Sagem Monetel
SAFRAN Group

Référence / Reference : SMO/SPE-0299 Révision / *Revision* : A

Titre /*Title* : **Parameter management in export Telium software**

Programme / *Subject* : TELIUM

Approbation de la révision / *Revision Approval* : **A**		
	Nom ***Name***	**Fonction** ***Function***
Etabli par : *Written by:*	Wilfrid Marsal	Software Engineer
Vérifié ou Approuvé par : *Checked or approved by:*	Martine Aizac Matthieu Gatesoupe	SDK Project Manager SDK Project Manager
Autorisé par : *Authorized by:*	Christophe Barthelemy	SDK Product leader

Révision *Issue*	**Date de validité / d'application** *Validity/application date*	**Nb de pages** *Nb of pages*	**Nb de pages annexes** *Nb of appendices*	**Objet et description de la modification** *Object and description of modification*
A	February 2008			Initial

TABLE OF CONTENT

1. INTRODUCTION

The parameters management provide some functions to manipulate tags used in Telium product. These tags allow having several product configurations. The parameter management allows to select or to switch between several set of parameters.

This management is implemented on the TLV_Tree data organization. It allows to store and to dynamically organize data in the shape of a tree fully allocated in the dynamic memory RAM. (See ref[1] : the SMO/SFO-0083 TLV Tree Reference Manual for more explanations)

The aim of the new parameter management is to:

- Have a dynamic parameter management (and not a fixed list of parameters),
- Have several sets of parameters usable by the application.

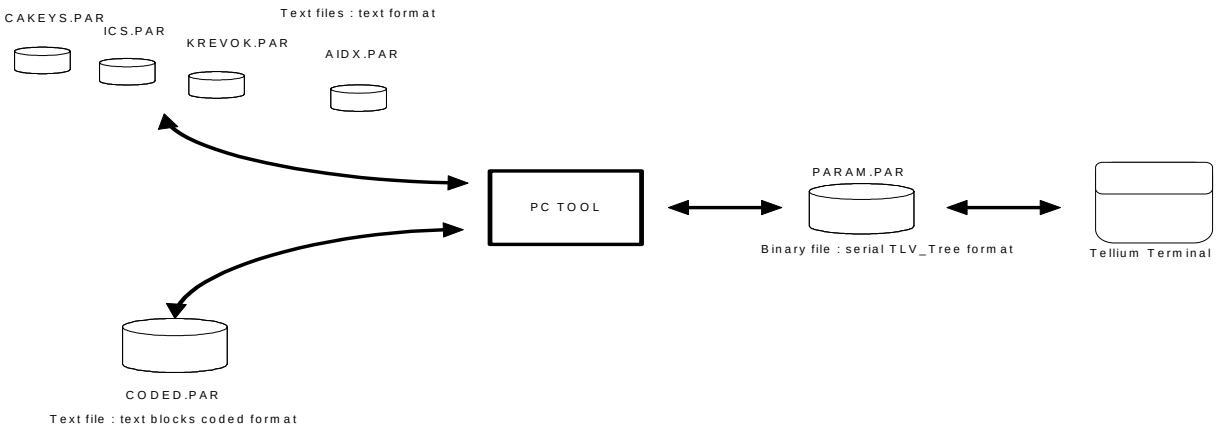
The modification impacts the cu_Param.c, cu_Base.c, cu_entry.c custom C files.
The parameters include in AIDX.PAR, CAKEYS.PAR, ICS.PAR, and KREVOK.PAR file are now managed by a PARAM.PAR file having an open structure organization.
The PARAM.Par file organisation is not dedicated to a specific application.

Warning: The Blackl.par file is not impacted by this modification.

Compatibility with previous software versions: Terminal interface continue to manage text files (AIDX.PAR, CAKEYS.PAR, ICS.PAR). Theses data are included in the TLV_Tree management. The reader interface manager is included by the cu_param.c file.

Coding parameter tool:
The PC “parameter” tool allows the file transformation of one or several text files (ics, krevok…) in a serial TLV_Tree format param.par file. This tool allows the reverse operation for control.

Input text file format:		Output binary file format:
Text files with tag list and values	⇔	Serial TLVTree format file.
Text files with block format tag list and values	⇔	Serial TLVTree format file.



Terminal software constraints:
The parameter tags are usable with a global set management. Each mark (or set of tags) is separately usable by the terminal application.

The param.par file may be built made by a PC software tool. No modification with a text editor may be directly made on this parameter file.

Tests/verifications:
The EMV parameters checksum will allow checking new parameter management.

2. NOTATION:

A set of parameters is usable by his mark activation. Software application can use one or several set of parameter simultaneously.
A mark is a node (see ref[1] document) with a specific data string formated as « MARQ_XXX ».

3. PRINCIPLES :

When a mark is activated, a set of tag (tags child of the selected mark) becomes usable by the application. These tags are usable moreover or in place of previous set of tags usable.

Example: in our bellow tree organisation, the 11 mark activation allows the 131 and et 132 tags usable. Theses new values and tag definitions are used instead of default values (If theses tags were defined).

With each mark activation, the selected mark list is updated.
On each tag parameter value request, we looked for tags associated to the activated marks in the TLV_Tree data read in the parameters file. If the specific tag is not founded, we look in the default TLV_Tree data.

Mark management function allows:
- TLV_Tree mark activation/deactivation (A mark activation allow to the application to known and to use all the child tags)
- Child mark activation of a main mark. This specific mark activation will be made one by one.
- Update functionality (reset, value affectation, default or parameter file recovery) for one or a list of tag specified by a mark.

As previously, the terminal software must know the tag parameters and marks management interface and it must known tags and mark organisation.

Default parameters:
Default parameters must be specified in the block mode. Each set of tags are separated from others with braces. (see example in annexe 1). The data reader will generate a default TLV_Tree.

TLVTree data organisation:

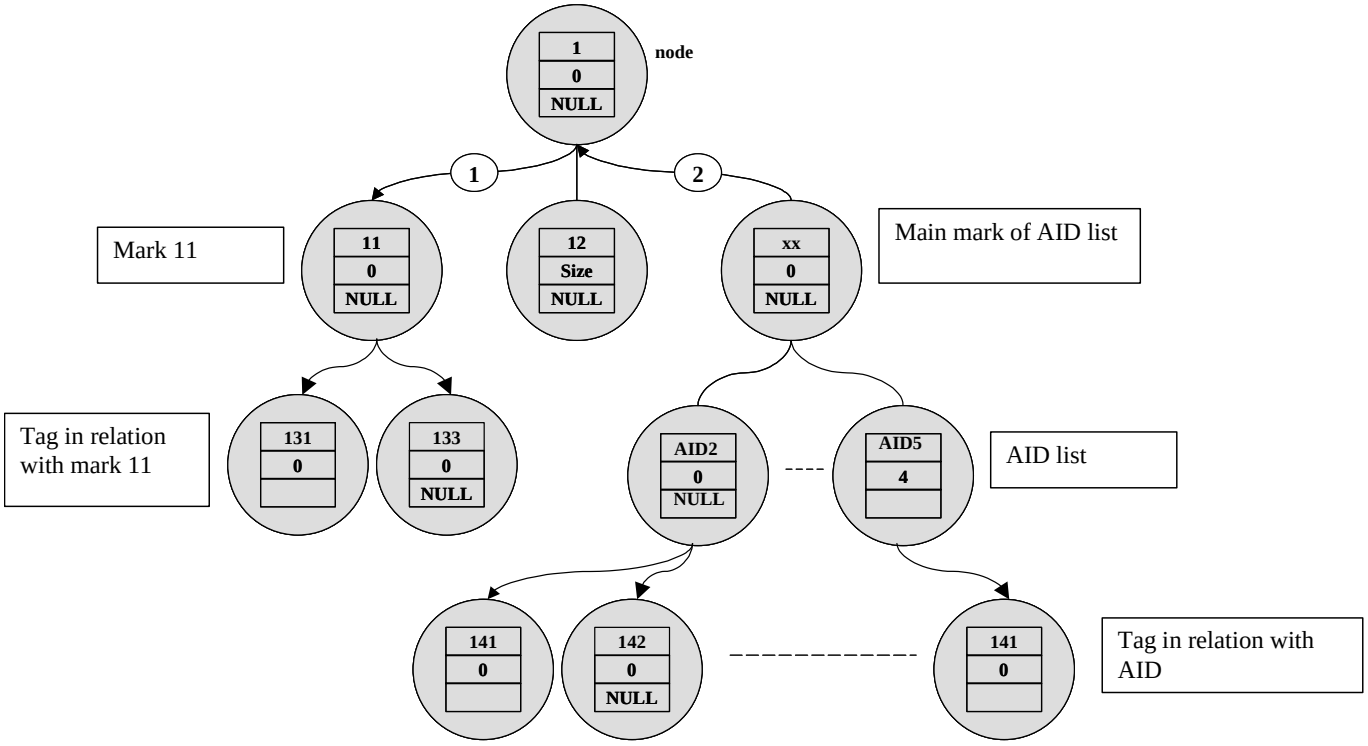
« Node root » : root of the tree and used as origin in all elements management.
« Child » : base element linked to a parent (or to the root for the higher elements).
Each element may contain some childs or datas (tags).

TLV_Tree organisation and parameter files:

Find a specific tag (or mark) or manage a list will allows to the target application to active a specific configuration.

A parameter organisation may be under the root node:

- A mark gathering the AID list and parameters in relation with,
- A mark gathering keys,
- A mark ...
- For each AID (AID1, AID2, …), a set of tag is usable when the mark in relation with this AID is selected.



Merchant functions for parameters:
Merchant can choose the terminal configuration for a specific test: éval, visa or démo (défaut). This function allows selecting the set of AIS managed by the terminal (to accept CB cards).

Default AID list :

```
07 B0 12 34 56 78 12 34
07 A0 00 00 00 04 10 10
07 A0 00 00 00 03 10 10
07 A0 00 00 00 99 90 90
07 A0 00 00 00 03 20 10
```

Eval AID list :

```
07 B0 12 34 56 78 12 34
07 A0 00 00 00 04 10 10",
05 A0 00 00 00 10
```

Visa AID list :

```
07 A0 00 00 00 03 10 10
07 A0 00 00 00 99 90 90
07 A0 00 00 00 03 20 10
07 A0 00 00 00 03 20 10
```

Main data (buffer, arrays, data organisation) :

Static variables in relation with parameter management. :

1er TLV_Tree: defaut	Defaut parameters (internal terminal data),
2nd TLV_Tree: param	Parameters read from parameters file.,
Array: activ mark list	Array used to store marks already selected for the parameters.

Marks management and speed research for a tag value:
The speed research for a tag is in relation with the quantity of data stored in the TLV-Three size to across. For our application, the data size isn't too large. It is not useful to manage a specific mechanism for accelerate the data access.

Tools available:

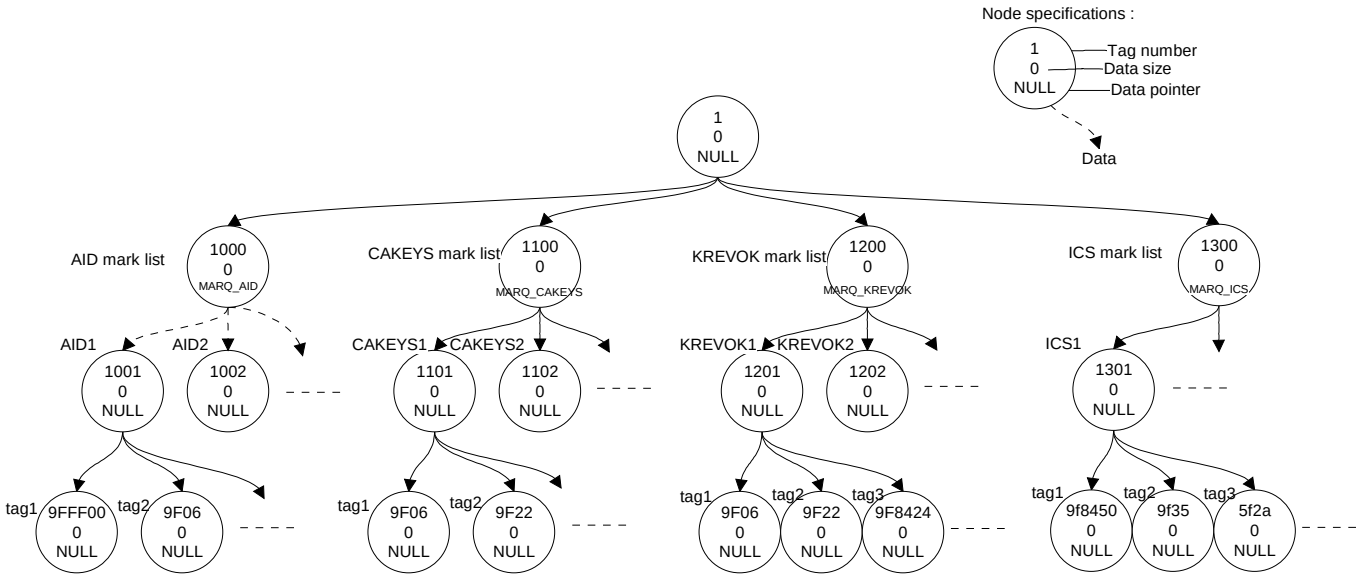
The data format used to exchange information in a file is the binary serial TLV_Tree. This format not readable by a text editor must be managed by a PC tool.

The parameter PC tool allow to :

- Perform the text parameter files (AIDX.PAR + KREVOK.PAR + CAKEYS.PAR + ICS.PAR) transformation in a binary serial TLV_Tree file.
- Perform the block text coded parameter file in a binary serial TLV_Tree file,
- Perform a binary serial TLV_Tree file transformation in one or several text file,
- Perform a binary serial TLV_Tree file transformation in a text block coded file.

4. PARAMETERS MANAGEMENT: DATA ORGANISATION

Under the root node, four main marks allow to organize the main categories of data: AID, CAKEYS, KREVOK and ICS. This structure is able to manage easily the parameters coming from the test parameter files as AIDx.PAR, ICS.PAR … Data extracted from each file must be placed under a main mark.



5. PARAMETER MANAGEMENT FUNCTIONS IN THE TERMINAL SOFTWARE

- Syntax: **TLV_TREE_NODE CUPAR_InitTree (TLV_TREE_NODE * pTree)**

Description: TLV_Tree structure initialisation. This structure allows storing parameters.
Tree TLV_Trees are usable:

- pTreeAppli : TLVTree allowing to store tags modified by the application
- pTreeDefault : TLVTree allowing to store default terminal parameters
- pTreeParam : TLVTree allowing to store parameters updated by parameter file.

Input parameters:
None.

Output parameters:
pTree : pointer on initialised TLV_Tree structure.

- Syntax: **TLV_TREE_NODE ReadArray(char *ptr_ac_x_ParamTable,**
  int i_x_ArraySize)

Description: TLV_Tree data structure initialisation with parameters read in a block descripted array. See annexe 1 for an block descripted array example..

Input parameters:
ptr_ac_x_ParamTable : array data
i_x_ArraySize : array size

Output parameters:
Pointer on TLV_Tree with data usable by interface functions for tags and marks.

- Syntax: **void CUPAR_WriteBinFile(TLV_TREE_NODE p_node,**
  char *ptr_x_fileName,
  unsigned int i_x_Mark)

Description: Update data file (binary serial TLV_Tree format) with a part of parameters included by the TLV_Tree given in parameter.

Input parameters:
p_node : pointer on TLV_Tree containing data parameters.
ptr_x_filename : file name to update : binary serial TLV_Tree format.
i_x_Mark : Parameter mark in the TLV_Tree structure. This mark is used as origin for data to update in the parameter file.

Output parameters :
None

- Syntax: **void CUPAR_WriteAllBinFile(TLV_TREE_NODE p_node,**
  char *ptr_x_fileName,
  unsigned int i_x_Mark)

Description: Update all data in a binary serial TLV_Tree format file with parameters included by the TLV_Tree given in parameter.

Input parameters:
p_node : pointer on the TLV_Tree structure including parameter data.
ptr_x_filename : file name to update : binary serial TLV_Tree format

Output parameters:
None

- Syntax: **int CUPAR_ReadBinFile(TLV_TREE_NODE p_node,**
  char *pc_x_DirFile)

Description: Read a parameter file (binary serial TLV_Tree format) and update a TLV_Tree structure.

Input parameters:
pc_x_DirFile : path and parameter file name.
p_node : pointer on TLV_Tree structure to update with data read in file..

Output parameters:
0 if no error, error code else.

- Syntax: **void CUPAR_WriteTxtFiles (TLV_TREE_NODE p_node,**
  char *ptr_x_fileName,
  unsigned int i_x_Mark,
  unsigned char c_x_tracetype)

Description: Update a text data file with parameter data included in the TLV_Tree structure. Several target file data organisations are supported: with or without marks.

Input parameters:
P_node: pointer on TLV_Tree structure containing parameters data.
Ptr_x_fileName: path and parameter file name to update.
i_x_Mark : Parameter mark in the TLV_Tree structure. This mark is used as origin for data to update in the parameter file
c_x_tracetype : parameter file update type
C_TRACE_ALL_CHILD: all tags of all children will be set in file.
C_TRACE_WITHOUT_MARK: all tags and marks will be set in file.
C_TRACE_MARK: all tags without marks will be set in file.

Output parameters:
None

- Syntax: **void PAR_ResetMarkList(void)**

Description: Actives mark reset. After this function, PAR_ReadParameter() function call cannot found a tag value updated with the parameter file or default values. Only tag updated by application may be accessed.

Input parameters:
None

Output parameters:
None

- Syntax: **int PAR_IsAMark(TLV_TREE_NODE pMark)**

Description: Specify if the parameter pointed by the parameter is a mark.

Input parameters:
pMark : node pointer

Output parameters:
Tag number in relation with the node given, or 0 if the parameter given isn't a pointer on a mark.

- Syntax: **TLV_TREE_NODE PAR_FoundMarkInTree(TLV_TREE_NODE pTLVTree, unsigned int i_x_NumMark)**

Description: Look for a mark in a TLV_Tree structure.

Input parameters:
pTLVTree : pointer on a TLV_Tree structure.
I_x_NumMark : mark number to look for.

Output parameters:
Pointer on mark founded, 0 else.

- Syntax: **TLV_TREE_NODE PAR_FoundMark(unsigned int i_x_NumMark)**

Description: Look for a mark in all structure available. First, the research is made in the TLV_Tree structure updated by the parameter file. If not found, research continues with the default parameters TLV_Tree structure.

Input parameters:
I_x_NumMark : mark number to look for.

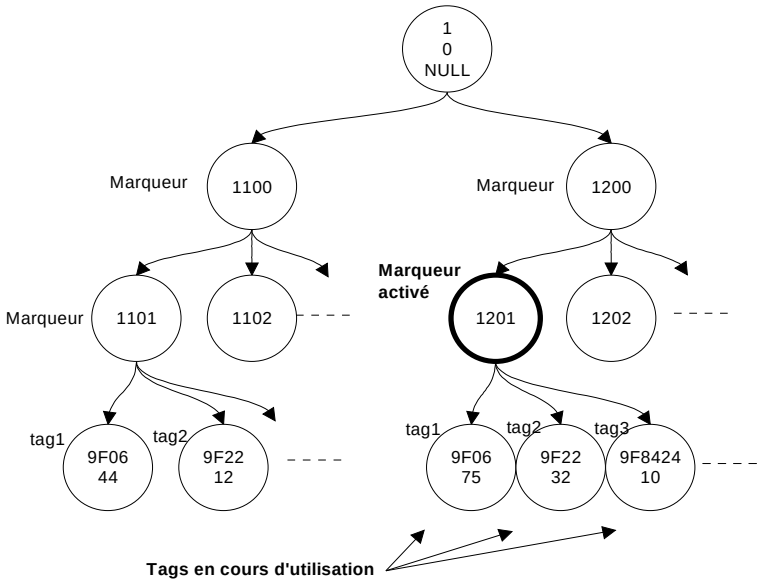
Output parameters:
Pointer on mark found, 0 else.

- Syntax: **int PAR_SelectMark(unsigned int i_x_NumMark)**

Description: Mark activation. All children tags will be available for a next PAR_ReadParameter() function call.
Remark: if this mark is already activated, the previous activation is suppressed. This cancellation allows managing a parameter update.

Input parameters:
I_x_NumMark: mark number to activate.

Output parameters:
0 if mark has been activated, error code else (mark not found …).



Valeur du tag 9F06 => 75 renvoyé par la fonction de lecture PAR_ReadParameter()

- Syntax: **unsigned long PAR_SelectFatherMark(DataElementExt data)**

Description: Active father mark of a tag.
Remark: This function could be used to select a set of parameter in relation with a specific data.

Input parameters:
DataElementExt data: tag reference.

Output parameters:
FatherMark selected if a father has been founded and selected, 0 else.

- Syntax: **int PAR_DeSelectMark(unsigned int i_x_NumMark)**

Description: mark deactivation.

Input parameters:
I_x_NumMark: mark number to deactivate.

Output parameters:
0 if mark has been deactivated, error code else (mark not found as activated…).

- Syntax: **int PAR_IsMarkSelected(unsigned int i_x_NumMark)**

Description: Give information if a mark has been activated.

Input parameters:
I_x_NumMark: mark number to precise.

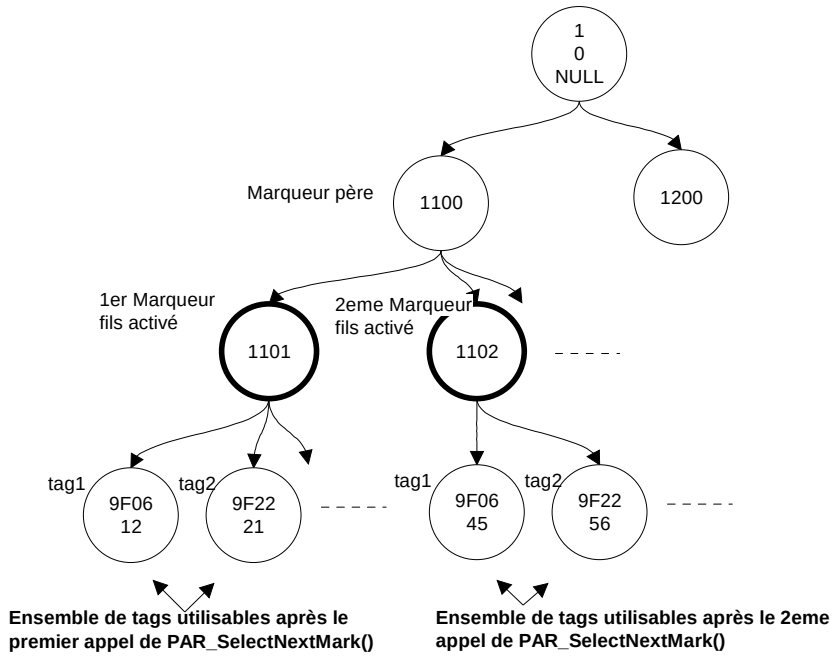
Output parameters:
1 if mark is already activated, 0 else.

- Syntax: **int PAR_SelectNextMark(unsigned int i_x_NumMark,**
  unsigned char uc_x_FirstAccess)

Description: select all children tag. Each child is selected at a time.

Input parameters:
I_x_NumMark: mark father number used as reference to select (active) each child.
Uc_x_FirstAccess : first function call indicator. On first function call, first child must be activated. In other case, next child must be activated.

Output parameters:
Child tag number activated, 0 else.



- Syntax: **int PAR_MarkAction(unsigned int i_x_NumMark,**
  unsigned char y_x_ActionType)

Description: Action on a specific mark.
This function doesn't manage a multi mark action.

Input parameters:
I_x_NumMark: mark number for the action.
Y_x_ActionType:

C_SELECT	: mark activation
C_UNSELECT	: mark deactivation.
C_SELECT_NEXT_CHILD	: next child activation.
C_UNSELECT_CHILD	: child deactivation.

Output parameters:
0 if no error on action, error code else

- Syntax: **int PAR_MarkInfos(unsigned int i_x_NumMark)**

Description: general function giving all information on a mark (Mark may be defined in default or in file parameters).

Input parameters:
I_x_NumMark: mark number to precise.

Output parameters:
Information sur le marqueur :

C_MARK_NOT_DEFINE	: undefined mark
C_MARK_NO_ACTIVATED	: Inactivated defined mark.
C_MARK_ACTIVATED	: Activated defined mark.

- Syntax: **int PAR_ReadParameter(unsigned long l_x_Tag,**
  unsigned char * value)

Description: Tag value access (Value taken for the tag is extracted from the active mark list) default value or value updated by parameter file may be used.

Remark : if several tags are available,
The active mark list allows finding the value of asked tag. If several tags are available for a specific tag number asked, the last mark activated will select the good tag.

Input parameters:
i_x_Tag: tag number

Output parameters:
Tags value pointed.

Return: 0 if tag defined, 1 else

- Syntax: **int PAR_ReadParameterList (const unsigned long cst_tag_list[],**
  unsigned int i_x_NbTag,
  DEL * outputDEL)

Description: Read a list of parameters in param or default TLVTree structure.
Input parameters:
Cst_tag_list: tag list to read
I_x_NbTag : nb tag to read

Output parameters:
OutputDEL : Data Element List updated with Tags read.

Return:
Nothing

- Syntax: **Int PAR_SetParameter(unsigned int i_x_Tag,**
  unsigned char ParameterAction,
  unsigned char * TagValue)

Description: this function initialises or affects one or several tags. Tags initialisation may be made with a value given in parameter, with the parameter file value, with default parameter value or value may be reseted. This initialisation may be made on one or a set of tag if the given parameter is a mark. This function works with marks. Tags managed by a mark may be in the TLV_Tree structure updated by the parameter file or in the default TLV_Tree structure.

Remark: a tag initialisation or tag affect could be made only if the tag is available (a father mark is activated).

Input parameters:
i_x_Tag : tag (or mark) number for the action
y_x_ParameterAction : Action type
C_RESET_TAG : reset tag (set 0 value)
C_RESET_MARQ : reset all children tag of mark (set 0 value)
C_DEFAULT_TAG : default value recovery for a tag
C_DEFAULT_MARQ: default value recovery for all children tags of a mark.
C_VALUE_TAG : set a new value to the tag.
C_VALUE_MARQ : set a new value to all children tag of a mark (may be used to reset a sat of tag).
unsigned char * TagValue : Pointer on tag value.

Output parameters:
0 if no error, error code else.

- Syntax: **int PAR_Build_Parameters(unsigned char c_x_WorkType,**
  char * ptr_TxtBuf,
  char * ptr_OutBuf,
  char * ptr_rStringTmp,
  unsigned char option,
  unsigned int i_x_Mark,
  char * ac_x_Mark)

Description: Update an output binary serial TLV_Tree file with parameters read in a text file. Input text file format may be text or block.

Text input file format:

Text format = list of tags and value

```
9F31 = 14 21 AA 23
9F42 = 12
9F10 = 12 A0 00
…
```

Block format = set of tag delimited by the «{» or «}» mark. See annexe 1 for example.

```
0 = {
        9FFF00= 41 49       44 31;
        9F06= 07 B0 1234 5678 1234;
        …
}
```

Input parameters:
c_x_WorkType : input file type
C_WORK_TEXT_FILE_TO_BIN : text file
C_WORK_CODED_FILE_TO_BIN: block text file
ptr_TxtBuf : path and input file name
ptr_OutBuf : path and output file name.
ptr_rStringTmp : text information on function process
option : if set to 1, the output file (binary serial TLV_Tree format) will be read and interpreted. All informations will be displayed.
i_x_Mark : origine mark (tag) to set data read.
ac_x_Mark : origine mark (text) to set data read.

Output parameters:
0 if no error, error code else

- Syntax: **void CUPAR_Print_Param_Supported(TLV_TREE_NODE pnode,**
  char *ptr_x_DataName,
  unsigned int i_x_Marq,
  unsigned char c_x_tracetype,
  unsigned long ul_x_tag)

Description: send parameters on printer

Input parameters:
Pnode : pointeur TLV_Tree
ptr_x_DataName : parameter set name

i_x_Marq : mark origin in the TLV_Tree to trace parameters.
c_x_tracetype : allow to trace all nodes (parameters) or just a part or just one
C_TRACE_ALL_CHILD : print all parameters,
C_TRACE_ONE_TAG : print tag given in pararmeter,
C_TRACE_MARQ : print all marks,
C_TRACE_WITHOUT_MARQ: print all tags without marks.
ul_x_tag : tag number to print if only 1 tag must be printed.

Output parameters:
None

- Syntax: **int CUPAR_ReadTextParamFile (TLV_TREE_NODE * ptr_pTree,**
  unsigned int i_x_tag,
  char * ptr_x_tagMark,
  char *fileName)

Description: Read a parameter text file (AIDx.par, CAKEYS.par, KREVOK.par, ICS.par) and update a TLV_Tree structure.

Input parameters:
ptr_pTree : TLV_Tree structure to update with parameters read.
I_x_tag: mark origin to set parameters in TLV_Tree structure.
Warning: if several same tags must be stored, several incremented marks will be create.
pc_x_DirFile : path and parameter file name.
pc_x_filename : input parameter file name.

Output parameters:
Pointer on TLV_Tree structure with parameters updated.
0 if no error, error code else.

- Syntax: **int CUPAR_ReadCodedParamFile (TLV_TREE_NODE * ptr_pTree,**
  char *fileName)

Description: Read a text block file and update a TLV_Tree structure.

Input parameters:
ptr_pTree : TLV_Tree to create or to update with parameters read.
pc_x_filename : input parameter file name.

Output parameters:
0 if no error, error code else.

- Syntax: **int CUPAR_read_next_element_text (char *buf,**
  int lg,
  int *indice,
  int max_len,
  unsigned long *tag,
  int *length,
  char *value)

Description: Read a text file element and extract a tag element.

Input parameters:
Buf : data read in file
lg : data buffer length
max_len : maximum length of an element

Output parameters:
Tag : Tag number of the read element
Length : data tag length
Value :data tag
indice : index on input buffer
TRUE if function is successful, FALSE else

- Syntaxe : **int CUPAR_read_next_element_coded (char *buf,**
  int lg,
  int *indice,
  int max_len,
  unsigned long *tag,
  int *length,
  char *value,
  unsigned char *py_x_BlockMark)

Description: Read a block text file element and extract a tag element.

Input parameters:
Buf : data read in file
lg : data buffer length
max_len : maximum length of an element

Tag: Tag number of the read element
Length: data tag length
Value: data tag
indice : index on input buffer
py_x_BlockMark : TRUE if a new data block is begining, FALSE if a data block is ending. 0xff else.

Return:
TRUE is function correctly executed, FALSE else.

6. ANNEXE

6.1 ANNEXE 1 : PARAMETER FILE, TEXT FORMAT WITH « BLOCKS » ORGANISATION:

Text block format file: unique PARAM.PAR file.
Syntax: node list with blocks delimiters: '{', '}'
And Tag list and values.

Remark, the block text reader eliminates punctuation as spaces, tabs, points, commas, …
Fields following "//" marks are not used.
Each mark or « node » must be followed by brackets showing the beginning and the end of a block..

```
0 = {
        1000 = {
                9FFF00= 41 49       44 31;
                9F06= 07 B0 1234 5678 1234;
                9F1A= 00 56;
                5F2A= 09 78;
                5F36= 02;
                9F812B= 00 00 01 F4
                9F8124= 9F 37 04
        }
        1010 = {
                9FFF00= 11 49       44 31;
                9F1A= 10 56;
                5F2A= 19 78;
                9F8124= 9F 37 04
                1020 = {
                        9F00=  44 31;
                }
        }
}
```

- ⇨ main mark: N° 0.
- ⇨ 2 children mark: n° 1000 and 1010.
  - o mark n°1000 with 7 tags (9fff00, 9f06, 9f1a, 5f2a, 5f36, 9f812b, 9f8124).
  - o mark n°1010 with 4 tags (9fff00, 9f1a, 5f2a, 9f8124) and 1 child (mark n° 1020)
    - ▪ mark 1020 with 1 tag (9f00)

6.2 ANNEXE 2 : PARAMETER FILES, TEXT FORMAT WITHOUT ORGANISATION :

6.2.1.1 Blackl.par

9F8805= 54 13 33 90 00 00 15 96;

=> fichier à laisser en l'état

6.2.1.2 Krevok.par

9F06 = A0 00 00 00 04;
9F22 = F8;
9F8424 = 00 10 00;

6.2.1.3 AidX.par

9FFF00= 41 49 44 31;
9F06= 07 B0 12 34 56 78 12 34;
9F1A= 00 56;
5F2A= 09 78;
5F36= 02;
9F812B= 00 00 01 F4;
9F8124= 9F 37 04;
9F8125= 9F 08 02;
9F8126= 00;
9F8127= 00;
9F8128= 00 00 00 00 00;
9F8129= 00 00 00 00 00;
9F812A= 00 00 00 00 00;
9F09= 00 02;
9F1B= 00 00 27 10;
9F841D= 00;

6.2.1.4 Ics.par

9f8450=PBOC;
9f35=22;
5f2a=008C;
9f1a=0840;
9f33=e0 e9 c8;
9f40=68 00 f0 b0 01;
9f8142=YES;
9F8195=YES;
9f844b=YES;
9f8440=NO;
9f8441=YES;
9f8442=NO;
….

6.2.1.5 Cakeys.par

9F06=A000000004;
9F22=01;
9F8123=A6E6FB72179506F860CCCA8C27F99CECD94C7D4F3191D303BBEE37481C7AA15F233BA755E9E4376345A9A67E7994BDC1C680BB3522D8C93EB0CCC91AD31AD450DA30D337662D19AC03E2B4EF5F6EC18282D491E19767D7B24542DFDEFF6F62185503532069BBB369E3BB9FB19AC6F1C30B97D249EEE764E0BAC97F25C873D973953E5153A42064BBFABFD06A4BB486860BF6637406C9FC36813A4A75F75C31CCA9F69F8DE59ADECEF6BDE7E07800FCBE035D3176AF8473E23E9AA3DFEE221196D1148302677C720CFE2544A03DB553E7F1B8427BA1CC72B0F29B12DFEF4C081D076D353E71880AADFF386352AF0AB7B28ED49E1E672D11F9;
9F8122=010001;
9F8121=0A86684ABB1B1CFDAFBDBBFE7D3C8C1C0AB10B10;

9F06=A000000003;
9F22=03;
9F8123=98F0C770F23864C2E766DF02D1E833DFF4FFE92D696E1642F0A88C5694C6479D16DB1537BFE29E4FDC6E6E8AFD1B0EB7EA0124723C333179BF19E93F10658B2F776E829E87DAEDA9C94A8B3382199A350C077977C97AFF08FD11310AC950A72C3CA5002EF513FCCC286E646E3C5387535D509514B3B326E1234F9CB48C36DDD44B416D23654034A66F403BA511C5EFA3;
9F8122=03;
9F8121=3D7B6A188BBC655381E2F94C3F668C375FC677D8;
…

6.3 ANNEXE 3, MEMORY SIZE:

The TLV_Tree structure organisation doesn't use a fix memory size. The following parameters file give the following data size:

Input parameter file :
4 AID with 16 tags each
1 KREVOK file with 3 tags
1 CAKEYS with 37 set of keys. Each set of key include 5 tags,
1 ICS file with 38 tags.

- ⇨ Input param.par file size: 18koctets
- ⇨ Binary data, buffer with serial TLV_Three format: 19,5koctets
- ⇨ TLV_Tree structure stored in memory : 22.3koctets ram