Persistence

Inhalt

Persistence as a black box from application point of view 3

The responsibility of Persistence in the system context 4

Application point of view 4

Infrastructure point of view 4

DataSeparation 5

PersistenceConcept 6

Level 1 logical view of Persistence 6

Responsibility Description 7

Persistence Management: Key-Value Management -> Shared data management 7

Persistence Management: Key-Value Management -> Local data management 7

Persistence Management: Key-Value Management -> Specific data management 7

Persistence Management: File and folder management -> Local and shared file management 7

Persistence Management: User and Shared data management -> Persistence Administration and Health 7

Concept Manifest 9

Library construction 9

Overview of the solution for the different types of persistence data 10

Persistence Interface Concept 11

Key - Value Access Interface Concept 11

Interface prototype 12

API usage examples 13

Persistence Resource Configuration table example (PRC-table) 14

File / Folder Access Interface Concept 15

Data organization 16

Format for Installation Configuration Table 17

Interface Definition 18

Client Library Interface Overview 18

ClientLibrary API - file access 19

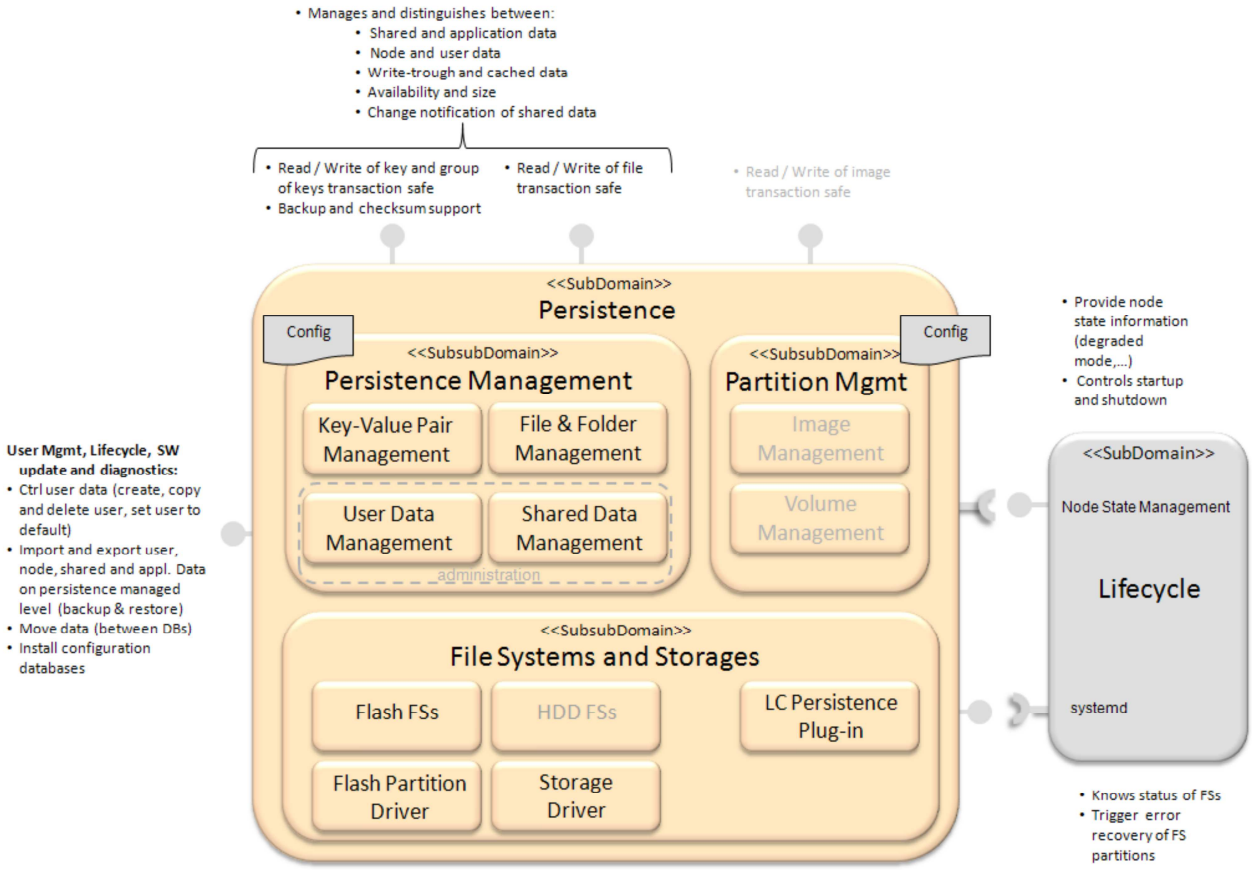
ClientLibrary API - key-value 21

Persistence Administration Service Interface Overview 23

PersAdminService 23

Persistence as a black box from application point of view

The following figure shows the interfaces of Persistence to the rest of the system. The provided interfaces which can be used by applicationsor others and the requested interfaces which has be either fulfilled or stubbed by the product development.
Additionally Persistence will be dependent on the subdomain Log&Trace.



*) Blocks with gray font are currently not in scope of the concept.

The responsibility of Persistence in the system context

Application point of view

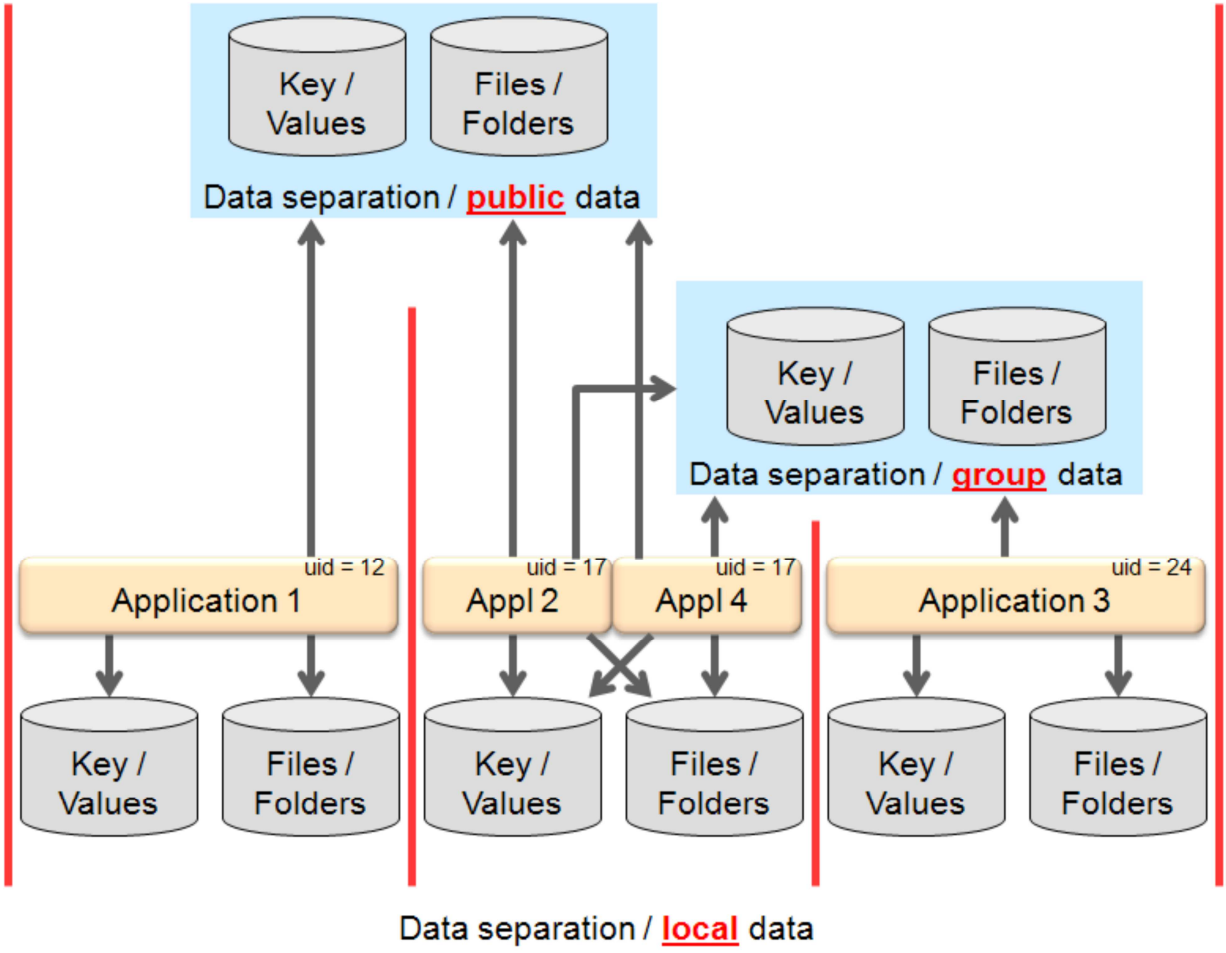
1. Read / write data items / key-values fast and reliable
2. Read / write files fast and reliable
3. Provide data storage for local data items and files (means no other application can access this data)
    a. Change notification is not needed for these data items
    b. No central/system wide data management required (means data items are not visible as an managed interface)
4. Provided data storage for shared data items and files (means a group of application or all can access this data)
    a. Change notification is needed for these data items
    b. The configuration of the data need to be managed as an interface
5. Committed write (write-through) and cached write shall be possible
6. User data and non-user (node) data are separated by hidden organization
7. Re-deployment / merging of processes shall be supported without having large impact on the organization and application code
8. Default data are managed separately and cannot be modified by the running application (for user and node data)
9. Configurable default data shall be supported (means the original default data is copied and changed)

Infrastructure point of view

1. The physical storage / partition for data items / group of item / files can be configured central and system-wide
2. Shared data need to be system-wide registered
3. Committed write (write-through) configured data items need to be system-wide registered
    a. To get control on flash write activities
4. User and non-user related data can be found easily
    a. For backup and restore reason (in ECU)
    b. For export and import reason (cross ECU)
5. Data items can be set back to default in case of unexpected behavior of the application.

DataSeparation

Access and grouping of the persistence data
The concept is to use the POSIX rights management for user access rights in order to provide data separation/security betweenapplications or/and groups of applications.
Note one or more applications can be started with the same POSIX UID.



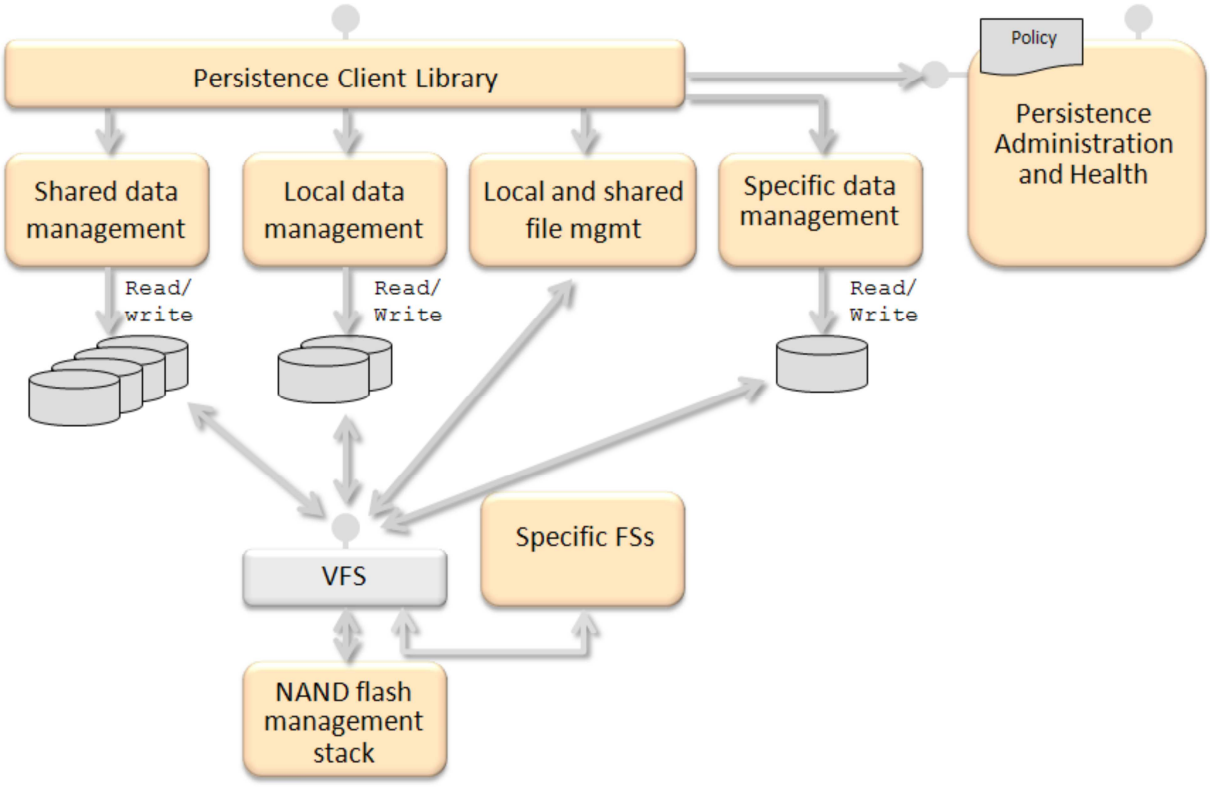
PersistenceConcept

Level 1 logical view of Persistence

The first-level structure definition:

- Clear separation between administration activities and Persistence users (application SW)
- Best performance including reduction of layering
- Reuse of existing open source solutions
- Meet application SW expectations
- Offering GENIVI SW platform interfaces to abstract from the chosen solution
- Extendable with product/customer specifics

The following figure presents the level 1 logical view of the concept for Persistence (only what is in scope currently!).



Responsibility Description

Persistence Management: Key-Value Management -> Shared data management

Abstract:

- Synchronized data write support
- Maximum on read performance
- Data change notification
- Access policy support
- Write policy support and configuration
- Hide user and node data items organization
- For small data size only (kByte size)

Persistence Management: Key-Value Management -> Local data management

Abstract:

- Ensures data isolation (only accessible by the owned application)
- Write policy support and configuration only for committed write requests
- Hide user and node data items organization
- Maximum on read and write performance
- For small data size only (kByte size)

Persistence Management: Key-Value Management -> Specific data management

Abstract:

This opportunity can extend the concept for custom solution for early, secure, factory settings a.s.o storages / databases.

Persistence Management: File and folder management -> Local and shared file management

Abstract:

- Simplify posix file system operations
- Manage error conditions
- Ensure reliable data write activities
- Abstracts the write policies and knows about configured committed write operations per file
- Hide user and node file organization
- Does not care about access policy of the file(s) (is the responsibility of the file system and be setup by the administration)

Persistence Management: User and Shared data management -> Persistence Administration and Health

Abstract:

- Create default application folder structure including links to shared data and deploy the default content (provided by the installation
- process)
- Create the configured local database for each application
- Create the configured shared databases
- Provide application specific links to shared databases (group/ public)
- Setup of application file system access policies
- Delete, copy, backup and restore files (files and databases)
- Manage partitions and volumes
- Handles mount issues
- Handles full files system partitions

- Observing usage of file systems?

File Systems and Storages: Persistence Management plug-ins -> NAND flash management stack

Abstract:

- Read / write of files
- Offers cached-write and write-through
- Offers lock for write-back and write-through
- Offers different mount points with different policies at the same time (access rights and write method)
- Transaction safe operations on files
- Statistics of file system usage
- Access policy support
- Volume / quota support
- Support of raw NAND chips
- It is currently open if there can also be combined solution for NAND and managed-NAND.

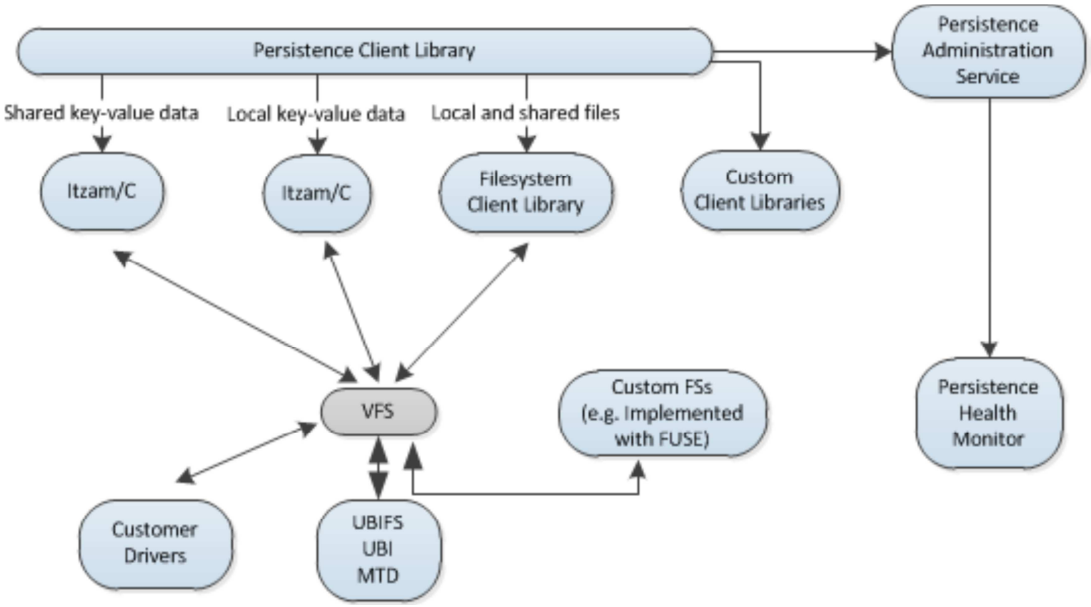
File Systems and Storages: Persistence Management plug-ins -> Specific filesystems

Abstract:

- If the NAND management stack (filesystem and more) does not support the required features a custom FS has to cover the gap:
- Offers cached-write and write-through
- Offers lock for write-back and write-through
- Offers different mount points with different policies at the same time (access rights and write method)
- Transaction safe operations on files
- Statistics of file system usage
- Volume / quota support
- Physical abstraction of raw NAND and managed NAND
- Additionally the possibility of a custom FS enables:
- Extensions of the concept for custom solution for early, secure, factory settings a.s.o file systems

Concept Manifest

The following figure presents the refinement of the concept for Persistence.



Library construction

The persistence client library provides a plugin API to load the other libraries shown in the image above with dlopen:

- Itzam/C library
- filesystem client library
- different custom client libraries

The "sub"-libraries (custom client libraries and custom filesystem libraries) will be loaded only when they are needed, for example, to provide a fast start-up.
Itzam/C is the standard client library which will be loaded by default and is needed to read the client configuration data (later named as persistence resource table).

Overview of the solution for the different types of persistence data

Here we are, the answer of the white cells:

	Node			User [n]				
Application [n]	Custom Client Lib	Custom Client Lib**	gvdb*	Custom Client Lib	Custom Client Lib**	gvdb*	WT	
	Custom Client Lib	Custom Client Lib**	gvdb*	Custom Client Lib	Custom Client Lib**	gvdb*	C	Key Value
	Custom FS	Custom FS**	UBI_FS	Custom FS	Custom FS**	UBI_FS	WT	
	Custom FS	Custom FS**	UBI_FS	Custom FS	Custom FS**	UBI_FS	C	File
Shared	Custom Client Lib	Custom Client Lib**	dconf	Custom Client Lib	Custom Client Lib**	dconf	WT	
	Custom Client Lib	Custom Client Lib**	dconf	Custom Client Lib	Custom Client Lib**	dconf	C	Key Value
	UBI_FS/RO	UBI_FS/WR**	UBI_FS	UBI_FS/RO	UBI_FS/WR**	UBI_FS	WT	
	UBI_FS/RO	UBI_FS/WR**	UBI_FS	UBI_FS/RO	UBI_FS/WR**	UBI_FS	C	File
	RO	RW		RO	RW			
	Secured		Normal	Secured		Normal		

*) This will change perhaps, because gvdb has no good write performance which was not expected by experts at the concept creation phase, which is needed for local application data. A new proposal is in work which should solve this issue. Currently we are aiming to replace gvdb/dconf by Itzam/C

**) Within a software loading or diagnostics session those data items can be written only by authorized application.

UBI_FS: UBI file system (http://www.linux-mtd.infradead.org/doc/ubifs.html)
dconf: allow the notification (https://live.gnome.org/dconf)
gvdb: possibility of cached and uncached granularity is limited to 4 kB page.

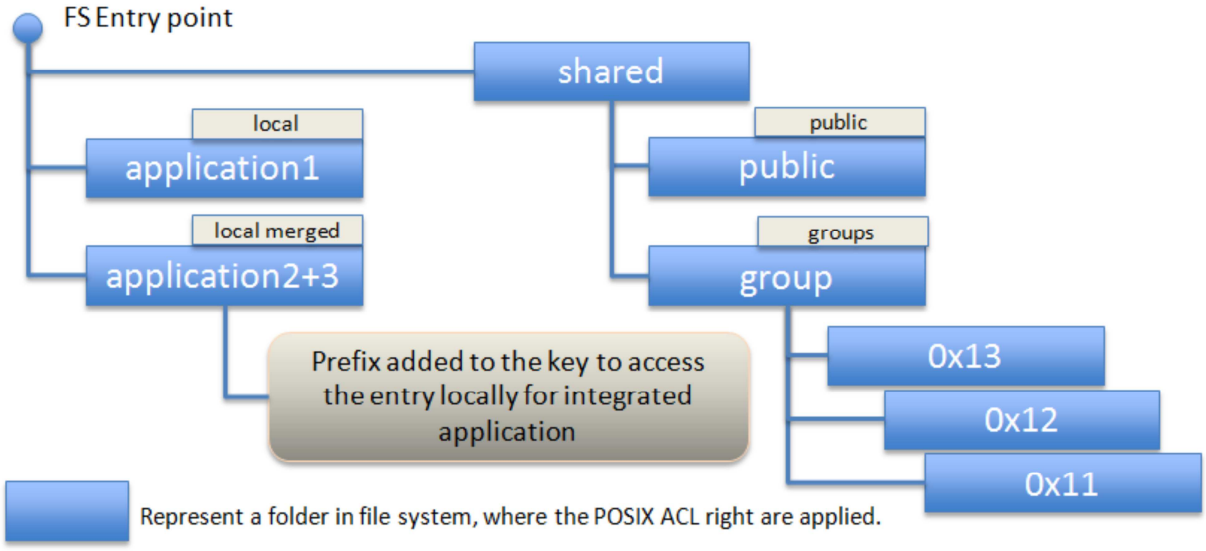
Persistence Interface Concept

Key - Value Access Interface Concept

Access level overview

Name	Definition
Local	folder local to an application. The access limitation is defined over the application UID (POSIX)
Local merged	folder local to an application which is containing merged application(s). The incubated application accesses the local application data environment where by the key are preceded by a prefix corresponding to the merged logical database identifier.
Groups	folder which can be accessed by the application with the corresponding GID assignment
public	folder which can be accessed by every applications in the system

File system access overview



The application SW have to deal with the following function parameters:

<table>
<tr><td colspan="3">Logical DB ID
(visibility / name space / location)</td><td>Resource ID</td><td>User No</td><td>Seat No</td></tr>
<tr><td></td><td>single persistence client</td><td>multiple persistence client</td><td rowspan="4">string [path + key]</td><td rowspan="3">unsigned int</td><td rowspan="3">unsigned int</td></tr>
<tr><td>single application</td><td>LOCAL</td><td>group</td></tr>
<tr><td>multiple application</td><td>local merge</td><td>PUBLIC</td></tr>
<tr><td colspan="3">Defines for LOCAL and PUBLIC data, for local merge and group the group ID will be used</td><td>node = 0x00 (no user)</td><td>no set = 0x00</td></tr>
</table>

Rational

The resource ID is system-wide not unique in general. The resource ID must be unique only for shared data. That means a namespace is needed.
Additional the available/chosen database:

- Does not support write-through / cached write on data item level
- Does not support right policy management

Therefore an identifier is needed which is called "logical DB-ID".
Inside the persistence client library (hidden) the data is stored in different locations -> location management is needed. The following table shows the different write policies and storage types as an example:

Write Policy	Storage Type
Cache	Early
Write Through	Normal
	ReadOnly
	Custome 1 (Secure)
	Custome 2 (Emergency)
	Custome 3 (HW Info)

*) The custom 1,2,3 and just further examples.
The different storage type will be registered to Persistence client library.

Interface prototype

Interface definition for key-value and file access will follow on next pages.

```
// return value positive: Contains the size
// return value negative: Error code
long get_size (unsigned int ldbid, char* resource_id, unsigned char user_no, unsigned
char seat_no, long* prc_handle)

// return value positive: Contains the size valid data in buffer
// return value negative: Error code
long read_data (unsigned int ldbid, char* resource_id, unsigned char user_no, unsigned
char seat_no, long* prc_handle, unsigned char* buffer, unsigned long buffer_size) INT32
```

API usage examples

The following table gives you examples about the parameters usage in different locations of the key-value management.

Persistence Client API	Persistence lib (from client resource table)	data base "input"	Comment
0, "/language/country_code", 0, 0	DB-ID="/sys/Secure"	Secure: "/language/country_code" or may a hash	public shared value (country_code)
0xFF, "/pos/last position", 0, 0	DB-ID="/Data/mnt-c/Appl-1/cached.itz"	Itzam/C: "/Node/pos/last position"	local value (last position)
0, "/language/current_language", 3, 0	DB-ID="/Data/mnt-wt/Shared/Public/wt.itz"	Itzam/C: "/User/3/language/current_language"	public shared user value (current_language)
0xFF, "/status/open_document", 3, 2	DB-ID="/Data/mnt-c/Appl-1/cached.itz"	Itzam/C: "/User/3/Seat/2/status/open_document"	local user seat value (open_document)
0, "/last_play_time", 0, 0	DB-ID="/sys/emergency"	Emergency: "hash"	(public, shared) (last play time)
20, "/address/home_address", 4, 0	DB_ID="/Data/mnt-c/Shared/Group/20/cached.itz"	Itzam/C: "/User/4/address/home_address"	group shared user value (home address)
0xFF, "/pos/last satellites", 0, 0	DB-ID="/Data/mnt-wt/Appl-1/wt.itz"	Itzam/C: "/Node/pos/last satellites"	local value (last satellites)
0x84, "/links/last link", 2, 0	DB-ID="/Data/mnt-wt/Appl-2/wt.itz"	Itzam/C: "/84/User/2/links/last link"	local value (last link)
0xFF "/media/mediaDB.db", 1, 1	DB-ID="/Data/mnt-c/Appl-1/"	file: "/User/1/Seat/1/mediaDb.db"	local file

Persistence Resource Configuration table example (PRC-table)

The following table is the corresponding resource table for this configuration example above for one application.

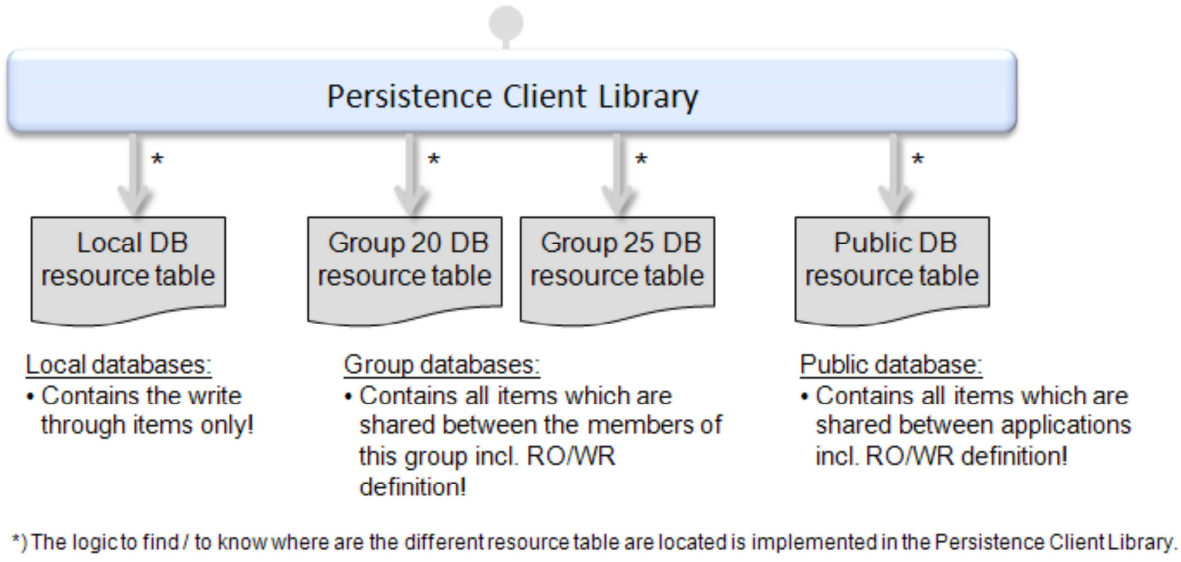
Pos	Logical DB ID	Resource ID	LDBID+RID hash	DB-ID (path+name)	Access rights	(Mount point info)
1	0	"/language/country_code"	hash	"/sys/Secure"	RO	NA
2	0	"/language/current_language"	hash	"/Data/mnt-wt/Shared/Public/wt.itz"	RO	WT
3	0	"/last_play_time"	hash	"/sys/emergency"	RW	NA
4	0x20	"/address/home_address"	hash	"/Data/mnt-c/Shared/Group/20/cached.itz"	RO	Cache
5	0xFF	"/pos/last satellites"	hash	"/Data/mnt-wt/Appl-1/wt.itz"	RW	WT
6	0x84	"/links/last link"	hash	"/Data/mnt-wt/Appl-2/wt.itz"	RW	WT
7	**TBD**	example for direct file access				
8	**TBD**	example for direct file access				
9	0xFF	"/pos/last position"	hash	"/Data/mnt-c/Appl-1/cached.itz"	RW	Cache
10	0xFF	"/status/open_document"	hash	"/Data/mnt-c/Appl-1/cached.itz"	RW	Cache

Note the last two entries in this table won't in that table finally, because this is the default and uncritical case. Therefore this does not need to be managed. This is just here to get a better understanding of the entire concept.

The responsibility of these resource tables are the following:

- Mapping of "Logical DB-ID + Resource ID" to the "DB-ID"
- Definition of the access rights
- Supports pre-compiled hash for a fast find/match
- Supports of fix position management within one Lifecycle for a fast find/match.

The runtime resource table is physical a set of tables! The following picture shows a typical table setup:



*) The logic to find / to know where are the different resource table are located is implemented in the Persistence Client Library.

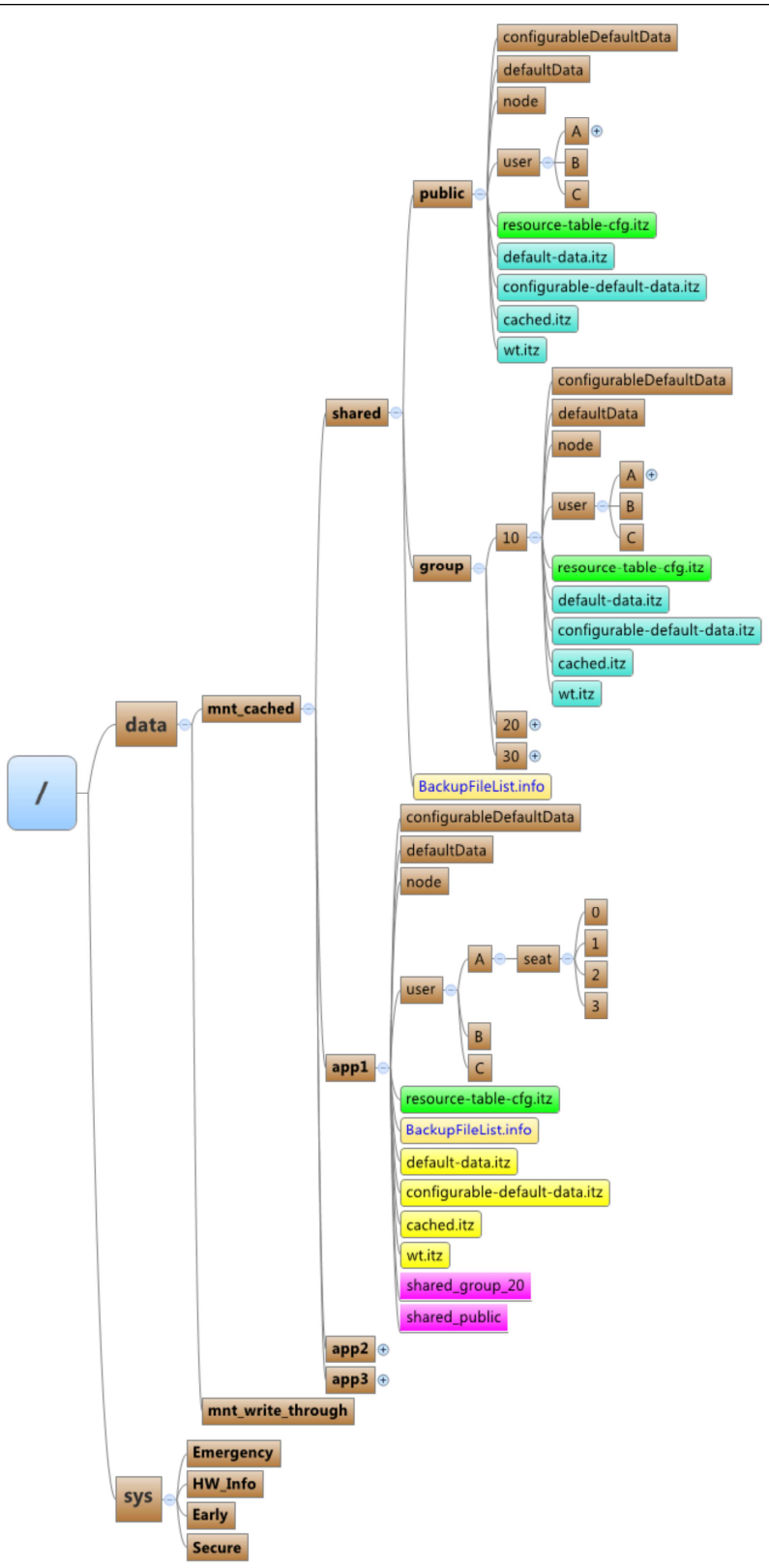
File / Folder Access Interface Concept

As the key-value interface is only to handle small data (kByte) the file / folder access interface concept offers the possibility to handle larger data (MByte).
It also allows applications to organize the data itself instead to store the data by key-value.

- The entire "/Data" tree is twice mounted:
  - As "Committed write" / "Write-Through"
  - As "Write-back" / "Cached-Write"
- Only write-through required files are listed in the resource tables.
- May be only one resource table for each type of table can be used (means a local resource table contains the list of the key-values and the file).
- No further redirection needed because the setup of the folder structure with the applied right policies organizes the visible of data to applications.
- Partial read of the file is possible
- Seeking within files is possible
- A backup of a file will not be created automatically, either there is an entry in the backup file list or the users triggers the backup creation.

For details see section "Data organization" below.

Data organization



Simplification of the data structure
In the context of the design of the Persistence Administrator a simplification of the structure was identified:
- minimize the size of the default data, the default data is only available once in the system also in case of user data
- remove the merge default data databases (wt / cache) to a single default database for the key values

Colors

Brown:
The brown colored items are folders.

Green:
Green refers to Itzam/C databases to store the path of the resource (key-value database, file store, local database, ...)

Turquoise:
Turquoise is a Itzam/C key-value database used to store shared data. This means different applications have access to this database.

Yellow:
Yellow is a Itzam/C key-value database to store applications data (only this application has access to this database)

Purple:
Purple are the links to shared persistence data the application has access to. There are public shared data (ever application has access) and shared group data (only a defined group has access)

Folders

data:
Persistent data will stored here.

sys:
Special persistent data will be stored here, like HW information, early data, secure data, emergency data. This folder can be extended to customer needs.

mnt_cached:
This folder will contain the cached persistent data.

mnt_write_through:
This folder will contain persistent data that will be directly written to memory device.

shared:
Here are the shared data, public (everyone has access) and group (only a special group of applications has access).

app1:
Each application has it's onw folder to stare persistent data.

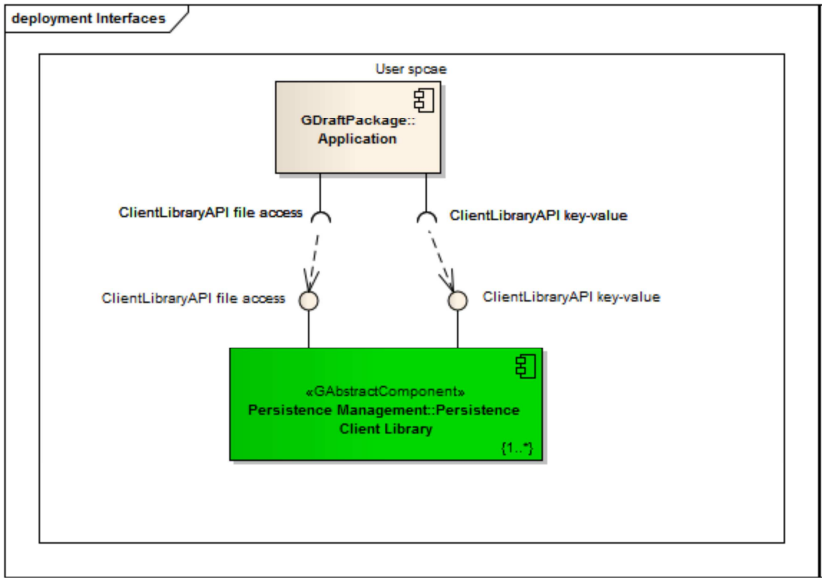
Format for Installation Configuration Table

In the intention to get the configuration manageable outside of the persistence environment of the target a portable format has to be selected.
In the first place the XML format may fulfill the requirements configurable, readable and version support.
A preferred format should be JSON where the same requirements are assumed but the footprint of the format is less.
Over the format the application, version and key are identified.
Presentation Configuration Definition for Persistence Administrator

```
{
    /* name of the application */
    "config_appl" : "ApplicationPlatform",
    /* version, string xx.xx.xx */
    "version" : "0.1.0",
    "resources" : {
        /* name of the entry */
        "ERG_WTCONTAINER_1234" : {
            /* policy: cached, wrtite through, NA */
            "policy" : "cached",
            /* permission: RW. RO */
            "permission" : "RW",
            /* storage: local,... */
            "storage" : "local",
            /* type: key-value, file */
            "type" : "key-value",
            /* unite: byte per user */
            "max_size" : "32",
            /* name of the responsible */
            "responsible" : "ApplicationPlatform",
            /* hash key to access data */
            "customID" : "0x84000001"
        }
    }
}
```

Interface Definition

Client Library Interface Overview



ClientLibrary API - file access

```
/**
* @brief close the given POSIX file descriptor
*
* @param fd the file descriptor to close
*
* @return zero on success. On error, -1 is returned, and errno is set
appropriately
*/
int pclFileClose(int fd);

/**
* @brief get the size of the file given by the file descriptor
*
* @param fd the POSIX file descriptor
*
* @return positive value. On error, -1 is returned, and errno is set appropriately
*/
int pclFileGetSize(int fd);

/**
* @brief map a file into the memory
*
* @param addr if NULL, kernel chooses address
* @param size the size in bytes to map into the memory
* @param offset in the file to map
* @param fd the POSIX file descriptor of the file to map
*
* @return a pointer to the mapped area, or on error the value MAP_FAILED
*/
void* pclFileMapData(void* addr, long size, long offset, int fd);

/**
* @brief open a file
*
* @param ldbid logical database ID
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
*
* @return positive value: the POSIX file descriptor; negative value: Error code
*/
int pclFileOpen(unsigned int ldbid, const char* resource_id, unsigned int user_no,
unsigned int seat_no);

/**
* @brief read persistent data from a file
*
* @param fd POSIX file descriptor
* @param buffer buffer to read the data
* @param buffer_size the size buffer for reading
*
* @return positive value: the size read; negative value: error code
*/
int pclFileReadData(int fd, void * buffer, unsigned long buffer_size);




/**
* @brief remove the file
*
* @param ldbid logical database ID
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
*
* @return positive value: success; negative value: error code
*
*/
int pclFileRemove(unsigned int ldbid, const char* resource_id, unsigned int
```

```
user_no, unsigned int seat_no);

/**
* @brief reposition the file descriptor
*
* @param fd the POSIX file descriptor
* @param offset the reposition offset
* @param whence the direction to reposition
SEEK_SET
The offset is set to offset bytes.
SEEK_CUR
The offset is set to its current location plus offset bytes.
SEEK_END
The offset is set to the size of the file plus offset bytes.
*
* @return positive value: resulting offset location; negative value: error code
*/
int pclFileSeek(int fd, off_t offset, int whence);

/**
* @brief unmap the file from the memory
*
* @param address the address to unmap
* @param size the size in bytes to unmap
*
* @return on success 0; negative value: error code
*/
int pclFileUnmapData(void* address, long size);

/**
* @brief write persistent data to file
*
* @param fd the POSIX file descriptor
* @param buffer the buffer to write
* @param buffer_size the size of the buffer to write in bytes
*
* @return positive value: bytes written; negative value: error code
*/
int pclFileWriteData(int fd, const void * buffer, unsigned long buffer_size);
```

ClientLibrary API - key-value

```
/// function callback definition for change notifications
typedef (*changeNotifyCallback_t)(void*)
/**
* @brief delete persistent data
*
* @param ldbid logical database ID
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
*
* @return positive value: success; negative value: error code
*/
int pclKeyDelete(unsigned int ldbid, const char* resource_id, unsigned int user_no,
unsigned int seat_no);

/**
* @brief gets the size of persistent data in bytes
*
* @param ldbid logical database ID
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
*
* @return positive value: the size; negative value: error code
*/
int pclKeyGetSize(unsigned int ldbid, const char* resource_id, unsigned int
user_no, unsigned int seat_no);

/**
* @brief close the access to a key-value identified by key handle
*
* @param key_handle key value handle return by key_handle_open()
*
* @return positive value: success; negative value: error code
*/
int pclKeyHandleClose(int key_handle);

/**
* @brief gets the size of persistent data in bytes identified by key handle
*
* @param key_handle key value handle return by key_handle_open()
*
* @return positive value: the size; negative value: error code
*/
int pclKeyHandleGetSize(int key_handle);

/**
* @brief open a key-value
*
* @param ldbid logical database ID
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
*
* @return positive value: the key handle to access the value; negative value:
Error code
*/
int pclKeyHandleOpen(unsigned int ldbid, const char* resource_id, unsigned int
user_no, unsigned int seat_no);

/**
* @brief reads persistent data identified by key handle
*
* @param key_handle key value handle return by key_handle_open()
* @param buffer the buffer for persistent data
* @param buffer_size size of buffer for reading
*
* @return positive value: the bytes read; negative value: error code
*/
int pclKeyHandleReadData(int key_handle, unsigned char* buffer, unsigned long
buffer_size);
```

```
/**
* @brief register a change notification for persistent data
*
* @param key_handle key value handle return by key_handle_open()
* @param if a key will be changed, the passed function callback will be called.
*
* @return positive value: registration OK; negative value: error code
*/
int pclKeyHandleRegisterNotifyOnChange(int key_handle, changeNotifyCallback_t
callback);

/**
* @brief writes persistent data identified by key handle
*
* @param key_handle key value handle return by key_handle_open()
* @param buffer the buffer containing the persistent data to write
* @param buffer_size the number of bytes to write
*
* @return positive value: the bytes written; negative value: error code
*/
int pclKeyHandleWriteData(int key_handle, unsigned char* buffer, unsigned long
buffer_size);

/**
* @brief reads persistent data identified by ldbid and resource_id
*
* @param ldbid logical database ID
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
* @param buffer the buffer to read the persistent data
* @param buffer_size size of buffer for reading
*
* @return positive value: the bytes read; negative value: error code
*/
int pclKeyReadData(unsigned int ldbid, const char* resource_id, unsigned int
user_no, unsigned int seat_no, unsigned char* buffer, unsigned long buffer_size);

/**
* @brief register a change notification for persistent data
*
* @param ldbid logical database ID of the resource to monitor
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
* @param if a key will be changed, the passed function callback will be called.
*
* @return positive value: registration OK; negative value: error code
*/
int pclKeyRegisterNotifyOnChange(unsigned int ldbid, const char* resource_id,
unsigned int user_no, unsigned int seat_no, changeNotifyCallback_t callback);

/**
* @brief writes persistent data identified by ldbid and resource_id
*
* @param ldbid logical database ID
* @param resource_id the resource ID
* @param user_no the user ID
* @param seat_no the seat number
* @param buffer the buffer containing the persistent data to write
* @param buffer_size the number of bytes to write
*
* @return positive value: the bytes written; negative value: error code
*/
int plugin_handle_get_size(int handle);
```

Persistence Administration Service Interface Overview

PersAdminService

```
/** Module version
The lower significant byte is equal 0 for released version only
*/
#define PERSIST_ADMINSERVICE_INTERFACE_VERSION (0x02010000U)
/** \defgroup PAS_RETURNS persAdmin Return Values
* ::PAS_SUCCESS, ::PAS_ERROR_CODE, ::PAS_FAILURE_INVALID_PARAMETER
* These defines are used to define the return values of the given low level access
functions
* - ::PAS_SUCCESS: the function call succeded
* - ::PAS_ERROR_CODE..::PAS_FAILURE: the function call failed
* \{
*/
/** Error code return by the SW Package, related to SW_PackageID. */
#define PAS_PACKAGEID 0x013 /**<
Software package identifier, use for return value base */
#define PAS_BASERETURN_CODE (PAS_PACKAGEID << 16) /**<
Basis of the return value containing SW PackageID */
#define PAS_SUCCESS 0x00000000 /**< the function call
succeded */
#define PAS_ERROR_CODE (-(PAS_BASERETURN_CODE)) /**<
basis of the error (negative values) */
#define PAS_FAILURE_INVALID_PARAMETER (PAS_ERROR_CODE - 1) /**<
Invalid parameter in the API call */
#define PAS_FAILURE_BUFFER_TOO_SMALL (PAS_ERROR_CODE - 2) /**< The
provided buffer can not accommodate the available data size */
#define PAS_FAILURE_OUT_OF_MEMORY (PAS_ERROR_CODE - 3) /**< not
enough memory, malloc failed, no handler available */
#define PAS_FAILURE_INVALID_FORMAT (PAS_ERROR_CODE - 4) /**<
format of the import source is not as expected (internal layout, type, etc) */
#define PAS_FAILURE_NOT_FOUND (PAS_ERROR_CODE - 5) /**< one
of the following resource file, folder or key not found */
#define PAS_FAILURE_INCOMPLETE_OPERATION (PAS_ERROR_CODE - 6) /**<
operation not completed due to shut-down notification */
#define PAS_FAILURE_ACCESS_DENIED (PAS_ERROR_CODE - 7) /**<
tried to access a file without having the right */
#define PAS_FAILURE_DBUS_ISSUE (PAS_ERROR_CODE - 8) /**<
related to DBUS */
#define PAS_FAILURE_OS_RESOURCE_ACCESS (PAS_ERROR_CODE - 9) /**<
related to mutex, queues, threads, etc.*/
#define PAS_FAILURE (PAS_ERROR_CODE - 0xFFFF) /**<
should be the max. value for error */
#define PAS_WARNING_CODE (PAS_BASERETURN_CODE) /**<
basis of the warning (positive values) */
/** \} */
/** \defgroup PERS_ADMIN_HELPER Configuration parameter
* \{
*/

#define PERSIST_SELECT_ALL_USERS (0xFFFFFFFF) /**< 32bit value used
to allow access to all users */
#define PERSIST_SELECT_ALL_SEATS (0xFFFFFFFF) /**< 32bit value used
to allow access to all seats */
/** enumerator used to identify the type of selected data for backup, import,
export */
typedef enum _PersASSelectionType_e
{
PersASSelectionType_All = 0, /**< select all data/files:
(node+user)->(application+shared) */
PersASSelectionType_User = 1, /**< select user data/files:
(user)->(application+shared) */
PersASSelectionType_Application = 2, /**< select application data/files:
(application)->(node+user) */
/** insert new entries here ... */
PersASSelectionType_LastEntry /**< last entry */
} PersASSelectionType_e;
```

```
/** enumerator used to identify the type of selected data for backup, import,
export
* \since V2.1.0
*/
typedef enum _PersASDefaultSource_e
{
PersADefaultSource_Factory = 0, /**< select from factory definition */
PersADefaultSource_Configurable= 1, /**< select from user factory or configurable
default if exist */
/** insert new entries here ... */
PersADefaultSource_LastEntry /**< last entry */
} PersADefaultSource_e;
/** \} */
/**
* \brief Allow creation of a backup on different level (application, user or
complete)
*
* \param type represent the quality of the data to backup: all, application, user
* \param backup_name name of the backup to allow identification
* \param applicationID the application identifier
* \param user_no the user ID
* \param seat_no the seat number (seat 0 to 3)
*
* \return positive value: number of bytes written; negative value: error code
(\ref PAS_RETURNS)
*/
long persAdminDataBackupCreate(PersASSelectionType_e type, const char* backup_name,
const char* applicationID, unsigned int user_no, unsigned int seat_no);
/**
* \brief Allow recovery of from backup on different level (application, user or
complete)
*
* \param type represent the quality of the data to backup: all, application, user
* \param backup_name name of the backup to allow identification
* \param applicationID the application identifier
* \param user_no the user ID
* \param seat_no the seat number (seat 0 to 3)
*
* \return positive value: number of bytes restored; negative value: error code
(\ref PAS_RETURNS)
*/
long persAdminDataBackupRecovery(PersASSelectionType_e type, const char*
backup_name, const char* applicationID, unsigned int user_no, unsigned int
seat_no);
/**
* \brief Allow to identify and prepare the data to allow an export from system
*
* \param type represent the quality of the data to backup: all, application, user
* \param dst_folder name of the destination folder for the data
*
* \return positive value: number of bytes written; negative value: error code
(\ref PAS_RETURNS)
*/
long persAdminDataFolderExport(PersASSelectionType_e type, const char* dst_folder);
/**
* \brief Allow the import of data from specified folder to the system respecting
different level (application, user or complete)
*
* \param type represent the quality of the data to backup: all, application, user
* \param src_folder name of the source folder of the data
*
* \return positive value: number of bytes imported; negative value: error code
(\ref PAS_RETURNS)
*/
long persAdminDataFolderImport(PersASSelectionType_e type, const char* src_folder);
/**
* \brief Allow to extend the configuration for persistency of data from specified
level (application, user).
* Used during new persistency entry installation
*
* \param resource_file name of the persistency resource configuration to integrate
*
* \return positive value: number of modified entries in the resource
```

```
configuration; negative value: error code (\ref PAS_RETURNS)
*/
long persAdminResourceConfigAdd(const char* resource_file);
/**
* \brief Allow the modification of the resource properties from data (key-values
and files)
*
* \param resource_file name of the persistency resource configuration to integrate
*
* \return positive value: number of modified properties in the resource
configuration; negative value: error code (\ref PAS_RETURNS)
*/
long persAdminResourceConfigChangeProperties(const char* resource_file);
/**
* \brief Allow the copy of user related data between different users
*
* \param src_user_no the user ID source
* \param src_seat_no the seat number source (seat 0 to 3)
* \param dest_user_no the user ID destination
* \param dest_seat_no the seat number destination (seat 0 to 3)
*
* \return positive value: number of bytes copied; negative value: error code (\ref
PAS_RETURNS)
*/
long persAdminUserDataCopy(unsigned int src_user_no, unsigned int src_seat_no,
unsigned int dest_user_no, unsigned int dest_seat_no);
/**
* \brief Delete the user related data from persistency containers
*
* \param user_no the user ID
* \param seat_no the seat number (seat 0 to 3)
*
* \return positive value: number of bytes deleted; negative value: error code
(\ref PAS_RETURNS)
*/
long persAdminUserDataDelete(unsigned int user_no, unsigned int seat_no);
/**
* \brief Allow restore of values from default on different level (application, user
or complete)
*
* \param type represents the data to restore: all, application, user
* \param defaultSource source of the default to allow reset
* \param applicationID the application identifier
* \param user_no the user ID
* \param seat_no the seat number (seat 0 to 3)
*
* \return positive value: number of bytes restored; negative value: error code
(\ref PAS_RETURNS)
*
* \since V2.1.0
*/
long persAdminDataRestore(PersASSelectionType_e type, PersASDefaultSource_e
defaultSource,
const char* applicationID, unsigned int
user_no, unsigned int seat_no);
```