERR	0
ERR	Seconds since ascending node must be zero or positive.	Computation not performed	PV_CFI_SWATHCALC_SEC_LIM_ERR	1
ERR	MicroSeconds must be zero or positive.	Computation not performed	PV_CFI_SWATHCALC_MICROSEC_1ST_ERR	2
ERR	MicroSeconds can not be bigger than 999999.	Computation not performed	PV_CFI_SWATHCALC_MICROSEC_2ND_ERR	3
ERR	Orbit number is not included in the Orbit Event File.	Computation not performed	PV_CFI_SWATHCALC_ORB_NUM_OEF_ERR	4
ERR	Seconds since ascending node must be less than orbital period.	Computation not performed	PV_CFI_SWATHCALC_SEC_ORB_PER_ERR	5
ERR	Input time greater than orbital period.	Computation not performed	PV_CFI_SWATHCALC_TIME_ERR	6
ERR	Repeat Days Cycle of this orbit is not the same than the swath template.	Computation not performed	PV_CFI_SWATHCALC_REP_CYCLE_ERR	7
ERR	Orbits Cycle Length of this orbit is not the same than the swath template.	Computation not performed	PV_CFI_SWATHCALC_CYCLE_LENGTH_ERR	8
ERR	No spherical triangle.	Computation not performed	PV_CFI_SWATHCALC_SPHER_TRIANG_ERR	9
WARN	The requested orbit does not belong to Swath cycle	Information message to the user. The orbit does not corresponds to swath cycle.	PV_CFI_SWATHCALC_STF_CYC_WARN	10
ERR	Error while reading OSF information.	Computation not performed	PV_CFI_SWATHCALC_PV_OSF_RECORDS_READ_ERR	32
ERR	Error while computing information of the orbit.	Computation not performed	PV_CFI_SWATHCALC_PV_ORBIT_INFO_ERR	33

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error while reading SWATH FILE.	Computation not performed	PV_CFI_SWATHCALC_PV_SWATH_READ_ERR	34
ERR	Error while checking orbit orbital change	Computation not performed	PV_CFI_SWATHCALC_CALL_ORBIT_RANGE_CYCLE_ERR	35

7.4.6 Runtime performances

The following runtime performances have been measured (OEF previously loaded):

Calls	Ultra Sparc [sec]
100	0.01

7.5 pv_anxutc

7.5.1 Overview

The **pv_anxutc** function converts an orbit-relative time into a UTC time.

pv_anxutc requires access to one file to produce its results:

- the Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (**pg_genoef** function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.

7.5.2 Calling sequence pv_anxutc

For C programs, the call to **pv_anxutc** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{
       long          orbit, second, microsec,
                     ierr[20], status;
       double        utc;
       char          *orbit_event_file;

       status = pv_anxutc (
                          orbit_event_file,
                          &orbit, &second, &microsec,
                          &utc,
                          ierr);

* test status */
}
```

For FORTRAN programs **pv_anxutc** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
        INTEGER*4    ORBIT, SECOND, MICROSEC,
       &             IERR(20), STATUS
        REAL*8       UTC
        CHARACTER*(*)    ORBIT_EVENT_FILE

#include"ppf_visibility.inc"

        STATUS = PV_ANXUTC (
                             ORBIT_EVENT_FILE, SWATH_FILE,
       &                     ORBIT, SECOND, MICROSEC,
       &                     UTC,
       &                     IERR)
```

```
C test status
```

7.5.3 Input parameters pv_anxutc

c name	c type	Array Element	Description	Units	Range
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
orbit	long		Orbit number		> 0
second	long		Seconds since ascending node	s	>= 0 < orbital period
microsec	long		Micro seconds within second	μs	0 =< =< 999999

Table 13: Input parameters for pv_anxutc

7.5.4 Output parameters pv_anxutc

c name	c type	Array Element	Description	Unit	Range
pv_anxutc	long		Function status flag, 0 = No error > 0 Warnings, results generated < 0 Error, no results generated		
utc	double		resulting UTC time	mjd 2000	
ierr[10]	long		Error status flags		

Table 14: Output parameters for pv_anxutc

7.5.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_anxutc** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg** (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_anxutc** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code** (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error in input parameter "orb_num".	Computation not performed	PV_CFI_ANXUTC_ORB_NUM_1ST_ERR	0
ERR	Error in input parameter "orb_num". orb_num<ABS_START_ORBIT.	Computation not performed	PV_CFI_ANXUTC_ORB_NUM_2ND_ERR	1
ERR	Error in input parameter "orb_num". orb_num>ABS_STOP_ORBIT.	Computation not performed	PV_CFI_ANXUTC_ORB_NUM_3RD_ERR	2
ERR	Error in input parameter "seconds".	Computation not performed	PV_CFI_ANXUTC_SEC_ERR	3
ERR	Error in input parameter "microsec".	Computation not performed	PV_CFI_ANXUTC_MICROSEC_ERR	4
ERR	Error in input parameter "seconds" plus "microsec" is greater than orbital period.	Computation not performed	PV_CFI_ANXUTC_SEC_MICROSEC_ERR	5
ERR	Error while reading OSF information.	Computation not performed	PV_CFI_ANXUTC_PV_OSF_RECORDS_READ_ERR	32
ERR	Error while computing information of the orbit.	Computation not performed	PV_CFI_ANXUTC_PV_ORBIT_INFO_ERR	33

7.5.6 Runtime performances

The following runtime performances have been measured:

Calls	Ultra Sparc [sec]
1000	2.1

7.6 pv_utcanx

7.6.1 Overview

The **pv_utcanx** function converts a UTC time into an orbit-relative time.

pv_utcanx requires access to one file to produce its results:

- the Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (**pg_genoef** function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.

7.6.2 Calling sequence pv_utcanx

For C programs, the call to **pv_utcanx** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{
       long          orbit, second, microsec,
                     ierr[10], status;
       double        utc;
       char          *orbit_event_file;

       status = pv_utcanx (
                          orbit_event_file,
                          &orbit, &second, &microsec,
                          &utc,
                          ierr);

* test status */
}
```

For FORTRAN programs **pv_utcanx** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
        INTEGER*4    ORBIT, SECOND, MICROSEC,
       &             IERR(10), STATUS
        REAL*8       UTC
        CHARACTER*(*)    ORBIT_EVENT_FILE

#include"ppf_visibility.inc"

        STATUS = PV_UTCANX (
                             ORBIT_EVENT_FILE, SWATH_FILE,
       &                     ORBIT, SECOND, MICROSEC,
       &                     UTC,
       &                     IERR)
```

```
C test status
```

7.6.3 Input parameters pv_utcanx

c name	c type	Array Ele-ment	Description	Units	Range
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves compu-tation time)		
utc	double		UTC time	mjd 2000	

Table 15: Input parameters for pv_utcanx

7.6.4 Output parameters pv_utcanx

c name	c type	Array Ele-ment	Description	Unit	Range
pv_anxutc	long		Function status flag, 0 = No error > 0 Warnings, results gener-ated < 0 Error, no results generated		
orbit	long		resulting orbit number		> 0
second	long		resulting seconds since ascending node	s	>= 0 < orbital period
microsec	long		resulting micro seconds within second	μs	0 =< =< 999999
ierr[10]	long		Error status flags		

Table 16: Output parameters for pv_utcanx

7.6.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_utcanx** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg** (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_utcanx** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code** (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error in input parameter "utc".	Computation not performed	PV_CFI_UTCANX_UTC_ERR	0
ERR	UTC is out of OEF limits (it comes before 1st orbit).	Computation not performed	PV_CFI_UTCANX_UTC_OEF_1ST_ORB_ERR	1
ERR	UTC is out of OEF limits (it comes after last orbit).	Computation not performed	PV_CFI_UTCANX_UTC_OEF_LAST_ORB_ERR	2
ERR	Error while reading OSF information.	Computation not performed	PV_CFI_UTCANX_PV_OSF_RECORDS_READ_ERR	32
ERR	Error while computing information of the orbit.	Computation not performed	PV_CFI_UTCANX_PV_ORBIT_INFO_ERR	33

7.6.6 Runtime performances

The following runtime performances have been measured:

Calls	Ultra Sparc [sec]
1000	2.2

7.7 pv_orbitinfo

7.7.1 Overview

The **pv_orbitinfo** function retrieves from an Orbit Event File or an Orbit Scenario File, the orbit information related with a certain orbit (specified by means of absolute orbit number, or relative orbit number and cycle number) or related with the first orbit of a phase, where the phase number is the input parameter.

pv_orbitinfo requires to access to one file to produce its results:

- the Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (pg_genoef function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.

7.7.2 Calling sequence pv_orbitinfo

For C programs, the call to **pv_orbitinfo** is (input parameters are underlined, some may be input or output depending on the calling mode):

```
#include "ppf_visibility.h"
{
       long             mode, abs_orbit, rel_orbit, cycle, phase;
       long             repeat_cycle, cycle_length;
       long             ierr[10], status;
       char             *orbit_event_file;
       double           phasing, mlst, mjd_anx[2], pos_anx[3],
vel_anx[3], xm_anx[6];

       status = pv_orbitinfo (&mode, orbit_event_file,&abs_orbit,
                        &rel_orbit, &cycle, &phase,
                        &repeat_cycle, &cycle_length, &phasing, &mlst,
                        mjd_anx, pos_anx, vel_anx, xm_anx, ierr);
/* test status */
}
```

For FORTRAN programs **pv_orbitinfo** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
#include"ppf_visibility.inc"
        INTEGER*4    MODE, ABS_ORBIT, REL_ORBIT, CYCLE, PHASE
        INTEGER*4    REPEAT_CYCLE, CYCLE_LENGT
        INTEGER*4    IERR(10), STATUS
        REAL*8       PHASING, MLST, MJD_ANX(2), POS_ANX(3)
        REAL*8       VEL_ANX(3), XM_ANX(6)
        CHARACTER*(*)    ORBIT_EVENT_FILE

       STATUS = PV_ORBITINFO (MODE, ORBIT_EVENT_FILE,ABS_ORBIT,
       &                    REL_ORBIT, CYCLE, PHASE, REPEAT_CYCLE,
       &                    CYCLE_LENGTH,PHASING, MLST, MJD_ANX,
       &                    POS_ANX, VEL_ANX, XM_ANX, IERR)

C test status
```

7.7.3 Input parameters pv_orbitinfo

c name	c type	Array Element	Description	Units	Range
mode	long *		Input / Output flag = pv_abs_infoinput: absolute orbit = pv_rel_infoinput: relative orbit, cycle = pv_phase_infoinput: phase, output given for 1st orbit of phase = + pv_osv_infooutput: including Cartesian and Kepler state-vectors		

Depending on the mode flag, input parameters shall be:

mode = PV_ABS_INFO [+PV_OSV_INFO]

c name	c type	Array Element	Description	Units	Range
mode	long *		Input / Output flag		
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
abs_orbit	long *		Absolute orbit number, if mode = pv_abs_info [+pv_osv_info]		≥ start_oef ≤ stop_oef

mode = PV_REL_INFO [+PV_OSV_INFO]

c name	c type	Array Element	Description	Units	Range
mode	long *		Input / Output flag		
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string ("") the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
rel_orbit	long *		Relative orbit number, if mode = pv_rel_info [+pv_osv_info]		
cycle	long *		Cycle number, if mode = pv_rel_info [+pv_osv_info]		

mode = PV_PHASE_INFO [+PV_OSV_INFO]

c name	c type	Array Element	Description	Units	Range
mode	long *		Input / Output flag		
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string ("") the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		

phase	long *		Phase number, if mode = pv_phase_info [+pv_osv_info]		

7.7.4 Output parameters pv_orbitinfo

Denpeding on the mode flag, output parameters, shall be:

mode = PV_ABS_INFO [+PV_OSV_INFO]

c name	c type	Array Ele-ment	Description	Unit	Range
pv_orbitinfo	long		Function status flag, = 0 No error ≥ +1 Warnings, results gener-ated ≤ -1 Error, no results generated		
rel_orbit	long *		Relative orbit number		
cycle	long *		Cycle number		
phase	long *		Phase number		
repeat_cycle	long *		Repeat cycle of the requested orbit, after repeat_cycle days the Earth-fixed track repeats	days	
cycle_length	long *		Cycle_length of the requested orbit, after cycle_length orbits the Earth-fixed track repeats	orbits	
phasing	double *		The longitude of ascending node of the requested orbit	deg	
mlst	double *		The mean local solar time of the requested orbit, atascending node	hour	
mjd_anx[2]	double	[0]	UTC at ascending node of requested orbit (UT1time) if mode ≥ pv_osv_info	decimal days (Process-ing format)	≥ -18262 < +36525
		[1]	ΔUT1 at ascending node of requested orbit (UT1time) if mode ≥ pv_osv_info	s (Process-ing format)	≥ -1.0 ≤ +1.0
pos_anx[3]	double	all	Osculating position vector at ascending node of requested orbit (Earth fixed CS) ifmode ≥ pv_osv_info	m	

c name	c type	Array Ele-ment	Description	Unit	Range
vel_anx[3]	double	all	Osculating velocity vector at ascending node of requested orbit (Earth fixed CS) if mode ≥ pv_osv_info	m/s	
xm_anx[6]	double	all	Mean kepler at ascending node of requested orbit (TOD) if mode ≥ pv_osv_info	m/s	
		[0]	a =Semi-major axis	m	
		[1]	e = Eccentricity	-	
		[2]	i =Inclination	deg	
		[3]	Ω = Right-ascension ascending node	deg	
		[4]	ω =Argument of perigee	deg	
		[5]	M = Mean anomaly	deg	
ierr[10]	long	all	Error status flags		

mode = PV_REL_INFO [+PV_OSV_INFO]

c name	c type	Array Ele-ment	Description	Unit	Range
pv_orbitinfo	long		Function status flag, = 0 No error ≥ +1 Warnings, results gener-ated ≤ -1 Error, no results generated		
abs_orbit	long *		Absolute orbit number		≥ start_oef ≤ stop_oef
phase	long *		Phase number		
repeat_cycle	long *		Repeat cycle of the requested orbit, after repeat_cycle days the Earth-fixed track repeats	days	
cycle_length	long *		Cycle_length of the requested orbit, after cycle_length orbits the Earth-fixed track repeats	orbits	
phasing	double *		The longitude of ascending node of the requested orbit	deg	

c name	c type	Array Ele-ment	Description	Unit	Range
mlst	double *		The mean local solar time of the requested orbit, atascending node	hour	
mjd_anx[2]	double	[0]	UTCat ascending node of requested orbit (UT1 time) if mode ≥ pv_osv_info	decimal days (Processing format)	≥ -18262 < +36525
		[1]	ΔUT1 at ascending node of requested orbit (UT1 time) if mode≥ pv_osv_info	s (Processing format)	≥ -1.0 ≤ +1.0
pos_anx[3]	double	all	Osculating position vector at ascending node of requested orbit (Earth fixed CS) if mode ≥ pv_osv_info	m	
vel_anx[3]	double	all	Osculating velocity vector at ascending node of requested orbit (Earth fixed CS) if mode ≥ pv_osv_info	m/s	
xm_anx[6]	double	all	Mean kepler at ascending node of requested orbit (TOD) if mode ≥ pv_osv_info	m/s	
		[0]	a = Semi-major axis	m	
		[1]	e = Eccentricity	-	
		[2]	i = Inclination	deg	
		[3]	Ω = Right-ascension ascending node	deg	
		[4]	ω = Argument of perigee	deg	
		[5]	M = Mean anomaly	deg	
ierr[10]	long	all	Error status flags		

mode = PV_PHASE_INFO [+PV_OSV_INFO]

c name	c type	Array Element	Description	Unit	Range
pv_orbitinfo	long		Function status flag, = 0 No error ≥ +1 Warnings, results generated ≤ -1 Error, no results generated		
abs_orbit	long *		Absolute orbit number		≥ start_oef ≤ stop_oef
rel_orbit	long *		Relative orbit number		
cycle	long *		Cycle number		
repeat_cycle	long *		Repeat cycle of the requested orbit, after repeat_cycle days the Earth-fixed track repeats	days	
cycle_length	long *		Cycle_length of the requested orbit, after cycle_length orbits the Earth-fixed track repeats	orbits	
phasing	double *		The longitude of ascending node of the requested orbit	deg	
mlst	double *		The mean local solar time of the requested orbit, at ascending node	hour	
mjd_anx[2]	double	[0]	UTC at ascending node of requested orbit (UT1 time) if mode ≥ pv_osv_info	decimal days (Processing format)	≥ -18262 < +36525
		[1]	ΔUT1 at ascending node of requested orbit (UT1 time) if mode ≥ pv_osv_info	s (Processing format)	≥ -1.0 ≤ +1.0
pos_anx[3]	double	all	Osculating position vector at ascending node of requested orbit (Earth fixed CS) if mode ≥ pv_osv_info	m	

c name	c type	Array Ele-ment	Description	Unit	Range
vel_anx[3]	double	all	Osculating velocity vector at ascending node of requested orbit (Earth fixed CS) if mode ≥ pv_osv_info	m/s	
xm_anx[6]	double	all	Mean kepler at ascending node of requested orbit (TOD) if mode ≥ pv_osv_info	m/s	
		[0]	a =Semi-major axis	m	
		[1]	e = Eccentricity	-	
		[2]	i =Inclination	deg	
		[3]	Ω = Right-ascension ascending node	deg	
		[4]	ω = Argument of perigee	deg	
		[5]	M= Mean anomaly	deg	
ierr[10]	long	all	Error status flags		

7.7.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_orbitinfo** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library pv_vector_msg (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the pv_utcanx CFI function by calling the function of the PPF_VISIBILITY software library pv_vector_code (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Cannot read OEF/OSF file. Reading function returned error.	Computation not performed	PV_CFI_ORBITINFO_OEF_READ_ERR	0
ERR	Input parameters are wrong.	Computation not performed	PV_CFI_ORBITINFO_INPUT_PARAMETER_ERR	32
ERR	Cannot execute Genstate. Function returned error.	Computation not performed	PV_CFI_ORBITINFO_GENSTATE	64

7.7.6 Runtime performances

The following runtime performances have been measured:

Calls	Ultra Sparc [sec]
100	1.7

7.8 pv_starvistime

7.8.1 Overview

The **pv_starvistime** function computes stars visibility segments, the orbital segments for which a given star is visible with a given instrument from the satellite.

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as seconds elapsed since the ascending node crossing.

In addition, **pv_starvistime** calculates for every start and end of the visibility segment a coverage flag, determining which side of the FOV the event took place.

pv_starvistime requires access to several files to produce its results:

- The Reference Orbit Event File, describing all major events occurring during each orbit of the corresponding scenario. It is produced off-line by the PPF_GENREF CFI software (**pg_genoef** function). The Reference Orbit Event File can be replaced by the Orbit Scenario File that was used to generate it.
- Two Inertial Reference Swath Template Files.
- (*Optional*) The Star's Database File, describing the location in right ascension and declination of a star, described by its corresponding identifier.

The time intervals used by **pv_starvistime** are expressed in absolute orbit numbers. This is valid for both:

- input parameter "Orbit Range": first and last absolute orbit to be considered
- output parameter "Star Visibility Segments": time segments with time expressed as {absolute orbit number, number of seconds since ANX, number of microseconds}

Users who need to use UTC times must make use of the conversion routines provided in PPF_VISIBILITY (**pv_utcanx** and **pv_anxutc** functions).

7.8.2 Swath Definition

pv_starvistime calculates stars visibility segments for FOV corresponding to limb-sounding instruments observing inertial objects. The corresponding template files are generated off-line by the PPF GENREF CFI software (**pg_genswath** function), such as those shown in the table below:

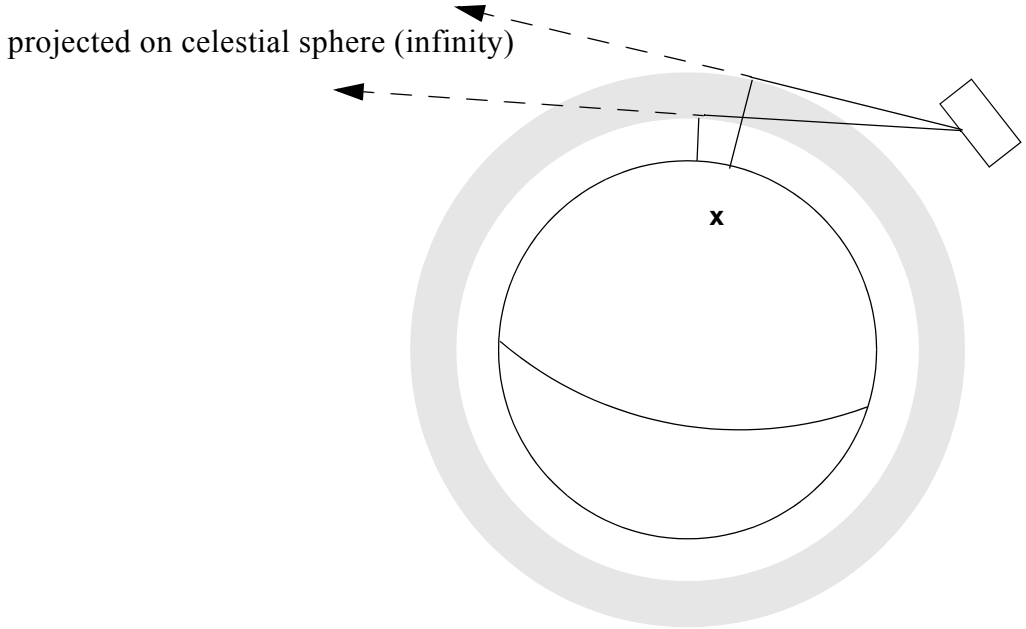
Instrument	Mode	File Prefix = swath	pg_genswath algorithm	Swath Type	Remarks
GOMOS	Occultation	GOMOIL GOMOIH	INERTIAL	Inertial direction	IFOV much smaller than swath. IFOV Very dependent on star availability. 2 swaths defined: - 1 for high altitude (GOMOIH) - 1 for low altitude (GOMOIL)
MIPAS	Rearward Sideward	MIPIRH MIPIRL MIPIXH MIPIXL	INERTIAL	Inertial direction	2 swaths defined for rearward mode: - 1 for high altitude (MIPIRH) - 1 for low altitude (MIPIRL) 3 swaths defined for sideward mode: - 1 for high altitude (MIPIXH) - 1 for back mode (MIPIXB) - 1 for forward mode (MIPIXF)

7.8.2.1 Inertial Swaths

The FOV for a Limb-sounding instrument observing inertial objects is calculated using two main parameters.

- The FOV projection on the celestial sphere is determined by two set of swaths, one corresponding to a higer (TOP) and a lower (BOTTOM) altitude over the ellipsoid, hence defining the elevation range of the FOV.

Figure 10 Two tangent altitudes over the ellipsoid



- The azimuth range is defined as such, the extremes corresponding to the left and right sides. In addition genswath generates coordinates for a middle point.

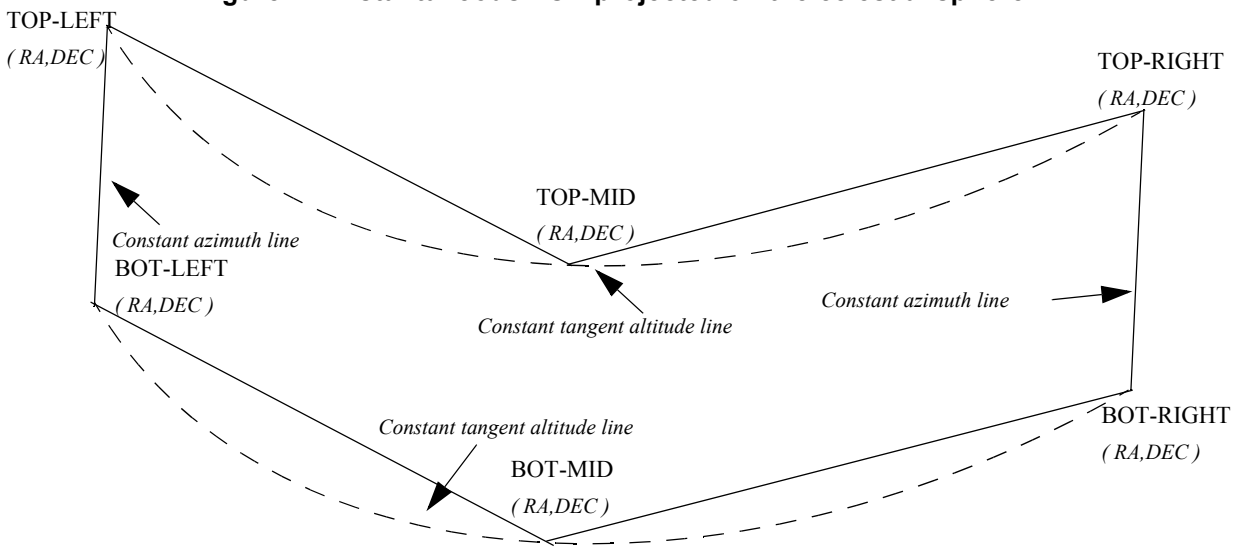
The instantaneous FOV projected on the celestial sphere can be represented as a series of points defined by their Right Ascension and Declination coordinates .

The top and bottom lines sweep the azimuth range at a constatnt tanget altitude, whilst the left and right side have a constant azimuth value with changing tangent altitude.

The shape of FOV should be similar to that shown in the diagram below with the dotted lines, whilst the algorithm implemented in pv_starvistime uses a simplified model joining the points with straight line.

As the satellite evolves around the orbit and the FOV sweeps the celestial sphere, a star can enter the FOV. **pv_starvistime** calculates that time and returns a flag indicating which part of the FOV (*LEFT*, *TOP*, *RIGHT* or *BOTTOM*) first detected the star. The same is done when the star exits the FOV.

Figure 11 Instantaneous FOV projected on the celestial sphere



7.8.2.2 Usage Hints

7.8.2.3 Splitting swaths

As it was shown in *figure 11*, the accuracy and precission of pv_starvistime strongly depends on how close the projection used in the algorithm is to the real world. Higher accuracy can be obtained splitting the azimuth range in sub-swaths.

Furthermore, splitting the swath would be necessary if the FOV was to cover an azimuth range larger than 180 degrees.

Note: It is important to note that if the FOV covers the value of 90 or 270 degrees in azimuth, one of the extremes (*LEFT* or *RIGHT*) of the STF must correspond to that azimuth value.

7.8.2.4 Orbital Changes

Since the reference swath template file is generated from a reference orbit, it is not recommended to use **pv_starvistime** for a range of orbits that includes an orbital change (e.g. change in the repeat cycle or cycle length). If this would happen, **pv_starvistime** will automatically ignore those orbits from the orbital change onwards, i.e. the actual stop orbit shall be the previous one to the first change in repeat cycle or cycle length.

For version 2 of Orbit Event/Orbit Scenario and Swath Template files, only the visibility segments of orbits corresponding to the orbital change of the Swath Template file reference orbit are returned.

7.8.3 Calling sequence pv_starvistime

For C programs, the call to **pv_starvistime** is (input parameters are underlined):

```
#include"ppf_visibility.h"
#define MAX_SEGMENTS <your value here>
{
       long          start_orbit, stop_orbit,
                     max_segments, number_segments,
                     bgn_orbit[MAX_SEGMENTS],
                     bgn_second[MAX_SEGMENTS],
                     bgn_microsec[MAX_SEGMENTS],
                     bgn_coverage[MAX_SEGMENTS],
                     end_orbit[MAX_SEGMENTS],
                     end_second[MAX_SEGMENTS],
                     end_microsec[MAX_SEGMENTS],
                     end_coverage[MAX_SEGMENTS],
                     ierr[10], status;
       double        star_ra, star_dec, star_ra_deg, star_dec_deg,
                     min_duration;
       char          *orbit_event_file,
                     *swath_file_upper, *swath_file_lower;
       char          star_id[8],*star_db_file;

       max_segments = MAX_SEGMENTS;
       status = pv_starvistime (
                           orbit_event_file, &start_orbit, &stop_orbit,
                           swath_file_upper, swath_file_lower, star_id,
                           star_db_file, &star_ra, &star_dec,
                           &max_segments, &min_duration,
                           &star_ra_deg, &star_dec_deg, &number_segments,
                           bgn_orbit, bgn_second, bgn_microsec,
bgn_coverage,
                           end_orbit, end_second, end_microsec,
end_coverage,
                           ierr);

/* test status */
}
```

For FORTRAN programs **pv_starvistime** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
        INTEGER*4    START_ORBIT, STOP_ORBIT,
       &             MAX_SEGMENTS, NUMBER_SEGMENTS,
       &             BGN_ORBIT(MAX_SEGMENTS),
       &             BGN_SECOND(MAX_SEGMENTS),
       &             BGN_MICROSEC(MAX_SEGMENTS),
```

```
      &            BGN_COVERAGE(MAX_SEGMENTS),
      &            END_ORBIT(MAX_SEGMENTS),
      &            END_SECOND(MAX_SEGMENTS),
      &            END_MICROSEC(MAX_SEGMENTS),
      &            END_COVERAGE(MAX_SEGMENTS),
      &            IERR(10), STATUS
       REAL*8      STAR_RA, STAR_DEC, STAR_RA_DEG, STAR_DEC_DEG
       CHARACTER*(*)    ORBIT_EVENT_FILE, STAR_DB_FILE
       CHARACTER*(*)    SWATH_FILE_UPPER, SWATH_FILE_LOWER
       CHARACTER*8      STAR_ID

#include"ppf_visibility.inc"

       STATUS = PV_STARVISTIME (
                   ORBIT_EVENT_FILE, START_ORBIT, STOP_ORBIT,
      &            SWATH_FILE_UPPER, SWATH_FILE_LOWER,
      &            STAR_ID, STAR_DB_FILE, STAR_RA, STAR_DEC,
      &            MAX_SEGMENTS, MIN_DURATION,
      &            STAR_RA_DEG, STAR_DEC_DEG, NUMBER_SEGMENTS,
      &            BGN_ORBIT, BGN_SECOND, BGN_MICROSEC, BGN_COVERAGE,
      &            END_ORBIT, END_SECOND, END_MICROSEC, END_COVERAGE,
      &            IERR)
C test status
```

7.8.4 Input parameters pv_starvistime

c name	c type	Array Ele-ment	Description	Units	Range
orbit_event_file	char *		It can be either a Reference Orbit Event File or an Orbit Scenario File. The orbit event file describes the Envisat-1 reference orbit. Defines start_oef and stop_oef, the first and last orbit of current file. The scenario file describes the orbital changes and the repeat cycle and cycle length. If empty string (“”) the file last read in a previous call to a PPF_VISIBILITY CFI function is used (saves computation time)		
start_orbit	long*		First orbit, segment filter. Segments will be filtered as from the beginning of first orbit (within orbit range from orbit_event_file) If set to zero then first orbit of stop_orbit orbital change is selected.	absolute orbit number	(=0) or (>= start_oef and <= stop_oef)
stop_orbit	long*		Last orbit, segment filter. Segments will be filtered until the end of last orbit (within orbit range from orbit_event_file) If set to zero then last orbit of start_orbit orbital change is selected.	absolute orbit number	= 0 or >= start_orbit <= stop_oef
swath_file_upper	char *		File name of the inertial swath-file for the appropiate instrument mode, which defines the upper limit of the FOV. This file is read each time the function is called		
swath_file_lower	char *		File name of the inertial swath-file for the appropiate instrument mode, which defines the lower limit of the FOV. This file is read each time the function is called		
star_id[8]	char		identification of the star, as defined in the star_db_file. This parameter is used **ONLY IF** star_db_file is not equal empty string(“”)		EXACTLY 8 characters
star_db_file	char *		File name of the star database file		

Table 17: Input parameters for pv_starvistime

c name	c type	Array Ele-ment	Description	Units	Range
star_ra	double*		Right Ascension of Star, in TOD. This parameter is used **ONLY IF** star_db_file is equal empty string ("")	deg	(-180.0, 180.0)
star_dec	double*		Declination of Star, in TOD. This parameter is used **ONLY IF** star_db_file is equal empty string ("")	deg	(-90.0, 90.0)
max_segments	long*		Size of the segment arrays. If more segments are detected. Only max_segments segments will be given on output.		>0
min_duration	double*		Minimum duration for segments. Only segments with a duration longer than min_duration will be given on output.	s	>= 0.0

Table 17: Input parameters for pv_starvistime

7.8.5 Output parameters pv_starvistime

c name	c type	Array Ele-ment	Description	Unit	Range
pv_starvistime	long		Function status flag, 0 = No error > 0 Warnings, results generated < 0 Error, no results generated		
star_ra_deg	double*		Right Ascension of the star, in TOD, for the UTC halfway start_orbit and stop_orbit.	deg	(-180.0, 180.0)
star_dec_deg	double*		Declination of the star, in TOD, for the UTC halfway start_orbit and stop_orbit.	deg	(-90.0, 90.0)
number_segment	long*		Number of visibility segments returned to the user (it can be the number of visibility segments found or max_segments, when the number of visibility segments found is greater than max_segments)		>= 0

Table 18: Output parameters for pv_starvistime

c name	c type	Array Ele-ment	Description	Unit	Range
bgn_orbit [max_segments]	long	all	Orbit number, begin of visibility segment i bgn_orbit[i-1], i = 1, number_segments		> 0
bgn_second [max_segments]	long	all	Seconds since ascending node, begin of visibility segment i bgn_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
bgn_microsec [max_segments]	long	all	Micro seconds within second begin of visibility segment i bgn_microsec[i-1], i = 1, number_segments	µs	0 =< =< 999999
bgn_coveragec [max_segments]	long	all	Coverage flag for swath entry: PV_STAR_UNDEFINED = 0, PV_STAR_UPPER = 1, PV_STAR_LOWER = 2, PV_START_LEFT = 3 , PV_STAR_RIGHT=4		0,1,2,3,4
end_orbit [max_segments]	long	all	Orbit number, end of visibility segment i end_orbit[i-1], i = 1, number_segments		> 0
end_second [max_segments]	long	all	Seconds since ascending node, end of visibility segment i end_second[i-1], i = 1, number_segments	s	>= 0 < orbital period
end_microsec [max_segments]	long	all	Micro seconds within second end of visibility segment i end_microsec[i-1], i = 1, number_segments	µs	0 =< =< 999999
end_coveragec [max_segments]	long	all	Coverage flag for swath exit: PV_STAR_UNDEFINED = 0, PV_STAR_UPPER = 1, PV_STAR_LOWER = 2, PV_START_LEFT = 3 , PV_STAR_RIGHT=4		0,1,2,3,4
ierr[10]	long		Error status flags		

Table 18: Output parameters for pv_starvistime

7.8.6 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_starvistime** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg** (see RD 3).

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_starvistime** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code** (see RD 3).

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error in input parameter to starvistime	Computation not performed	PV_CFI_STARVISTIME_INPUTS_CHECK_ERR	0
ERR	Error reading the Orbit event file	Computation not performed	PV_CFI_STARVISTIME_OEF_READ_ERR	1
WARN	Warning, start orbit is outside range of OEF/OSF	Computation performed Message to inform the user	PV_CFI_STARVISTIME_FIRST_ORBIT_WARN	2
WARN	Warning, stop orbit is outside range of OEF/OSF	Computation performed Message to inform the user	PV_CFI_STARVISTIME_LAST_ORBIT_WARN	3
ERR	Error updating star's positionfrom JD200 to determined UTC.	Computation not performed	PV_CFI_STARVISTIME_STAR_RADEC_ERR	4
ERR	Error obtaining orbital information.	Computation not performed	PV_CFI_STARVISTIME_ORBIT_INFO_ERR	5
WARN	Warning, there is an orbital change within the requested orbits	Visibility segments from that point are not returned	PV_CFI_STARVISTIME_ORBIT_CHANGE_WARN	6
ERR	Error reading the upper swath template file.	Computation not performed	PV_CFI_STARVISTIME_SWATH_UPPER_READ_ERR	7
ERR	Error reading the lower swath template file.	Computation not performed	PV_CFI_STARVISTIME_SWATH_LOWER_READ_ERR	8
ERR	Error, starvistime can only operate with an inertial swath	Computation not performed	PV_CFI_STARVISTIME_INERTIAL_SWATH_ERR	9
ERR	Error, orbital information does not coincide with reference swath	Computation not performed	PV_CFI_STARVISTIME_INCONSISTENT_SWATH_ERR	10

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error reading the star data file	Computation not performed	PV_CFI_STARVISTIME_READ_STAR_ERR	11
ERR	Low swath altitude is above the upper limit described by the higher swath altitude	Computation not performed	PV_CFI_STAVISTIME_ALT_ERR	12
ERR	Error determining transitions	Computation not performed	PV_CFI_STARVISTIME_STAR_MAIN_ERR	13
ERR	Error checking orbit range cycle	Computation not performed. The input orbit range is incompatible with STF cycle	PV_CFI_STARVISTIME_CALL_ORBIT_RANGE_CYCLE_ERR	14
ERR	The requested orbit range is not contained in swath cycle	Computation not performed The input orbit range is not compatible with STF cycle	PV_CFI_STARVISTIME_ORBIT_RANGE_CYCLE_ERR	15
WARN	The orbit %li is not contained in swath cycle	Message to inform the user. The orbits not compatible with swath cycle are not taken into account in computation	PV_CFI_STARVISTIME_ORBIT_RANGE_CYCLE_WARN	16
ERR	Error calculating start/stop orbits	Computation not performed. Error trying to compute start/stop orbits when 0 is entered as start_orbit or stop_orbit.	PV_CFI_STARVISTIME_ORB_DEFAULT_VALUES_ERR	17

7.8.7 Runtime performances

The following runtime performances have been measured, for 5000 orbits, running in a Sun Ultra Sparc, under different input conditions:

	800 swath points	1200 swath points
DECLINATION : -35.0 deg	25.28 seconds	37.75 seconds
DECLINATION : 0.0 deg	23.07 seconds	34.31 seconds
DECLINATION : 75.0 deg	12.45 seconds	18.47 seconds

It can be seen that the above time depends strongly both on the position in declination of the star and the number of points contained by the Swath Template files.

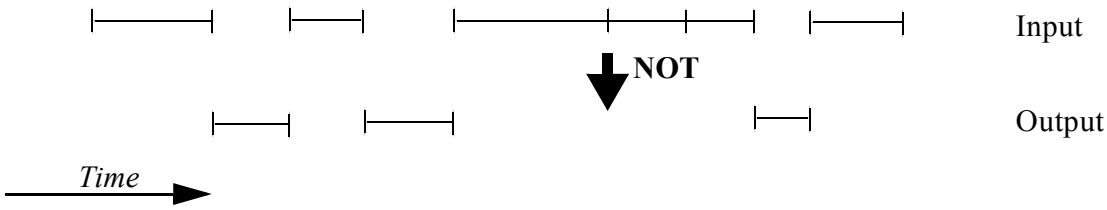
7.9 pv_time_segments_not

7.9.1 Overview

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as an orbit number and the seconds elapsed since the ascending node crossing.

The **pv_time_segments_not** function computes the compliment of a list of orbital segments (see Figure 12)

Figure 12 pv_time_segments_not function



Note that the intervals from the first orbit to the first segment and from the last segment to the end of mission are not returned.

The input segments list need to be sorted according to the start time of the segments. If this list is not sorted, it should be indicated in the function interface with the corresponding parameter (see below). In this case the input list will be modified accordingly.

The time intervals used by **pv_time_segments_not** can be expressed in absolute or relative orbit numbers. This is valid for both:

- input parameter: first and last orbit to be considered. In case of using relative orbits, the corresponding cycle numbers should be used, otherwise, the cycle number will be a dummy parameter.
- output parameter: time segments with time expressed as {absolute orbit number (or relative orbit and cycle number), number of seconds since ANX, number of micro seconds}

The orbit representation (absolute or relative) for the output segments will be the same as in the input orbits. Moreover, the segments will be ordered chronologically.

The **pv_time_segments_not** requires access to the following files to produce its results:

- the Orbit Scenario File: only if the orbits are expressed in relative numbers.

7.9.2 Calling sequence pv_time_segments_not

For C programs, the call to **pv_time_segments_not** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{

       long     orbit_type, order_switch,
                num_segments_in,
                *bgn_orbit_in, *bgn_secs_in,
                *bgn_microsecs_in, *bgn_cycle_in,
                *end_orbit_in, *end_secs_in,
                *end_microsecs_in, *end_cycle_in,
                num_segments_out,
                *bgn_orbit_out, *bgn_secs_out,
                *bgn_microsecs_out, *bgn_cycle_out,
                *end_orbit_out, *end_secs_out,
                *end_microsecs_out, *end_cycle_out,
                ierr[1], status;
        char    *orbit_scenario_file;

       status = pv_time_segments_not(
                        orbit_scenario_file,
                        &orbit_type,  &order_switch,
                        &number_segments_in,
                        bgn_orbit_in, bgn_secs_in,
                        bgn_microsecs_in, bgn_cycle_in,
                        end_orbit_in, end_secs_in,
                        end_microsecs_in, end_cycle_in,
                        &num_segments_out,
                        &bgn_orbit_out, &bgn_secs_out,
                        &bgn_microsecs_out, &bgn_cycle_out,
                        &end_orbit_out, &end_secs_out,
                        &end_microsecs_out, &end_cycle_out,
                        ierr);

/* test status */
}
```

For FORTRAN programs **pv_time_segments_not** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
       INTEGER*4   ORBIT_TYPE, ORDER_SWITCH,
      &            NUM_SEGMENTS_IN,
      &            BGN_ORBIT_IN, BGN_SECS_IN,
      &            BGN_MICROSECS_IN, BGN_CYCLE_IN,
      &            END_ORBIT_IN, END_SECS_IN,
      &            END_MICROSECS_IN, END_CYCLE_IN,
      &            NUM_SEGMENTS_OUT,
      &            BGN_ORBIT_OUT, BGN_SECS_OUT,
      &            BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
      &            END_ORBIT_OUT, END_SECS_OUT,
      &            END_MICROSECS_OUT, END_CYCLE_OUT,
                   IERR(1), STATUS;
       CHARACTER*(*)*ORBIT_SCENARIO_FILE


#include"ppf_visibility.inc"

      STATUS = PV_TIME_SEGMENTS_NOT(
                        ORBIT_SCENARIO_FILE,
                        ORBIT_TYPE, ORDER_SWITCH,
                        NUMBER_SEGMENTS_IN,
                        BGN_ORBIT_IN, BGN_SECS_IN,
                        BGN_MICROSECS_IN, BGN_CYCLE_IN,
                        END_ORBIT_IN, END_SECS_IN,
                        END_MICROSECS_IN, END_CYCLE_IN,
                        NUM_SEGMENTS_OUT,
                        BGN_ORBIT_OUT, BGN_SECS_OUT,
                        BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
                        END_ORBIT_OUT, END_SECS_OUT,
                        END_MICROSECS_OUT, END_CYCLE_OUT,
                        IERR)

C test status
```

7.9.3 Input parameters pv_time_segments_not

Table 19: Input parameters of pv_time_segments_not

c name	c type	Array Element	Description	Units	Range
orbit_scenario_file	char *		The scenario file describes the orbital changes and the repeat cycle and cycle length. This is only necessary when using relative orbits, otherwise an empty string (“”)can be used.	-	-
orbit_type	long	-	Define the type of orbit representation, i.e. absolute or relative orbits in the input/output parameters	-	Complete (see table 1)
order_switch	long	-	Indicates if the input list is sorted by start times. If input segments are already sorted, the flag should be set to PV_TIME_ORDER to save computation time.	-	Complete (see table 1)
num_segments_in	long	-	Number of segments in the input list.	-	>0
bgn_orbit_in	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs_in	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs_in	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle_in	long*	all	Array of cycle numbers for the beggining of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit_in	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs_in	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs_in	long*	all	Array of seconds within a second for the end of the segments	-	>0 <999999
end_cycle_in	long*	all	Array of cycle numbers for the end of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL

7.9.4 Output parameters pv_time_segments_not

Table 20: Output parameters of pv_time_segments_not

c name	c type	Array Ele-ment	Description	Unit	Range
pv_time_segments_not	long		Function status flag, = 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
num_segments_out	long	-	Number of segments in the output list.	-	>0
bgn_orbit_out	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs_out	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs_out	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 < 999999
bgn_cycle_out	long*	all	Array of cycle numbers for the beggining of the segments.	-	>0
end_orbit_out	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs_out	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs_out	long*	all	Array of microseconds within a second for the end of the segments	-	>0 < 999999
end_cycle_out	long*	all	Array of cycle numbers for the end of the segments.	-	>0 or NULL
ierr[	long	0	Error status flags		

Memory Management: Note that the output visibility segments arrays are pointers to integers instead of static arrays. The memory for these dynamic arrays is allocated within the **pv_time_segments_not** function. So the user will only have to declare those pointers but not to allocate memory for them. However, once the function has returned without error, the user will have the responsibility of freeing the memory for those pointers once they are not used.

7.9.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_time_segments_not** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg**.

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_time_segments_not** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code**.

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error allocating internal memory.	Computation not performed	PV_CFI_TIME_SEGMENTS_NOT_MEMORY_ERR	0
ERR	Error getting absolute orbit vector from relative orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_NOT_REL_TO_ABS_ORBIT_ERR	1
ERR	Error getting relative orbit vector from absolute orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_NOT_ABS_TO_REL_ORBIT_ERR	2
ERR	Error sorting input list.	Computation not performed	PV_CFI_TIME_SEGMENTS_NOT_SORTING_ERR	3

7.9.6 Runtime performances

The following runtime performance has been measured.

Table 21: Runtime performances of pv_time_segments_not function

Ultra Sparc [ms]
TBD

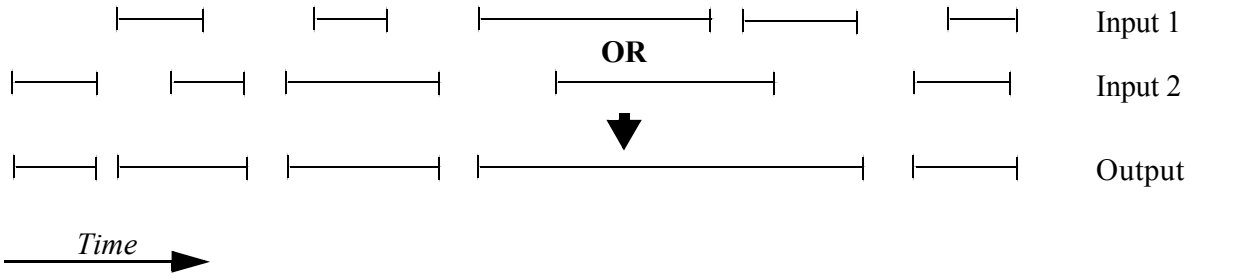
7.10 pv_time_segments_or

7.10.1 Overview

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as an orbit number and the seconds elapsed since the ascending node crossing.

The **pv_time_segments_or** function computes the union of a list of orbital segments (see Figure 13)

Figure 13 pv_time_segments_or function



The input segments list need to be sorted according to the start time of the segments. If this list is not sorted, it should be indicated in the function interface with the corresponding parameter (see below). In this case the input list will be modified accordingly.

The time intervals used by **pv_time_segments_or** can be expressed in absolute or relative orbit numbers. This is valid for both:

- input parameter: first and last orbit to be considered. In case of using relative orbits, the corresponding cycle numbers should be used, otherwise, the cycle number will be a dummy parameter.
- output parameter: time segments with time expressed as {absolute orbit number (or relative orbit and cycle number), number of seconds since ANX, number of microseconds}

The orbit representation (absolute or relative) for the output segments will be the same as in the input orbits. Moreover, the segments will be ordered chronologically.

The **pv_time_segments_or** requires access to the following files to produce its results:

- the Orbit Scenario File: only if the orbits are expressed in relative numbers.

7.10.2 Calling sequence pv_time_segments_or

For C programs, the call to **pv_time_segments_or** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{

       long     orbit_type, order_switch,
                num_segments_1,
                *bgn_orbit_1, *bgn_secs_1,
                *bgn_microsecs_1, *bgn_cycle_1,
                *end_orbit_1, *end_secs_1,
                *end_microsecs_1, *end_cycle_1,
                num_segments_2,
                *bgn_orbit_2, *bgn_secs_2,
                *bgn_microsecs_2, *bgn_cycle_2,
                *end_orbit_2, *end_secs_2,
                *end_microsecs_2, *end_cycle_2,
                num_segments_out,
                *bgn_orbit_out, *bgn_secs_out,
                *bgn_microsecs_out, *bgn_cycle_out,
                *end_orbit_out, *end_secs_out,
                *end_microsecs_out, *end_cycle_out,
                ierr[1], status;
       char     *orbit_scenario_file;

       status = pv_time_segments_or (
                        orbit_scenario_file,
                         &orbit_type, &order_switch,
                         &number_segments_1,
                         bgn_orbit_1, bgn_second_1,
                        bgn_microsec_1, bgn_cycle_1,
                         end_orbit_1, end_second_1,
                        end_microsec_1, end_cycle_1,
                         &number_segments_2,
                         bgn_orbit_2, bgn_second_2,
                        bgn_microsec_2, bgn_cycle_2,
                        end_orbit_2, end_second_2,
                        end_microsec_2, end_cycle_2,
                         &num_segments_out,
                         &bgn_orbit_out, &bgn_secs_out,
                        &bgn_microsecs_out, &bgn_cycle_out,
                         &end_orbit_out, &end_secs_out,
                        &end_microsecs_out, &end_cycle_out,
                        ierr);

/* test status */
}
```

For FORTRAN programs **pv_time_segments_or** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
      INTEGER*4   ORBIT_TYPE, ORDER_SWITCH,
     &            NUM_SEGMENTS_1,
     &            BGN_ORBIT_1, BGN_SECS_1,
     &            BGN_MICROSECS_1, BGN_CYCLE_1,
     &            END_ORBIT_1, END_SECS_1,
     &            END_MICROSECS_1, END_CYCLE_1,
     &            NUM_SEGMENTS_2,
     &            BGN_ORBIT_2, BGN_SECS_2,
     &            BGN_MICROSECS_2, BGN_CYCLE_2,
     &            END_ORBIT_2, END_SECS_2,
     &            END_MICROSECS_2, END_CYCLE_2,
     &            NUM_SEGMENTS_OUT,
     &            BGN_ORBIT_OUT, BGN_SECS_OUT,
     &            BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
     &            END_ORBIT_OUT, END_SECS_OUT,
     &            END_MICROSECS_OUT, END_CYCLE_OUT,
                  IERR(1), STATUS
      CHARACTER*(*)*ORBIT_SCENARIO_FILE


#include"ppf_visibility.inc"

      STATUS = PV_TIME_SEGMENTS_OR(
     &                 ORBIT_SCENARIO_FILE,
     &                 ORBIT_TYPE, ORDER_SWITCH,
     &                 NUMBER_SEGMENTS_1,
     &                 BGN_ORBIT_1, BGN_SECS_1,
     &                 BGN_MICROSECS_1, BGN_CYCLE_1,
     &                 END_ORBIT_1, END_SECS_1,
     &                 END_MICROSECS_1, END_CYCLE_1,
     &                 NUMBER_SEGMENTS_2,
     &                 BGN_ORBIT_2, BGN_SECS_2,
     &                 BGN_MICROSECS_2, BGN_CYCLE_2,
     &                 END_ORBIT_2, END_SECS_2,
     &                 END_MICROSECS_2, END_CYCLE_2,
     &                 NUM_SEGMENTS_OUT,
     &                 BGN_ORBIT_OUT, BGN_SECS_OUT,
     &                 BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
     &                 END_ORBIT_OUT, END_SECS_OUT,
     &                 END_MICROSECS_OUT, END_CYCLE_OUT,
                       IERR)

C test status
```

7.10.3 Input parameters pv_time_segments_or

Table 22: Input parameters of pv_time_segments_or

c name	c type	Array Element	Description	Units	Range
orbit_scenario_file	char *		The scenario file describes the orbital changes and the repeat cycle and cycle length. This is only necessary when using relative orbits, otherwise an empty string (“”)can be used.	-	-
orbit_type	long	-	Define the type of orbit representation, i.e. absolute or relative orbits in the input/output parameters	-	Complete (see table 1)
order_switch	long	-	Indicates if the input list is sorted by start times. If input segments are already sorted, the flag should be set to PV_TIME_ORDER to save computation time.	-	Complete (see table 1)
num_segments_1	long	-	Number of segments in the input list 1.	-	>0
bgn_orbit_1	long*	all	Array of orbit numbers for the beggining of the segments in list 1	-	>0
bgn_secs_1	long*	all	Array of seconds elapsed since ANX for the beggining of the segments in list 1	-	>0 <nodal period
bgn_microsecs_1	long*	all	Array of microseconds within a second for the beggining of the segments in list 1	-	>0 <999999
bgn_cycle_1	long*	all	Array of cycle numbers for the beggining of the segments in list 1. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit_1	long*	all	Array of orbit numbers for the end of the segments in list 1	-	>0
end_secs_1	long*	all	Array of seconds elapsed since ANX for the end of the segments in list 1	-	>0 <nodal period
end_microsecs_1	long*	all	Array of microseconds within a second for the end of the segments in list 1	-	>0 <999999
end_cycle_1	long*	all	Array of cycle numbers for the end of the segments in list 1. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
num_segments_2	long	-	Number of segments in the input list 2.	-	>0

Table 22: Input parameters of pv_time_segments_or

c name	c type	Array Element	Description	Units	Range
bgn_orbit_2	long*	all	Array of orbit numbers for the beggining of the segments in list 2	-	>0
bgn_secs_2	long*	all	Array of seconds elapsed since ANX for the beggining of the segments in list 2	-	>0 <nodal period
bgn_microsecs_2	long*	all	Array of microseconds within a second for the beggining of the segments in list 2	-	>0 <999999
bgn_cycle_2	long*	all	Array of cycle numbers for the beggining of the segments in list 2. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit_2	long*	all	Array of orbit numbers for the end of the segments in list 2	-	>0
end_secs_2	long*	all	Array of seconds elapsed since ANX for the end of the segments in list 2	-	>0 <nodal period
end_microsecs_2	long*	all	Array of microseconds within a second for the end of the segments in list 2	-	>0 <999999
end_cycle_2	long*	all	Array of cycle numbers for the end of the segments in list 2. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL

7.10.4 Output parameters pv_time_segments_or

Table 23: Output parameters of pv_time_segments_or

c name	c type	Array Ele-ment	Description	Unit	Range
pv_time_segments_or	long		Function status flag, = 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
num_segments_out	long	-	Number of segments in the output list.	-	>0
bgn_orbit_out	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs_out	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs_out	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle_out	long*	all	Array of cycle numbers for the beggining of the segments.	-	>0
end_orbit_out	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs_out	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs_out	long*	all	Array of microseconds within a second for the end of the segments	-	>0 <999999
end_cycle_out	long*	all	Array of cycle numbers for the end of the segments.	-	>0 or NULL
ierr	long	0	Error status flags		

Memory Management: Note that the output visibility segments arrays are pointers to integers instead of static arrays. The memory for these dynamic arrays is allocated within the **pv_time_segments_or** function. So the user will only have to declare those pointers but not to allocate memory for them. However, once the function has returned without error, the user will have the responsibility of freeing the memory for those pointers once they are not used.

7.10.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_time_segments_or** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg**.

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_time_segments_or** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code**.

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error allocating internal memory.	Computation not performed	PV_CFI_TIME_SEGMENTS_OR_MEMORY_ERR	0
ERR	Error getting absolute orbit vector from relative orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_OR_REL_TO_ABS_ORBIT_ERR	1
ERR	Error getting relative orbit vector from absolute orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_OR_ABS_TO_REL_ORBIT_ERR	2
ERR	Error sorting input list.	Computation not performed	PV_CFI_TIME_SEGMENTS_OR_SORTING_ERR	3

7.10.6 Runtime performances

The following runtime performance has been measured.

Table 24: Runtime performances of pv_time_segments_or function

Ultra Sparc [ms]
TBD

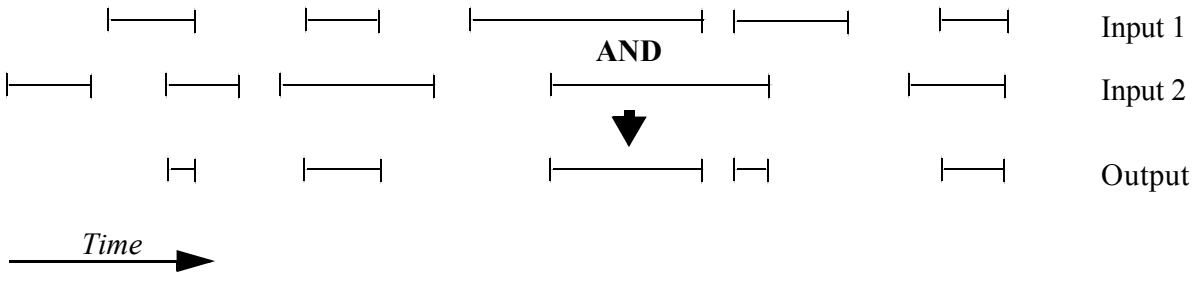
7.11 pv_time_segments_and

7.11.1 Overview

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as an orbit number and the seconds elapsed since the ascending node crossing.

The **pv_time_segments_and** function computes the intersection of a list of orbital segments (see Figure 14)

Figure 14 pv_time_segments_and function



The input segments list need to be sorted according to the start time of the segments. If this list is not sorted, it should be indicated in the function interface with the corresponding parameter (see below). In this case the input list will be modified accordingly.

The time intervals used by **pv_time_segments_and** can be expressed in absolute or relative orbit numbers. This is valid for both:

- input parameter: first and last orbit to be considered. In case of using relative orbits, the corresponding cycle numbers should be used, otherwise, the cycle number will be a dummy parameter.
- output parameter: time segments with time expressed as {absolute orbit number (or relative orbit and cycle number), number of seconds since ANX, number of microseconds}

The orbit representation (absolute or relative) for the output segments will be the same as in the input orbits. Moreover, the segments will be ordered chronologically.

The **pv_time_segments_and** requires access to the following files to produce its results:

- the Orbit Scenario File: only if the orbits are expressed in relative numbers.

7.11.2 Calling sequence pv_time_segments_and

For C programs, the call to **pv_time_segments_and** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{

       long     orbit_type, order_switch,
                num_segments_1,
                *bgn_orbit_1, *bgn_secs_1,
                *bgn_microsecs_1, *bgn_cycle_1,
                *end_orbit_1, *end_secs_1,
                *end_microsecs_1, *end_cycle_1,
                num_segments_2,
                *bgn_orbit_2, *bgn_secs_2,
                *bgn_microsecs_2, *bgn_cycle_2,
                *end_orbit_2, *end_secs_2,
                *end_microsecs_2, *end_cycle_2,
                num_segments_out,
                *bgn_orbit_out, *bgn_secs_out,
                *bgn_microsecs_out, *bgn_cycle_out,
                *end_orbit_out, *end_secs_out,
                *end_microsecs_out, *end_cycle_out,
                ierr[1], status;
       char     *orbit_scenario_file;

       status = pv_time_segments_and (
                        orbit_scenario_file,
                         &orbit_type, &order_switch,
                         &number_segments_1,
                         bgn_orbit_1, bgn_second_1,
                        bgn_microsec_1, bgn_cycle_1,
                         end_orbit_1, end_second_1,
                        end_microsec_1, end_cycle_1,
                         &number_segments_2,
                         bgn_orbit_2, bgn_second_2,
                        bgn_microsec_2, bgn_cycle_2,
                        end_orbit_2, end_second_2,
                        end_microsec_2, end_cycle_2,
                         &num_segments_out,
                         &bgn_orbit_out, &bgn_secs_out,
                        &bgn_microsecs_out, &bgn_cycle_out,
                         &end_orbit_out, &end_secs_out,
                        &end_microsecs_out, &end_cycle_out,
                        ierr);

/* test status */
}
```

For FORTRAN programs **pv_time_segments_and** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
      INTEGER*4   ORBIT_TYPE, ORDER_SWITCH,
     &            NUM_SEGMENTS_1,
     &            BGN_ORBIT_1, BGN_SECS_1,
     &            BGN_MICROSECS_1, BGN_CYCLE_1,
     &            END_ORBIT_1, END_SECS_1,
     &            END_MICROSECS_1, END_CYCLE_1,
     &            NUM_SEGMENTS_2,
     &            BGN_ORBIT_2, BGN_SECS_2,
     &            BGN_MICROSECS_2, BGN_CYCLE_2,
     &            END_ORBIT_2, END_SECS_2,
     &            END_MICROSECS_2, END_CYCLE_2,
     &            NUM_SEGMENTS_OUT,
     &            BGN_ORBIT_OUT, BGN_SECS_OUT,
     &            BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
     &            END_ORBIT_OUT, END_SECS_OUT,
     &            END_MICROSECS_OUT, END_CYCLE_OUT,
                  IERR(1), STATUS
      CHARACTER*(*)*ORBIT_SCENARIO_FILE


#include"ppf_visibility.inc"

      STATUS = PV_TIME_SEGMENTS_AND(
     &                 ORBIT_SCENARIO_FILE,
     &                 ORBIT_TYPE, ORDER_SWITCH,
     &                 NUMBER_SEGMENTS_1,
     &                 BGN_ORBIT_1, BGN_SECS_1,
     &                 BGN_MICROSECS_1, BGN_CYCLE_1,
     &                 END_ORBIT_1, END_SECS_1,
     &                 END_MICROSECS_1, END_CYCLE_1,
     &                 NUMBER_SEGMENTS_2,
     &                 BGN_ORBIT_2, BGN_SECS_2,
     &                 BGN_MICROSECS_2, BGN_CYCLE_2,
     &                 END_ORBIT_2, END_SECS_2,
     &                 END_MICROSECS_2, END_CYCLE_2,
     &                 NUM_SEGMENTS_OUT,
     &                 BGN_ORBIT_OUT, BGN_SECS_OUT,
     &                 BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
     &                 END_ORBIT_OUT, END_SECS_OUT,
     &                 END_MICROSECS_OUT, END_CYCLE_OUT,
                       IERR)

C test status
```

7.11.3 Input parameters pv_time_segments_and

Table 25: Input parameters of pv_time_segments_and

c name	c type	Array Element	Description	Units	Range
orbit_scenario_file	char *		The scenario file describes the orbital changes and the repeat cycle and cycle length. This is only necessary when using relative orbits, otherwise an empty string (“”)can be used.	-	-
orbit_type	long	-	Define the type of orbit representation, i.e. absolute or relative orbits in the input/output parameters	-	Complete (see table 1)
order_switch	long	-	Indicates if the input list is sorted by start times. If input segments are already sorted, the flag should be set to PV_TIME_ORDER to save computation time.	-	Complete (see table 1)
num_segments_1	long	-	Number of segments in the input list 1.	-	>0
bgn_orbit_1	long*	all	Array of orbit numbers for the beggining of the segments in list 1	-	>0
bgn_secs_1	long*	all	Array of seconds elapsed since ANX for the beggining of the segments in list 1	-	>0 <nodal period
bgn_microsecs_1	long*	all	Array of microseconds within a second for the beggining of the segments in list 1	-	>0 <999999
bgn_cycle_1	long*	all	Array of cycle numbers for the beggining of the segments in list 1. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit_1	long*	all	Array of orbit numbers for the end of the segments in list 1	-	>0
end_secs_1	long*	all	Array of seconds elapsed since ANX for the end of the segments in list 1	-	>0 <nodal period
end_microsecs_1	long*	all	Array of microseconds within a second for the end of the segments in list 1	-	>0 <999999
end_cycle_1	long*	all	Array of cycle numbers for the end of the segments in list 1. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
num_segments_2	long	-	Number of segments in the input list 2.	-	>0

Table 25: Input parameters of pv_time_segments_and

c name	c type	Array Element	Description	Units	Range
bgn_orbit_2	long*	all	Array of orbit numbers for the beggining of the segments in list 2	-	>0
bgn_secs_2	long*	all	Array of seconds elapsed since ANX for the beggining of the segments in list 2	-	>0 <nodal period
bgn_microsecs_2	long*	all	Array of microseconds within a second for the beggining of the segments in list 2	-	>0 <999999
bgn_cycle_2	long*	all	Array of cycle numbers for the beggining of the segments in list 2. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit_2	long*	all	Array of orbit numbers for the end of the segments in list 2	-	>0
end_secs_2	long*	all	Array of seconds elapsed since ANX for the end of the segments in list 2	-	>0 <nodal period
end_microsecs_2	long*	all	Array of microseconds within a second for the end of the segments in list 2	-	>0 <999999
end_cycle_2	long*	all	Array of cycle numbers for the end of the segments in list 2. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL

7.11.4 Output parameters pv_time_segments_and

Table 26: Output parameters of pv_time_segments_and

c name	c type	Array Element	Description	Unit	Range
pv_time_segments_and	long		Function status flag, = 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
num_segments_out	long	-	Number of segments in the output list.	-	>0
bgn_orbit_out	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs_out	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs_out	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle_out	long*	all	Array of cycle numbers for the beggining of the segments.	-	>0
end_orbit_out	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs_out	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs_out	long*	all	Array of microseconds within a second for the end of the segments	-	>0 <999999
end_cycle_out	long*	all	Array of cycle numbers for the end of the segments.	-	>0 or NULL
ierr	long	0	Error status flags		

Memory Management: Note that the output visibility segments arrays are pointers to integers instead of static arrays. The memory for these dynamic arrays is allocated within the **pv_time_segments_and** function. So the user will only have to declare those pointers but not to allocate memory for them. However, once the function has returned without error, the user will have the responsibility of freeing the memory for those pointers once they are not used.

7.11.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_time_segments_and** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg**.

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_time_segments_and** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code**.

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error allocating internal memory.	Computation not performed	PV_CFI_TIME_SEGMENTS_AND_MEMORY_ERR	
ERR	Error getting absolute orbit vector from relative orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_AND_REL_TO_ABS_ORBIT_ERR	
ERR	Error getting relative orbit vector from absolute orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_AND_ABS_TO_REL_ORBIT_ERR	
ERR	Error sorting input list.	Computation not performed	PV_CFI_TIME_SEGMENTS_AND_SORTING_ERR	

7.11.6 Runtime performances

The following runtime performance has been measured.

Table 27: Runtime performances of pv_time_segments_and function

Ultra Sparc [ms]
TBD

7.12 pv_time_segments_sort

7.12.1 Overview

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as an orbit number and the seconds elapsed since the ascending node crossing.

The **pv_time_segments_sort** function sorts a list of orbital segments following two different criteria:

- Absolute orbits: the segments are sorted by their start time
- Relative orbits

The time intervals used by **pv_time_segments_sort** can be expressed in absolute or relative orbit numbers. This is valid for both:

- input parameter: first and last orbit to be considered. In case of using relative orbits, the corresponding cycle numbers should be used, otherwise, the cycle number will be a dummy parameter.
- output parameter: time segments with time expressed as {absolute orbit number (or relative orbit and cycle number), number of seconds since ANX, number of microseconds}

The orbit representation (absolute or relative) for the output segments will be the same as in the input orbits. Note that the sort criteria does not have any relation with the chosen orbit representation. The following example clarifies this:

Input orbits: 6, 8, 4, 5, 9, 3 (absolute)

Let's suppose that the cycle length is 4 orbits. Then the relative orbits are:

input orbits: 2, 4, 4, 1, 1, 3 (relative)

When ordering this array, we have the following possibilities(table 28) depending on the orbit representation and the sort criteria chosen:

Table 28: pv_time_segments_sort function

Input	Sort Criteria	Output
absolute orbits 6, 8, 4, 5, 9, 3	absolute orbits	absolute orbits 3, 4, 5, 6, 8, 9
	relative orbits	absolute orbits 5, 9, 6, 3, 4, 8
relative orbits 2, 4, 4, 1, 1, 3	absolute orbits	relative orbits 3, 4, 1, 2, 4, 1
	relative orbits	relative orbits 1, 1, 2, 3, 4, 4

The **pv_time_segments_sort** requires access the following files to produce its results:

- the Orbit Scenario File: only if the orbits are expressed in relative numbers.

7.12.2 Calling sequence pv_time_segments_sort

For C programs, the call to **pv_time_segments_sort** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{

       long     orbit_type, sort_criteria,
                num_segments,
                *bgn_orbit, *bgn_secs,
                *bgn_microsecs, *bgn_cycle,
                *end_orbit, *end_secs,
                *end_microsecs, *end_cycle,
                ierr[1], status;
       char     *orbit_scenario_file;

       status = pv_time_segments_sort (
                        orbit_scenario_file,
                         &orbit_type, &sort_criteria,
                         &number_segments,
                         bgn_orbit, bgn_second,
                        bgn_microsec, bgn_cycle,
                         end_orbit, end_second,
                        end_microsec, end_cycle,
                        ierr);

/* test status */
}
```

For FORTRAN programs **pv_time_segments_sort** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
       INTEGER*4   ORBIT_TYPE, ORDER_SWITCH,
      &            NUM_SEGMENTS,
      &            BGN_ORBIT, BGN_SECS,
      &            BGN_MICROSECS, BGN_CYCLE,
      &            END_ORBIT, END_SECS,
      &            END_MICROSECS, END_CYCLE,
                   IERR(1), STATUS
       CHARACTER*(*)*ORBIT_SCENARIO_FILE


#include"ppf_visibility.inc"

       STATUS = PV_TIME_SEGMENTS_SORT(
      &                 ORBIT_SCENARIO_FILE,
      &                 ORBIT_TYPE, ORDER_SWITCH,
      &                 NUMBER_SEGMENTS,
      &                 BGN_ORBIT, BGN_SECS,
      &                 BGN_MICROSECS, BGN_CYCLE,
      &                 END_ORBIT, END_SECS,
      &                 END_MICROSECS, END_CYCLE,
                        IERR)

C test status
```

7.12.3 Input parameters pv_time_segments_sort

Table 29: Input parameters of pv_time_segments_sort

c name	c type	Array Element	Description	Units	Range
orbit_scenario_file	char *		The scenario file describes the orbital changes and the repeat cycle and cycle length. This is only necessary when using relative orbits, otherwise an empty string (“”)can be used.	-	-
orbit_type	long	-	Define the type of orbit representation, i.e. absolute or relative orbits in the input/output parameters	-	Complete (see table 1)
sort_criteria	long	-	sorting criteria to be used: absolute or relative orbits	-	Complete (see table 1)
num_segments	long	-	Number of segments in the input.	-	>0
bgn_orbit	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle	long*	all	Array of cycle numbers for the beggining of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs	long*	all	Array of microseconds within a second for the end of the segments.	-	>0 <999999
end_cycle	long*	all	Array of cycle numbers for the end of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL

7.12.4 Output parameters pv_time_segments_sort

Table 30: Output parameters of pv_time_segments_sort

c name	c type	Array Element	Description	Unit	Range
pv_time_segments_and	long		Function status flag, = 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
ierr	long	0	Error status flags		

7.12.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_time_segments_sort** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg**.

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_time_segments_sort** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code**.

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error allocating internal memory.	Computation not performed	PV_CFI_TIME_SEGMENTS_SORT_MEMORY_ERR	0
ERR	Error getting absolute orbit vector from relative orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_SORT_CHANGING_ORBIT_ERR	1

7.12.6 Runtime performances

The following runtime performance has been measured.

Table 31: Runtime performances of pv_time_segments_sort function

Ultra Sparc [ms]
TBD

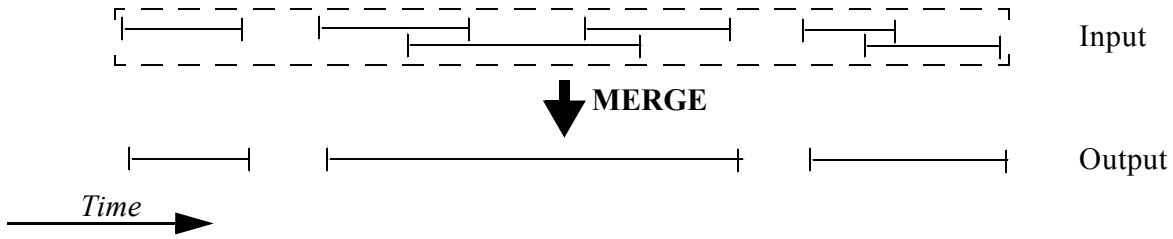
7.13 pv_time_segments_merge

7.13.1 Overview

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as an orbit number and the seconds elapsed since the ascending node crossing.

The **pv_time_segments_merge** function merges all the overlapped segments within a list (see Figure 15)

Figure 15 pv_time_segments_merge function



The input segments list need to be sorted according to the start time of the segments. If this list is not sorted, it should be indicated in the function interface with the corresponding parameter (see below). In this case the input list will be modified accordingly.

The time intervals used by **pv_time_segments_merge** can be expressed in absolute or relative orbit numbers. This is valid for both:

- input parameter: first and last orbit to be considered. In case of using relative orbits, the corresponding cycle numbers should be used, otherwise, the cycle number will be a dummy parameter.
- output parameter: time segments with time expressed as {absolute orbit number (or relative orbit and cycle number), number of seconds since ANX, number of microseconds}

The orbit representation (absolute or relative) for the output segments will be the same as in the input orbits. Moreover, the segments will be ordered chronologically.

The **pv_time_segments_merge** requires access to the following files to produce its results:

- the Orbit Scenario File: only if the orbits are expressed in relative numbers.

7.13.2 Calling sequence pv_time_segments_merge

For C programs, the call to **pv_time_segments_merge** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{

       long     orbit_type, order_switch,
                num_segments,
                *bgn_orbit, *bgn_secs,
                *bgn_microsecs, *bgn_cycle,
                *end_orbit, *end_secs,
                *end_microsecs, *end_cycle,
                num_segments_out,
                *bgn_orbit_out, *bgn_secs_out,
                *bgn_microsecs_out, *bgn_cycle_out,
                *end_orbit_out, *end_secs_out,
                *end_microsecs_out, *end_cycle_out,
                ierr[1], status;
        char    *orbit_scenario_file;

       status = pv_time_segments_merge(
                        orbit_scenario_file,
                        &orbit_type,  &order_switch,
                        &number_segments,
                        bgn_orbit, bgn_secs,
                        bgn_microsecs, bgn_cycle,
                        end_orbit, end_secs,
                        end_microsecs, end_cycle,
                        &num_segments_out,
                        &bgn_orbit_out, &bgn_secs_out,
                        &bgn_microsecs_out, &bgn_cycle_out,
                        &end_orbit_out, &end_secs_out,
                        &end_microsecs_out, &end_cycle_out,
                        ierr);

/* test status */
}
```

For FORTRAN programs **pv_time_segments_merge** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
      INTEGER*4     ORBIT_TYPE, ORDER_SWITCH,
      &             NUM_SEGMENTS,
      &             BGN_ORBIT, BGN_SECS,
      &             BGN_MICROSECS, BGN_CYCLE,
      &             END_ORBIT, END_SECS,
      &             END_MICROSECS, END_CYCLE,
      &             NUM_SEGMENTS_OUT,
      &             BGN_ORBIT_OUT, BGN_SECS_OUT,
      &             BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
      &             END_ORBIT_OUT, END_SECS_OUT,
      &             END_MICROSECS_OUT, END_CYCLE_OUT,
                    IERR(1), STATUS;
       CHARACTER*(*)*ORBIT_SCENARIO_FILE


#include"ppf_visibility.inc"

       STATUS = PV_TIME_SEGMENTS_MERGE(
      &                  ORBIT_SCENARIO_FILE,
      &                  ORBIT_TYPE, ORDER_SWITCH,
      &                  NUMBER_SEGMENTS,
      &                  BGN_ORBIT, BGN_SECS,
      &                  BGN_MICROSECS, BGN_CYCLE,
      &                  END_ORBIT, END_SECS,
      &                  END_MICROSECS, END_CYCLE,
      &                  NUM_SEGMENTS_OUT,
      &                  BGN_ORBIT_OUT, BGN_SECS_OUT,
      &                  BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
      &                  END_ORBIT_OUT, END_SECS_OUT,
      &                  END_MICROSECS_OUT, END_CYCLE_OUT,
                         IERR)

C test status
```

7.13.3 Input parameters pv_time_segments_merge

Table 32: Input parameters of pv_time_segments_merge

c name	c type	Array Element	Description	Units	Range
orbit_scenario_file	char *		The scenario file describes the orbital changes and the repeat cycle and cycle length. This is only necessary when using relative orbits, otherwise an empty string (“”)can be used.	-	-
orbit_type	long	-	Define the type of orbit representation, i.e. absolute or relative orbits in the input/output parameters	-	Complete (see table 1)
order_switch	long	-	Indicates if the input list is sorted by start times. If input segments are already sorted, the flag should be set to PV_TIME_ORDER to save computation time.	-	Complete (see table 1)
num_segments_in	long	-	Number of segments in the input list.	-	>0
bgn_orbit	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle	long*	all	Array of cycle numbers for the beggining of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs	long*	all	Array of microseconds within a second for the end of the segments	-	>0 <999999
end_cycle	long*	all	Array of cycle numbers for the end of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL

7.13.4 Output parameters pv_time_segments_merge

Table 33: Output parameters of pv_time_segments_merge

c name	c type	Array Ele-ment	Description	Unit	Range
pv_time_segments_merge	long		Function status flag, = 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
num_segments_out	long	-	Number of segments in the output list.	-	>0
bgn_orbit_out	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs_out	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs_out	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle_out	long*	all	Array of cycle numbers for the beggining of the segments.	-	>0
end_orbit_out	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs_out	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs_out	long*	all	Array of microseconds within a second for the end of the segments	-	>0 <999999
end_cycle_out	long*	all	Array of cycle numbers for the end of the segments.	-	>0 or NULL
ierr	long	0	Error status flags		

Memory Management: Note that the output visibility segments arrays are pointers to integers instead of static arrays. The memory for these dynamic arrays is allocated within the **pv_time_segments_merge** function. So the user will only have to declare those pointers but not to allocate memory for them. However, once the function has returned without error, the user will have the responsibility of freeing the memory for those pointers once they are not used.

7.13.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_time_segments_merge** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg**.

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_time_segments_merge** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code**.

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error allocating internal memory.	Computation not performed	PV_CFI_TIME_SEGMENTS_MERGE_MEMORY_ERR	0
ERR	Error getting absolute orbit vector from relative orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_MERGE_REL_TO_ABS_ORBIT_ERR	1
ERR	Error getting relative orbit vector from absolute orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_MERGE_ABS_TO_REL_ORBIT_ERR	2
ERR	Error sorting input list.	Computation not performed	PV_CFI_TIME_SEGMENTS_MERGE_SORTING_ERR	3

7.13.6 Runtime performances

The following runtime performance has been measured.

Table 34: Runtime performances of pv_time_segments_merge function

Ultra Sparc [ms]
TBD

7.14 pv_time_segments_delta

7.14.1 Overview

An orbital segment is a time interval along the orbit, defined by start and stop times expressed as an orbit number and the seconds elapsed since the ascending node crossing.

The **pv_time_segments_delta** function makes the segments within a list, longer or shorter, depending on the user election. After increasing/decreasing the longitude of the segments, these are sorted and merged to avoid possible overlapping. Therefore, at the end the list is sorted and without overlapped segments.

The time intervals used by **pv_time_segments_delta** can be expressed in absolute or relative orbit numbers. This is valid for both:

- input parameter: first and last orbit to be considered. In case of using relative orbits, the corresponding cycle numbers should be used, otherwise, the cycle number will be a dummy parameter.
- output parameter: time segments with time expressed as {absolute orbit number (or relative orbit and cycle number), number of seconds since ANX, number of microseconds}

The orbit representation (absolute or relative) for the output segments will be the same as in the input orbits.

The **pv_time_segments_delta** requires access to the following files to produce its results:

- the Orbit Scenario File: only if the orbits are expressed in relative numbers.

7.14.2 Calling sequence pv_time_segments_delta

For C programs, the call to **pv_time_segments_delta** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{

       long     operation, orbit_type,
                delta_secs, delta_microsecs,
                num_segments,
                *bgn_orbit, *bgn_secs,
                *bgn_microsecs, *bgn_cycle,
                *end_orbit, *end_secs,
                *end_microsecs, *end_cycle,
                num_segments_out,
                *bgn_orbit_out, *bgn_secs_out,
                *bgn_microsecs_out, *bgn_cycle_out,
                *end_orbit_out, *end_secs_out,
                *end_microsecs_out, *end_cycle_out,
                ierr[1], status;
        char    *orbit_scenario_file;

       status = pv_time_segments_delta(
                        orbit_scenario_file,
                        &operation, &orbit_type,
                        &delta_secs, &delta_microsecs,
                        &number_segments,
                        bgn_orbit, bgn_secs,
                        bgn_microsecs, bgn_cycle,
                        end_orbit, end_secs,
                        end_microsecs, end_cycle,
                        &num_segments_out,
                        &bgn_orbit_out, &bgn_secs_out,
                        &bgn_microsecs_out, &bgn_cycle_out,
                        &end_orbit_out, &end_secs_out,
                        &end_microsecs_out, &end_cycle_out,
                        ierr);

/* test status */
}
```

For FORTRAN programs **pv_time_segments_delta** has the following calling sequence (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
      INTEGER*4   OPERATION, ORBIT_TYPE,
     &            DELTA_SECS, DELTA_MICROSECS,
     &            NUM_SEGMENTS,
     &            *BGN_ORBIT, *BGN_SECS,
     &            *BGN_MICROSECS, *BGN_CYCLE,
     &            *END_ORBIT, *END_SECS,
     &            *END_MICROSECS, *END_CYCLE,
     &            NUM_SEGMENTS_OUT,
     &            *BGN_ORBIT_OUT, *BGN_SECS_OUT,
     &            *BGN_MICROSECS_OUT, *BGN_CYCLE_OUT,
     &            *END_ORBIT_OUT, *END_SECS_OUT,
     &            *END_MICROSECS_OUT, *END_CYCLE_OUT,
                  IERR(1), STATUS;
      CHARACTER*(*)*ORBIT_SCENARIO_FILE


#include"ppf_visibility.inc"

      STATUS = PV_TIME_SEGMENTS_DELTA(
     &                 ORBIT_SCENARIO_FILE,
     &                 ORBIT_TYPE, ORDER_SWITCH,
     &                 NUMBER_SEGMENTS,
     &                 BGN_ORBIT, BGN_SECS,
     &                 BGN_MICROSECS, BGN_CYCLE,
     &                 END_ORBIT, END_SECS,
     &                 END_MICROSECS, END_CYCLE,
     &                 NUM_SEGMENTS_OUT,
     &                 BGN_ORBIT_OUT, BGN_SECS_OUT,
     &                 BGN_MICROSECS_OUT, BGN_CYCLE_OUT,
     &                 END_ORBIT_OUT, END_SECS_OUT,
     &                 END_MICROSECS_OUT, END_CYCLE_OUT,
                       IERR)

C test status
```

7.14.3 Input parameters pv_time_segments_delta

Table 35: Input parameters of pv_time_segments_delta

c name	c type	Array Element	Description	Units	Range
orbit_scenario_file	char *		The scenario file describes the orbital changes and the repeat cycle and cycle length. This is only necessary when using relative orbits, otherwise an empty string (“”)can be used.	-	-
operation	long		Define the type of operation is to be performed in the segments, i.e. increase or decrease the segments duration	-	Complete (see table 1)
orbit_type	long	-	Define the type of orbit representation, i.e. absolute or relative orbits in the input/output parameters	-	Complete (see table 1)
delta_secs	long		Number of seconds to add/subtract to the segments	-	>=0
delta_microsecs	long		Number of microseconds to add/subtract to the segments	-	>=0
num_segments_in	long	-	Number of segments in the input list.	-	>0
bgn_orbit	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle	long*	all	Array of cycle numbers for the beggining of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL
end_orbit	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs	long*	all	Array of microseconds within a second for the end of the segments	-	>0 <999999
end_cycle	long*	all	Array of cycle numbers for the end of the segments. When using absolute orbits, a NULL pointer can be used.	-	>0 or NULL

7.14.4 Output parameters pv_time_segments_delta

Table 36: Output parameters of pv_time_segments_delta

c name	c type	Array Element	Description	Unit	Range
pv_time_segments_delta	long		Function status flag, = 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
num_segments_out	long	-	Number of segments in the output list.	-	>0
bgn_orbit_out	long*	all	Array of orbit numbers for the beggining of the segments	-	>0
bgn_secs_out	long*	all	Array of seconds elapsed since ANX for the beggining of the segments	-	>0 <nodal period
bgn_microsecs_out	long*	all	Array of microseconds within a second for the beggining of the segments	-	>0 <999999
bgn_cycle_out	long*	all	Array of cycle numbers for the beggining of the segments.	-	>0
end_orbit_out	long*	all	Array of orbit numbers for the end of the segments	-	>0
end_secs_out	long*	all	Array of seconds elapsed since ANX for the end of the segments	-	>0 <nodal period
end_microsecs_out	long*	all	Array of microseconds within a second for the end of the segments	-	>0 <999999
end_cycle_out	long*	all	Array of cycle numbers for the end of the segments.	-	>0 or NULL
ierr	long	0	Error status flags		

Memory Management: Note that the output visibility segments arrays are pointers to integers instead of static arrays. The memory for these dynamic arrays is allocated within the **pv_time_segments_delta** function. So the user will only have to declare those pointers but not to allocate memory for them. However, once the function has returned without error, the user will have the responsibility of freeing the memory for those pointers once they are not used.

7.14.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_time_segments_delta** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg**.

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_time_segments_delta** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code**.

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error allocating internal memory	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_MEMORY_ERR	0
ERR	Error getting absolute orbit vector from relative orbits	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_REL_TO_ABS_ERR	1
ERR	Error getting relative orbit vector from absolute orbits	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_ABS_TO_REL_ERR	2
ERR	Error transforming from orbits to processing times.	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_ORBIT_TO_TIME_ERR	3
ERR	Error transforming from processing times to orbits.	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_TIME_TO_ORBIT_ERR	4
ERR	Error modifying time segment duration	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_TIME_ADD_ERR	5
ERR	Error, incorrect operation defined	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_OPERATION_ERR	6
ERR	Error sorting input list	Computation not performed	PV_CFI_TIME_SEGMENTS_DELTA_SORT_ERR	7

7.14.6 Runtime performances

The following runtime performance has been measured.

Table 37: Runtime performances of pv_time_segments_delta function

Ultra Sparc [ms]
TBD

7.15 pv_compute_mlst_drift

7.15.1 Overview

The **pv_compute_mlst_drift** function compute the MLST and MLST drift for a given orbit using as inputs an orbit scenario file.

7.15.2 Calling sequence pv_compute_mlst_drift

For C programs, the call to **pv_compute_mlst_drift** is (input parameters are underlined):

```
#include"ppf_visibility.h"
{
        char    *osf;
        long    abs_orbit;
        double  mlst_drift;
        long    status, ierr[1];

        status = pv_compute_mlst_drift(osf, &abs_orbit,
                                       &mlst_drift, ierr);
         /* test status */
}
```

7.15.3 Input parameters pv_compute_mlst_drift

Table 38: Input parameters of pv_compute_mlst_drift

c name	c type	Array Ele-ment	Description	Units	Range
osf	char *		The scenario file describes the orbital changes and the repeat cycle and cycle length. This is only necessary when using relative orbits, otherwise an empty string (“”)can be used.	-	-
abs_orbit	long*	-	Requested absolute orbit number	-	>0

7.15.4 Output parameters pv_compute_mlst_drift

Table 39: Output parameters of pv_compute_mlst_drift

c name	c type	Array Ele-ment	Description	Unit	Range
pv_compute_mlst_drift	long		Function status flag, = 0 No error > 0 Warnings, results generated < 0 Error, no results generated		
mlst_drift	double	-	MLST drift at ANX of the requested orbit	s/year	
ierr	long	0	Error status flags		

7.15.5 Warnings and errors

Next table lists the possible error messages that can be returned by the **pv_compute_mlst_drift** CFI function after translating the returned status vector into the equivalent list of error messages by calling the function of the PPF_VISIBILITY software library **pv_vector_msg**.

This table also indicates the type of message returned, i.e. either a warning (WARN) or an error (ERR), the cause of such a message and the impact on the performed calculation, mainly on the results vector.

The table is completed by the error code and value. These error codes can be obtained translating the status vector returned by the **pv_compute_mlst_drift** CFI function by calling the function of the PPF_VISIBILITY software library **pv_vector_code**.

Error type	Error message	Cause and impact	Error Code	Error No
ERR	Error reading the Orbit Sce-nario file.	Computation not performed	PV_CFI_COMPUTE_MLST_OSF_READ_ERR	0
ERR	Orbit number is before the range of OEF/OSF	Computation not performed	PV_CFI_COMPUTE_MLST_WRONG_ORBIT_ERR	1

7.15.6 Runtime performances

The following runtime performance has been measured.

Table 40: Runtime performances of pv_compute_mlst_drift function

Ultra Sparc [ms]
TBD

8 LIBRARY PRECAUTIONS

The following precautions shall be taken into account when using PPF_VISIBILITY software library:

- When a message like

PPF_VISIBILITY >>> ERROR in *pv_function*: Internal computation error # *n*

or

PPF_VISIBILITY >>> WARNING in *pv_function*: Internal computation warning # *n*

appears, run the program in *verbose* mode for a complete description of warnings and errors,

and call for maintenance if necessary.

- In some cases, the string [PL] or [PO] or [PP] appears at the end of the error/warning message. In these cases, run the program in *pl_verbose* or *po_verbose* or *pp_verbose* mode for a complete description of warnings and errors coming from other libraries.

- When the string ““, that is, empty string, is the *orbit_event_file* input string in any function of this library, the information coming from the Orbit Event File (or Orbit Scenario File) used in the last call to any function of PPF_VISIBILITY within the same program is utilized. The first PPF_VISIBILITY called in a program must have a complete *orbit_event_file* string (““ is not allowed for the first function called). All the following functions called with ““ as *orbit_event_file* input string afterwards, use the same information, until another *orbit_event_file* different from ““ is used as input, and so on.

9 KNOWN PROBLEMS

The following precautions shall be taken into account when using the CFI software libraries:

CFI function	Problem	Work around solution