esa

DOCUMENT

document title/ titre du document

GENERIC IPF INTERFACE SPECIFICATIONS

reference/*réference*	MMFI-GSEG-EOPG-TN-07-0003
issue/*édition*	1
revision/*révision*	8
date of issue/*date d'édition*	03/08/2009
status/*état*	Approved
Document type/*type de document*	ICD

European Space Agency
Agence spatiale européenne

ESRIN
Via Galileo Galilei - Casella Postale 64 - 00044 Frascati - Italy
Tel. (39) 06 941801 - Fax (39) 06 94180 280 www.esa.int

APPROVAL

Title *Titre*	Generic IPF Interface Specification	issue 1 *issue*	revision 8 *revision*

author *auteur*	A.Buongiorno	date 03/08/09 *date*

approved by	P.Lecomte	date 06/10/09

CHANGE LOG

reason for change /*raison du changement*	issue/*issue*	revision/*revision*	date/*date*
Preliminary Release for external distribution	1	6	02/05/2007
EOP-G Reviewed version	1	7	26/02/2009
EOP-G Reviewed version	1	8	03/08/2009

CHANGE RECORD

Issue: 1 Revision: 7

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*
Updated release for external distribution	All	All

Issue: 1 Revision: 7

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*
Tag name update from Order to List_of_Orders, following tag list	58	
Tag name not used " PREFIX " removed from File_name_type list	32	
IPF WS configuration file description update to non-compulsory one	23, 57	
Product Report file formatting rules updated	62	
Product report format updated to explicitly include the job order tag	63	
Corrected position of Tag pool in respect of the Dyn processing parameter tag	26	

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*
Added ”count” attribute to pool tag	26	
Appendix E update: removal of unclear examples	87	
Added IPF configuration file interface description	76	
Added Sensing_Time_flag into the Task Table	26	
Corrected time format for start/stop	41	
Footnote into appendix B for correlating time values definitions	80	
Note added into the output tag: File_name_type	34	
Updated selection criteria information	78	
Added new optional tag into the task table about the number of CPU the task is using	35	
Updated job order tag level into Product Report file	64	

Issue: 1 Revision: 8

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*
Updated release after comments from external reviewers	All	All

Issue: 1 Revision: 8

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*
Updated Title of the document in line with previous 1.6 version	Cover page	

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*
Updated text for time format with microseconds	41	
Corrected job order tag level into Product Report file	64	

TABLE OF CONTENTS

INTRODUCTION

1.1 Purpose and scope

This document contains a set of mission-independent guidelines for the specification of the interface between the Instrument Processing Facility (IPF) and the Processors Management Layer, that are elements of the Processing Facility (PF) of the Payload Data Segment (PDGS).
The purpose of such a guideline is twofold: on the one side it is intended to decouple as much as possible the PDGS and IPF development, and on the other one it aims at establishing common practice across the various ESA future missions for the development and the integration of new processors.
To this purpose the document specifies all interfaces, conventions and design constraints that must be satisfied by any processor that is to be integrated in the PDGS environment.
The contents of this document are supposed to be integrated by PF specific ICDs to be issued in the frame of each mission development.

1.2 How to use this document

The chapter organisation of this document follows a learning-path that starts from the introduction of basic PF vocabulary and ends up with the detailed specification of each interface.

The first part of the document contains a general description of the PF system as well as some considerations and evaluations about the main system characteristics.
The reader is not supposed to have any prerequisite for reading this section.

The second part is mostly intended for reference: all the interfaces used in the PF have been identified, classified and specified in dedicated sections (one section for each interface)
A set of appendixes is thought of as reference as well: each appendix describes specific implementation details.

1.3 Document structure

The body of this document consists of four main sections, including this introduction, plus appendices.

Section 2 (The Processing Facility) is a general introduction to the PF system architecture. This section has a twofold purpose:

- to provide basic information about the reference architecture characteristics and strengths
- to introduce the basic PF vocabulary

In Section 3 (Interface Definition), all the PF Management Layer <->IPF interfaces are identified and classified according to their level of necessity for the IPF implementation.
This section is supposed to be read by everybody.

Section 4 (Interface Specification) is the part of the document to be used as reference: every interface as identified in section 3 is specified in a dedicated sub-section.
This section is supposed to be read by everybody.

1.4 Conventions

This document specifies the structure of several XML files. In order to keep the explanation as simple as possible their description will be based on tables and on the conventions described in the next sub-section.

1.4.1 SPECIFICATION OF XML FILES

A 7-columns table is used to specify the contents of each XML file covered by this specification (see next snapshot). Each raw describes a tag.

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain

- **Level.** This is a very simple, graphical symbol showing the indentation level:
  - `>` top level (no indentation)
  - `1` 1 level indentation
  - `=2` 2 levels indentation
  - `==3` 3 levels indentation
  - **…**
- **Tag name**
- **Descendants.** The list of the tags enclosed into the specified tag. The following symbols can be found in the description of each descendant:
  - ⊕ exclusive OR among possible descendant tags
  - ∩ AND among possible descendant tags
  - ∪ OR among possible descendant tags
  - ∅ no descendant tags
  - **All** top level tag enclosing anything in the file
- **Cardinality.** The number of occurrences of the tag. Possible values:
  - [0,n] at most n times
  - 1 exactly one
  - + at least one

* as many as wanted including zero.

- **Tag Contents.** Contents of the tag. In case the tag can assume a finite set of values, then all of the possible values are listed and separated by ∪**.** In all of the other cases a verbose description is used**.** The default value, if any, is written by using italics/bold characters. ∅ is also possible if the tag is empty.

- **Attribute name.** Name of tag's attribute, if any; ∅ if none

- **Attribute domain.** Allowed range for each attribute. It could hold: ∅ to flag that tag is not supposed to hold any value. In case of integer values, the following symbols are used as well:
  [0,n] any integer in the range 0…n
  [n,n] exactly n
  \+ any integer greater than or equal to 1
  * any integer greater than or equal to 0.

1.5 Reference Documents

Document Title	Identifier	Internal Reference
EE Ground Segment(s) Payload Data Segment PDGS-IPF ICD Generic Interface Guidelines	ESA-ID-ACS-GS-001 Issue:2.2, Date 01/08/2006	[PDGS-IPF GDLNS]
GOCE Payload Data Segment Generic IPF Processor ICD	GO-ID-ACS-GS-0113, Issue: 2.0, Date 14/05/2006	[GP-ICD]

1.6 Acronyms and abbreviations

I/F Interface

PF Processing Facilty

IPF Instrument Processing Facility

PDGS Payload Data Ground Segment

ML Management Layer

WS	Workstation

THE PROCESSING FACILITY (PF)

This section addresses the following basic questions:

- What is the Processing Facility?
- What is an Instrument Processing Facility?
- What is the Management Layer?
- How does the PF work?
- What are the PF strengths?

What is the Processing Facility?

The PF is a sub-system of the Payload Data Segment (PDGS), i.e. that part of the Ground Segment of a specific mission that processes the data coming from the payload(s) of the spacecraft. Typically the main purpose of the PDGS is to transform raw telemetry data in a set of higher-level products for the final users, e.g.: Level 0, Level1, Level 2, etc,.

Regardless of the mission, the most general, minimal block diagram of the PDGS architecture can be depicted as in figure 2-1 where the PF is presented as well.

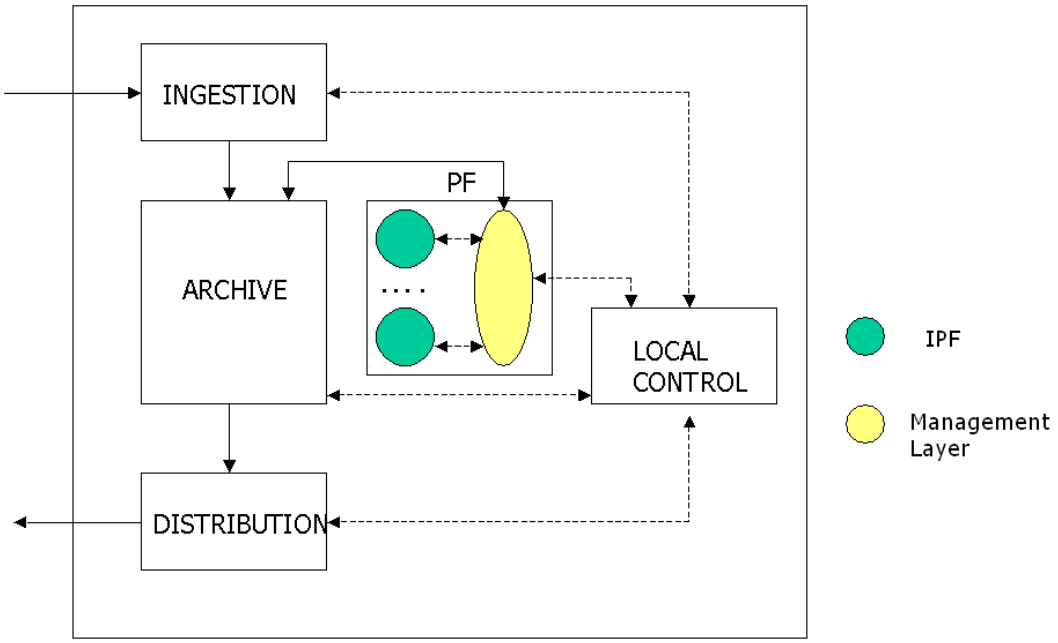


Figure 2-1

The following functional blocks are shown:

- The INGESTION is the section of the PDGS that ingests into the system whatever needed for the PDGS production and operations (raw data, externally provided auxiliary data, orders, etc…)

- The ARCHIVE is the PDGS part that stores all the received input and all the generated products. On command this section is capable to retrieve data for distribution or product generation.
- The DISTRIBUTION section takes care of disseminating the products to the users according to the chosen distribution policy
- LOCAL CONTROL provides tools for controlling and monitoring the overall PDGS operations.
- The PF transforms raw data into higher-level products

The PF architecture is based on two cornerstones: the **Instrument Processing Facility** – that encapsulates the algorithmic and computational part of the product generation process - and the **Management Layer** – that controls the Processors' operations and implements the interface to the other PDGS elements.

All the interfaces between the Management Layer and IPF are file-based.

According to this the first basic question can be now answered.

A PF is that architectural part of the PDGS that consists of a Management Layer and set of Instruments Processors .
The Management Layer interfaces the IPF on one side and the PDGS on the other one.
All the interfaces are file-based.

What is the Instrument Processing Facility?

The **Instrument Processing Facility** is the component of the PF that includes the scientific processing of the payload data and that must generate products at the required level.

From the architectural point of view…

The IPF is:

- A collection of executables. Each Executable is named: Task
- The specification of the Task execution order
- The specification of the Task input and output data as well as of all the processing parameters

Each Processor Task can input data from files and output its results to (newly generated) files.
Of course, the output of one Task can be an input to the next Task (or one of the next Tasks) in the calling sequence.

Each Task outputs formatted log messages (syslog style ASCII strings) on *stdout* and *stderr* with the purpose to report the most significant events occurring during the processing. Log messages are

categorised and by configuration the operator can decide which message categories have to be issued.

Every Tasks of a Processor can be run either sequentially (i.e. at a given time only one Task is running in the PF) or in parallel to other Tasks of the same Pool.
Accordingly, the definition of Pool is: a group of Tasks that can be executed and controlled in parallel by the Management Layer.
When each Task of a Process has to be run in sequence then there are as many Pools as the number of Tasks and – of course – each Pool will contain just one Task.
Two kinds of Pools do exist: non-detached (foreground execution) and detached (background execution) Pools.
Non-detached Pools are run in sequence, whilst detached Pools can be run in parallel with other detached Pools or with at most one non-detached Pool.
Examples of Pool implementation are given in Appendix G.

The list of input files that a given Task has to use as well as the list of the output files that the Task has to generate is contained in a dedicated file that is called: **Job Order**.
The Management Layer passes the Job Order to the Task as parameter on the command line.
The Job Order is the only parameter passed to the Task on the command line; moreover the same Job order is passed to each Task of the same Processor.

Dynamic Processing Parameters (i.e. parameters that might change at every run of an IPF) can be inserted into the Processing Order. The Management Layer extracts these parameters from the Order and passes them to the Processor by directly copying the Parameters from the Order into a dedicated section of the Job Order.
Possible alternatives to this mechanism should be considered following the mission specific requirements.

Upon completion each Task returns an Exit Code.

How does the PF work?

Physically the PF is supposed to consist of several workstations.
Each PF workstation hosts:

- one or more IPF
- a set of configuration files

What is the Management Layer?

The Management Layer is the component of the PF that interfaces the PDGS and that commands and controls the IPF.

In particular:

- The Management Layer receives processing orders by the PDGS.
- The Management Layer retrieves the input needed by an Instrument Processor from the PDGS Archive
- The Management Layer starts and controls the execution of the IPF.
- The Management Layer reports to the PDGS about the processing activities.
- The Management Layer puts the IPF outputs into the PDGS archive.

The basic criteria driving the PF design are:

- **Functional Encapsulation**: the Management Layer takes care of the control functions; the Instrument Processors take care of the input files/parameters verification, algorithm implementation, output formatting and writing.
- **Simple, file-based interfaces**: in the PF, files are the bricks used to build both data flow (that's quite usual) and the control flow. Actually, the Management Layer and the IPF "exchange" files; the Management Layer and the PDGS "exchange" files.

The file-based IPF interfaces driving the control flow are of two types: static (configuration files) and dynamic (files generated at run-time)

Run-Time Control flow Generated Files are:

- *JOB ORDER*
- *LOGGING*
- EXIT CODE

Static Control flow Configuration Files are:

- *IPF WS CONFIGURATION* file
- *TASK TABLE* files

The PDGS generates a *PROCESSING ORDER* (simply *ORDER* in the following) to command a Processor execution or –correspondingly– to command the generation of a Product (or set of related Products).

These *ORDERs* are put in a centralised queue that can be accessed by each Management Layer that regularly pools the queue and checks for some Order to be fulfilled.

To fulfil an ORDER the Management Layer has to verify that all the inputs needed to generate the ordered Products are available in the PDGS Archive. To do this, the Management Layer makes use of the *TASK TABLE* file on the one side and of specific queries to the PDGS Archive on the other one.

On each Processing Workstation there is one TASK TABLE file for each Processor.
Each TASK TABLE contains the list of Tasks belonging to the Processor as well as the list of files each Task needs in input and the list of files each Task is supposed to generate in output.
More specifically:

- The TASK TABLE contains the lists of file types
- For each file type the TASK TABLE specifies if a file is mandatory or not for the fulfilment of the processing.
- The TASK TABLE specifies also the retrieval policy of each input file type (i.e. the type of query to be used)

According to these contents of the TASK TABLE, the Management Layer queries the PDGS Archive and checks for the presence of the needed input files.

If all the inputs needed to fulfil a specific processing are available, then the Management Layer:

- Creates the working directory on the local disk
- Puts in the working directory all input files needed (downloads from the Archive) by the Processor.
- Generates the Job Order
- Starts in sequence each Processor Task according to the order specified in the Task Table. The Management Layer passes the Job Order to each Task on the command line.

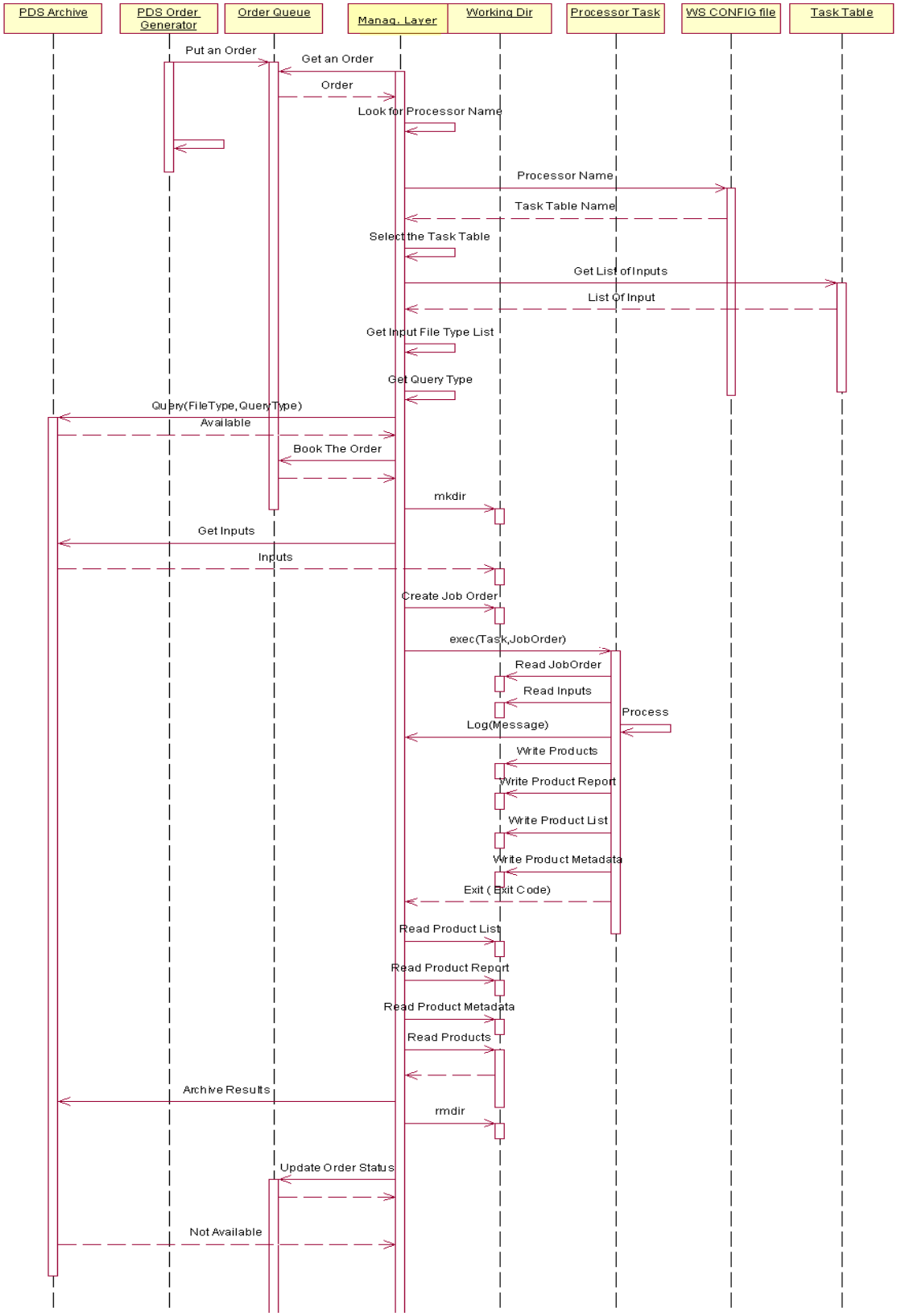
During its execution each Task logs the most significant events occurring during the processing.
Log messages are categorised in five groups: Error, Warning, Information, Progress (they provide an insight of the percentage of the work done up to the message issuing time) and Debug Messages. This classification allows the operator to filter (by configuration) the kinds of messages that a given Processor has to issue during execution.
Different settings control the message filtering level on *stderr* and *stdout*. Examples of events that are logged (the complete list can be found in the specification section of this document) are:

- Start and stop time of processing (information message)
- Names of used files
- Missing input file (warning message if it is not mandatory, error message if it is)
- XML file parsing error (error messages)

Upon execution accomplishment each Task returns an *EXIT CODE* to the Management Layer. According to this value the Management Layer makes the decision whether to start the next Task in the *TASK TABLE* list or not.

When the last Task of the Processor finishes, the Management Layer moves the files to be inventoried to the PDGS Archive and removes the working directory from disk.

The following picture shows an example of the sequence of operations described so far.


PDS Archive
PDS Order Generator
Order Queue
Manag. Layer
Working Dir
Processor Task
WS CONFIG file
Task Table
Put an Order
Get an Order
Order
Look for Processor Name
Processor Name
Task Table Name
Select the Task Table
Get List of Inputs
List Of Input
Get Input File Type List
Get Query Type
Query(FileType, QueryType)
Available
Book The Order
mkdir
Get Inputs
Inputs
Create Job Order
exec(Task,JobOrder)
Read JobOrder
Read Inputs
Process
Log(Message)
Write Products
Write Product Report
Write Product List
Write Product Metadata
Exit (Exit Code)
Read Product List
Read Product Report
Read Product Metadata
Read Products
Archive Results
rmdir
Update Order Status
Not Available

What are the PF strengths?

- **Flexibility.** The PF design is completely independent from mission specific requirements. Actually the key-idea behind is that the system has simply to be event-driven: the PF reacts to events that are supposed to occur in time. Typically these events are inputs that become available at a given time and that have to be used to produce outputs. This is a typical operational scenario of every space mission (and not only).

- **Scalability.** The PF is a parallel, loosely coupled system and – accordingly - is highly scalable. Thanks to the Management Layer design the PF hardware configuration can be sized taking into account only mission specific workload without impacting on any aspect of the architecture.

- **Configurability.** The set of configuration parameters provided by the PF architecture allows the same kind of processors installed on different workstation to be tuned in order to cope with different, specific needs. In particular, the same kind of processors but with different versions can be hosted and managed inside of the same PF.

- **Reliability.** It is easy to realise that the PF is a highly reliable system thanks to the implementation of a distributed control strategy. If some workstation (or some software component on it) failed for some reason, the only impact would be that that WS wouldn't contribute to empty the order queue, however all the rest of the system would work on.

- **Controllability.** Monitoring and controlling the PF is very easy because there is no need of dedicated interfaces for this purpose: the Management Layer provides all these functionality out-off-the-shelf

- **Operability.** The PF can be designed to be operated automatically, i.e. without operator intervention. This goal is achieved thanks to the event-driven architecture on the one side and on the other one thanks to a small set of configuration files. Each of them controls specific operational aspects allowing a strong and easy control of every operational aspect. Manual operations are possible and easy: it is just matter to insert a manually generated order in the order queue or Job Orders to start the processors.

INTERFACE DEFINITION

In this section all the possible IPF interfaces between the Managent Layer are identified, listed and classified.

An interface can be classified as:

- **Compulsory**
- **Non-compulsory**
- **Recommended.**

Compulsory interfaces are the architecture cornerstones: the system cannot work without these interfaces implemented.

Non-compulsory interfaces list includes interfaces that are widely used in existing PF, nevertheless simpler PF or new PF could not need all these interfaces.

The decision about which Non-compulsory interfaces has to be implemented is driven by the Management Layer architecture characteristics. The Management Layer implementation /maintenance team shall provide such information to the IPF implementation team.
The decision shall be in any case supervised and approved by the ESA PDGS development responsible.

The Non-compulsory interfaces to be implemented shall be documented into the relevant PF specific ICDs.

Recommended interfaces are internal to the IPF and basically they drive Processors behaviours. The implementation of this kind of interfaces is considered a good –and often widely used - IPF design practice but the system can work properly also without them.

Recommended interfaces have to be specified into the specific IPF documentation as well as detailed into the field of compulsory interfaces where this information it is foreseen to be placed (e.g. Processing Configuration parameters file in the Task Table and Job Order configuration file tags).

3.1 Compulsory Interfaces

Interface Name	Description	Static/Dynamic Information	Generating Source
Job Order	The Job Order contains the set of input files to the Processor, a specification of the output product types and other processing parameters. The same Job Order file is supplied to every Task of the same Processor	Dynamic	Management Layer
Logging	Each Task must log major processing event according to a specific logging syntax	Dynamic	IPF
Exit Code	The Exit Code of the processors is used to convey processing success/failure information to the Management Layer.	Dynamic	IPF
Processor Task Table	The Task Table specifies the Processor structure (list of Tasks), the list of input/output to each Task as well as the calling sequence of the Tasks and the input files selection rules. There is one Processor Task Table for each Processor.	Static	IPF

3.2 Non-compulsory Interfaces

Interface Name	Description	Static/Dynamic Information	Generating Source
Product Report	The Processor generates the Product Report file whenever an Order is successfully fulfilled. The Product Report contains summary information on the generated Product as well as a brief report about the performed processing (exit code, log messages, etc…)	Dynamic	IPF
Product List	The Product List contains a list of the Product files generated by an Order .	Dynamic	IPF
Product Metadata	The Product Metadata file contains all the input needed by the PDGS to fill up database (Archive) tables upon inventorying the Product the Product Metadata refers to.	Dynamic	IPF
IPF WS Configuration	The IPF WS Configuration file lists the Processors installed on each workstation. There is one IPF WS Configuration file on each workstation	Static	PF

3.3 Recommended Interfaces

Interface Name	Description	Static/Dynamic Information	Source
IPF Processing Configuration Parameters	The IPF Processing Configuration Parameters file contains static parameters and specific configuration settings driving the behaviour of each installed Processor. This file can be passed by the management layer to the IPF via the Job Order.	Static	IPF

INTERFACE SPECIFICATION

4.1 Processor Task Table

4.1.1 PURPOSE

This I/F contains the definition of the processor, in terms of:

- number of composing executables (tasks),
- list of input, output and intermediate file types for each executable.

The table is loaded by Management Layer to know the exact sequence of Tasks to be executed, their location and the location and version of the processor specific configuration file(s).

4.1.2 TYPE

This interface is of type: **Compulsory**.
This interface is implemented by means of an XML file.

4.1.3 FORMAT

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
>	Ipf_Task_Table	All	Ø	1	Ø	Ø
1	Processor_Name	Ø	String known to PDGS	1	Ø	Ø
1	Version	Ø	String known to PDGS	1	Ø	Ø
1	Test	Ø	Yes U ***No***	1	Ø	Ø
1	Min_Disk_Space	Ø	***1024*** U + **NOTE** : this and the next field must be worst case estimates, plus a safety margin	1	units	***MB***
1	Max_Time	Ø	***0*** U + **NOTE** : 0 stands for "no-limit"	1	sec	Ø
1	List_of_Cfg_Files	Cfg_Files	Ø **NOTE** : The list of configuration file s shall be the one provided into the Specific Processor ICD	1	Ø	Ø
=2	Cfg_Files	Version ∩ File_Name	Ø	*	Ø	Ø
==3	Version	Ø	Any string	[0,1]	Ø	Ø
==3	File_Name	Ø	Any valid pathname **NOTE** : This is the complete absolute pathname of the file	1	Ø	Ø
1	List_of_Dyn_ProcParam	Dyn_ProcParam	Ø **NOTE** : This is an optional Tag, It is provided if the Management Layer has no other source for getting such information when it computes the Job Order relevant tags. This tag provides a list of dynamic parameters attributes, including the definition of default values, when necessary.	[0,1]	count	+
=2	Dyn_ProcParam	Param_Name∩	Ø	+		

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
		Param_type_∩ Param default ∩				
==4	Param_Name	Ø	String	+		
==4	Param_Type	Ø	String or number or datenumber	+		
==4	Param_Default	Ø	Default value for the parameter	[0,1]		
1	Sensing_Time_flag	Ø	true U ***false*** This flag is set to true for all IPFs that use a sensing time interval as main processing parameter (e.g. the start/stop time of the segment to be processed or a sub part of it. If "true", a start/stop time will be inserted into the tag name : Sensing_Time of the JO.	[0,1]		
1	List_of_Pools	Pool	Ø	1	count	+
=2	Pool	Detached ∩ Killing_Signal ∩ List_of_Tasks	Ø	+	Ø	Ø
==3	Detached	Ø	true U ***false*** **NOTE** : true means that a background execution is foreseen; false means that a foreground execution is foreseen	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
==3	Killing_Signal	Ø	[1,8] U ***9*** U [10,30]	1	Ø	Ø
==3	List_of_Tasks	Task	Ø	1	count	+
===4	Task	Name ∩ Version ∩ Critical ∩ Criticality_Level ∩ File_Name ∩ List_Of_Inputs ∩ List_Of_Outputs ∩ List_Of_Breakpoints ∩ Number_of_CPUs	Ø	+	Ø	Ø
====5	Name	Ø	Any string	1	Ø	Ø
====5	Version	Ø	Any string	1	Ø	Ø
====5	Critical	Ø	***true*** U false **NOTE** : If set to true, the failure of the task will cause the failure of the whole processor.	1	Ø	Ø
====5	Criticality_Level	Ø	Integer > 1 **NOTE** : used in case of Pools composed by more than one task. If a critical task with a criticality level of "n" fails it will cause the kill of all the tasks in the pool having criticality level less than or equal to "n". The tasks with a criticality level higher than "n" will complete their execution before the whole processor is interrupted and marked as "failed".	1	Ø	Ø
====5	File_Name	Ø	Any valid valid pathname	1	Ø	Ø
====5	List_of_Inputs	Input	∅	1	count	+

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
====6	Input	∅ ∩ (Mode ∩ Mandatory ∩ List_of_Alternatives)	∅	+	id	string **NOTE**: this attribute is used to name the input and re-use it later on without need to re-specify all details
					ref	string **NOTE**: this attribute is used to retrieve the details from a previous input definition; in this latter case, contents of the section are required to be empty (i.e., it is not allowed to redefine an already named input)

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=====7	Mode	Ø	**NOTE** :This tag specify in which mode this input has to be passed to the processor. The Mode Tag, includes one or a combination of modes, and allows the provision of different sets of input files, depending of the processing applicable condition. The tag names can be chosen according to the processing operational needs and configuration. Examples of name operationally used in existing Processing Facilities are herewith provided: NRT U SYSTEMATIC U REPROCESSING U SUBS U ***ALWAYS*** • NRT: near real time orders • SYSTEMATIC: routine production orders • REPROCESSING: Batch reprocessing (with new auxiliary files) of previously executed orders • SUBS: "slice" orders, that is, orders dedicated to extract a sub-portion from a previously generate product • ALWAYS: self evident	1	Ø	Ø
=====7	Mandatory	Ø	***Yes*** U No **NOTE** : This field specifies the Management Layer behaviour in case the relevant file is not found while checking order preconditions: = Yes: abort, the order cannot be executed without this file = No: continue and do not put this file in the job order, the processor is able to work without this file	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=====7	List_of_Alternatives	Alternative	Ø	1	count	+
=======8	Alternative	Order ∩ Origin ∩ Retrieval_Mode ∩ T0 ∩ T1 ∩ File_Type∩ File_Name_Type	Ø	+	Ø	Ø
========9	Order	Ø	* **NOTE**: for each input specification, the possible alternate file types are grouped in the file subsection; each alternative is marked with a preference number: in case more than one alternative file type is available, the one with lowest preference number will be retrieved and passed to the task; if more alternatives are marked with the same preference number, pick-up by Management Layer is random among them	1	Ø	Ø
========9	Origin	Ø	***DB*** U PROC U LOG **NOTE**: this field specifies if the file comes from one of the previous tasks (i.e., it is an intermediate file - PROC), or has to be fetched from the database (DB) or it is the log file (LOG) as generated by the Management Layer. The union over all the tasks of all the files that have to be fetched from the database constitutes the set that is queried upon order booking	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=======9	Retrieval_Mode	Ø **NOTE**: this field specifies if an exact time relationship have to be respected when retrieving the input file from DB, or if the closest in time file has to be picked-up; in general, the PDGS has the capability to use several (and also implement new ones) policies for intput file retrieval; this document reports the ones currently available (for more details, see in the appendix).	ValCover U LatestValCover U ValIntersect U LatestValIntersect U ***LatestValidityClosest*** U BestCenteredCover U LatestValCoverClosest U LargestOverlap U LargestOverlap85 U LatestValidity U LatestValCoverNewestValidity **NOTE**: 1. this feld is only meaningful if **origin** is DB. 2. Exact specification of this field shall be reported for each input inside the processor dedicated ICD 3. This field is ignored if Origin is not DB	1	Ø	Ø
=======9	T0	Ø	Any double >= 0 (***0***) **NOTE** : see next row	1	Ø	Ø
=======9	T1	Ø	Any double >= 0 (***0***) **NOTE**: When these fields are set the Management Layer enlarges the retrieval time window respectively on the left (T0) and on the right (T1) of the specified amount of seconds, with respect to the commanded processing time window, before searching for the input file in the DB	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=======9	File_Type	Ø	string	1	Ø	Ø
=======9	File_Name_Type	Ø	***Physical*** U Logical U Stem U Regexp **NOTE**: 1. each file name is composed by three parts: A/B.C, where A is a directory path, B is a filename in fixed format as per the EO format standards (class, type, time) and C is an extension (e.g., HDR, DBL, EEF, …); the TL is able to generate filenames in the forms: - B ➔ logical - A/B ➔ stem - A/B.C ➔ physical the Regexp option will map a filename of the type A/R, where A is a directory path as in previous options, while R is a regular expression that will match the naming convention of filenames expected to be generated by the processor (usually, this option will be used in association with a previous DBPROC type output – see next section -, or when the processor can generate several output files from a single input file, and the actual segmentation of the file cannot be known a-priori). **IMPORTANT: the regular expression R is not in the format used generally by Linux shells (Bash, csh, sh), but in the POSIX 1003.2 regular expressions format: refer to "man 7 regex" command for details** (this means, for instance, that "any file" shall be specified as ".*", not as "*"). Each	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			processor shall specify default regular expression to be used for each filetype .In case no such default is provided, the PDGS (in particular, the Management Layer and the Management Layer Emulator) will use the default expression ".*<filetype>.*" 2. In combination with the value **PROC** for the field Origin, the **Physical** and **Stem** conventions lead to generation of the same filename in the Job Order by Management Layer.			
===[5]	List_of_Outputs	Output	Ø	1	count	+
=====[6]	Output	Destination ∩ Mandatory ∩ Type ∩ File_Name_Type	Ø	+	Ø	Ø
======[7]	Destination	Ø	***DB*** U PROC U DBPROC **NOTE**: same meaning as Input.Origin DBPROC reflects an output that shall both be archived and used as input in a following task(s)	1	Ø	Ø
======[7]	Mandatory	Ø	***Yes*** U No **NOTE**: If the attribute is set to **No**, then the processor is not expected to always generate the file, that is, its absence in the processing directory upon task exiting does not imply a failure, unless explicitly signalled by the processor itself through the exit code	1	Ø	Ø
======[7]	Type	Ø	String	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			NOTE: for files with destination DB, this field will specify the regular expression matching the file naming convention, known to the processor but not to the Management Layer; upon successful termination of all tasks composing the processor, the Management Layer will collect outputs with destination DB matching the given regular expression, and forward them to the archive for distribution			
======7	File_Name_Type	Ø	***Physical*** U Logical U Stem U Regexp U Directory **NOTE**: 1. If **Directory**, the File_Name field will point to a directory (i.e., no actual file part); otherwise, same meaning as the corresponding field in the Input.Origin 2. In combination with the value **PROC** for the field Destination, the **Physical** and **Stem** conventions lead to generation of the same filename in the Job Order by Management Layer. 3. "Physical" can be used if the naming convention for the output file is known in advance	1	Ø	Ø
====5	List_of_Breakpoints	Breakpoint	Ø	1	count	*
=====6	Breakpoint	File_Name ∩ File_Name_Type	Ø	*	Ø	Ø
======7	File_Name	Ø	A valid Pathname according to			

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			File_Name_Type **NOTE**: It is the responsibility of the IPF contractor to define in a dedicated ICD the list of valid breakpoint name recognized by the IPF. The format of the breakpoints is not supposed to be known by the PDGS			
======7	File_Name_Type	Ø	***Physical*** U Logical U Stem U Regexp U Directory			
====5	Number_of_CPUs	Ø	Any Integer > 0 This optional parameter is an information for the Management layer that specifies the number of CPUs used by a specific task	[0,1]	Ø	Ø

4.1.4 TRANSPORT MECHANISM

The file is installed with the relevant processor. In other words there is Task Table for each installed processor.

4.1.5 FREQUENCY / CONDITIONS FOR TRANSFER

This file is installed on the workstation once for all at the end of the processor installation.

4.2 Job Order

4.2.1 PURPOSE

The Job Order file contains all the settings controlling processing options or algorithm performance, including (but not limited to):

- Configuration settings
- Input/Output/Intermediate file list and location
- Specific processor configuration file
- Dynamic Processing parameters

4.2.2 TYPE

This interface is of type: **Compulsory**.
This interface is implemented by means of an XML file. Its name has a fixed format that looks like in the following:
JobOrder.<order_id>.xml

where the **JobOrder** string is fixed, and the **order_id** component is an integer uniquely identifying the production order for which the processor is being invoked.

4.2.3 FORMAT

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
>	Ipf_Job_Order	Ipf_Conf ∩ List_of_Ipf_Procs	Ø	1	Ø	Ø
1	Ipf_Conf	Processor_Name ∩ Version ∩ Stdout_Log_Level ∩ Stderr_Log_Level ∩ Test ∩ Breakpoint_Enable ∩ Acquisition_Station ∩ Processing_Station ∩ Config_Files ∩ Sensing_Time∩ Dynamic_Processing_Parameters	Ø	1	Ø	Ø
=2	Processor_Name	Ø	The processor name as set into the task Table	1	Ø	Ø
=2	Version	Ø	The processor version as set into the task Table	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=2	Stdout_Log_Level	Ø	DEBUG U ***INFO*** U PROGRESS U WARNING U ERROR **NOTE**: 1 Logging levels are ordered from low-to-high details as in the list. E.g. setting INFO forces PROGRESS, WARNING and ERROR messages to be issued as well. Setting WARNING forces ERROR messages to be issued as well. 2 Logging level put in the Job Order is read from the WS Configuration File (see later on in this document)TBC 3 ERROR level messages will always be issued (i.e., cannot be disabled).	1	Ø	Ø
=2	Stderr_Log_Level	Ø	DEBUG U ***INFO*** U PROGRESS U WARNING U ERROR **NOTE**: 1. Logging levels are ordered from low-to-high details as in the list. E.g. setting INFO forces PROGRESS, WARNING and ERROR messages to be issued as well. Setting WARNING forces ERROR messages to be issued as well. 2. Logging level put in the Job Order is read from the WS Configuration File (see later on in this document)TBC 3. ERROR level messages will always be issued (i.e., cannot be disabled).	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=2	Test	Ø	true U ***false*** **NOTE**: If Test is set to **true**, then the products must be flagged as test products, according to the specific product format TBC	1	Ø	Ø
=2	Breakpoint_Enable	Ø	true U ***false*** **NOTE**: this flag is enabled for troubleshooting orders, and is used in LOGICAL AND with the on/off switch of each breakpoint file specified in the Breakpoint File section.	1	Ø	Ø
=2	Acquisition_Station	Ø	String with mission specific content.	[0,1]	Ø	Ø
=2	Processing_Station	Ø	String with mission specific content	1	Ø	Ø
=2	Config_Files	Conf_File_Name **NOTE**: The config_Files section will contain one entry (named tag) for each configuration file needed by the processor. The list of the configuration files is read from the Task Table, in the section List_of_Cfg_Files . The actual description and format of configuration files shall be found in the processor specific ICDs (e.g., [IPF ICD]). In case of no configuration file such tag shall be present in the Job Order with null occurrence.	Ø	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
==3	Conf_File_Name	Ø ***Note**: Configuraiton file name as specified into the task table*	Absolute pathname and filename of the configuration file	*	Ø	Ø
=2	Sensing_Time	Start ∩ Stop	Ø Note: The sensing start/stop time is a largerly used parameter for Lv0->Lv1 generation,representing the requested data window to be processed; in general, it refers to the "main" input product, i.e., the product whose availability/generation caused the processing order to be emitted (e.g., for L1B products, this is the start-stop time of the L0 product to be processed), but the processors must be prepared to accept sensing time windows that are actually a sub-portion of the complete input reference product. It is set as optional parameter because not all IPFs are triggered by a time window parameter (e.g. some IPFs are triggered by a geographical coordinates window)	[0,1]	Ø	Ø
==3	Start	Ø	The start time of the processing of main input data in the format (year,month,day,hour,minute,seconds,micro seconds) YYYYMMDD_HHMMSSuuuuuu	1	Ø	Ø
==3	Stop	Ø	The stop time for the processing of main input data in the format (year,month,day,hour,minute,seconds,micro seconds) YYYYMMDD_HHMMSSuuuuuu	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=2	Dynamic_Processing_ Parameters	Processing_Parameter	Ø **NOTE**: The dynamic processing parameters are all those that are needed in input to the processors task. They should be defined into the Specific Processor ICD in terms of value, format and allowed ranges..	[0,1]	Ø	Ø
==3	Processing_Parameter	Name ∩ Value	Ø	+	Ø	Ø
===4	Name	Ø	A string defining the name of the parameter: e.g., Center Latitude, Center Longitude, Order_type, etc,…	1	Ø	Ø
===4	Value	Ø	A string defining the value of the parameter: e.g. A Latitude/Longitude in the format : +LLL.lllllll	1	Ø	Ø
1	List_of_Ipf_Procs	Ipf_Procs	Ø	1	count	+
=2	Ipf_Proc	Task_Name ∩ Task_Version ∩ BreakPoint ∩ List_of_Inputs ∩ List_of_Outputs	Ø	+	Ø	Ø
==3	Task_Name	Ø	String	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
==3	Task_Version	Ø	Any String TBC **NOTE**: The Task_Name and Task_Version fields will match one of the values specified in the processor task table (see related section). This check is ensured by the Management Layer.	1	Ø	Ø
==3	BreakPoint	List_of_Brk_Files	Ø Note: this is an optional field used during IPF testing/maintenance mode for those processors that implement intermediate output information for troubleshooting purposes.	[0,1]	Ø	Ø
===4	List_of_Brk_Files	Brk_File	Ø **NOTE**: 1 this tag and its descendants are ignored if tag BreakPoint.Enable is OFF 2 The number of breakpoint files is determined by the Management Layer starting from the processor task table	1	count	*
====5	Brk_File	Enable ∩ File_Type ∩File_Name_Type ∩File_Name	Ø	*	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
====6	Enable	Ø	ON U ***OFF*** **NOTE**: this flag is used in LOGICAL AND with the global Enable_Breakpoints on/off switch in the Ipf_Conf section. The way this setting is obtained is implementation specific, that is, some Management Layer implementation could not provide automatic configuration of this flag on a single order basis. In this case, this flag is set to the value specified by the master switch.	1	Ø	Ø
====6	File_Type	Ø	Breakpoint file type according to processor specific ICD	*	Ø	Ø
====6	File_Name_Type	Ø	***Physical*** U Logical U Stem U Regexp U Directory **NOTE**: same meaning as Input.Origin in the Task Table	*		
====6	File_Name	Ø	Breakpoint file name according to processor specific ICD	*	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
==3	List_of_Inputs	Input	∅	1	count	* **NOTE**: This number is determined by the Management Layer starting from the processor task table
===4	Input	File_Type ∩ File_Name_Type ∩ List_of_File_Names ∩ List_of_Time_Intervals	Ø	*	Ø	Ø
====5	File_Type	Ø	(File type according to processor specific ICD) U LOG **NOTE**: This string is read from the Processor Task Table. While running the processor, the TL creates a logfile with messages intercepted on stdout and stderr of the tasks (see later on, Logging interface). This logfile, in turn, can be passed to the processor as input file (usually, to a report generation task that will translate/filter the log into a product generation report). This constitute a special input case, in the sense that normally inputs are either retrieved from the archive, or generated in the working directory by one of the previous processor tasks (interim files). In this case, the special file type LOG is inserted in the File_Type field.	1	Ø	Ø
====5	File_Name_Type	Ø	***Physical*** U Logical U Stem	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			U Regexp U Directory **NOTE**: same meaning as Input.Origin in the Task Table			
====5	List_of_File_Names	File_Name	Ø	1	count **NOTE**: The number of files for each type is determined by the start/stop time window. It is not guaranteed to be the same for each input type	*
=====6	File_Name	Ø	Any legal and valid filename **NOTE**: In case of Regexp value for the File_Name_Type field, this field will contain a a filename of the type A/R, where A is the path of the working dir, and R is a regular expression that will match the naming convention of filenames expected to be generated by the processor	*	Ø	Ø
====5	List_of_Time_Intervals	Time_Interval	Ø	[0,1]	count	*
=====6	Time_Interval	Start ∩ Stop∩ File_Name	Ø **NOTE**: The time validity interval list is a generic partitioning in subintervals of the sensing time, indicating (in case of multiple files or file overlap) which of the input files should be used in the specified interval. Processors can sidestep this hint, but this shall be documented in the processor specific ICD	*	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
======7	Start	Ø	A time stamp in the format: YYYYMMDD_HHMMSSuuuuuu	1	Ø	Ø
======7	Stop	Ø	A time stamp in the format: YYYYMMDD_HHMMSSuuuuuu	1	Ø	Ø
======7	File_Name	Ø	Any of the filename listed in the previous List_of_File_Names field **NOTE**: 1 a null string ("") in this field means that there is a hole in the data, not covered by any of the input files 2 in case of Regexp File_Name_Type this will contain the same string found in List_of_File_Names.File_Name	1	Ø	Ø
==3	List_of_Outputs	Output	Ø	1	count	* **NOTE**: This number is determined by the Management Layer starting from the processor task table
==4	Output	File_Type ∩ File_Name_Type ∩ File_Name	Ø	*	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
====5	File_Type	Ø	File type according to processor specific ICD	1	Ø	Ø
====5	File_Name_Type	Ø	***Physical*** U Logical U Stem U Regexp U Directory **NOTE**: If Directory, the File_Name field will point to a directory (i.e., no actual file part); otherwise, same meaning as the corresponding field in the Input section	1	Ø	Ø
====5	File_Name	Ø	Any legal and valid filename **NOTE**: 1. If this field maps to an existing directory path, then it means that the output is going to be archived, and shall be generated by the task in the specified directory according to the file naming conventions established in its ICD for the file type reported in the type field; in this case, the File_Name_Type tag is set to Directory 2. Same as for Input in case of Regexp File_Name_Type	1	Ø	Ø

4.2.4 TRANSPORT MECHANISM

The Management Layer generates the Job Order file and puts it in the working directory before starting the Processor.
The complete filename (including path) of the file is passed to the task(s) as the only command line parameter. Note that the file contains settings for all the tasks (i.e., the job order is unique), the same job order file will be passed to all tasks by the Management Layer.

4.2.5 FREQUENCY / CONDITIONS FOR TRANSFER

File is put in the processing directory on an as-needed basis, as soon as all other needed inputs are available in the processing directory itself and have been checked-out from the archive, before starting execution of the first task.

4.3 Logging

4.3.1 PURPOSE

The Logging interface allows the PDGS to monitor the status of the running processor.
To this purpose the Processors generate a set of messages and forwards them to the Management Layer. The Management Layer, in turn, is able to catch these messages and to log them into the log file.
In some cases the Management Layer will forward them to the PDGS for immediate presentation to the PDGS operator.

This I/F specifies:

- Events, warnings and errors that all processors shall log in order to allow the PDGS to properly monitor the system.
- The logging syntax that all processors shall comply with

4.3.2 TYPE

This interface is of type: **Compulsory**.

This interface is stream-based: the processor is required to output the log messages to the stdout and stderr according to the rules as specified in the next sub-sections. The Management Layer saves all of the messages into a log file that will be the input to generate the product report when required. A dedicated Processor Task should carry out this latter operation.

4.3.3 FORMAT

Processors messages are syslog style ASCII strings, variable length, NL terminated.

They are written to stdout (all kinds of messages but error), and to stderr (error messages and – in some implementation - progress messages). It is up to the PDGS Management Layer to intercept these messages and process them, generating appropriate events to the PDGS, storing them in files and routing such files to the archive.
By convention, the stderr channel is used to pass important information to the PDGS in real-time. In other words, whatever received on this channel is immediately routed to the PDGS.
Message format consists of the following fields (i.e., substrings), separated by 1 blank character:

1. Date and Time: 26 characters, in the format yyyy-MM-ddThh:mm:ss.nnnnnn, where the dash, colon, period and T characters are sub-field separators, and the sub-field meaning is as follows:

   a. yyyy → year, four digits
   b. MM → month, two digits, left padded with 0 (zero)
   c. dd → day, two digits, left padded with 0 (zero)
   d. hh → hours, two digits, left padded with 0 (zero)
   e. mm → minutes, two digits, left padded with 0 (zero)
   f. ss → seconds, two digits, left padded with 0 (zero)
   g. nnnnnn → microseconds, six digits, right padded with 0 (zero)

2. **Node Name**: variable length alphanumeric string (no whitespace characters allowed), identifying which of the workstations the processor is running on; each node will have its name set up during configuration; the node name to be put in the message shall be the string retrieved through the **gethostname** system call

3. **Processor Name**: variable length alphanumeric string (no whitespace characters allowed), identifying the processor or the processor component; value for this field shall be specified in the processor specific ICDs

4. **Processor Version**: 5 characters, in the format ***nn.nn*** (the embedded period is included in the count, and must be present)

5. **PID**: 12 characters, in the format ***[nnnnnnnnnn]***, (the squared brackets are included in the count, and must be present), specifying the system level process id of the processor, obtained through the **getpid** system call

6. **Header Separator:** 1 colon character; please, note that **this has not to be considered an informative field**, therefore, **no space shall be inserted between the PID and the separator**;

7. **Message Type:** 3 characters, in the format ***[c]*** (the squared brackets are included in the count, and must be present), identifying the type of message being issued; the character used inside the brackets has to be one listed in the next table:

[c]	*Purpose*	*Description*	*Channel*
[D]	Debug messages	Basic Software Tracing messages	stdout
[I]	Informational messages	It contains info on kind/status of operations	stdout
[P]	Progress messages	It contains info on progress of operations	stderr
[W]	Warning messages	An error occurred, the processor was able to continue	stdout
[E]	Error messages	An error occurred, the processor was not able to continue	stderrr

Note, that all and **only messages intercepted on stderr will generate error/alarm** events to the PDGS, regardless the message type field inside them.
In other words, whatever logged on the stderr descriptor by the processor gets forwarded to the PDGS operator.

8. **Message Text:** variable length string, no restrictions, only NL characters not allowed; additional conventions specific for each processor shall be implemented inside this string (hence, they are transparent to PDGS), and shall be defined inside the processor specific ICD. Note that the message format specified in this document does not foresee the task name, but only the processor name. If the task name is deemed important for some processor, this constraint shall be foreseen constraint in the processor specific ICD, where further syntax on the "free" text can be imposed

A typical message will look like in the following example:

2004-02-24T04:02:07.458000 ipf1ws1 IPF1 01.04 [0000013875]: [I] Processor starting. Config file is /home/IPF1/order0105.xml

Unformatted messages coming out of the processor or messages formatted in a way different from the one specified in this section is not allowed. However, it could happen– for mission specific needs – that some Processor is made of Task reused by previous development. In this case the Management Layer should take care of intercepting these "foreign" messages and pack them into a structure compatible with this specification. In particular, the "foreign" message should be inserted into the field: Message Text of a message structure built by the Management Layer. The type of message field should be filled with a default value (DEBUG is recommended).

4.3.4 TRANSPORT MECHANISM

This interface relies on OS basic services (stdout and stderr).
The processor output is saved into a file that the Management Layer generates in the processing directory,.
In order to allow the Management Layer to generate the logfile preserving the correct messages timing, the tasks shall prevent their stderr and stdout descriptors to be buffered. To achieve this result, tasks shall flush the stderr and stdout descriptors after writing each message.
The file generated by the Management Layer has the fixed name LOG.<order_id>, and is created and maintained in the working dir. It is forbidden to processors to directly write inside this file, or to use a file with this name for its internal purposes

4.3.5 FREQUENCY / CONDITIONS FOR TRANSFER

This section reports the common set of messages that have to be issued by all tasks of all processors.
The complete list of error messages for each processor will be find in each processor's ICD and/or User Manual.

INFORMATIONAL EVENTS			
Name	**Description**	**Trigger**	**Parameters**
Start	The task is starting.	Main function entered	1. processor version 2. job order filename
Input	List of input filenames and processing parameters as extracted from the job order.	End of scanning of input section of the job order file	1. file name
Output	List of output filenames as generated by the processor	End of processors outputs files writing	1. file name
Stop	The task is exiting.	Main function exiting	1. exit code 2. exit code description
Progress	The task has performed a certain percentage of its work; exact frequency of messages shall be specified in the processor ICD	The indicated amount of work has been accomplished	1. Progress indication (exact format shall be specified in the processor dedicated ICD; default is percentage accomplished of total estimated amount of work)
Processing elapsed time	Time elapsed since the processing task started	Start/end of the task	1. User time 2. System time
WARNING EVENTS			
Name	**Description**	**Trigger**	**Parameters**
Default	One of the expected parameters has not been found in the configuration file; a default value is being used.	Unexpected EOF on the configuration file	1. parameter name 2. default value used
Incomplete product	The generated product is shorter than expected	Unexpected EOF on one of the input files	1. expected length in sec 2. actual length in sec 3. filename of the input file that caused EOF
No input	An input file, corresponding to an optional file type, specified in the Job Order cannot be open	File open system call failure	1. file name 2. file type 3. system call error code 4. system call error description (can be obtained through the errno system call)

Missing input	An input file, corresponding to an optional file type, is not specified in the Job Order	End of Job Order file scanning	1. file name 2. file type 3. system call error code 4. system call error description (can be obtained through the errno system call)
No configuration	A configuration space related xml file cannot be open (but a default file is available to the processor)	File open system call failure	1. file name 2. system call error code 3. system call error description (can be obtained through the errno system call) 4. indication of the default configuration file being used from now on
ERROR EVENTS			
Name	**Description**	**Trigger**	**Parameters**
No ipf config	The configuration file cannot be open	File open system call failure	1. file name 2. system call error code 3. system call error description (can be obtained through the **errno** system call)
No proc config	The special configuration file cannot be open	File open system call failure	1. file name 2. system call error code 3. system call error description (can be obtained through the **errno** system call)
No configuration	A configuration space related xml file cannot be open (and no alternate file is being used by the processor)	File open system call failure	1. file name 2. system call error code 3. system call error description (can be obtained through the errno system call)
XML parse	Error detected while parsing an XML configuration file	XML access routine failure	1. routine name 2. returned code 3. faulty tag

No input	An input file cannot be open	File open system call failure	1. file name 2. file type 3. system call error code 4. system call error description (can be obtained through the errno system call)
No output	An output file cannot be created	File open system call failure	1. file name 2. file type 3. system call error code 4. system call error description (can be obtained through the **errno** system call)
No breakpoint	A breakpoint file cannot be open	File open system call failure	1. file name 2. file type 3. system call error code 4. system call error description (can be obtained through the **errno** system call)

4.4 Exit Code

4.4.1 PURPOSE

This I/F specifies the exit code that any task has to provide to the Management Layer (usually through a return statement at the end of the main function, or through a call to the **exit** system call)

4.4.2 TYPE

This interface is of type: **Compulsory**.
Each Exit Code value is an 8-bit unsigned integer.

4.4.3 FORMAT

The following table reports the codes that must be returned, and their meaning.

Value	Mnemonic	Description
0	OK	Product generation or task execution successfully completed. The Management Layer will **continue** with execution of next task of the processor task table.
1-127	INCOMPLETE	Incomplete product generation or product shorter than expected (warning exit code). The Management Layer will **continue** with execution of next task of the processor task table.
128-255	FAILURE	No product generated or task execution failed/aborted (error exit code). The Management Layer will **abort** execution of the processor task table.

4.4.4 TRANSPORT MECHANISM

Operating system services (in particular, the **exit** system call).

4.4.5 FREQUENCY / CONDITIONS FOR TRANSFER

The exit code is automatically generated by the Operating System upon termination of the task, for whatever reason.

4.5 IPF WS Configuration

4.5.1 PURPOSE

This XML configuration file contains all the information that might be needed by some Management Layers to identify which kind of order can be satisfied on a processing workstation, that is, including the list of processors installed on the workstation.

4.5.2 TYPE

This interface is of type: **Non-Compulsory**.
This interface is implemented by means of an XML file.

4.5.3 FORMAT

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
>	Proc_Table	All	Ø	1	Ø	Ø
1	List_of_Processors	Processor	Ø	1	count	+
=2	Processor	Processor_Name ∩ Version ∩ Log_Level ∩ Stdout_Log_Level ∩ Stderr_Log_Level ∩ Task_Table ∩ List_of_Orders	Ø	+		
==3	Processor_Name	Ø	Any string. The name combined to the version fields shall allow to refer to a valid task table file.	1	Ø	Ø
==3	Version	Ø	Any string. The name combined to the version fields shall allow to refer to a valid task table file.	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
==3	Log_Level	Ø	DEBUG U ***INFO*** U PROGRESS U WARNING U ERROR **NOTE**: 1. The Management Layer overrides the programmed Logging Level and sets the Logging level to DEBUG in case of troubleshooting orders 2. this field represents the default value, in case next two settings (stdout and stderr specific) are set to DEFAULT	1	Ø	Ø
==3	Stdout_Log_Level	Ø	***DEFAULT*** U DEBUG U INFO U PROGRESS U WARNING U ERROR **NOTE**: DEFAULT means "whatever value set in the Log_Level tag"	0-1	Ø	Ø
==3	Stderr_Log_Level	Ø	***DEFAULT*** U DEBUG U INFO U PROGRESS U WARNING U ERROR **NOTE**: same as for Stdout_Log_Level tag	0-1	Ø	Ø
==3	Task_Table	Ø		1	Ø	Ø
==3	List_of_Orders	Order	Ø	1	count	+

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
===4	Order	Order_Type ∩ Status	Ø	+	Ø	Ø
===5	Order_Type	Ø	(Order type string according to specific mission) U Trblshoot	1	Ø	Ø
===5	Status	Ø	***enabled*** ∪ disabled	1	Ø	Ø

4.5.4 TRANSPORT MECHANISM

The file is installed locally to each IPF workstation.

4.5.5 FREQUENCY / CONDITIONS FOR TRANSFER

This file is initially installed on the workstation when the processing facility is integrated inside the PDGS, as part of the Management Layer installation procedure. After that, it can be updated whenever one of the following conditions occurs:

1. a new processor is installed on the workstation: the corresponding processor subsection will be inserted in the file (note that in case of processor version upgrade, a new processor subsection need to be inserted only if the previous version is not to be discontinued, otherwise updating the existing section is enough). The processor sub-section relevant to the new processor has to be taken out from the WS configuration file actually delivered with the Processor installation package.
2. a processor is removed from the workstation: the corresponding processor subsection will be removed from the file
3. the configuration of the processor on this workstation has changed: the existing processor subsection will be updated in the file; typical examples of this case are the following:
    a. processor version upgrade, with the previous version discontinued: the version number is modified
    b. load distribution on the workstation is being reallocated: order enabling is modified in the order processing subsection for one or more processors

a processor is being discontinued on this workstation: the processor all orders will be disabled in the processor's order processing subsection

4.6 Product Report

4.6.1 PURPOSE

A generation report shall be produced for each product generated. This report contains at least:

- version of software used to perform the generation
- list of names of all input files used to perform the generation
- start and stop date and time of operation
- full name of files generated
- synthetic exit code (SUCCESS/FAILURE) for the generation result
- list of errors and warnings, in clear, self-explanatory text
- the job order

4.6.2 TYPE

This interface is of type: **Non-compulsory**.
This interface shall be implemented by means of an XML file.
The format described hereafter refers to the data block that can be encapsulated in a standard product structure by adding the header part (although this latter may be empty).
The file name convention for the product report will follow the rules defined for the products if it shall be generated as such, otherwise the file name will be defined in line with the specific rules for temporary/internal files.

The format of the data block is reported in next paragraph.

4.6.3 FORMAT

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
>	Product_Report	All	Ø	1	Ø	Ø
1	File_Name	Ø	Name of the report file without extension and path	1	Ø	Ø
1	List_of_Messages	Message	Ø	1	count	+
=2	Message	Date_Time ∩ Node_Name ∩ Processor_Name ∩ Processor_Version ∩ PID∩Type ∩Text	Ø	+	Ø	Ø
==3	Date_Time	Ø	Time at which the message has been issued by the task. Same format as specified in the logging section	1	Ø	Ø
==3	Node_Name	Ø	Workstation Name according to the logging specification	1	Ø	Ø
==3	Processor_Name	Ø	Processor Name according to the logging specification	1	Ø	Ø
==3	Processor_Version	Ø	Processor Version according to the logging specification	1	Ø	Ø
==3	PID	Ø	OS processor identifier	1	Ø	Ø
==3	Type	Ø	Format and valid values of fields are reported in section Logging	1	Ø	Ø
==3	Text	Ø	The Message Text field will be in the format of a CDATA xml tag, in order to allow any character to be included in the	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			message without xml parsers and schema driven editors complain, should special characters like '<' and '>' be used inside the text			
1	Job Order	Ø	This tag will include the job order used for generating the product as CDATA xml tag.	1	Ø	Ø

4.6.4 TRANSPORT MECHANISM

The file shall be available in the processing directory, where it shall be generated by a dedicated processor task that must be executed after all other processor's tasks (this is achieved by using a specific task order into the task table). The product report generator task shall filter and translate the Log Message file (see next section 4.4).

4.6.5 FREQUENCY / CONDITIONS FOR TRANSFER

The file is not generated if one tasks executed before the report generator failed. In this case, the Log Message File will be available for troubleshooting.

4.7 Product List

4.7.1 PURPOSE

This interface is of type: **Non-compulsory**.
This file contains the list of product files that shall be inventoried by the PDGS.

4.7.2 TYPE

ASCII FILE-BASED

4.7.3 FORMAT

The file is composed by a list of plain filenames, one line per filename.

4.7.4 TRANSPORT MECHANISM

The file is available in the processing directory, where it has been generated by one of the tasks in the processor task-table (a dedicated pre-inventory task running after all other tasks is the preferred way).

4.7.5 FREQUENCY / CONDITIONS FOR TRANSFER

- This file is generated whenever an IPF accomplishes its job successfully.
- The file is not generated in case one of the tasks that have to be executed before the pre-inventory generator fails.
- Product files not listed in this file will not be inventoried by the PDGS.
- At each file name in the list shall correspond a metadata file (see previous section 4.6)
- The name of this file shall be in the format
  <order_id>.LIST,
  where order_id is extracted from the Job Order filename
- In case the file is not generated, the Management Layer will build it with all metadata files it finds in the working dir after the processor completion

4.8 Product Metadata

4.8.1 PURPOSE

This interface is of type: **Non-compulsory**.
This file contains the data needed by the PDGS to properly fill-up database tables upon inventory of a product.

4.8.2 TYPE

This interface is implemented by means of an XML file.

4.8.3 FORMAT

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
>	Level1_Inventory_Metadata	All	Ø	1	Ø	Ø
1	File_Name	Ø	Name of the File to be inventoried (string)	1	Ø	Ø
1	File_Version	Ø	Version of the file to be inventoried (string)	1	Ø	Ø
1	Source	Ø	Processor name passed by the PDGS Management Layer inside the Job Order (string)	1	Ø	Ø
1	Source_Sw_Version	Ø	Processor version passed by the PDGS Management Layer inside the Job Order (string) **NOTE**: Source and source version are the processor name and processor version passed by the PDGS Management Layer inside the Job Order	1	Ø	Ø
1	Generation_Time	Ø	Times are UTC, and must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**	1	Ø	Ø
1	Validity_Start	Ø	Times are UTC, and	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**			
1	Validity_Stop	Ø	Times are UTC, and must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**	1	Ø	Ø
1	Start_Orbit_Number	Ø	*	1	Ø	Ø
1	Stop_Orbit_Number	Ø	*	1	Ø	Ø
1	Geographic_Localization	Time_Range ∩ List_of_Geo_Pnt	Ø	1	Ø	Ø
=2	Time_Range	Start_Time ∩ Stop_Time	Ø	1	Ø	Ø
==3	Start_Time	Ø	Times are UTC, and must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**	1	Ø	Ø
==3	Stop_Time	Ø	Times are UTC, and must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=2	List_Of_Geo_Pnt	Geo_Pnt **NOTE**: This section will contain at least two points mapping on the start and stop longitude and latitude. Other optional intermediate points can be computed on a regular time interval base (i.e., one point every 30 seconds of product). The interval between the last computed point and the stop point can be not accurate.	Ø	1	count	[2,*]
==3	Geo_Pnt	Latitude ∩ Longitude Time	Ø	2	Ø	Ø
===4	Latitude	Ø		1	Ø	Ø
===4	Longitude	Ø		1	Ø	Ø
===4	Time	Ø	Times are UTC, and must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**	1	Ø	Ø
1	Quality_Info	Ø	verbose description : default " "	1	Ø	Ø
1	Validity_Flag	Ø	***true*** U false	1	Ø	Ø
1	Validation_Date	Ø	Times are UTC, and must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			NOTE: The PDGS has the capability to store files which have been generated/acquired but have not yet been recognised as officially usable (i.e., they are waiting some sort of "imprinting" before being used), as well as the capability to "discontinue" file usage but still keep them for historical usage. In order for a file to be used, it need its validity flag to be true, and the validation date, accordingly, set to the date upon which the flag has been set to true. In case of processor generated products, however, these fields are directly set by the processor to true and to the product generation date (i.e., it is assumed that a product generated by an "official" processor does not need further validation before being used)			
1	Header_Size	Ø	*	1	Ø	Ø
1	Data_Size	Ø	*	1	Ø	Ø
1	File_Type	Ø	Type of the file to be inventoried (string)	1	Ø	Ø
1	File_Class	Ø	Class of the file to be inventoried (string)	1	Ø	Ø
1	Sensor_Id	Ø	TBD string	1	Ø	Ø
1	Acquisition_Station_Id	Ø	TBD string	1	Ø	Ø
1	Processing_Station_Id	Ø	TBD string	1	Ø	Ø
1	Sensor_Mode	Ø	TBD string	1	Ø	Ø
1	Phase	Ø	[A,W] U ***X*** U [Y,Z] U	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
			[0,9] **NOTE**: If not used, the Phase value shall be set to 'X'			
1	Satellite_Id	Ø	TBD string	1	Ø	Ø
1	Ascending_Flag	Ø	A U D	1	Ø	Ø
1	Equator_X_Longitude	Ø	TBD	1	Ø	Ø
1	Equator_X_Time	Ø	Times are UTC, and must follow the format: **UTC=yyyy-MM-ddThh:mm:ss**	1	Ø	Ø
1	Order_Id	Ø	* **NOTE**: The order_id is the one retrieved from the Job Order filename: it uniquely identifies the production order for which the product was generated	1	Ø	Ø

4.8.4 TRANSPORT MECHANISM

The file is available in the processing directory, where it has been generated by one of the tasks in the processor task-table (a dedicated pre-inventory task running after all other tasks is the preferred way).

4.8.5 FREQUENCY / CONDITIONS FOR TRANSFER

- This file is generated whenever an IPF accomplishes its job successfully.
- The file is not generated in case one of the tasks that has to be executed before the pre-inventory generator fails.
- There must be one such file for each generated product
- The name of this file shall be obtained by the product name substituting the product extension (or adding, if the product has no extension) with the extension .MTD
- **Generation of this file can be omitted in case the generated products obey to a standard in which the non-variable section of the product header contains all the information specified by previous section 4.7.3.** (In this case, the PDGS would still be able to inventory the products without having to know the internals of the product specification)

IPF Processing Configuration Parameters

4.8.6 PURPOSE

This XML configuration file contains the settings controlling processing options and/or algorithm performance. Typical examples of these switches are:

- Processing options
- Thresholds
- Breakpoint activation

4.8.7 TYPE

This interface is of type: **Recommended**.
This interface is an IPF private one, therefore the format choice is left to the IPF implementor, anyway the usage of XML syntax is recommended.

4.8.8 FORMAT

The Format of this file is mission specific.
However, since this interface has been widely implemented in existing systems, some guidelines are recommended in the following.

The Processing Configuration File can be arranged to contain as many named sections, as the processor-to-task decomposition requires. In general, there should exist one section for each processor task. Nevertheless, if several tasks share the same setting, then these settings can be grouped into a single section.
The most general layout, in case of usage of an XML file type, could look like in the following:

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
>	Ipf1_Conf	All	∅	1	∅	∅
1	Common	TBD	Set of Common Settings	1	∅	∅
1	Task*n*	TBD	Set of Setting for Task *n*			

4.8.9 TRANSPORT MECHANISM

The Management Layer copies the Processing Configuration File from the processor installation directory (specified in the Task Table) into the working directory.
The complete filename (including path) of the file is passed by the Job Order.

4.8.10 FREQUENCY / CONDITIONS FOR TRANSFER

This file is stored on the workstation as part of the processor installation in the PDGS.
More versions of the file may exist. Each version file shall be reported in the processor task table.

APPENDIX A: QUICK REFERENCE GUIDE

This section has been written with a twofold purpose:

- To provide the reader with a quick guess about the meaning of the most used IPF terms
- To give to the reader some insight on the complexity of some common operations to be carried out in the system.

None of the contents of this section is part of the specification.

For a rigorous definition of the terms hereafter reported the reader should refer to section 4 or to specific mission documentation.

Contents of this reference are:

What is … ?

The PF
The Processing Order
The Management Layer
The Processor
The PROCESSING CONFIGURATION file
The IPF CONFIGURATION file
The TASK TABLE file
The JOB ORDER

What is the PF?

The PF is that architectural part of the PDGS that consists of a Management Layer and set of Processors (IPF).
The Management Layer interfaces the Processors on one side and the PDGS on the other one. All the interfaces are file-based.

What is the Processing Order?

The Processing Order is the main command interface between the PDGS and the PF.
Processing Orders generated by the PDGS are stacked into a queue that is regularly polled by the Management Layer. When the Management Layer recognises that a specific Processing Order can be fulfilled then it starts an instance of the required Processor

What is the Management Layer?

The Management Layer is the architectural element of the PF that fulfils the PDGS Processing Orders by running a specific instance of a Processor.

What is the Processor?

An IPF Processor is:

- A collection of executables. Each Executable is named: Task
- The specification of the Task execution order
- The specification of the Task input and output

Processors are started and controlled by the Management Layer.

What is the PROCESSING CONFIGURATION file?

The PROCESSING CONFIGURATION file is a Recommended Interface, i.e. it is not part of the general architecture.
It is a configuration file local to the processor that contains the settings controlling processing options and/or performance.

What is the IPF WS CONFIGURATION file?

The IPF WS CONFIGURATION file is an optional interface and it consists of a Management Layer configuration file that contains the list of the installed IPF Processors on a given workstation as well as all the details needed to the Management Layer to take the decision if a given Order can be fulfilled by each listed IPF Processor or not.

What is the TASK TABLE file?

The TASK TABLE is a Management Layer configuration file that specifies the composition of a Processor, in terms of:

- number of composing executables (tasks) and their execution order,
- list of input, output and intermediate file types for each executable.

What is the JOB ORDER file?

The JOB ORDER file is the main interface between the Management Layer and a Processor.

The JOB ORDER contains the list of input/output files each Task of a Processor is supposed to use/produce.

The Management Layer passes on the command line the JOB ORDER to each Task of the Processor.

APPENDIX B: MANAGEMENT LAYER INPUT RETRIEVAL POLICIES

The Management Layer has to provide queries operating on database inventory tables.
Each query maps to a particular retrieval mode an input file in the task table. The retrieval modes are defined by the IPF according to the files selection rules it requests for the processing
The Management Layer is prevented from performing any operation on the retrieval modes as read-in from the Task Tables; in other words, the retrieval mode written in the Task Table has to be straight used in the query without any manipulation by the Management Layer. Any deviation from this simple mechanism would force the Management Layer to loose its generality, making that Management Layer implementation unacceptable.

Each retrieval mode available in the currently existing PDGS' refers to a specified time interval, that is usually the time coverage of the product to be processed, say t0 – t1.
This time interval can be enlarged with programmable delta intervals on the left (say dt_0) and on the right (say dt_1).

Retrieval modes based on different criteria (e.g. latitude and longitude) are – of course – possible and they should be decided according to specific mission requirements. However, the general and simple rule that the Management Layer simply uses the retrieval mode as it is read from the Task Table has to be always followed.

The currently available retrieval modes are:

ValCover	This mode gets all files that cover entirely time interval $[t_0 - dt_0 , t_1 + dt_1]$. Using this query in the scenario exhibited in fig B-1, it will return records R_2 and R_3
LatestValCover	This mode gets the latest file that covers entirely time interval $[t_0 - dt_0 , t_1 + dt_1]$. The latest record is the one with the more recent Generation Date. Using this query in the scenario exhibited in fig B-1, it will return record R_3
ValIntersect	This mode gets all files that cover partly time interval $[t_0 - dt_0 , t_1 + dt_1]$. Using this query in the scenario exhibited in fig B-1, it will return records R_1, R_2, R_3 and R_4.
LatestValIntersect	This mode gets the latest file that covers partly time interval $[t_0 - dt_0 , t_1 + dt_1]$. The latest record is the one with the more recent Generation Date. Using this query in the scenario exhibited in fig B-1, it will return record R4
LatestValidityClosest	This mode gets the latest file which is nearest to $((t_0–dt_0)+(t_1+dt_1))/2$. Using this query in the scenario exhibited in fig B-1,

	it will return record R_4
BestCenteredCover	This mode gets the latest file which covers entirely time interval $[t_0 - dt_0 , t_1 + dt_1]$, and for which is maximized the minimum distance of his extremes from the time interval borders. That is, if we name A and B the left and right endpoint of the file validity interval, the selected file is the one corresponding to $\mathbf{max_i(min(A_i - (t_0 - dt_0), B_i - (t_1 + dt_1))}$. Using this query in the scenario exhibited in fig B-1, it will return record R_3.
LatestValCoverClosest	This mode gets the file that : ❑ covers entirely time interval interval $[t_0 - dt_0 , t_1 + dt_1]$ and ❑ has got the start time closest to to-dt0. Basically the outcomes of this mode is the same as the following sequence is applied: ❑ Get files with "ValCover" mode ❑ Among the returned files select the one with start time closes to to-dt0. In Figure B-1 this would be product R2. If there are several files with the same start time choose the most recent ingestion time
LargestOverlap	This mode gets the file (only one) that satisfies both the following conditions: ❑ covers entirely time interval interval $[t_0 - dt_0 , t_1 + dt_1]$ ❑ has got the largest overlap. Basically the outcomes of this mode is the same as the following sequence is applied: ❑ Get files with "ValCover" mode ❑ Among the returned files select the one with the largest overlap. If there are several products with the same overlap (e.g. full coverage), the product with the start time that is closest to TO-TO is chosen. Note that in the full coverage case the result is identical to "ValCoverClosest".
LargestOverlap85	This mode is the same as LargestOverlap but only products with at least 85% coverage of the query interval are selected.
LatestValidity	This mode gets a product with the latest Validity Start Time. In Figure B-1 this would be product R6.
LatestValCoverNewestValidity	This mode applies first "LatestValCover". If no file is returned

	then it applies "Latest Validity"

Where:

. **t0** and **t1** are the values set into the "**Start**" and "**Stop**" tags

dt0, **dt1** are the values set into the **T0, T1** tags of the Task Table

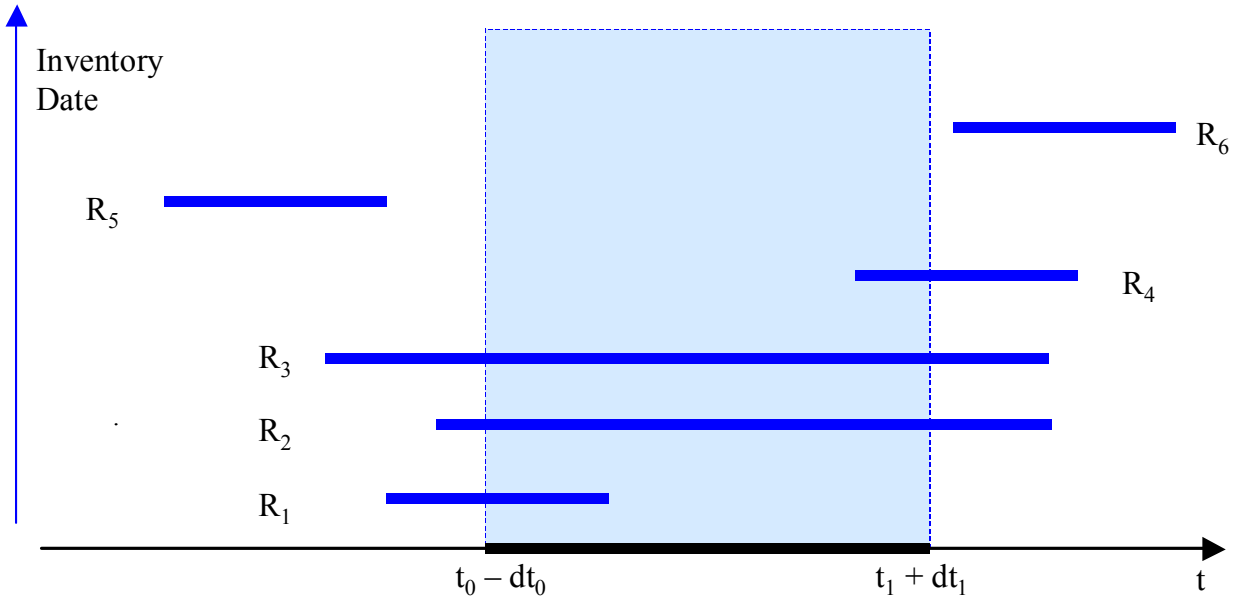


Figure B-1

Further retrieval policies can be added, however, in order to avoid confusion, the names of the retrieval policies described so far should be kept untouched either in the case they are used as they are and in case they are used as basic element for the definition of new policies (e.g. the LatestValCover and LatestValidity modes in the definition of the LatestValCoverNewestValidity Methods).

APPENDIX D: TASK TABLE CONTENTS

This appendix presents an example of how a general Task Table file looks like:

```
<Task_Table xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" xsi:noNamespaceSchemaLocation="TaskTable.xsd">
        <Processor_Name>DUMMY_PROCESSOR</Processor_Name>
        <Version>01.00</Version>
        <Test>No</Test>
        <Min_Disk_Space units="MB">10000</Min_Disk_Space>
        <Max_Time units="sec">0</Max_Time>
                <List_of_Cfg_Files >
                        < Cfg_Files>
                                < Version >1.3< /Version >
                                <File_Name>dummy.cfg </File_Name>
                        </Cfg_Files>
                </List_of_Cfg_Files>
        <List_of_Dyn_ProcParam count=”0>
        </List_of_Dyn_ProcParam >
        <List_of_Pools count="3">
                <Pool>
                        <Detached>false</Detached>
                        <Killing_Signal>15</Killing_Signal>
                        <List_of_Tasks count="1">
                                <Task>
                                        <Name>DUMMYTASK1</Name>
                                        <Version>1.0</Version>
                                        <Critical>true</Critical>
```

```
<Criticality_Level>1</Criticality_Level>
<File_Name>/TlStandAloneWorkArea/processors/DummyProcessor/DummyTask1</File_Name>
<List_of_Inputs count="1">
        <Input>
                <Mode>ALWAYS</Mode>
                <Mandatory>Yes</Mandatory>
                <List_of_Alternatives count="1">
                        <Alternative>
                                <Order>0</Order>
                                <Origin>DB</Origin>
                                <Retrieval_Mode>ValIntersect</Retrieval_Mode>
                                <T0>0</T0>
                                <T1>0</T1>
                                <File_Type>FILETYPE1</File_Type>
                                <File_Name_Type>Physical</File_Name_Type>
                        </Alternative>
                </List_of_Alternatives>
        </Input>
</List_of_Inputs>
<List_of_Outputs count="2">
        <Output>
                <Destination>DBPROC</Destination>
                <Mandatory>Yes</Mandatory>
                <File_Type>FILETYPE2</File_Type>
                <File_Name_Type>Regexp</File_Name_Type>
        </Output>
        <Output>
                <Destination>PROC</Destination>
```

```
                                        <Mandatory>No</Mandatory>
                                        <File_Type>INTERIM</File_Type>
                                        <File_Name_Type>Stem</File_Name_Type>
                                    </Output>
                                </List_of_Outputs>
                                <List_of_Breakpoints count="0"/>
                            </Task>
                        </List_of_Tasks>
                    </Pool>
                    <Pool>
                        <Detached>false</Detached>
                        <Killing_Signal>15</Killing_Signal>
                        <List_of_Tasks count="1">
                            <Task>
                                <Name>DUMMYTASK2</Name>
                                <Version>1.0</Version>
                                <Critical>true</Critical>
                                <Criticality_Level>1</Criticality_Level>
                                <File_Name>/TlStandAloneWorkArea/processors/DummyProcessor/DummyTask2</File_Name>
                                <List_of_Inputs count="3">
                                    <Input id="FileType2">
                                        <Mode>ALWAYS</Mode>
                                        <Mandatory>Yes</Mandatory>
                                        <List_of_Alternatives count="1">
                                            <Alternative>
                                                <Order>0</Order>
                                                <Origin>PROC</Origin>
                                                <Retrieval_Mode>ValCover</Retrieval_Mode>
```

```
                        <T0>0</T0>
                        <T1>0</T1>
                        <File_Type>FILETYPE2</File_Type>
                        <File_Name_Type>Regexp</File_Name_Type>
                    </Alternative>
                </List_of_Alternatives>
            </Input>
            <Input>
                <Mode>ALWAYS</Mode>
                <Mandatory>Yes</Mandatory>
                <List_of_Alternatives count="1">
                    <Alternative>
                        <Order>0</Order>
                        <Origin>PROC</Origin>
                        <Retrieval_Mode>ValCover</Retrieval_Mode>
                        <T0>0</T0>
                        <T1>0</T1>
                        <File_Type>INTERIM</File_Type>
                        <File_Name_Type>Stem</File_Name_Type>
                    </Alternative>
                </List_of_Alternatives>
            </Input>
            <Input id="TLMX">
                <Mode>ALWAYS</Mode>
                <Mandatory>Yes</Mandatory>
                <List_of_Alternatives count="1">
                    <Alternative>
                        <Order>0</Order>
```

```
                                                <Origin>DB</Origin>
                                                <Retrieval_Mode>ValCover</Retrieval_Mode>
                                                <T0>0</T0>
                                                <T1>0</T1>
                                                <File_Type>TLMX__VC1</File_Type>
                                                <File_Name_Type>Physical</File_Name_Type>
                                        </Alternative>
                                </List_of_Alternatives>
                        </Input>
                </List_of_Inputs>
                <List_of_Outputs count="1">
                        <Output>
                                <Destination>DB</Destination>
                                <Mandatory>No</Mandatory>
                                <File_Type>OUTFILETYPE</File_Type>
                                <File_Name_Type>Directory</File_Name_Type>
                        </Output>
                </List_of_Outputs>
                <List_of_Breakpoints count="0"/>
        </Task>
    </List_of_Tasks>
</Pool>
<Pool>
        <Detached>false</Detached>
        <Killing_Signal>15</Killing_Signal>
        <List_of_Tasks count="1">
                <Task>
                        <Name>DUMMYTASK3</Name>
```

```
                <Version>1.0</Version>
                <Critical>true</Critical>
                <Criticality_Level>1</Criticality_Level>
                <File_Name>/TlStandAloneWorkArea/processors/DummyProcessor/DummyTask1</File_Name>
                <List_of_Inputs count="2">
                    <Input ref="FileType2">
                        <Mode>ALWAYS</Mode>
                        <Mandatory>Yes</Mandatory>
                    </Input>
                    <Input ref="TLMX">
                        <Mode>ALWAYS</Mode>
                        <Mandatory>Yes</Mandatory>
                    </Input>
                </List_of_Inputs>
                <List_of_Outputs count="0"/>
                <List_of_Breakpoints count="0"/>
            </Task>
        </List_of_Tasks>
    </Pool>
</List_of_Pools>
</Task_Table>
```

APPENDIX E: CONFIGURATION SPACES

This section contains an example of how to specify contents of other possible configuration space to be provided to Processors by the Management Layer

Constant Parameters

The chosen example is taken from Cryosat, and is the Geophysical_Constans configuration space, containing the unique definition of the value for all the constants to be used by Cryosat Processors.

PURPOSE

This file contains the value of all scientific constants used by any IPF processors.

TYPE

This interface is implemented by means of an XML file.

FORMAT

Next table shows a tentative layout of the constant file, based on the constant file presently used in the CryoSat's PDGS.
The value of each used constant has to be frozen during

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
>	Ipf_Const	All	Ø	1	Ø	Ø
1	PI	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Pi greek	1	Ø	Ø
=2	Value	Ø	3.141592653589793238	1	unit	unitless
1	Deg_2_Rad	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	PI/180	1	Ø	Ø
=2	Value	Ø	0.017453292514387740	1	unit	unitless
1	C	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Speed of light in vacuum	1	Ø	Ø
=2	Value	Ø	2.99792458e8	1	unit	m/s
1	K	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Boltzmann Constant	1	Ø	Ø
=2	Value	Ø	1.380662e-23	1	unit	J/K
1	W	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Angular speed of Earth rotation	1	Ø	Ø
=2	Value	Ø	7.29211585e-5	1	unit	rad s^-1
1	Grav_Accl_Mu	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Geocentric Gravitational Constant = g * mass of Earth	1	Ø	Ø
=2	Value	Ø	3.9860044e14	1	unit	m^3 s^-2
1	J2	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Zonal harm coef of Earth grav potential	1	Ø	Ø
=2	Value	Ø	1082.65e-16	1	unit	unitless
1	J3	Description ∩ Value	Ø	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=2	Description	Ø	Zonal harm coef of Earth grav potential	1	Ø	Ø
=2	Value	Ø	-2.543e-6	1	unit	unitless
1	J4	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Zonal harm coef of Earth grav potential	1	Ø	Ø
=2	Value	Ø	-1.6715e-6	1	unit	unitless
1	Ellipsoid	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Name of reference Ellipsoid	1	Ø	Ø
=2	Value	Ø	WBS84	1	unit	unitless
1	R_Equator	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	semi-major axis of reference ellipsoid	1	Ø	Ø
=2	Value	Ø	6378137.000	1	unit	m
1	R_pole	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	semi-minor axis of reference ellipsoid	1	Ø	Ø
=2	Value	Ø	6356752.314	1	unit	m
1	Flattening	Description ∩ Value	Ø	1	Ø	Ø
=2	Description	Ø	Flattening of reference ellipsoid	1	Ø	Ø
=2	Value	Ø	0.00335281066	1	unit	unitless
1	R_Mean	Description ∩ Value	∅	1	Ø	Ø
=2	Description	Ø	Mean Earth Radius	1	Ø	Ø
=2	Value	Ø	6367444.657	1	unit	m
1	H_mean	Description ∩ Value	∅	1	Ø	Ø

Level	Tag name	Descendants	Tag Contents	Cardinality	Attribute name	Attribute domain
=2	Description	Ø	Mean Altitude	1	Ø	Ø
=2	Value	Ø	730000	1	unit	m
1	Sec_InDay	Description ∩ Value	∅	1	Ø	Ø
=2	Description	Ø	Seconds in a day	1	Ø	Ø
=2	Value	Ø	86400	1	unit	s

TRANSPORT MECHANISM

The Management Layer pushes this file onto the processing directory.

The Management Layer keeps a local copy of the Configuration files that are last downloaded. If for any reason the connection with the database is not available at retrieval time, this local copy is used.

FREQUENCY / CONDITIONS FOR TRANSFER

The file is pushed on the processing directory together with the input files, before starting the processing job. The last used copy is saved by the Management Layer, and will be reused if the communication with the Archive is for any reason interrupted or impossible.
Note that for testing purposes some Processors could need different settings of some of the constants values. In this case Processors are expected to be locally configured to use a different file. However, in this case, the alternate constant file shall have the same format specified in this section for the interface. In other words, only the constant settings may change, not the file format.

APPENDIX G: THE POOLS

The concept of Pool introduces the parallelism inside the Processor, i.e. the possibility to run several Tasks of a Processor at the same time.
In particular, the Management Layer executes in parallel the Tasks belonging to the same Pool.

Two kinds of Pools do exist: non-detached (foreground execution) and detached (background execution).

Non-detached Pools are run in sequence, according to the order specified in the Processor's Task Table.

Detached Pools can be run in parallel with other detached Pools and/or with at most one non-detached Pool.
Detached Pools provide reduced monitoring capabilities because the Management Layer doesn't intercept neither stdout nor stderr descriptors of background tasks.
The Management Layer is configurable on the behavior to follow upon last foreground task termination. In particular, it can be configured to wait for the background tasks to terminate or to force their exit without waiting. In this last case, the exit code of the processor is not affected.

In order to better explain how the Pool works, three basic examples are provided in the following.
Each example has been extrapolated from a realistic mission scenario, even if none of them refers to a specific mission.

First we will present the case occurring in most of the practical cases: the Tasks of a Processor are executed in sequence, i.e. at a given time only one Task is running in the IPF. In this case each Task is contained into a non-detached Pool

Next we will analyse a Processor with a two-tasks Pool.

Last, we will describe a Processor with several non-detached Pools plus one detached-Pool.

Case 1: the mono-task Pool

A very simple, yet very frequent case is represented by a Processor built up of a certain number of tasks to be executed in sequence.
Most of the level 1 Processors are of this type, and then let's analyze this specific case.
Let's consider a Level1 processor consisting of three tasks:

- A Pre-Processor Task, i.e. a Task that is supposed to decode and check the Level 0 product contents
- A Computer Task, i.e. a Task that implements the Level1 algorithms onto the Level0 data
- A Post-Processor Task, i.e. a Task that formats the Level1 data as computed by the computer task into a Level1 product.

In this case we will have a Task table containing three non-detached Pools with one Task each, i.e:

```
<List_of_Pools count="3">
        <Pool>
                <Detached>false</Detached>
                <Killing_Signal>15</Killing_Signal>
                <List_of_Tasks count="1">
                        <Task>
                                <Name>PreProcessor</Name>
                                …
                        </Task>
                </List_of_Tasks>
        </Pool>
        <Pool>
                <Detached>false</Detached>
                <Killing_Signal>15</Killing_Signal>
                <List_of_Tasks count="1">
                        <Task>
                                <Name>Processor</Name>
                                …
                        </Task>
                </List_of_Tasks>
        </Pool>
        <Pool>
                <Detached>false</Detached>
                <Killing_Signal>15</Killing_Signal>
                <List_of_Tasks count="1">
                        <Task>
                                <Name>PostProcessor</Name>
                                …
                        </Task>
                </List_of_Tasks>
        </Pool>
</List_of_Pools>
```

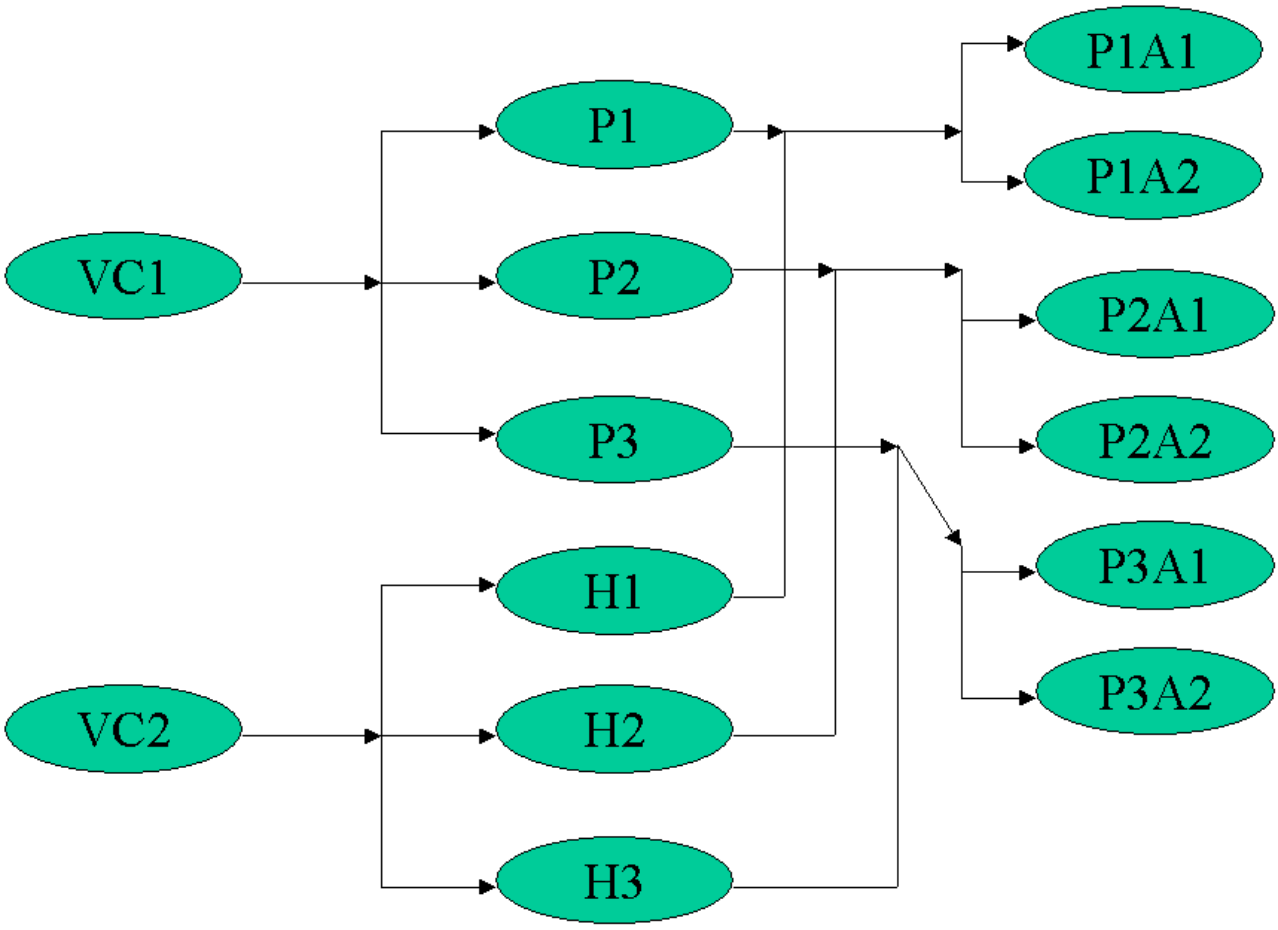
Case 2: the general non-detached Pool

In order to explain this case, let suppose a mission scenario where a satellite embarks several instruments and each of them can be operated in several modes.
Instrument Source Packets (ISPs) relevant to the same instrument are tagged with the same APID whereas the operative mode of each instrument is coded into the housekeeping telemetry. Housekeeping packets are tagged with a set of APIDs different from the instrument APID set.
Instrument data and housekeeping data are downloaded at each pass over the ground station on different virtual channels, let say them VC1 (instruments) and VC2 (housekeeping)

The breakdown strategy of the telemetry into Level 0 products foresees that each level 0 has to contain ISPs relevant to both a specified instrument and a well-defined mode.
To achieve this purpose we could organise the level 0 generation process in two steps:

- First, the Instrument Source Packets with the same APID are grouped into the same product (i.e. one product for each instrument and one product for each housekeeping telemetry type).
- Second, housekeeping info are used to generate level 0 products containing ISP relevant to a specific instrument and operated in a specific mode (e.g. a Level 0 shall contain data relevant to the Star Tracker when operated in nominal mode).

The sketch of the possible product flow is sketched in the following



In particular, at the first step the three products: P1, P2 and P3 are generated from VC1 whilst from VC2 the three products H1 H2 and H3 are derived.
At the second step we combine P1 and H1 data to get the two products P1A1 (Product 1-Activity 1) and P1A2 (Product 1-Activity 2) and we do the same for the other couples pf Px and Hx products.

The implementation of this Level 0 processor could straightly follow the product flow: the first Pool contains two Tasks (VC1_Processor and VC2_Processor) that in parallel generate the Px and the Hx products and then the second Pool contains a Post Processor that combines a couple Px and Hx products to get PxAy products.
The Task Table skeleton will look like in the following:

```
<List_of_Pools count="2">
    <Pool>
        <Detached>false</Detached>
        <Killing_Signal>15</Killing_Signal>
        <List_of_Tasks count="2">
            <Task>
                <Name>VC1_Processor </Name>
                …
            </Task>
            <Task>
                <Name>VC2_Processor </Name>
                …
            </Task>
        </List_of_Tasks>
    </Pool>
    <Pool>
        <Detached>false</Detached>
        <Killing_Signal>15</Killing_Signal>
        <List_of_Tasks count="1">
            <Task>
                <Name>PostProcessor</Name>
                …
            </Task>
        </List_of_Tasks>
    </Pool>
</List_of_Pools>
```

Case 3: the detached Pool

In order to explain this case, let suppose there is a PDGS facility that acquires telemetry data in real time and stores them on a disk. Data acquisition is triggered by a particular event (typically a watch-dog based mechanism based on the mission plan), however statistical data (e.g. histograms of values) have to be collected over all the acquisition phases.
A possible approach to solve this application problem is to implement the complete acquisition chain by means of non-detached Pools whereas the “Statistical” Task could be part of a detached Pool.

In this case the foreground Tasks are supposed to pass information to the background tasks by means of dedicated interfaces internal to the processor.
After the last non-detached Pool has been executed the Management Layer can either wait for the background Task to accomplish its execution or force the background task to exit.
The strategy is chosen by properly configuring for the Management Layer.

APPENDIX H: EXPECTED CONTENTS OF AN IPF DELIVERY

The minimal composition of an IPF package shall be:

1. self installing package, containing:
    a. processor task executables
    b. processor configuration files
    c. one task table for each processor, with full physical pathnames pointing to post-installation files (executables, configuration files, processing parameters, I/O products…)
2. Software release notes, as per applicable project standards
3. installation checkout (test) software and data, as per applicable project standards/requirements