GENIVI Alliance

GENIVI Document CS00027

Persistence Administration Service

Component Specification

Version 1.1.0

2015-01-23

Sponsored by:
GENIVI Alliance

Abstract:
This document describes the component “Persistence Administration Service”. A short overview of the environment will be given to get a better understanding of the realization. The definition is part of the GENIVI Persistence Concept.

Keywords:
GENIVI, Persistence Administration Service

GENIVI Alliance
2400 Camino Ramon, Suite 375
San Ramon, CA 94583, USA

Table of Contents

1 Overview

1.1 Purpose of the document

This document describes the component “Persistence Administration Service”. A short overview of the environment will be given to get a better understanding of the realization. The definition is part of the GENIVI Persistence Concept.

1.2 Persistence Overview

Persistence is about storing data permanently, to manage FLASH constraints and to manage Persistence within automotive lifecycle constraints.
The scope of this work package is a full development activity of Persistence. That means starting with studying the past / history of this package; specify the system- and software architecture; doing the design and implementation of the common parts including the integration and test.
The functional scope of persistence:

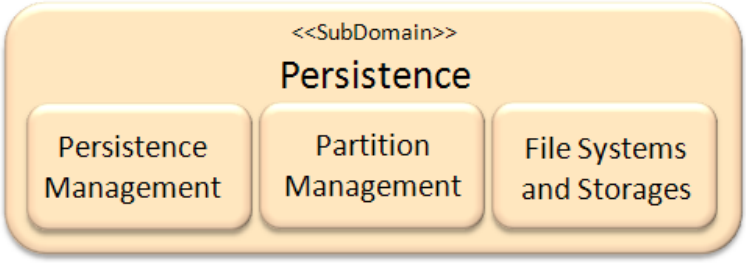


Fig. 1 – Functional scope of persistence

The current activity will cover in detail:

- Persistence Management
  - Key/Value Pair Management
  - File- and Folder Structure Management
  - User Data Management
  - Shared Data Management
- File Systems and Storages
  - Flash storages (focus NAND)
  - Persistence plug-in for Lifecycle (Health monitoring and repair)

Additionally the architecture will taking in account different deployments (one- and multi-node architectures) and how to handle OEM coding / variant data.
The current activities do not scope:

- Partition Management
- File Systems and Storages
  - other storages (except NAND flash)

1.3 Issues about "Persistent data"

1.3.1 Reliability

The problem with FLASH (NOR and/or NAND) devices is linked to reliability of semiconductor storage devices. In particular flash devices are sensitive to voltage drops during write cycles; reliability of flash devices heavily depends on the device driver, but also on the applications.

1.3.2 Lifetime

Managing persistent data is to avoid too often storing data during runtime in order to increase the lifetime of the FLASH device, because the total number of program-erase cycles per block is typically limited. Most available FLASH memory devices are guaranteed to have around 100,000 cycles, before the wear begins to deteriorate the integrity of the storage.
The number of program-erase cycles can be extended by a technique which is called wear leveling. Wear leveling can be done in software or it is already integrated in the hardware of the flash storing device.
The FLASH storing device must work properly at least during the lifetime of a car which is at minimum 10 years.

Exemplary estimation of the number of required program-erase cycles:

Mileage of a car per lifetime: 200.000km
Average mileage of one lifecycle: 10km
Number of lifecycles per lifetime: 20.000
Max program-erase cycles per lifecycle: 5

1.3.3 Defragmentation

Because of the physical limitation of manageable size (write-able size -> page, erase-able size -> block (NAND) or sector (NOR)) defragmentation is a known issue of Persistence data. Different solutions need to be offered to avoid forcing defragmentation already from concept point of view right from the beginning.

1.3.4 Availability during Start-up of the node/system

Typically to grant reliability persistent data must be checked before use it; this check can be a time-consuming process. But this time cannot be accepted for IVI system. Therefore more enhanced concepts are needed.

Considering that Persistence storage medium may have long start-up times (e.g. Hard Disks) the applications consuming persistent data must be informed when data can be accessed.

1.3.5 Getting everything stored in shutdown phase

To get a consistent system snapshot, power off must be delayed until all applications have their persistent data written. After all persistent data was written, the filesystem must be unmounted to ensure that the flash device is not accessed anymore. Unmounting the filesystem may also be a time consuming process, since it may include erase and/or write operations.

1.3.6 Data Retention

The persistence has to assume that the data retention. This is requested depending of the technology by accessing the data in regular cycles.

1.4 Persistence 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 applications or 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.

1.4.1 The responsibility of Persistence in the system context

1.4.1.1 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)

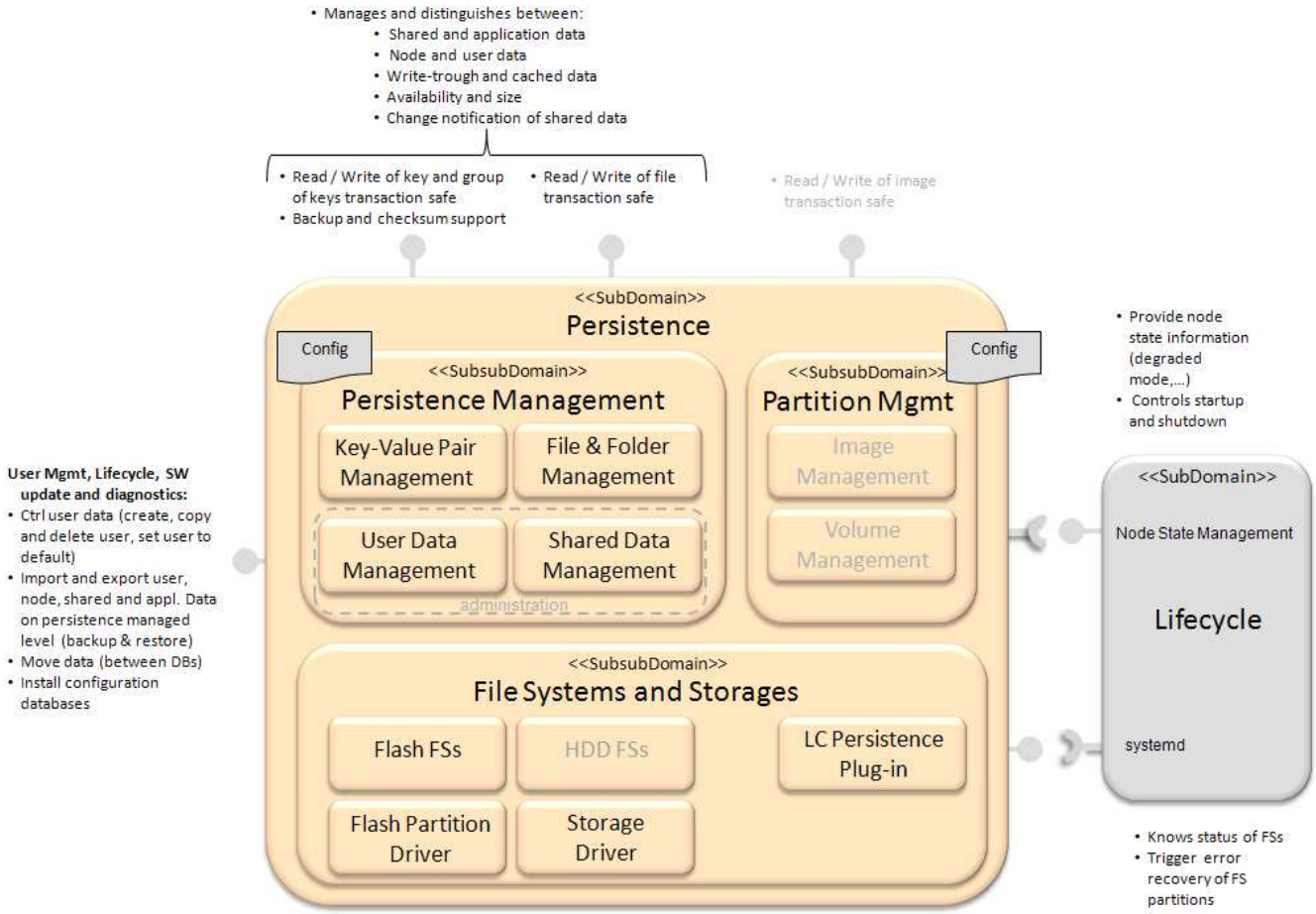


Fig. 2 –Persistence Infrastructure

1.4.1.2 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.

1.5 Persistence Concept

1.5.1 Logical view of Persistence

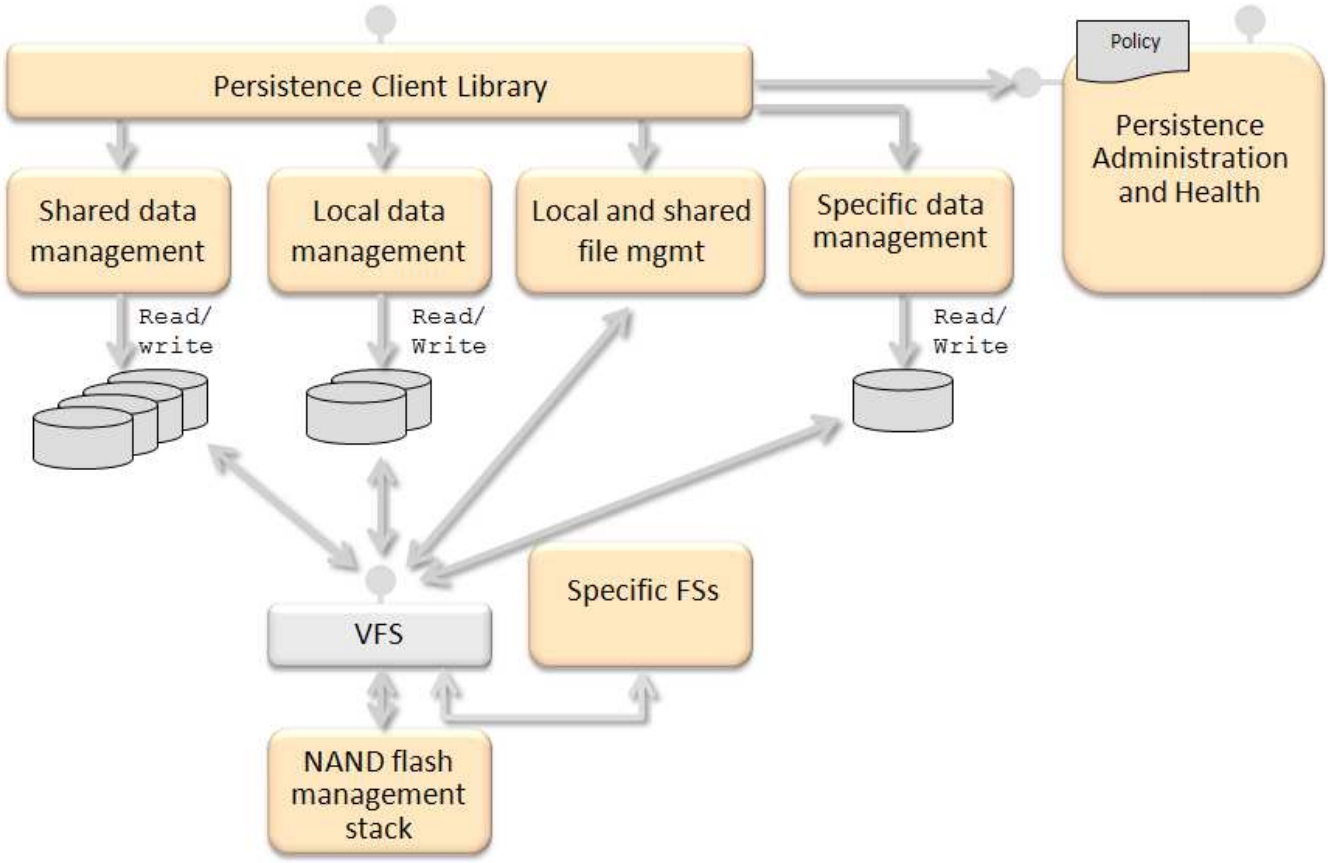


Fig. 3 –Persistence Logical View

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

1.5.2 Responsibility Description

1.5.2.1 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)

1.5.2.2 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)

1.5.2.3 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.

1.5.2.4 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)

1.5.2.5 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?

1.5.2.6 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.

1.5.2.7 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

2 Persistence Concept – Scope Persistence Administrator

2.1 Refinement Persistence Concept

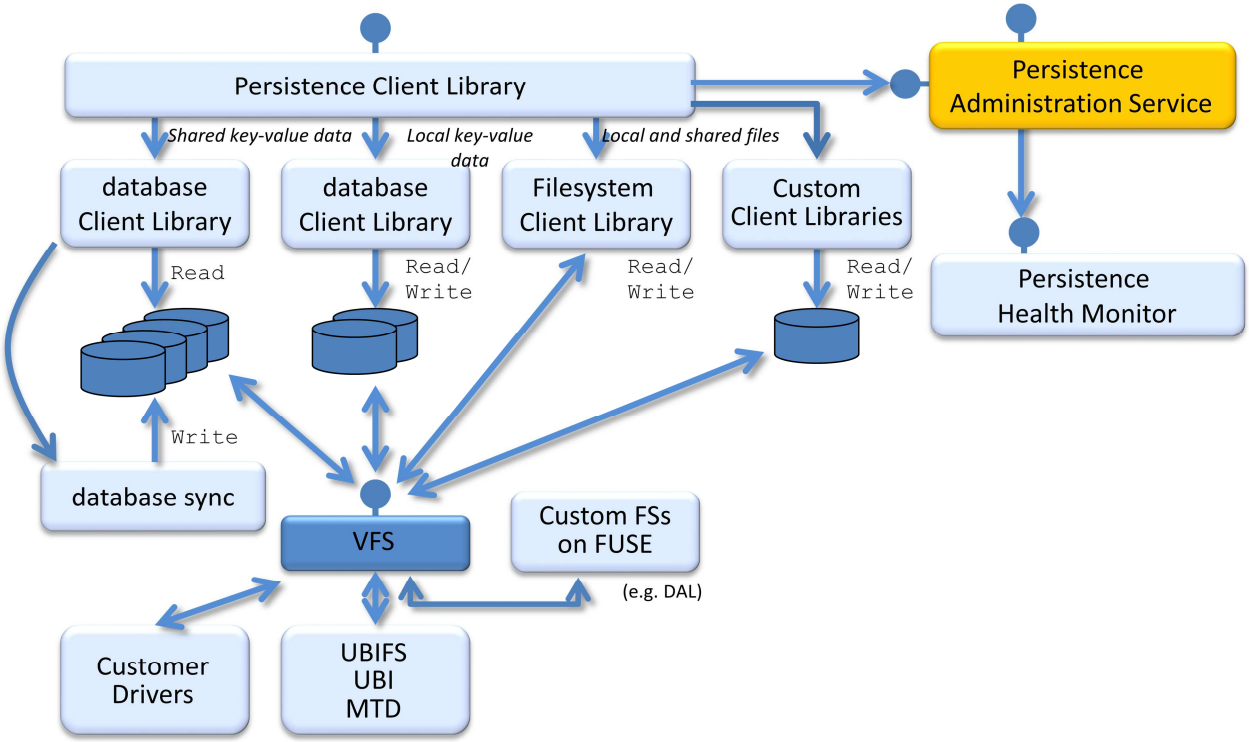


Fig. 4 – Refinement of persistence concept

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

2.2 Persistence Component / Requirements Persistence Administrator

2.2.1 System requirement

Requirement ID	Requirement Name	Priority	Text
VH-PERS-001	The flash shall work properly at least during the lifetime of a car (min. 10 years).	P1	Exemplary estimation of the number of required read/write cycles: Mileage of a car per lifetime: 200.000km Average mileage of one lifecycle: 10km => Number of lifecycles per lifetime: 20.000
VH-PERS-005	At least a portion of persistent data shall be readable very early in the platform start-up.	P1	Rationale: Settings are needed in the start-up process to start the right

			functionality, like coding data to get information which tuner is in the system (analog or digital). This information must available in kernel space.
VH-PERS-004	Persistence shall ensure that read data is not corrupt.	P1	Some error detection aka consistency check is needed. Rationale: Persistence must be robust against corruption of data e.g. triggered by sudden voltage drop at engine start. Corrupted data could cause unpredictable software behavior. It is better to use an older but valid set of data than an inconsistent or corrupted set of data.
VH-PERS-006	The user related portion of persistent data has to be deletable with a single user command. If there are factory default settings available, the data will be set back to this data.	P1	Rationale: Allow the "delete all personal information" Secure deletion is not required. The address book for example will be deleted completely and e.g. the current volume will be set back to default value.
VH-PERS-003	Ensure data consistency across groups of data.	P2	Rationale: There are sets of data e.g. within a lifecycle of the ECU which are related and shall always be stored together. If in case of an error not all data of the group could be stored successfully then the persistence should fall back to the last valid set of data for all applications using the data. Groups of data can be related data from different applications.
VH-PERS-002	Persistent data shall not be written during time periods where power drops are foreseeable.	P1	Rationale: Flash devices are very sensible to power drops. So flash write errors should be avoided by avoid writing in dangerous situations.

2.2.2 Software requirements

Requirement ID	Requirement Name	Priority	Text
SW-PERS-002	A Central policy of Persistence Management for read/write	P1	A Central policy of Persistence Management must be used which defines when Persistence data will be to read/written and what should be cached.
SW-PERS-003	Mechanism to request writing of data without caching/delay	P1	Apps need an mechanism to request writing of important persistent data immediately without any caching/delay
SW-PERS-004	The location to store persistent data must be configurable	P1	The term location refers to the path within the file system where the data will be stored.
SW-PERS-006	Check Persistence contents using hash for consistency	P1	A hash will be needed to detect inconsistency of data.
SW-PERS-007	There must be a recovery mechanism	P1	There must be a recovery mechanism if Persistence contents are considered invalid (e.g. rollback to earlier lifecycle)
SW-PERS-016	Persistence must be able to create internal data structure to store data by itself	P2	Persistence must be able to create internal data structure to store data by itself. In most cases it is sufficient when persistence creates the initial folder and file structure and optional deploy default content. The data structure should be hidden for users.
SW-PERS-017	Persistence must be able to setup file system write access policies	P2	Persistence must be able to setup file system write access policies Access policies can be e.g. to write through or write cached.
SW-PERS-018	Persistence must be able to setup file system access rights policies	P2	Persistence must be able to setup file system access rights policies. Which application has access to which data
SW-PERS-019	Persistence must be able to setup file system location	P2	Persistence must be able to setup file system location. Location of the persistent data

Tab. 1 – Software requirement related to Persistence Administrator

3 Software Component Persistence Administration Service

3.1 Requirement / responsibility overview

Responsibility	
the PAS shall provide the capability to change the policy to access the data without to lose the content (e.g. cache / write throw, local /share)	
Change visibility or policy of data	4.2.1
the PAS shall provide an interface to allow the installation of new application related data with the foreseen policy. The corresponding default values shall be in this context of this initialisation	
Configure and initialize data and policy for new application	
the PAS shal provide the capability to mount the environment of the persistence data, In case where the basis structure is not available the component has to create it and distribute the corresponding information in the system	
Configure persistence system	4.2.3
the PAS shall provide the capability to copy user related data to different destination. The tipical realisation allow to duplicate the current user configuration to an other.	
Copy user data	4.2.4
the PAS shall provide the capability to create backup of the current data. The selection of the data type (user/node and application/shared) shall be possible. The PAS provide a recovery point of the system in case where the complete data content is backup	
Create backup	4.2.5
The PAS shall offer the capability to delete the user related data, The selection of the related user has to be provided over the interface ćalled by the authorized application	
Delete user related data	4.2.6
The persistence shall provide the capability to recover the persistence content in error case. The data is corrupted, use the back or recreate the complete environment in case where the root /Data is not more available. In context of the error recovery a representable information of the recovery has to be logged in the syste,	
Error recovery	4.2.7
The PAS shall provide the capability to import the provided data in 3 differents ways: - local and shared user related data - local and shared application related data - complete provided data	
Import provided data	4.2.8
The PAS shall provide the capability to provide the data to export in 3 differents ways: - local and shared user related data - local and shared application related data - complete provided data	
Provide data for export	4.2.9
The PAS shall provide the capability to restore the data from previously created backup. In case of restore a complete previously created back the data recovery is executed	
Restore from backup	4.2.10
The RAS shall provide the capability to restore the data content to the default factory available in the system	
Restore to factory default	4.2.11
The RAS shall provide the capability to restire the data related data to default available in the system	
Restore to user default data	4.2.12

3.2 Administration Service UseCase Overview

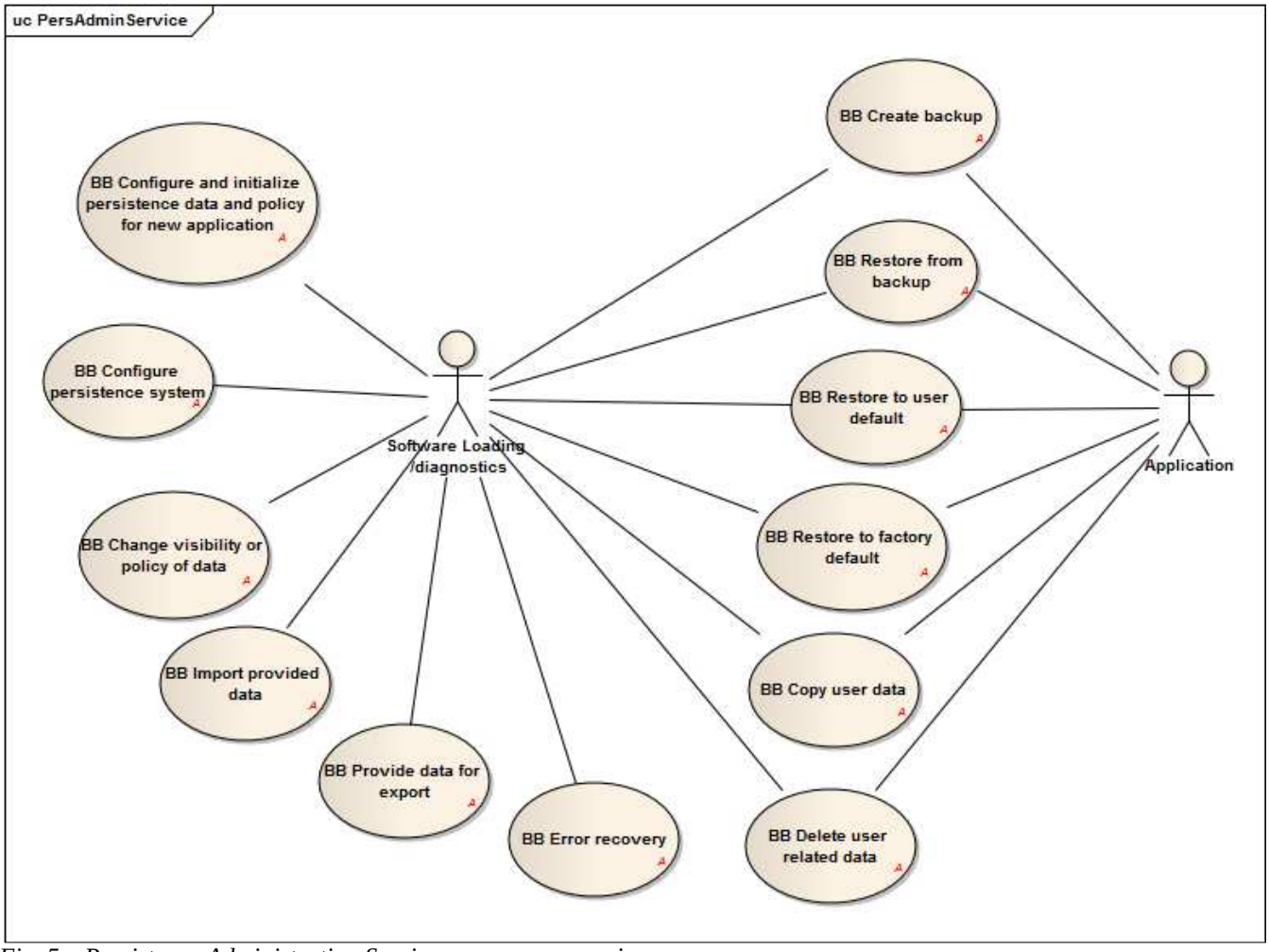


Fig. 5 – Persistence Administration Service - use cases overview

3.2.1 BB Change visibility or policy of data

Description:
The change of the visibility or policy causes to move data in the system.
Use cases:
- change of policy (move from cached to WT)
- change visibility (move from local to shared)
This use case is needed because limited functionality of the chosen solution.
Term "data" means files and key-values

Trigger: move between storage triggered by user or software loading and diagnostics

Pre-conditions:

- System up and running

Main Flow (change policy: cache <> uncached):

1. system requests to change the policy of data
2. check if the expected definition is already available in the current definition
3. check if definition is available in the resource configuration table:
    a. if key is not available in the configuration table

i. try to load data from cache / local area
4. check if data exists
   a. copy the data to the expected container
5. update the configuration table
6. return success

Alternative Flow [A2] (change policy):

1. system requests to change the policy of data
2. if the expected definition is already available in the current definition
   a. definition available in the configuration
   b. if not expected cached / local data
3. do nothing and return

Alternative Flow [A3] (change visibility: local <> shared):

1. system requests to change the visibility of data
2. check if the expected definition is already available in the current definition
3. check if definition is available in the resource configuration table:

- if key is not available in the configuration table
  try to load data from cache / local area

1. check if data exists

- copy the data to the expected container

1. update the configuration table
2. do nothing and return

Alternative Flow [A4] (change visibility):

1. system requests to change the visibility of data
2. if the expected definition is already available in the current definition
3. definition available in the configuration
4. if not available in the configuration data are cached / local data
5. do nothing and return

Exception Flow [Ex1]:

1. not enough memory
2. return error

Post-conditions:

- identified data have a new policy or visibility

Notes: Assumptions, issues and all other notes go here.

3.2.2 BB Configure and initialize persistence data and policy for new application

Description:
Configure persistence for new application:
- update the resource configuration table for the Persistence data with the policy and database assignment...
Initialize persistence data
- copy the provided factory default data (if available in the definition)
Term "data" means files and key-values-database

Trigger: The initialization and configuration can be started by the user or by software loading and diagnostics

Pre-conditions:

- System up and running

Main Flow:

1. check for validity of the resource configuration to update
2. update the resource configuration table for the Persistence data with the type, policy and database assignment
3. copy the provided factory default data (if available in the definition)

Exception Flow [Ex1]:

1. check of the resource configuration validity failed
2. return error code

Exception Flow [Ex2]:

1. check for validity of the resource configuration to update
2. no resource configuration to be updated
3. do nothing and return

Post-conditions:

data definition available in the system with the corresponding default values

Notes:

The requested resource configuration for the application has theoretically be part of the software package for its delivery.

3.2.3 BB Configure persistence system

Description:
Configuration of the /data area (different mount points, size definition...)
Customer client libraries database and file systems
dconf daemon starts

Trigger: the startup manager of the system (e.g. systemd) starts the configuration of the /data area

Pre-conditions:

- System up and running

Main Flow:

1. check availability of the Persistence root folder /data
2. mount the Persistence folder with the corresponding policy (cached / write through)
3. check the used size of the data in Persistence, create a report
4. starts the dconf daemons in the system to allow interprocess communication

Alternative Flow [A1]:

1. check availability of the Persistence root folder /data
2. /data doe not exist, create it
3. continue with Main Flow Pt 2

Exception Flow [Ex1]:

1. no more memory left on memory device
2. notification to system health monitor

Exception Flow [Ex1]:

1. other error case related to system start to be checked

Post-conditions:

- Persistence available system wide for key-value and file based realization

Notes:

Assumptions, issues and all other notes go here.

3.2.4 BB Copy user data

Description:
Possibility to copy of the user Persistence data to different destination
(e.g. default user to specific user)
Term "data" means files and key-values

Trigger: The copy of the user data can be started by the user or by software loading and diagnostics

Pre-conditions:

- System up and running

Main Flow (copy user to an other):

1. locate source user related data
2. check if user already exists in system
3. verify user related data from source
4. replace destination user related data with verified source user related data

Alternative Flow [A1]:

1. locate source user related data
2. check if user already exists in system
3. create user related structure
4. go to main flow Pt.3

Exception Flow [Ex1]:

1. source user does not exist in the system
2. return error code

Post-conditions:

- user data of the destination user are overwritten by the content of the source user

Notes:

Assumptions, issues and all other notes go here.

3.2.5 BB Create backup

Description:
Possibility to create a backup for the 3 different types:
- Local and shared user related data
- Local and shared application related data
- Complete (user, application and all public data)
Term "data" means files and key-values

Trigger: The backup can be started by the user or by software loading and diagnostics

Pre-conditions:

- System up and running

Main Flow complete backup (system recovery point):

1. walkthrough the shared and application data
2. identification of the file to backup over the corresponding definition list
3. synchronize the cached data in the system
4. copy all the files from backup file list to the corresponding backup files.

Alternative Flow application backup (application recovery point):

1. locate application data
2. identification of the file to backup over the corresponding definition list
3. synchronize the cached data in the system
4. copy all the files from backup file list to the corresponding backup files.

Alternative Flow application backup (user recovery point):

1. locate user data
2. identification of the file to backup over the corresponding definition list
3. synchronize the cached data in the system
4. copy all the files from backup file list to the corresponding backup files.

Exception Flow [Ex1]:

1. no more memory left on memory device
2. notification to system health monitor
3. return error to application /Diagnostic (rework)

Post-conditions:

- data has been stored within non volatile memory

Notes:

Assumptions, issues and all other notes go here.

3.2.6 BB Delete user related data

Description:
Current local and shared user relevant data must be deleted.
The delete can be started by the user or by software loading and diagnostics.
Term "data" means files and key-values

Trigger: The deletion of the user related data can be started by the user or by software loading and diagnostics.

Pre-conditions:

- System is up and running

Main Flow:

1. determine stored persistent user related data of a specific user
2. delete the user related data

Exception Flow [Ex1]:

1. data related to the specified user is not available
2. return an error code

Post-conditions:

- No user related data available anymore for the given user

Notes:

Assumptions, issues and all other notes go here.

3.2.7 BB Error recovery

Description:
Error recovery used for mount failed.
Use case definition will follow.
Last alternative will be to format of the partition / volume

Trigger: the startup manager of the system (e.g. systemd) or diagnostics starts the error recovery

Pre-conditions:

- System up and running

Main Flow:

1. call BB Restore from backup

Alternative Flow [A1]:

1. call BB Restore from factory default

Exception Flow [Ex1]:

1. no backup settings available
2. call alternative flow [A1]

Exception Flow [Ex2]:

1. no factory default settings available
2. return error code

Post-conditions:

- valid data in persistence is restored and available

Notes:

Assumptions, issues and all other notes go here.

3.2.8 BB Import provided data

Description:
Allow to import data in the system.
Possibility of provided data to be imported for the 3 different types:
- Local and shared user related data
- Local and shared application related data
- Complete (user, application and all public data)
This usecase supports only the source handling of the system usecase "Import data".
Term "data" means files and key-values

Trigger: System usecase "Import data" request the source handling

Pre-conditions:

- System up and running
- import data pool available in the system

Main Flow - import complete:

1. locate import data pool (user, application, public data)
2. verify import data
3. replace current data with verified backup data

Alternative Flow - import user related data:

1. locate import data pool (user related data)
2. verify import data
3. replace current data with verified import data

Alternative Flow - restore application related data:

1. locate import data pool (application related data)
2. verify import data
3. replace current data with verified import data

Exception Flow [Ex1]:

1. locate import data
2. no import data available
3. do nothing, return error code

Post-conditions:

- provide verified data to application

Notes:

the import data pool is created by a product specific module which is able to import the data in the system. The import in the Persistence data is done based on the define structure, similar to backup created.

3.2.9 BB Provide data for export

Description:
Prepare data to allow its export outside from the system.
Possibility to provide data to export for the 3 different types:
- Local and shared user related data
- Local and shared application related data
- Complete (user, application and all public data)
This usecase supports only the source handling of the system usecase "Export data".
Term "data" means files and key-values

Trigger: The preparation of the data for export can be started by the user or by software loading and diagnostics

Pre-conditions:

- System up and running

Main Flow complete export:

1. locate the shared and application data
2. identification of the file to export (databases and files)
3. synchronize the cached data in the system
4. if requested a copy of the file is created to assume to get a synchronized export. (e.g. wt-files)
5. return the list of all the files to be exported .

Alternative Flow application backup (application recovery point):

1. locate the application data
2. identification of the file to export (databases and files)
3. synchronize the cached data in the system
4. if requested a copy of the file is created to assume to get a synchronized export. (e.g. wt-files)
5. return the list of all the files to be exported .

Alternative Flow application backup (user recovery point):

1. locate the user related data
2. identification of the file to export (databases and files)

3. synchronize the cached data in the system
4. If requested a copy of the file is created to assume to get a synchronized export. (e.g. wt-files)
5. Return the list of all the files to be exported.

Exception Flow [Ex1]:

1. no more memory left on memory device
2. notification to system health monitor

Post-conditions:

- list of the files to export is provided for the system level usecase "export data"

Notes:

Assumptions, issues and all other notes go here.

3.2.10 BB Restore from backup

Description:
Restore the data from a previously created backup.
This functionality is the way to recover the system to a defined state.
The restore can be started by the user or by software loading and diagnostics
Term "data" means files and key-values

Trigger: restore triggered by user or software loading and diagnostics

Pre-conditions:

- System up and running
- backup available in the system

Main Flow - restore complete:

1. locate backup data (user, application, public data)
2. verify backup data
3. pass verified data to requested application
4. replace current data with verified backup data

Alternative Flow - restore user related data:

1. locate backup data (user related data)
2. verify backup data
3. pass verified data to requested application
4. replace current data with verified backup data

Alternative Flow - restore application related data:

1. locate backup data (application related data)
2. verify backup data
3. pass verified data to requested application
4. replace current data with verified backup data

Exception Flow [Ex1]:

1. locate backup data
2. no backup data available
3. do nothing, return warning

Post-conditions:

- provide verified data to application

Notes:

the restore from backup for a complete system allows to return to a defined recovery state of the system

3.2.11 BB Restore to factory default

Description:
Current values must be replaced by the factory default settings, the node and user data are set to the factory defaults available in the system.
The restore can be started by the user or by software loading and diagnostics.
Term "data" means files and key-values

Trigger: restore triggered by user or software loading and diagnostics

Pre-conditions:

- System up and running

Main Flow - restore complete:

1. locate factory default data (user, application, public data)
2. verify backup data
3. replace current data with verified backup data

Exception Flow [Ex1]:

1. locate default factory settings
2. no default settings available
3. return error code

Post-conditions:

- provide verified data to application

Notes:

The restore from to factory default is executed on complete system.

During the installation of an application the corresponding factory defaults have to be provided.

3.2.12 BB Restore to user default

Description:
Current user values must be replaced by the user adjusted default settings.
The restore can be started by the user or by software loading and diagnostics
Term "data" means files and key-values

Trigger: restore triggered by user or software loading and diagnostics

Pre-conditions:

- System up and running

Main Flow - restore complete:

1. locate user related default data (for all users, all applications, public data)
2. verify user related default data
3. replace current data with verified user related default data for existing users

Alternative Flow - restore one user to user default:

1. locate user related default data (for a specific user, all applications, public data)
2. check if user already exists in system, if not create structure
3. verify user related default data

4. replace current data with verified user related default data

Alternative Flow - restore application user related data to user default:

1. locate user related default data (for all user, one application, public data)
2. check if user already exists in system, if not create structure
3. verify user related default data
4. replace current data with verified user related default data

Alternative Flow - restore one user in related user application data to user default:

1. locate user related default data (for a specific user, one application, public data)
2. check if user already exists in system, if not create structure
3. verify user related default data
4. replace current data with verified user related default data

Exception Flow [Ex1]:

1. locate adjusted user default settings
2. no default settings available
3. return error code

Post-conditions:

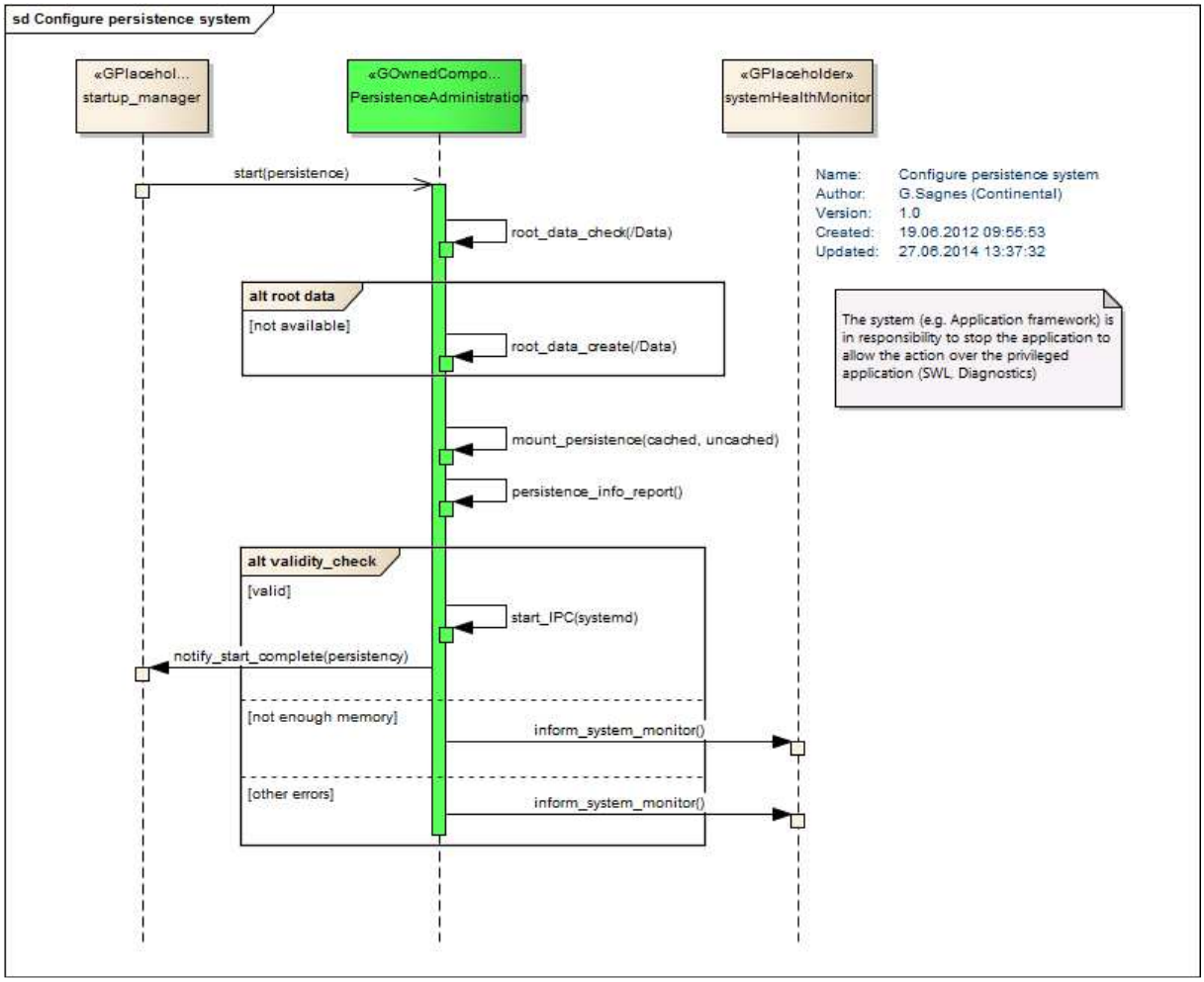
- provide verified data to application

Notes:

During the installation of an application the corresponding user defaults have to be provided.

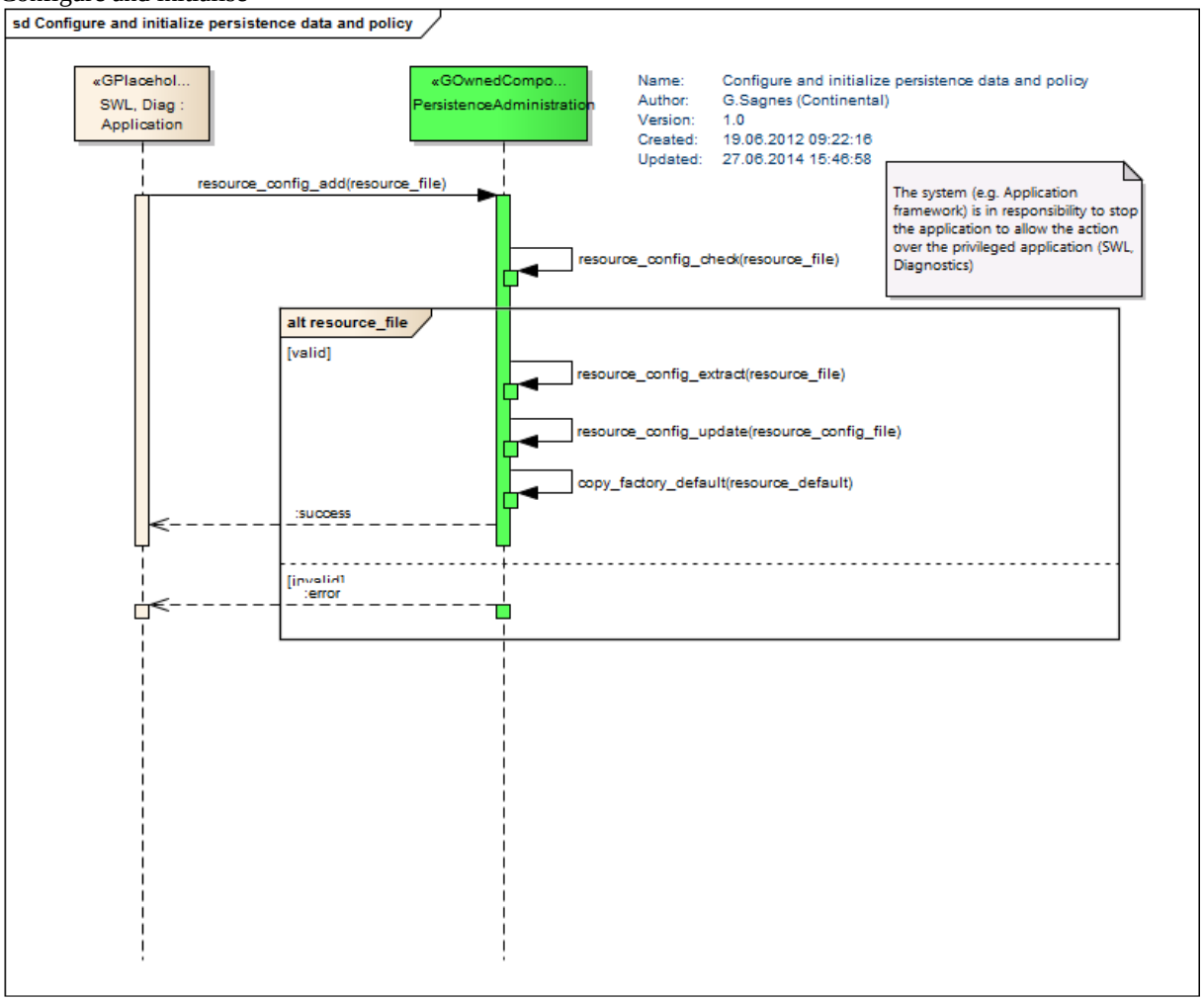
3.3 Persistence administration / collaboration

3.3.1 Configure persistence system

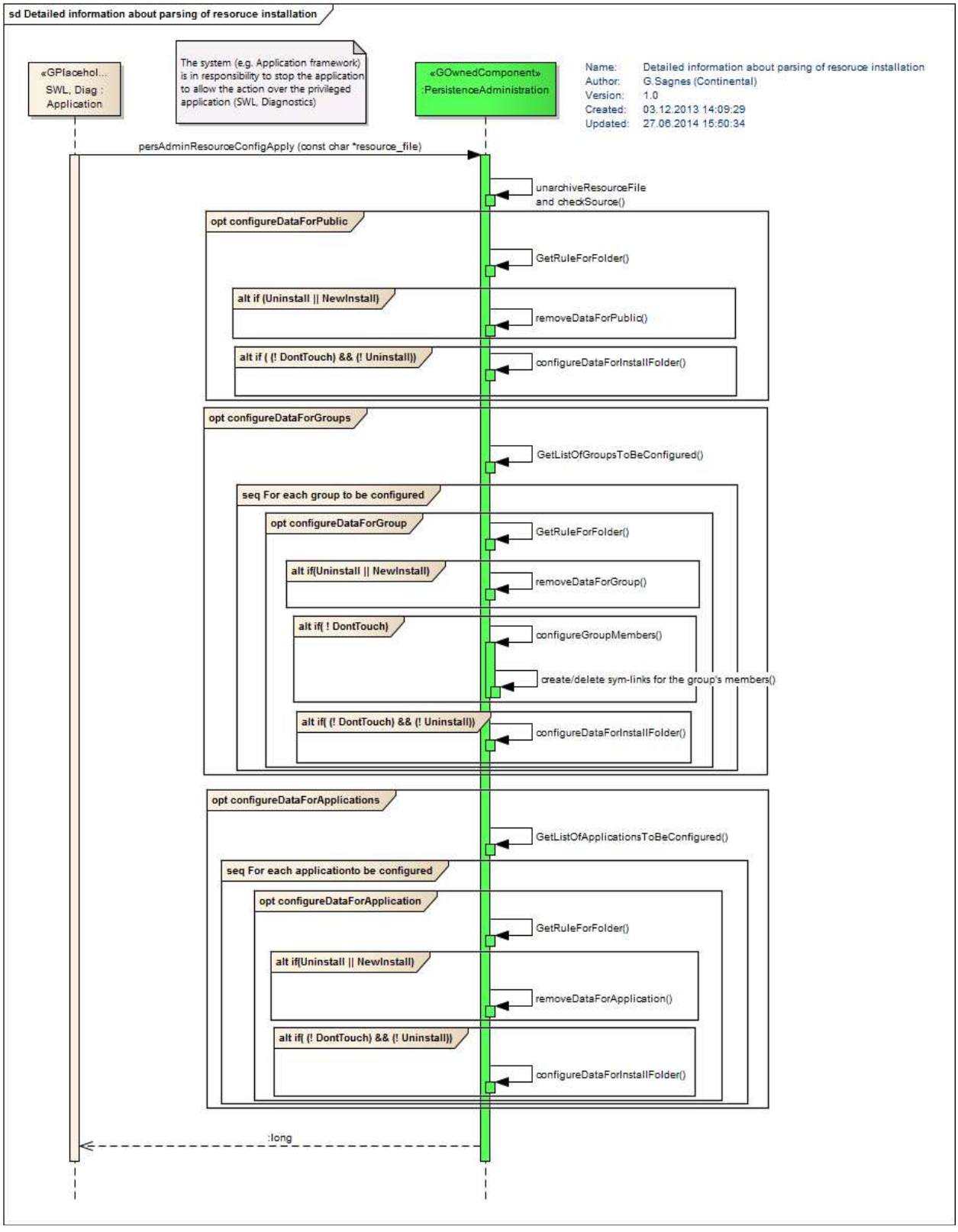


3.3.2 Configure and initialize persistence data and policy for new application

Configure and initialise

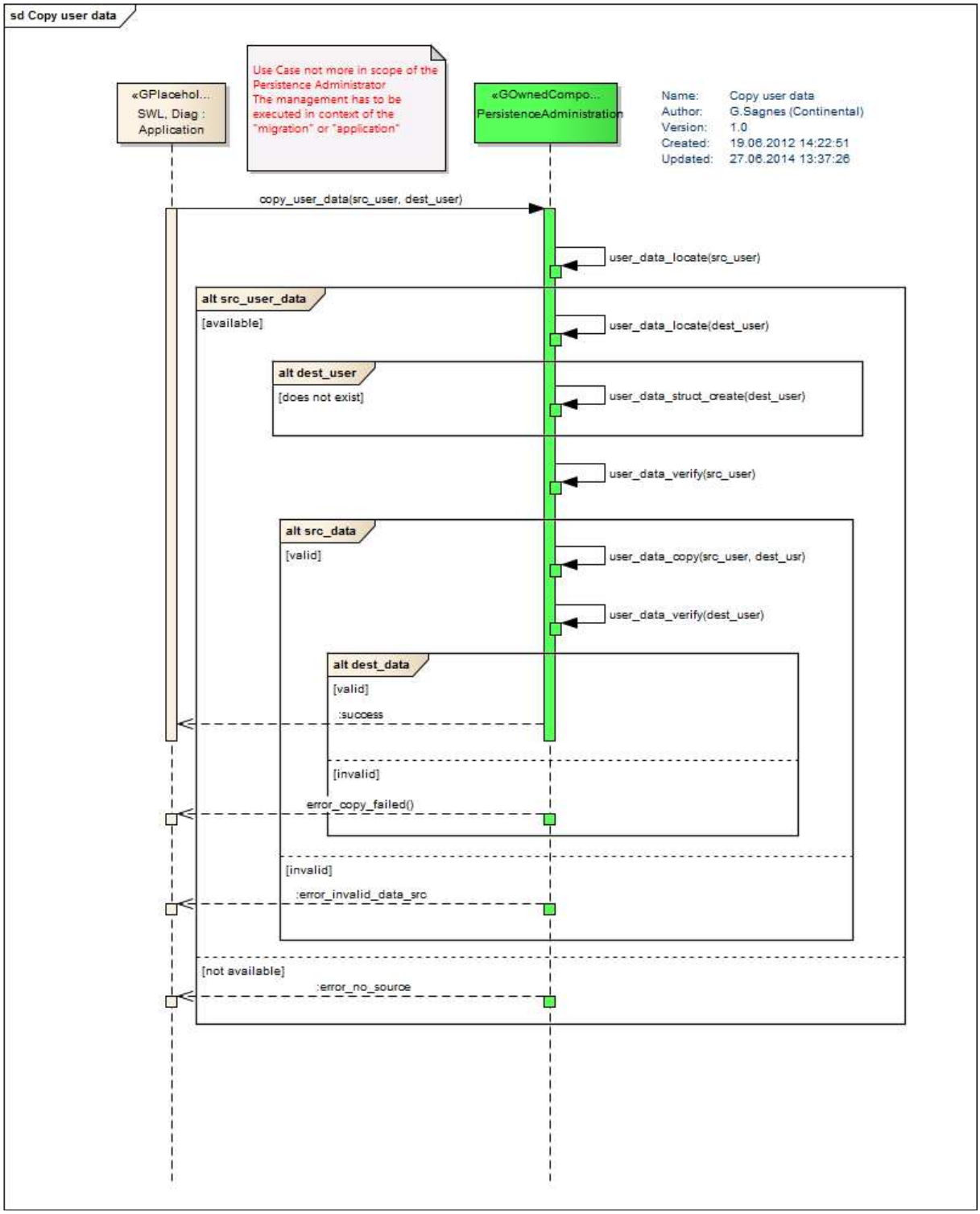


1 Details:

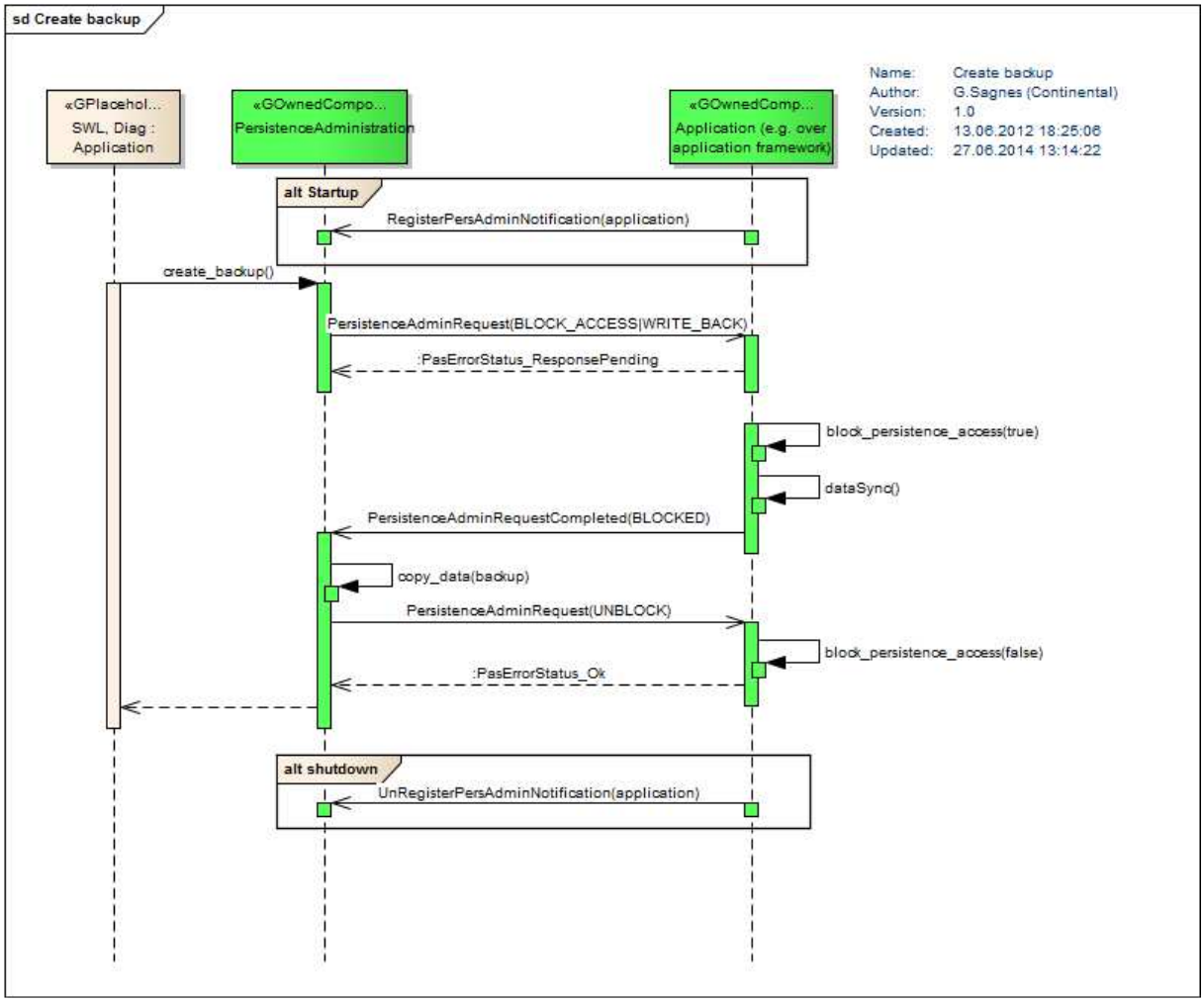


2
3

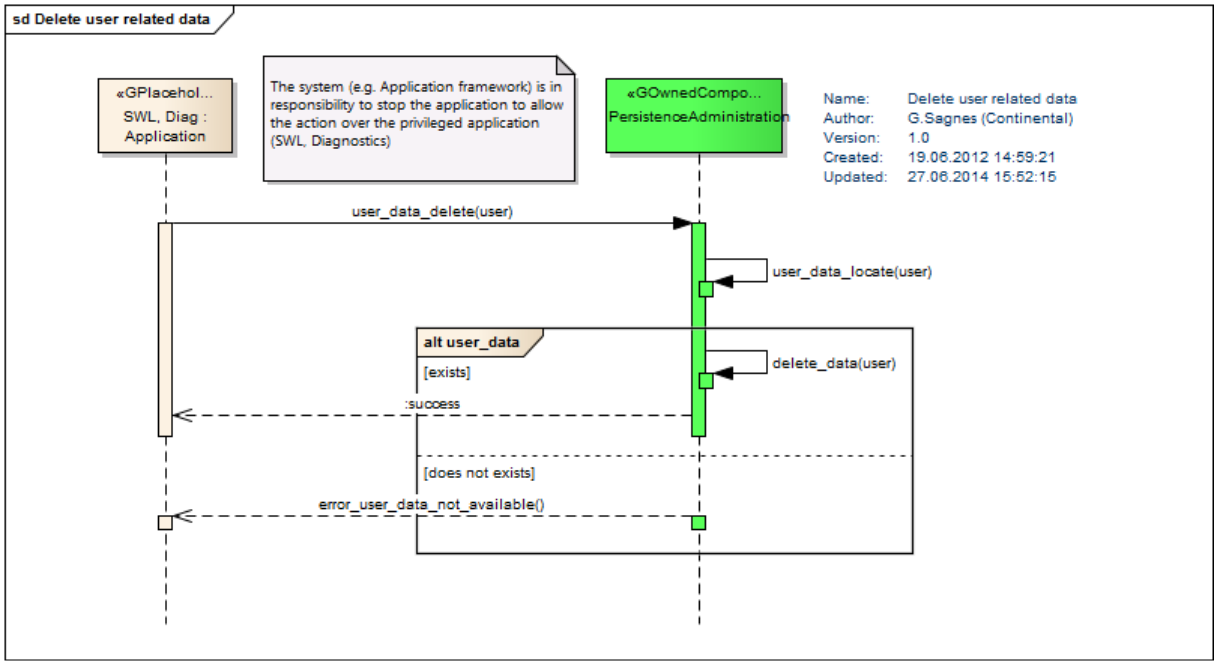
3.3.3 Copy user data



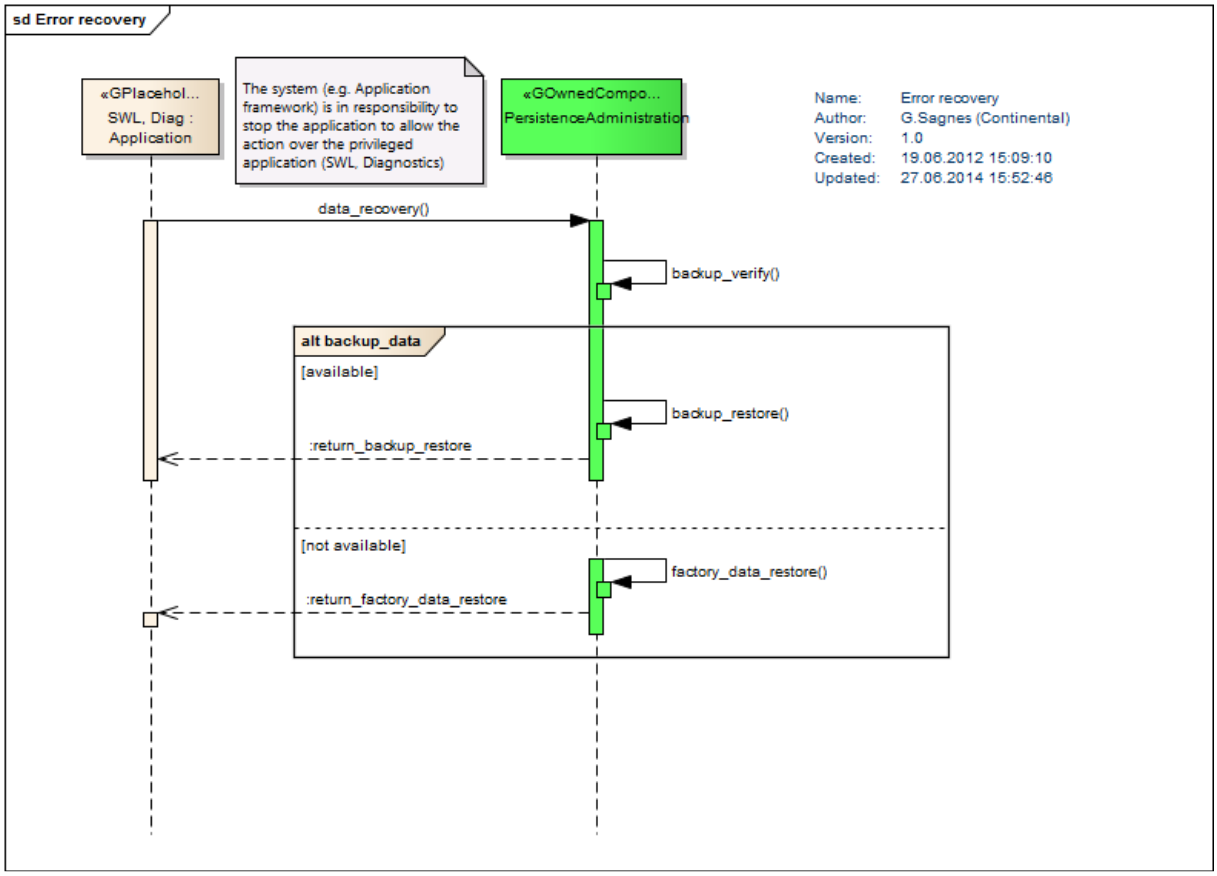
3.3.4 Create backup



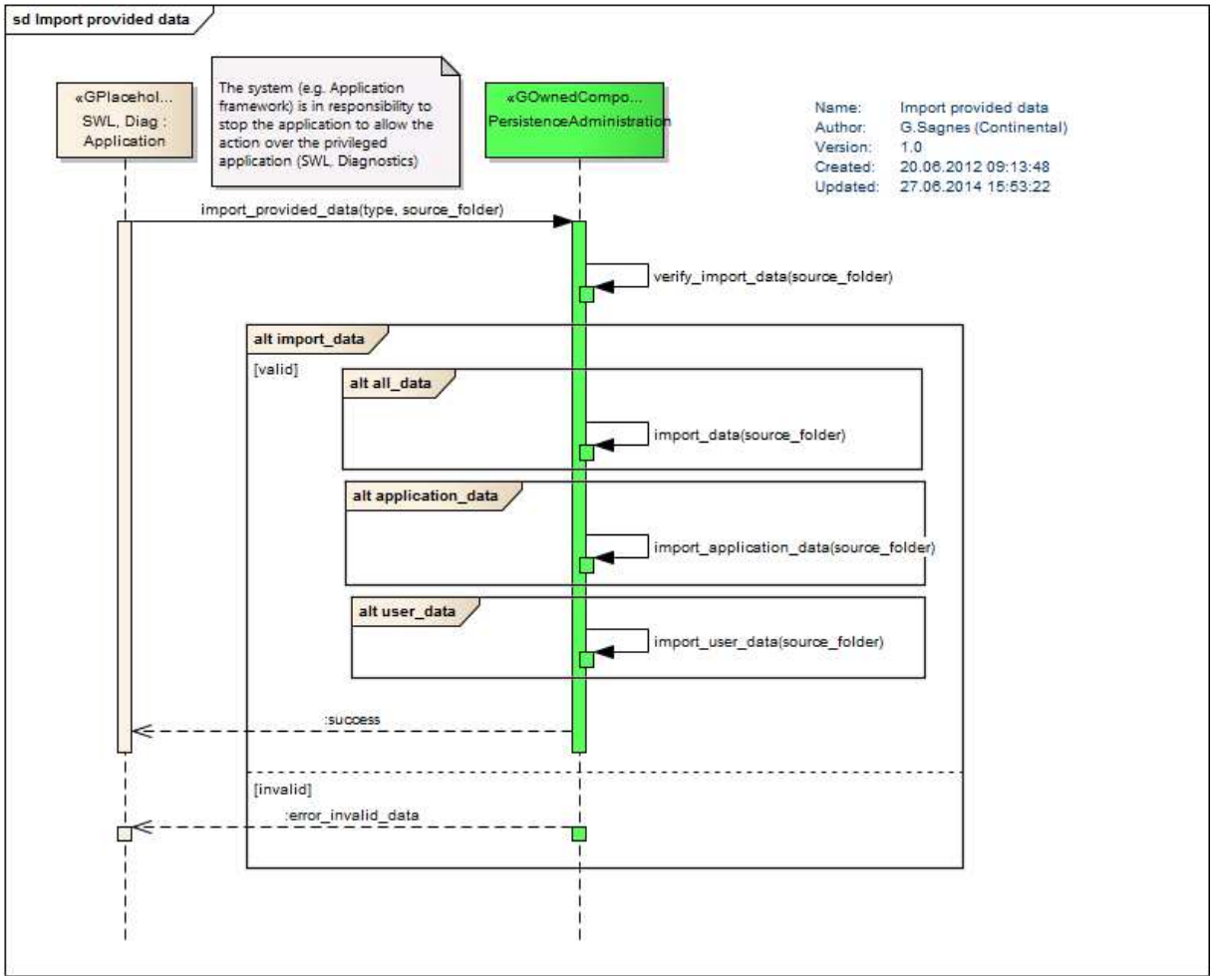
3.3.5 Delete user related data



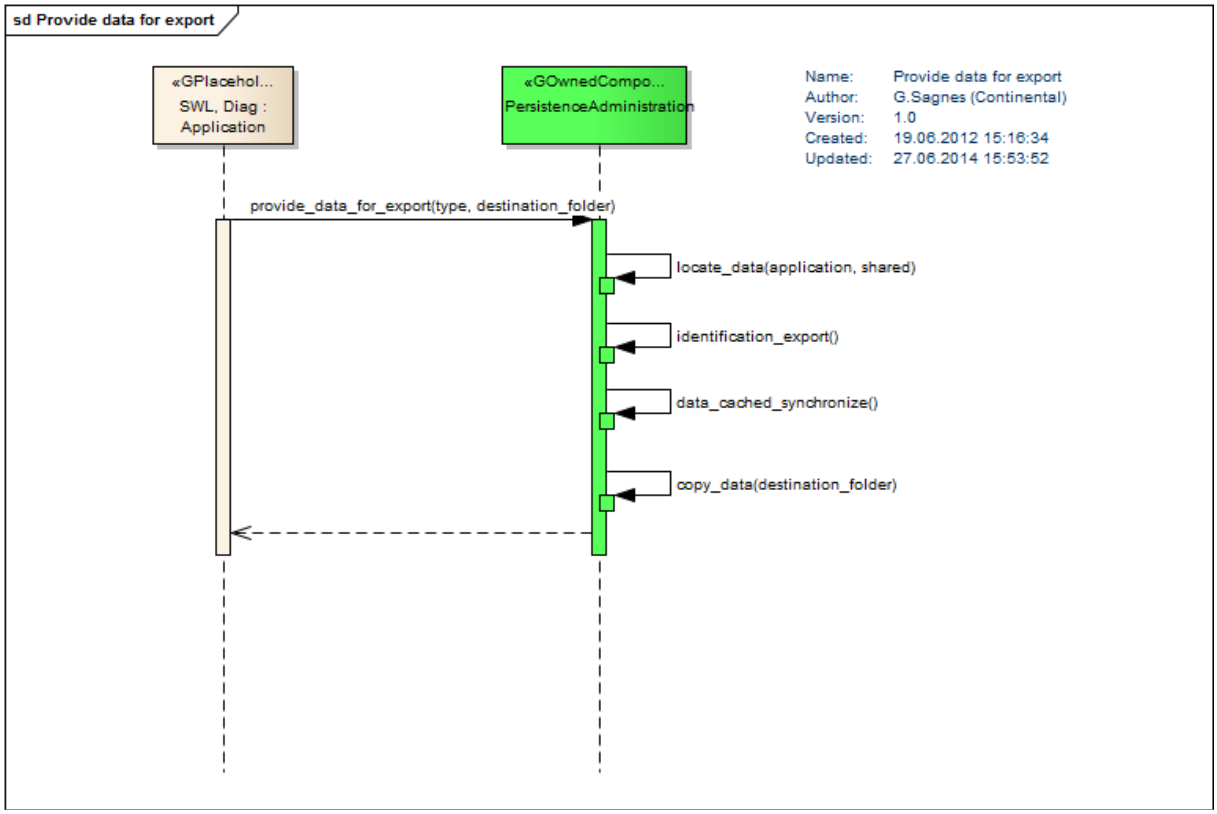
3.3.6 Error recovery



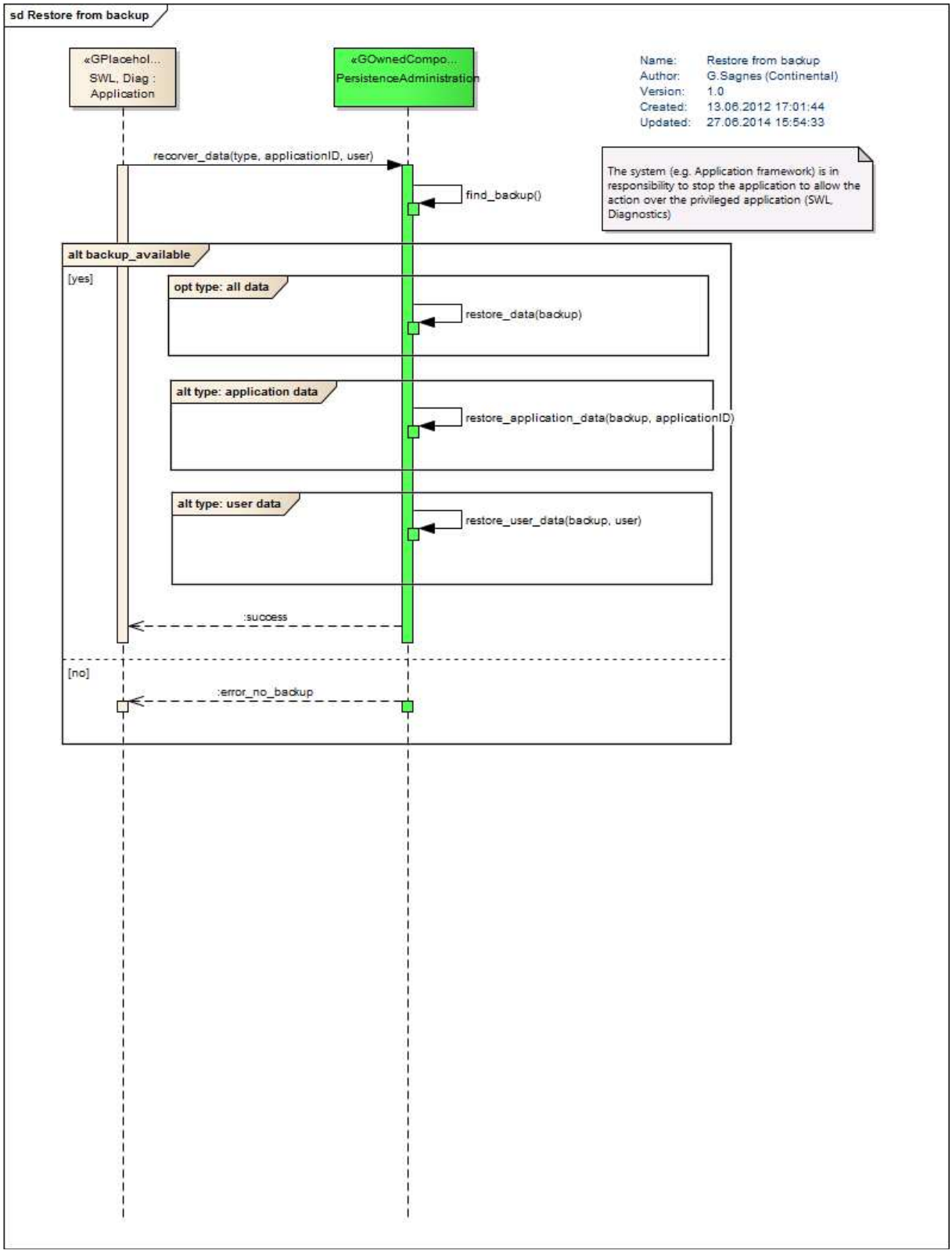
3.3.7 Import provided data



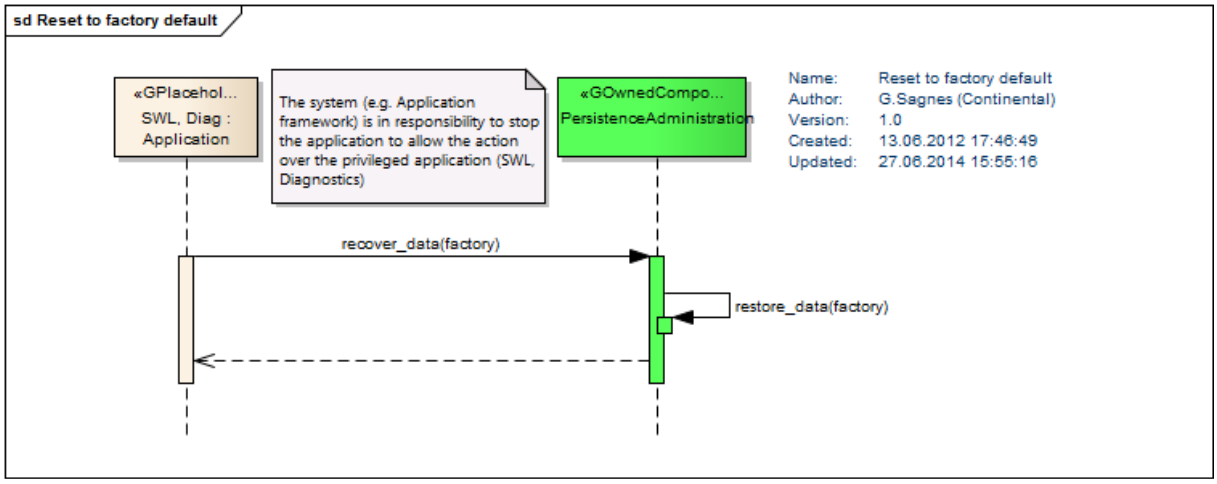
3.3.8 Provide data for export



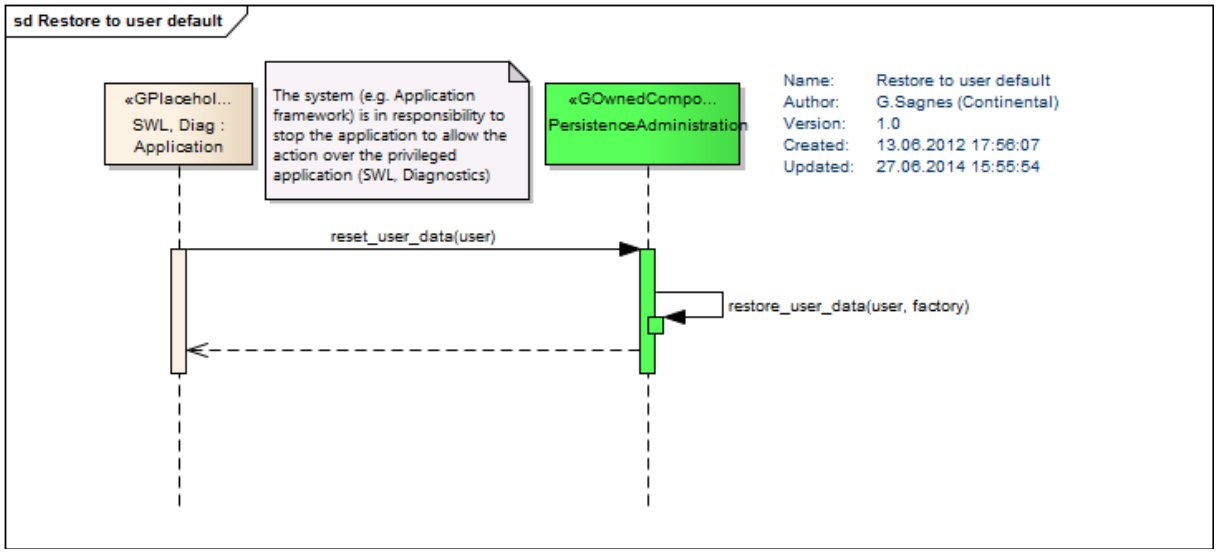
3.3.9 Restore from backup



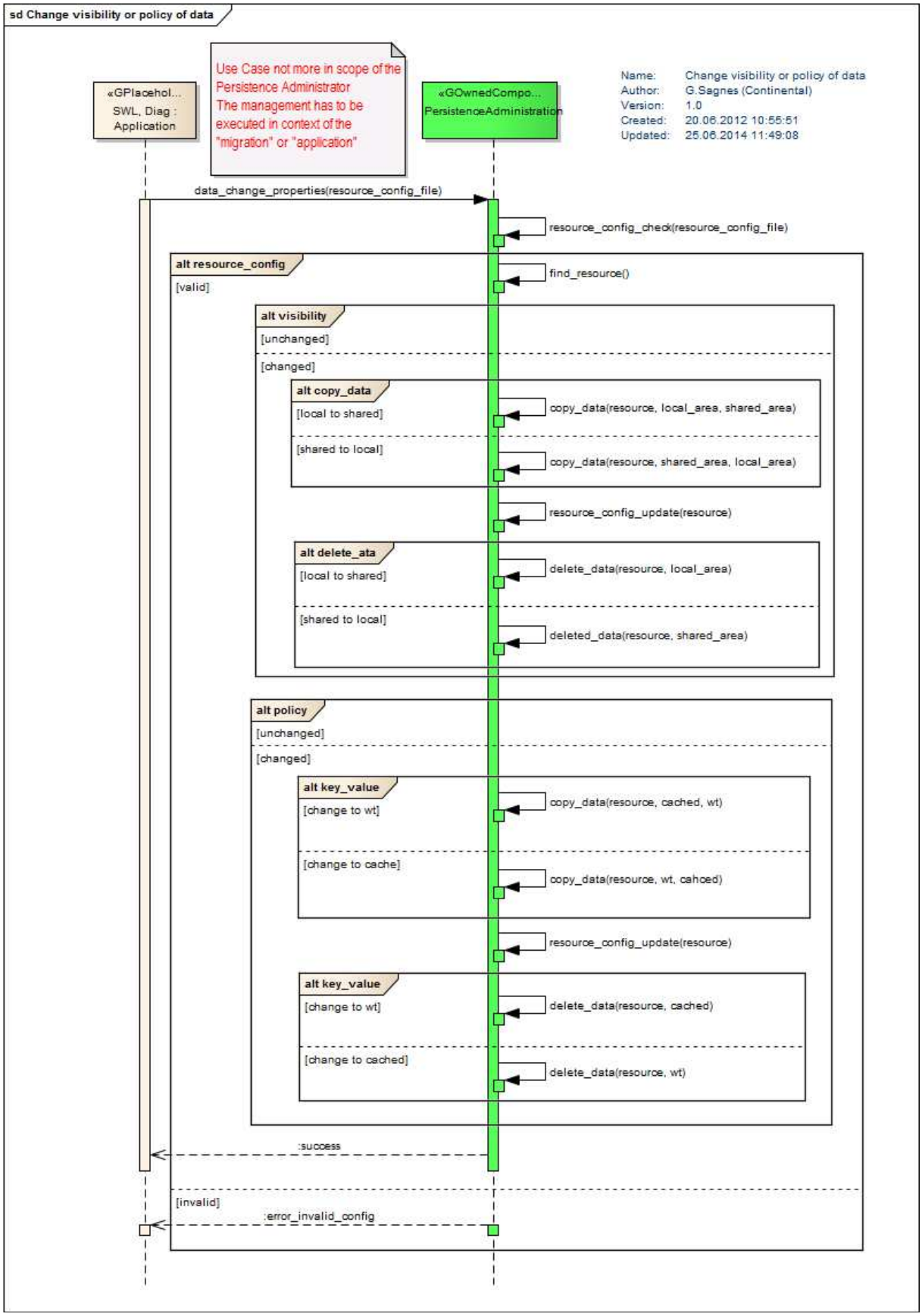
3.3.10 Restore to factory default



3.3.11 Restore to user default



3.3.12 Change visibility or policy of data



3.4 Persistence Component / Interfaces overview

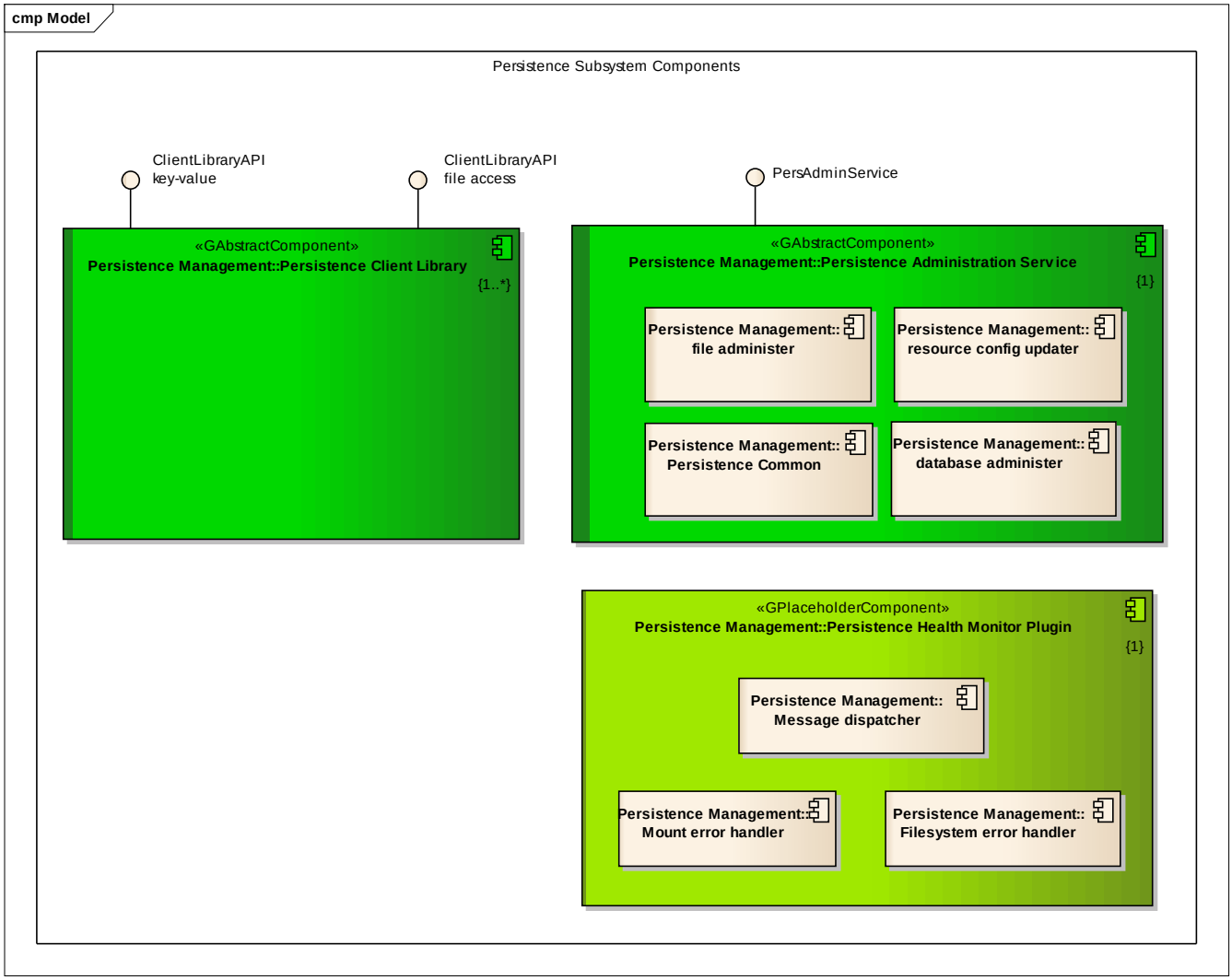


Fig. 6 – Persistence – Components / Interfaces overview

3.5 GENIVI Persistence Interfaces

The application will need only to use the Persistence Client Library (PCL) to access the data in the system.

3.5.1 Persistence Client Library

The responsibility of the Persistence Client library is:

- Shared data management
- Persistent data may be accessed by different applications
- Local data management
- Persistent data is accessible only by the "owned" application
- Specific data management
- Concept can be extended for custom solution for early, secure, factory …
- Local and shared file management
- Data can be stored in a file.

The file can be accessed by different applications or only the "owned" application.
The Persistence Client Library provides an API to application to read and write data persistently.

The API is divided in two parts a key-value API and a file API.

- The key-value API allows applications to store data in a schema-less way.
- The file API allows applications to store data directly into a file.

3.5.2 Persistence Administrator

The access to the Persistence Administrator Service (PAS) is limited to administrative application like software loading or housekeeping.

The responsibility of the Administration Service:

- Assume the file and folder structure management
- User data management
- Shared data management
- backup, import / export functionality

3.6 Persistence Administrator Public Interface

3.6.1 Administrator Interface overview (from Model)

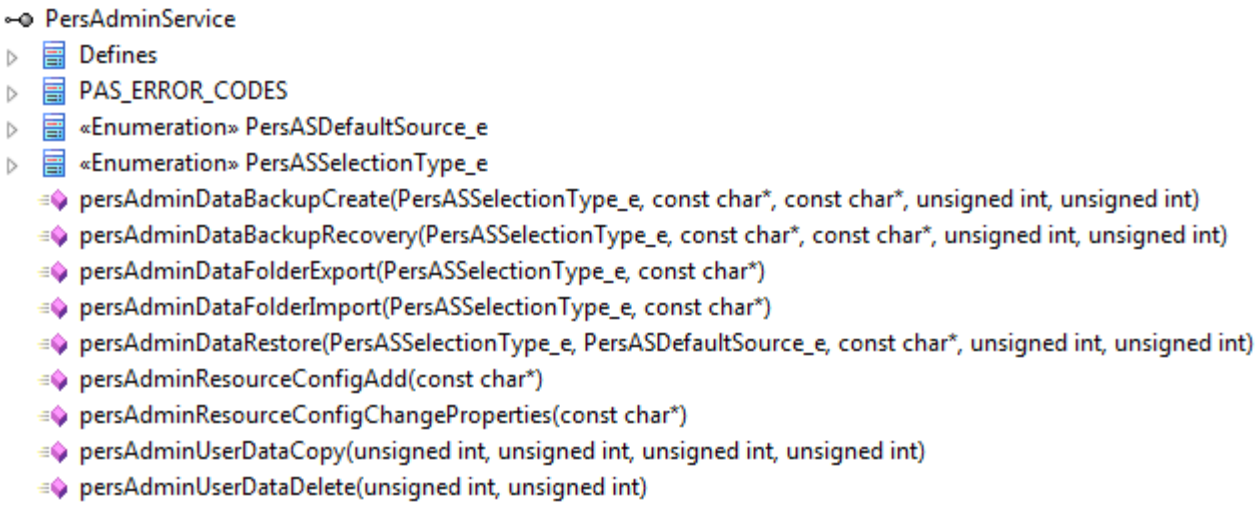


Fig. 7 – Persistence – Components / Interfaces overview

3.6.2 Interface documentation based on source code

Defines	
#define	PERSIST_ADMINSERVICE_INTERFACE_VERSION (0x02020000U)
Functions	
long	persAdminDataBackupCreate (PersASSelectionType_e type, const char *backup_name, const char *applicationID, unsigned int user_no, unsigned int seat_no)
	Allow creation of a backup on different level (application, user or complete)
long	persAdminDataBackupRecovery (PersASSelectionType_e type, const char *backup_name, const

	char *applicationID, unsigned int user_no, unsigned int seat_no)
	Allow recovery of from backup on different level (application, user or complete)
long	persAdminDataFolderExport (PersASSelectionType_e type, const char *dst_folder)
	Allow to identify and prepare the data to allow an export from system.
long	persAdminDataFolderImport (PersASSelectionType_e type, const char *src_folder)
	Allow the import of data from specified folder to the system respecting different level (application, user or complete)
long	persAdminResourceConfigAdd (const char *resource_file)
	Allow to extend the configuration for Persistence of data from specified level (application, user). Used during new Persistence entry installation.
long	persAdminResourceConfigChangeProperties (const char *resource_file)
	Allow the modification of the resource properties from data (key-values and files)
long	persAdminUserDataCopy (unsigned int src_user_no, unsigned int src_seat_no, unsigned int dest_user_no, unsigned int dest_seat_no)
	Allow the copy of user related data between different users.
long	persAdminUserDataDelete (unsigned int user_no, unsigned int seat_no)
	Delete the user related data from Persistence containers.
long	persAdminDataRestore (PersASSelectionType_e type, PersASDefaultSource_e defaultSource, const char *applicationID, unsigned int user_no, unsigned int seat_no)
	Allow restore of values from default on different level (application, user or complete)

1

4 Resource Installation File

4.1 Overview

The management of the key-values, files and folder are managed over a specific installation process. This action is based on the Resource Installation File (RIF).

The RIF will be used in the context of the API:

```
long persAdminResourceConfigAdd(const char* resource_file)
```

The Interface to allow the installation of the Resource of the system is kept simple and transparent. A possible adaptation of the format used can be easily done.

Format offers enough flexibility to allow :

- Installation of new application's data
- Update/uninsttall of applications' data (as a whole)
- Install/update/uninstall of individual resources for an application
  e.g. install a new key for an application, while the existent keys are left unchanged
  or update an application's data as a whole, but leaving untouched a certain key
- configuration of single or many applications
- special needs of System Configurator
- partial update (use of masks) of resources (only key type)

4.2 Data organization

The Resource Installation File (RIF) is a dedicated compressed folder structure:
In the root folder of the RIF – 3 folders represent the deployed persistence environment:

1. Apps
   - installation data folders – one per application
     `installRules.json`
     list the applications to be affected by the installation
     for each application, the type of install is specified
2. groups
   - installation data folders - one per group
     follow the naming convention
     `installRules.json`
     list the groups to be affected by the installation
     for each group, the type of install is specified
3. public
   - public – installation data folder
     to maintain the same structure as for the installation data folders for applications and groups
     `installRules.json`
     specifies the type of installation for the public data

The distribution of the resource and content is defined basically inside installRules.json and will be described in next chapter.

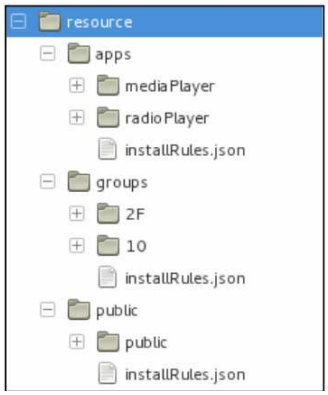


Fig. 8 – Content Resource Installation File (RIF)

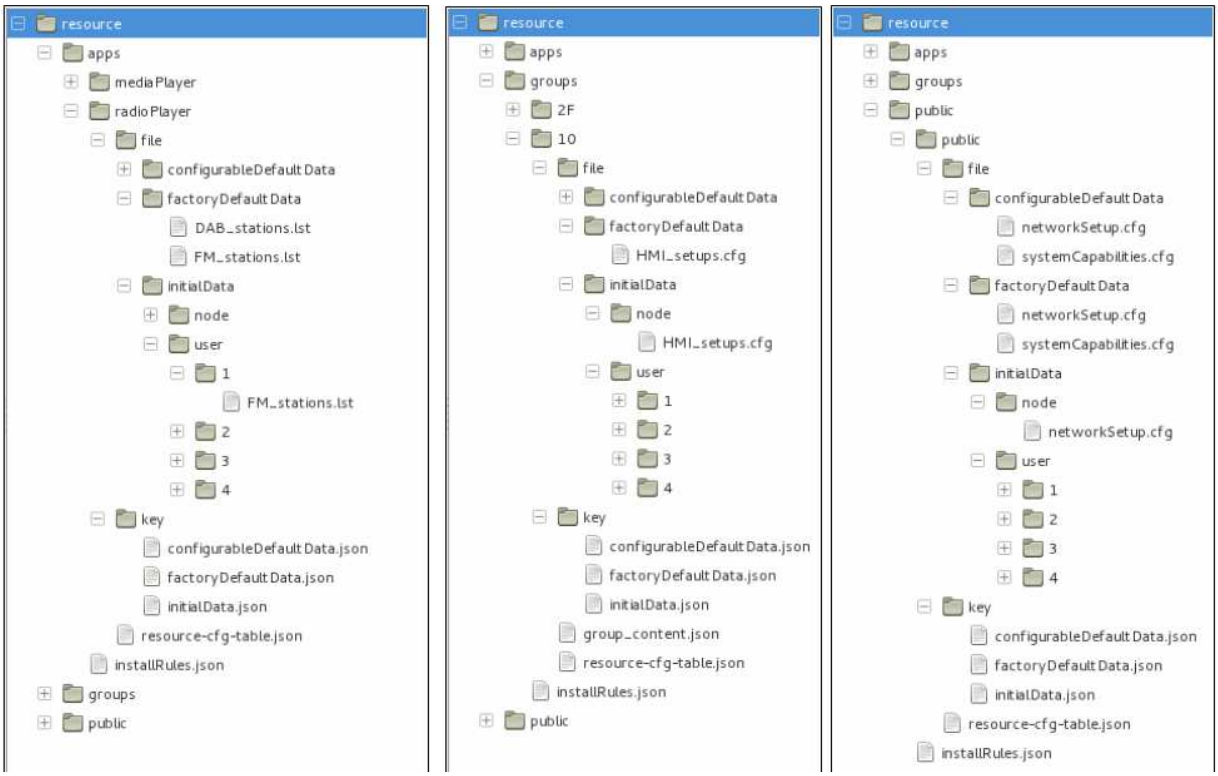


Fig. 9 – RIF Details

4.3 installRules.json format definition

It allows the clear specification of what actions to be performed for each application or group

Root *installRules.json*

```
{
    "[APPLICATIONNAME]":"[RULE]",
    …
}
```

PersAdminCfgInstallRules_NewInstall
Application/group data is overwritten (eventual existent data is deleted)

PersAdminCfgInstallRules_Uninstall
deletes application/group data
Consistency must be assured (if an application has to be uninstalled, it has also to be removed from all the groups it was part of)

PersAdminCfgInstallRules_DontTouch
Don't touch the application's or group's data

PersAdminCfgInstallRules_UpdateAll (merge) :
New resourceIDs are installed
resourceIDs no longer available in the new RCT are uninstalled
Data for resourceIDs available in both old and new RCT is merged (new overwrite old data)
Covers also the change of policy !!!

PersAdminCfgInstallRules_UpdateAllSkipDefaultFactory
PersAdminCfgInstallRules_UpdateAllSkipDefaultConfig
PersAdminCfgInstallRules_UpdateAllSkipDefaultAll
PersAdminCfgInstallRules_UpdateDefaultFactory
PersAdminCfgInstallRules_UpdateDefaultConfig
PersAdminCfgInstallRules_UpdateDefaultAll
PersAdminCfgInstallRules_UninstallNonDefault
PersAdminCfgInstallRules_UpdateSetOfResources
When only one ore a few resources need to be updated

Additional:
Exceptions are defined in installExceptions.json in each app/group's install folder

Affected data and allowed exceptions for each install rule		Default action on resource's data	Affected data				Valid exceptions		
			Factory default	Config default	Non-default	Cached (not managed over RCT)	update	dontTouch	delete
Install Rule per app/group	new-install	update	x	x	x	x	-	-	-
	un-install	delete	x	x	x	x	-	-	-
	dont-touch	-	-	-	-	-	-	-	-
	update-all	update	x	x	x	-	-	x	-
	update-all-skip-default-factory	update	-	x	x	-	-	x	-
	update-all-skip-default-config	update	x	-	x	-	-	x	-
	update-all-skip-default-all	update	-	-	x	-	-	x	-
	update-default-factory	update	x	-	-	-	-	x	-
	update-default-config	update	-	x	-	-	-	x	-
	update-default-all	update	x	x	-	-	-	x	-
	update-set-of-resources	exception	x	x	x	-	x	-	x
	uninstall-non-default	delete	-	-	x	-	-	x	-

Tab. 2 – Affected data and allowed exceptions for each install rule

4.4 resourceConfiguration.json format definition

This file is in responsibility to define the resources for the persistence (e.g. policy, storage, size…)

Root *resourceConfiguration.json*

{

```
    "config_appl" : “[APPLICATIONNAME]",
    "version" : “[VERSION]",
    "resources" : {
        “[ENTRYNAME]" : {
                "policy" : “[POLICY]",
                "permission" : “[PERMISSION]",
                "storage" : “[STORAGE]",
                “type”:”[TYPE]”,
                "max_size" : “[MAXSIZE]",
                "responsible" : “[APPLICATIONNAME]",
                “customPlugin” : [PLUGINNAME],
                "customID" : “[HASHVALUE]"
        },
    …
}
```

[POLICY]: *NA*, *cached*, *writethrough*
[PERMISSION]: *WO*, *RO*, *RW*
[STORAGE]: *local*, *shared*, *hwinfo*, *early*, *vlow*, *secured*, …
[PLUGINNAME]: *name of the plugin library*
[TYPE]: *file*, *key-value*
[STORAGE]: *an extension is possible there, where by a specific plug in library may be added as separate parameter*

4.5 factoryDefaultData.json format definition

This file is in responsibility to initialize the default values for the installed resources.

Root *factoryDefaultData.json*

```
{
    "config_appl" : “[APPLICATIONNAME]",
    "version" : “[VERSION]",
    "resources" : {
        “[ENTRYNAME]" : {
                “size" : “[SIZE]",
                “data" : “[DATA]",
        },
    …
    }
}
```

“[SIZE]“ is the real size of the default data not from the string but of the resulting binary buffer,
“[DATA]“ is a hexadecimal coded string of the buffer content

```
        "ERG_TEST_APPL1" : {
                "size" : "16",
                "data" : “000102030405060708090A0B0C0D0E0F"
        }
```

5 Acronyms

API	application programming interface
BB	Black Box (use case definition)
ECU	Electronic Control Unit
FS	File System
FUSE	File system in Userspace
MTD	Memory Technology Devices
NAND	Flash memory (s. **http://en.wikipedia.org/wiki/Flash_memory**)
NOR	Flash memory (s. **http://en.wikipedia.org/wiki/Flash_memory**)
OEM	Original Equipment Manufacturer
PAS	Persistence administration Service
PCL	Persistenc e Client Library
RIF	Resource Installation File – use for installation by the PAS
SW	Software
SWL	Software Loading – Application used in context of the Software Management
VFS	Virtual File System (s. **http://en.wikipedia.org/wiki/Virtual_file_system**)
UBIFS	Unsorted Block Image File System (s. **http://en.wikipedia.org/wiki/UBIFS**)
WT	Write Through (not cached managed)

Tab. 3 – Acronyms

6 Indices

This chapter lists all illustrations, tables and literature used in this document.

6.1 Illustrations

The following illustrations have been used in this document:

6.2 Tables

The following tables have been used in this document:

7 Open points

Tab. 4 – Open Items